

Description of STM32F0 HAL and low-layer drivers

Introduction

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

STM32Cube includes:

- [STM32CubeMX](#), a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as [STM32CubeF0](#) for STM32F0)
 - 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, USB, TCP/IP and Graphics.
 - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks).

The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors.

The HAL drivers are feature-oriented instead of IP-oriented. For example, the timer APIs are split into several categories following the IP functions, such as basic timer, capture and pulse width modulation (PWM). The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking enhances the firmware robustness. Run-time detection is also suitable for user application development and debugging.

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

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

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

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

It is compliant with MISRA C®:2004 standard.

This user manual is structured as follows:

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



1 General information

The [STM32CubeF0](#) MCU Package runs on STM32F0 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
CORDIC	Trigonometric calculation unit
CPU	Central processing unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
CSS	Clock security system
DAC	Digital to analog converter
DLYB	Delay block
DCMI	Digital camera interface
DFSDM	Digital filter sigma delta modulator
DMA	Direct memory access
DMAMUX	Direct memory access request multiplexer
DSI	Display serial interface
DTS	Digital temperature sensor
ETH	Ethernet controller
EXTI	External interrupt/event controller
FDCAN	Flexible data-rate controller area network unit
FLASH	Flash memory
FMAC	Filtering mathematical calculation unit
FMC	Flexible memory controller
FW	Firewall
GFXMMU	Chrom-GRC™
GPIO	General purpose I/Os
GTZC	Global TrustZone controller
GTZC-MPCBB	GTZC block-based memory protection controller
GTZC-MPCWM	GTZC watermark memory protection controller
GTZC-TZIC	GTZC TrustZone illegal access controller
GTZC-TZSC	GTZC TrustZone security controller

Acronym	Definition
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB host controller driver
HRTIM	High-resolution timer
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
ICACHE	Instruction cache
IRDA	Infrared data association
IWDG	Independent watchdog
JPEG	Joint photographic experts group
LCD	Liquid crystal display controller
LTDC	LCD TFT Display Controller
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
MCO	Microcontroller clock output
MDIOS	Management data input/output (MDIO) slave
MDMA	Master direct memory access
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND Flash memory
NOR	NOR Flash memory
NVIC	Nested vectored interrupt controller
OCTOSPI	Octo-SPI interface
OPAMP	Operational amplifier
OTFDEC	On-the-fly decryption engine
OTG-FS	USB on-the-go full-speed
PKA	Public key accelerator
PCD	USB peripheral controller driver
PSSI	Parallel synchronous slave interface
PWR	Power controller
QSPI	QuadSPI Flash memory
RAMECC	RAM ECC monitoring
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
SMARTCARD	Smartcard IC

Acronym	Definition
SMBUS	System management bus
SPI	Serial peripheral interface
SPDIFRX	SPDIF-RX Receiver interface
SRAM	SRAM external memory
SWPMI	Serial wire protocol master interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch sensing controller
UART	Universal asynchronous receiver/transmitter
UCPD	USB Type-C and power delivery interface
USART	Universal synchronous receiver/transmitter
VREFBUF	Voltage reference buffer
WWDG	Window watchdog
USB	Universal serial bus
PPP	STM32 peripheral or block

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: USART 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 (USART1, USART2...)
- All HAL APIs implement user-callback functions mechanism:
 - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
 - Peripherals interrupt events
 - Error events.
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

3.1 HAL and user-application files

3.1.1 HAL driver files

A HAL drivers are composed of the following set of files:

Table 2. HAL driver files

File	Description
<code>stm32f0xx_hal_ppp.c</code>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32f0xx_hal_adc.c, stm32f0xx_hal_irda.c, ...</i>
<code>stm32f0xx_hal_ppp.h</code>	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. <i>Example:stm32f0xx_hal_adc.h,stm32f0xx_hal_irda.h, ...</i>
<code>stm32f0xx_hal_ppp_ex.c</code>	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. <i>Example:stm32f0xx_hal_adc_ex.c,stm32f0xx_hal_flash_ex.c, ...</i>
<code>stm32f0xx_hal_ppp_ex.h</code>	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

File	Description
	Example: <i>stm32f0xx_hal_adc_ex.h,stm32f0xx_hal_flash_ex.h, ...</i>
<i>stm32f0xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
<i>stm32f0xx_hal.h</i>	<i>stm32f0xx_hal.c</i> header file
<i>stm32f0xx_hal_msp_template.c</i>	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32f0xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32f0xx_hal_def.h</i>	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
<i>system_stm32f0xx.c</i>	This file contains SystemInit() which 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 relocating the vector table in internal SRAM.
<i>startup_stm32f0xx.s</i>	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
<i>stm32f0xx_flash.icf (optional)</i>	Linker file for EWARM toolchain allowing mainly adapting the stack/heap size to fit the application requirements.
<i>stm32f0xx_hal_msp.c</i>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32f0xx_hal_conf.h</i>	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
<i>stm32f0xx_it.c/h</i>	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 <i>stm32f0xx_hal.c</i>) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
<i>main.c/h</i>	This file contains the main program routine, mainly: <ul style="list-style-type: none">• Call to HAL_Init()• assert_failed() implementation• system clock configuration• peripheral HAL initialization and user application code.

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

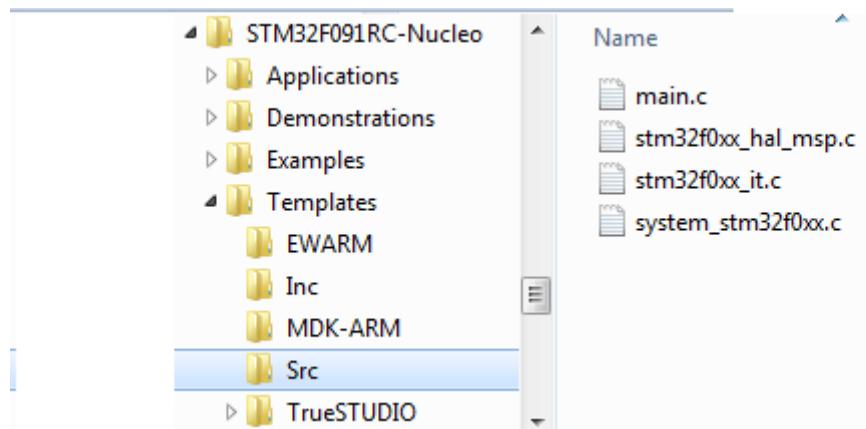
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Its features are the following:

- It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows configuring the CMSIS and HAL drivers accordingly.

- It provides ready to use user files preconfigured as defined below:
 - HAL is initialized
 - SysTick ISR implemented for HAL_Delay()
 - System clock configured with the maximum 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 allows working with several IP instances simultaneously.

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

The peripheral handle is used for the following purposes:

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

An example of peripheral structure is shown below:

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

Note:

1. *The multi-instance feature implies that all the APIs used in the application are reentrant and avoid using global variables because subroutines can fail to be reentrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:*
 - Reentrant code does not hold any static (or global) non-constant data: reentrant functions can work with global data. For example, a reentrant interrupt service routine can grab a piece of hardware status to work with (e.g. 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 frame.*/
    uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
    uint32_t Parity; /*!< Specifies the parity mode. */
    uint32_t Mode; /*!< Specifies whether the Receive or Transmit mode is enabled or disabled.*/
    uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled or disabled.*/
    uint32_t OverSampling; /*!< Specifies whether the Over sampling 8 is enabled or disabled,
                           to achieve higher speed (up to fPCLK/8).*/
}UART_HandleTypeDef;
```

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

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

3.2.3 Specific process structures

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

Example:

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

3.3 API classification

The HAL APIs are classified into three categories:

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

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

- **Extension APIs:**

This set of API is divided into two sub-categories :

- **Family specific APIs:** APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t SingleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDiff);
```

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

```
#if defined(STM32F042xx) || defined(STM32F048xx) || defined(STM32F072xB) || defined(
STM32F078xx) || \
defined(STM32F091xC) || defined(STM32F098xx)
#endif /* STM32F042xx || STM32F048xx || STM32F072xB || STM32F078xx || */
/* STM32F091xC || STM32F098xx */
```

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 ⁽¹⁾
Family specific APIs		X
Device specific APIs		X

1. In some cases, the implementation for a specific device part number may change. In this case the generic API is declared as weak function in the extension file. The API is implemented again to overwrite the default function.

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.

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

3.4 Devices supported by HAL drivers

Table 5. List of devices supported by HAL drivers

IP/Module	STM32F030x6	STM32F030x8	STM32F030x6	STM32F070x8	STM32F030xC	STM32F031x6	STM32F051x8	STM32F071xB	STM32F091xC	STM32F042x6	STM32F072xB	STM32F048xx	STM32F058xx	STM32F078xx	STM32F098xx
stm32f0xx_hal.c	Yes														
stm32f0xx_hal_adc.c	Yes														
stm32f0xx_hal_adc_ex.c	Yes														
stm32f0xx_hal_can.c	No	Yes	Yes	Yes	Yes	No	Yes	Yes							
stm32f0xx_hal_cec.c	No	No	No	No	No	No	Yes								
stm32f0xx_hal_comp.c	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f0xx_hal_cortex.c	Yes														
stm32f0xx_hal_crc.c	Yes														
stm32f0xx_hal_crc_ex.c	Yes														
stm32f0xx_hal_dac.c	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f0xx_hal_dac_ex.c	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f0xx_hal_dma.c	Yes														
stm32f0xx_hal_flash.c	Yes														
stm32f0xx_hal_flash_ex.c	Yes														
stm32f0xx_hal_gpio.c	Yes														
stm32f0xx_hal_i2c.c	Yes														
stm32f0xx_hal_i2c_ex.c	Yes														
stm32f0xx_hal_i2s.c	No	No	No	No	No	Yes									
stm32f0xx_hal_irda.c	No	No	No	No	No	Yes									
stm32f0xx_hal_iwdg.c	Yes														
stm32f0xx_hal_msp_template.c	NA														
stm32f0xx_hal_pcd.c	No	No	Yes	Yes	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No
stm32f0xx_hal_pcd_ex.c	No	No	Yes	Yes	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No
stm32f0xx_hal_pwr.c	Yes														
stm32f0xx_hal_pwr_ex.c	Yes	No	No	No											
stm32f0xx_hal_rcc.c	Yes														
stm32f0xx_hal_rcc_ex.c	Yes														
stm32f0xx_hal_rtc.c	Yes														
stm32f0xx_hal_rtc_ex.c	Yes														
stm32f0xx_hal_smartcard.c	No	No	No	No	No	Yes									
stm32f0xx_hal_smartcard_ex.c	No	No	No	No	No	No	Yes								

IP/Module	STM32F030x6	STM32F030x8	STM32F070x6	STM32F070x8	STM32F030xC	STM32F031x6	STM32F051x8	STM32F071xB	STM32F091xC	STM32F042x6	STM32F072xB	STM32F048xx	STM32F058xx	STM32F078xx	STM32F098xx
stm32f0xx_hal_smbus.c	Yes														
stm32f0xx_hal_spi.c	Yes														
stm32f0xx_hal_tim.c	Yes														
stm32f0xx_hal_tim_ex.c	Yes														
stm32f0xx_hal_tsc.c	No	No	No	No	No	No	Yes								
stm32f0xx_hal_uart.c	Yes														
stm32f0xx_hal_uart_ex.c	Yes														
stm32f0xx_hal_usart.c	Yes														
stm32f0xx_hal_wwdg.c	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	Device specific
File names	<i>stm32f0xx_hal_ppp (c/h)</i>	<i>stm32f0xx_hal_ppp_ex (c/h)</i>	<i>stm32f0xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	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 STM32F0 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in the CMSIS header: *stm32f030x6.h*, *stm32f030x8.h*, *stm32f031x6.h*, *stm32f038xx.h*, *stm32f042x6.h*, *stm32f048xx.h*, *stm32f051x8.h*, *stm32f058xx.h*, *stm32f071xB.h*, *stm32f072xB.h*, *stm32f078xx.h*, *stm32f091xC.h* and *stm32f098xx.h*. The platform is selected by enabling the compilation switch in the compilation toolchain directive or in the *stm32f0xx.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 (e.g. **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* (e.g. *ADC_InitTypeDef*).
- The structure containing the Specific configuration parameters for the PPP peripheral are named *PPP_xxxConfTypeDef* (e.g. *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* (e.g. *HAL_TIM_Init()*).
- The functions used to reset the PPP peripheral registers to their default values are named *HAL_PPP_Delinit* (e.g. *HAL_TIM_Delinit()*).
- 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_ADC_Start()* 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
<code>_HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Enables a specific peripheral interrupt
<code>_HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Disables a specific peripheral interrupt
<code>_HAL_PPP_GET_IT (__HANDLE__, __INTERRUPT__)</code>	Gets a specific peripheral interrupt status
<code>_HAL_PPP_CLEAR_IT (__HANDLE__, __INTERRUPT__)</code>	Clears a specific peripheral interrupt status
<code>_HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)</code>	Gets a specific peripheral flag status
<code>_HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)</code>	Clears a specific peripheral flag status
<code>_HAL_PPP_ENABLE(__HANDLE__)</code>	Enables a peripheral
<code>_HAL_PPP_DISABLE(__HANDLE__)</code>	Disables a peripheral
<code>_HAL_PPP_XXXX (__HANDLE__, __PARAM__)</code>	Specific PPP HAL driver macro
<code>_HAL_PPP_GET_IT_SOURCE (__HANDLE__, __INTERRUPT__)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the `stm32f0xx_hal_cortex.c` file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example : `STATUS = XX | (YY << 16)` or `STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)`.
- The PPP handles are valid before using the `HAL_PPP_Init()` API. The init function performs a check before modifying the handle fields.

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

- The macros defined below are used:

- Conditional macro:

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

- Pseudo-code macro (multiple instructions macro):

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

3.5.3

HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- `HAL_PPP_IRQHandler()` peripheral interrupt handler that should be called from `stm32f0xx_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_MspDelInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8. Callback functions

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

3.6 HAL generic APIs

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

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

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

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

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

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

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

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

Table 9. HAL generic APIs

Function group	Common API name	Description
<i>Initialization group</i>	HAL_ADC_Init()	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DelInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
<i>IO operation group</i>	HAL_ADC_Start()	This function starts ADC conversions when the polling method is used

Function group	Common API name	Description
IO operation group	<code>HAL_ADC_Stop()</code>	This function stops ADC conversions when the polling method is used
	<code>HAL_ADC_PollForConversion()</code>	This function allows waiting 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.
	<code>HAL_ADC_Start_IT()</code>	This function starts ADC conversions when the interrupt method is used
	<code>HAL_ADC_Stop_IT()</code>	This function stops ADC conversions when the interrupt method is used
	<code>HAL_ADC_IRQHandler()</code>	This function handles ADC interrupt requests
	<code>HAL_ADC_ConvCpltCallback()</code>	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
Control group	<code>HAL_ADC_ConfigChannel()</code>	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	<code>HAL_ADC_AnalogWDGConfig</code>	This function configures the analog watchdog for the selected ADC
State and Errors group	<code>HAL_ADC_GetState()</code>	This function allows getting in run time the peripheral and the data flow states.
	<code>HAL_ADC_GetError()</code>	This function allows getting in run time the error that occurred during IT routine

3.7 HAL extension APIs

3.7.1 HAL extension model overview

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

The extension model consists of an additional file, `stm32f0xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32f0xx_hal_ppp_ex.h`) for a given part number.

Below an example based on the ADC peripheral:

Table 10. HAL extension APIs

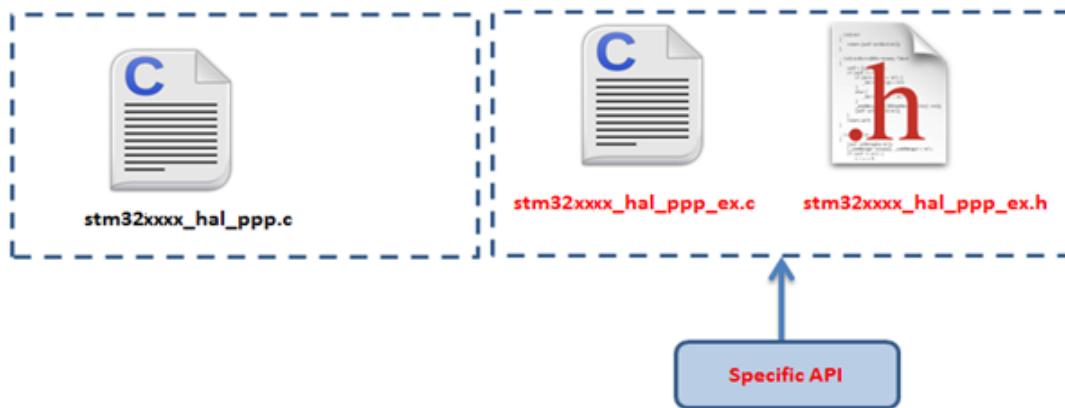
Function group	Common API name
<code>HAL_ADCEx_CalibrationStart()</code>	This function is used to start the automatic ADC calibration
<code>HAL_ADCEx_Calibration_GetValue()</code>	This function is used to get the ADC calibration factor
<code>HAL_ADCEx_Calibration_SetValue()</code>	This function is used to set the calibration factor to overwrite the automatic conversion result

3.7.2 HAL extension model cases

The specific IP 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 `stm32f0xx_hal_ppp_ex.c` extension file. They are named `HAL_PPPEX_Function()`.

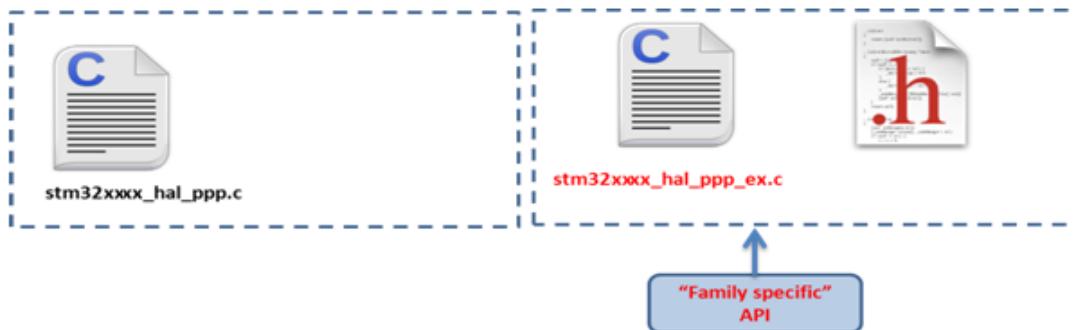
Figure 2. Adding device-specific functions

Example: `stm32f0xx_hal_rcc_ex.c/h`

```
#if defined(STM32F042xB) || defined(STM32F048xx) || \
defined(STM32F071xB) || defined(STM32F072xB) || defined(STM32F078xx) || \
defined(STM32F091xC) || defined(STM32F098xx)
void HAL_RCCEX_CRSConfig(RCC_CRSInitTypeDef *pInit);
void HAL_RCCEX_CRSSoftwareSynchronizationGenerate(void);
void HAL_RCCEX_CRSGetSynchronizationInfo(RCC_CRSSynchroInfoTypeDef *pSynchroInfo);
RCC_CRSStatusTypeDef HAL_RCCEX_CRSWaitSynchronization(uint32_t Timeout);
#endif /* STM32F042xB || STM32F048xx || */
/* STM32F071xB || STM32F072xB || STM32F078xx || */
/* STM32F091xC || STM32F098xx */
```

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 `stm32f0xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32f0xx_hal_conf.h` using the macro:

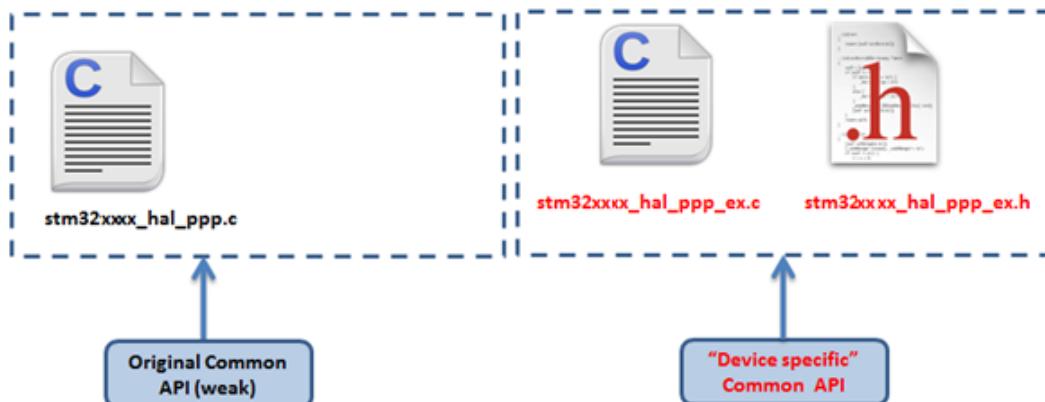
```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. Adding new peripherals

Example: stm32f0xx_hal_adc.c/h

Updating existing common APIs

In this case, the routines are defined with the same names in the stm32f0xx_hal_ppp_ex.c extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

Figure 5. Updating existing APIs

Updating existing data structures

The data structure for a specific device part number (e.g. 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.

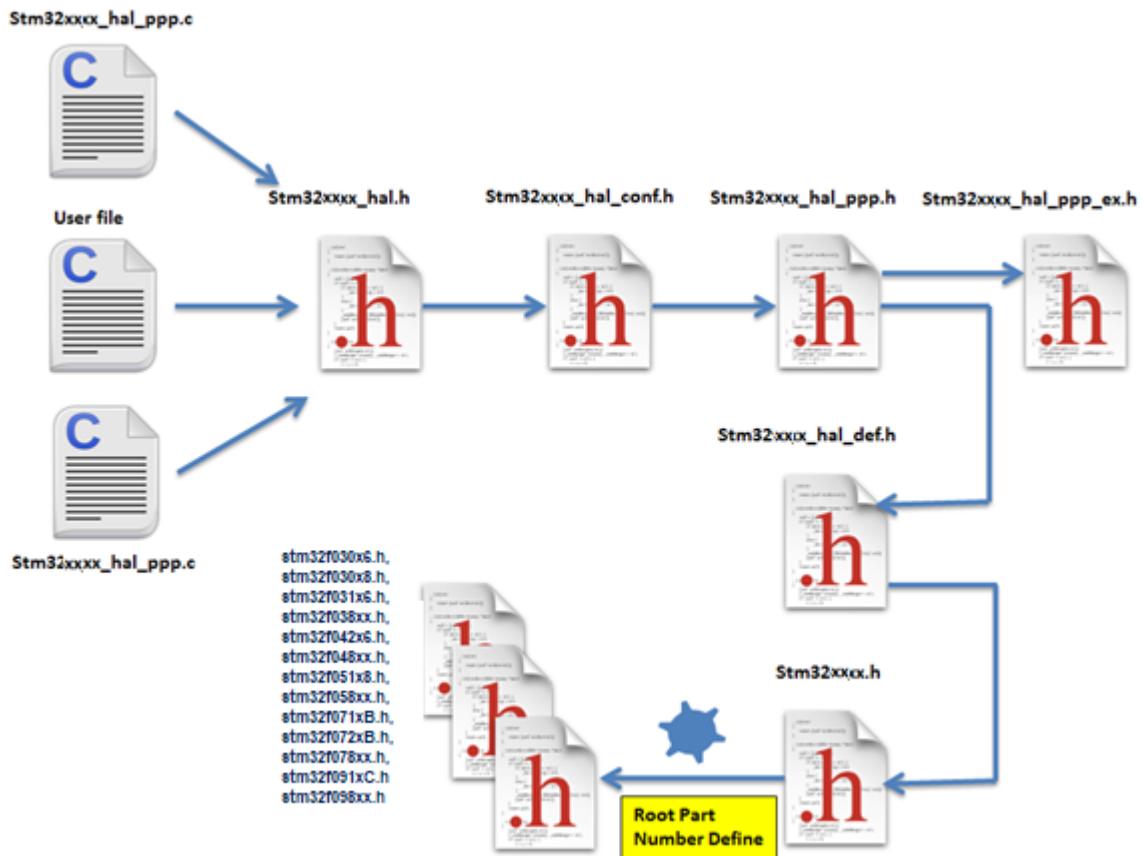
Example:

```
#if defined (STM32F072xB)
typedef struct
{
(...)
}PPP_InitTypeDef;
#endif /* STM32F072xB */
```

3.8 File inclusion model

The header of the common HAL driver file (stm32f0xx_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 6. 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 stm32f0xx_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 USE_HAL_USART_MODULE
#define USE_HAL_IRDA_MODULE
#define USE_HAL_DMA_MODULE
#define USE_HAL_RCC_MODULE
(...)
```

3.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in `stm32f0xx_hal_def.h`. The main common define enumeration is `HAL_StatusTypeDef`.

- **HAL Status**

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

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

- **HAL Locked**

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

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

In addition to common resources, the stm32f0xx_hal_def.h file calls the stm32f0xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:

- Declarations of peripheral registers and bits definition.
- Macros to access peripheral registers hardware (Write register, Read register...etc.).
- **Common macros**
 - Macro defining NULL

```
#ifndef NULL
#define NULL 0
#endif
```

- 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, stm32f0xx_hal_conf.h, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

Table 11. Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
HSI_STARTUP_TIMEOUT	Timeout for HSI start-up, expressed in ms	5000

Configuration item	Description	Default Value
LSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 (Hz)
LSE_STARTUP_TIMEOUT	Timeout for LSE start-up, expressed in ms	5000
HSI14_VALUE	Defines the value of the Internal High Speed oscillator for ADC expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	14 000 000 (Hz)
HSI48_VALUE	Defines the value of the Internal High Speed oscillator for USB expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	48 000 000 (Hz)
LSI_VALUE	Defines the value of the Internal Low Speed oscillator expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	40 000 (Hz)
VDD_VALUE	VDD value	3300 (mV)
USERTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE

Note: The `stm32f0xx_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.

Note: By default, the values defined in the `stm32f0xx_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 chapter 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 Clock

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 AHB and APB clock dividers
 - configures the number of Flash memory wait states
 - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (such as RTC, USB). In this case, the clock configuration is performed by an extended API defined in `stm32f0xx_hal_rcc_ex.c`:
`HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)`.

Additional RCC HAL driver functions are available:

- `HAL_RCC_DelInit()` Clock de-initialization function that returns clock configuration to reset state
- Get clock functions that allow retrieving various clock configurations (system clock, HCLK, PCLK1, PCLK2, ...)
- MCO and CSS configuration functions

A set of macros are defined in `stm32f0xx_hal_rcc.h` and `stm32f0xx_hal_rcc_ex.h`. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- `__HAL_PPP_CLK_ENABLE/ __HAL_PPP_CLK_DISABLE` to enable/disable the peripheral clock
- `__HAL_PPP_FORCE_RESET/ __HAL_PPP_RELEASE_RESET` to force/release peripheral reset

- `__HAL_PPP_CLK_SLEEP_ENABLE`/`__HAL_PPP_CLK_SLEEP_DISABLE` to enable/disable the peripheral clock during Sleep mode.

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 `stm32f0xx_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: <code>GPIO_PIN_x</code> or <code>GPIO_PIN_All</code> , where <code>x[0..15]</code>
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: <ul style="list-style-type: none">• <u>GPIO mode</u><ul style="list-style-type: none">– <code>GPIO_MODE_INPUT</code> : Input floating– <code>GPIO_MODE_OUTPUT_PP</code> : Output push-pull– <code>GPIO_MODE_OUTPUT_OD</code> : Output open drain– <code>GPIO_MODE_AF_PP</code> : Alternate function push-pull– <code>GPIO_MODE_AF_OD</code> : Alternate function open drain– <code>GPIO_MODE_ANALOG</code> : Analog mode• <u>External Interrupt mode</u><ul style="list-style-type: none">– <code>GPIO_MODE_IT_RISING</code> : Rising edge trigger detection– <code>GPIO_MODE_IT_FALLING</code> : Falling edge trigger detection– <code>GPIO_MODE_IT_RISING_FALLING</code> : Rising/Falling edge trigger detection• <u>External Event mode</u><ul style="list-style-type: none">– <code>GPIO_MODE_EVT_RISING</code> : Rising edge trigger detection– <code>GPIO_MODE_EVT_FALLING</code> : Falling edge trigger detection– <code>GPIO_MODE_EVT_RISING_FALLING</code> : Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: <code>GPIO_NOPULL</code> <code>GPIO_PULLUP</code> <code>GPIO_PULLDOWN</code>
Speed	Specifies the speed for the selected pins Possible values are: <code>GPIO_SPEED_FREQ_LOW</code> <code>GPIO_SPEED_FREQ_MEDIUM</code> <code>GPIO_SPEED_FREQ_HIGH</code>
Alternate	Peripheral to be connected to the selected pins. Possible values: <code>GPIO_AFx_PP</code> , where AFx: is the alternate function index and PPP: is the peripheral instance Example: use <code>GPIO_AF1_TIM2</code> to connect TIM2 IOs on AF1.

Structure field	Description
	<p>These values are defined in the GPIO extended driver, since the AF mapping may change between product lines.</p> <p>Note: Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.</p>

Please find below typical GPIO configuration examples:

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

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

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

- Configuring USART1 Tx (PA9, mapped on AF4) as alternate function:

```
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF4_USART1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

3.11.3

Cortex® NVIC and SysTick timer

The Cortex® HAL driver, `stm32f0xx_hal_cortex.c`, provides APIs to handle NVIC and SysTick. The supported APIs include:

- `HAL_NVIC_SetPriority()`
- `HAL_NVIC_EnableIRQ() / HAL_NVIC_DisableIRQ()`
- `HAL_NVIC_SystemReset()`
- `HAL_SYSTICK_IRQHandler()`
- `HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ() / HAL_NVIC_ClearPendingIRQ()`
- `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()`
- Wakeup pin configuration
 - `HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()`

- Low-power mode entry
 - HAL_PWR_EnterSLEEPMode()
 - HAL_PWR_EnterSTOPMode()
 - HAL_PWR_EnterSTANDBYMode()
- Backup domain configuration
 - HAL_PWR_EnableBkUpAccess() / HAL_PWR_DisableBkUpAccess()

3.11.5 EXTI

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

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

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

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

Table 13. Description of EXTI configuration macros

Macros	Description
PPP_EXTI_LINE_FUNCTION	Defines the EXTI line connected to the internal peripheral. Example: <code>#define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!<External interrupt line 16 Connected to the PVD EXTI Line */</code>
__HAL_PPP_EXTI_ENABLE_IT	Enables a given EXTI line Example: <code>__HAL_PWR_PVD_EXTI_ENABLE_IT()</code>
__HAL_PPP_EXTI_DISABLE_IT	Disables a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_DISABLE_IT()</code>
__HAL_PPP_EXTI_GET_FLAG	Gets a given EXTI line interrupt flag pending bit status. Example: <code>__HAL_PWR_PVD_EXTI_GET_FLAG()</code>
__HAL_PPP_EXTI_CLEAR_FLAG	Clears a given EXTI line interrupt flag pending bit. Example: <code>__HAL_PWR_PVD_EXTI_CLEAR_FLAG()</code>
__HAL_PPP_EXTI_GENERATE_SWIT	Generates a software interrupt for a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_GENERATE_SWIT()</code>
__HAL_PPP_EXTI_ENABLE_EVENT	Enables event on a given EXTI Line Example: <code>__HAL_PWR_PVD_EXTI_ENABLE_EVENT()</code>
__HAL_PPP_EXTI_DISABLE_EVENT	Disables event on a given EXTI line Example: <code>__HAL_PWR_PVD_EXTI_DISABLE_EVENT()</code>

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32f0xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCALLBACK()).

3.11.6 DMA

The DMA HAL driver allows enabling and configuring 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 allows programming the required configuration through the following parameters:

- Transfer direction
- Source and destination data formats
- Circular, Normal or peripheral flow control mode
- Channel priority level
- Source and destination Increment mode
- FIFO mode and its threshold (if needed)
- Burst mode for source and/or destination (if needed).

Two operating modes are available:

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

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

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

The most used DMA HAL driver macros are the following:

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

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

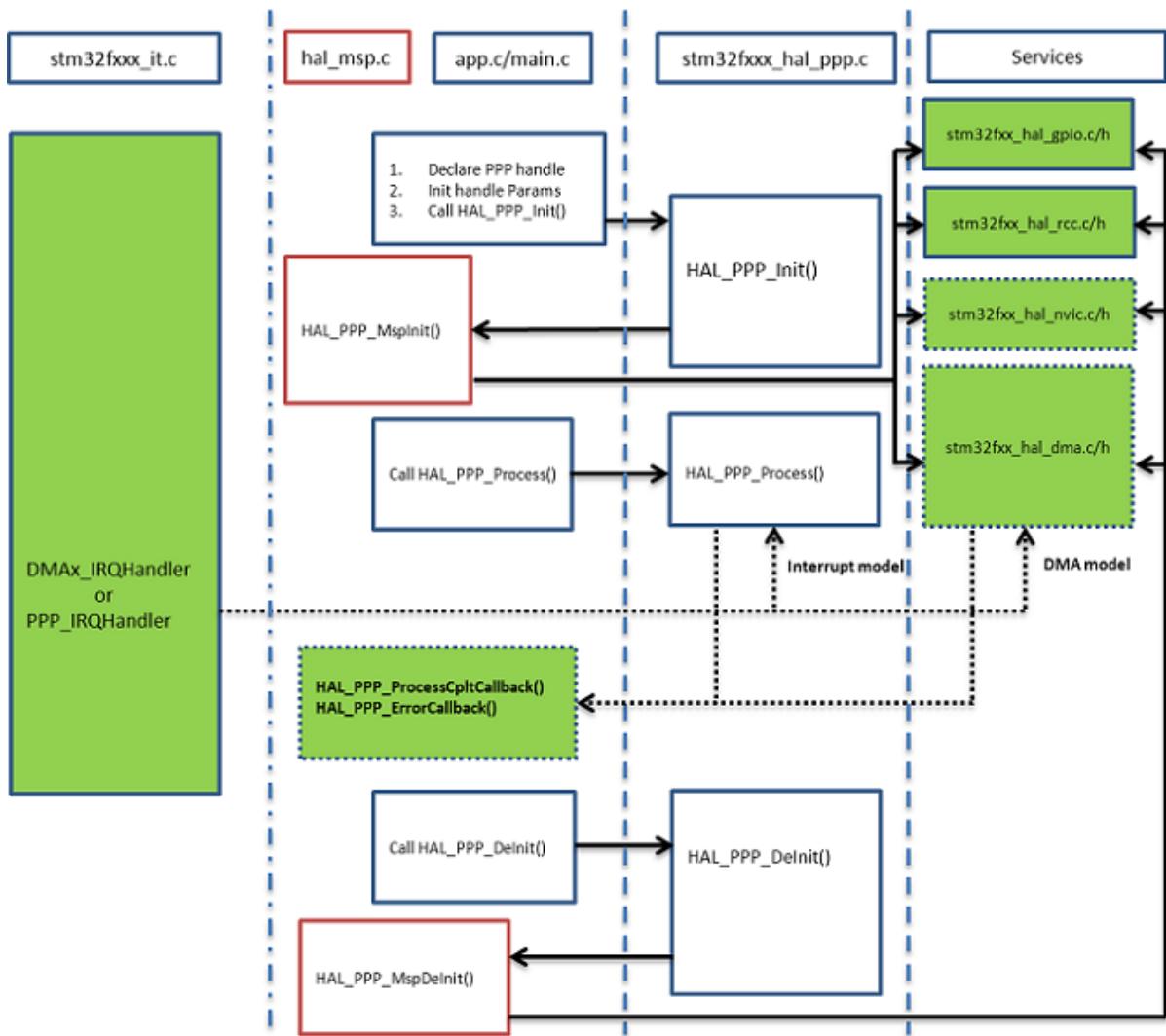
Note: DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

3.12 How to use HAL drivers

3.12.1 HAL usage models

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

Figure 7. HAL driver model



Note:

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

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

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

3.12.2 HAL initialization

3.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file `stm32f0xx_hal.c`.

- `HAL_Init()`: this function must be called at application startup to
 - initialize data/instruction cache and pre-fetch queue
 - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
 - call `HAL_MspInit()` user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). `HAL_MspInit()` is defined as “weak” empty function in the HAL drivers.
- `HAL_DeInit()`
 - resets all peripherals
 - calls function `HAL_MspDeInit()` which is user callback function to do system level De-Initializations.
- `HAL_GetTick()`: this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- `HAL_Delay()`: this function implements a delay (expressed in milliseconds) using the SysTick timer.
Care must be taken when using `HAL_Delay()` since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if `HAL_Delay()` is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be 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.

Please find below the typical Clock configuration sequence:

```
static void SystemClock_Config(void)
{
RCC_ClkInitTypeDef RCC_ClkInitStruct;
RCC_OscInitTypeDef RCC_OscInitStruct;
/* Enable HSE Oscillator and Activate PLL with HSE as source */
RCC_OscInitStruct.OscillatorType = RCC OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
Error_Handler();
}
/* Select PLL as system clock source and configure the HCLK, PCLK1 clocks dividers */
RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1);
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
{
Error_Handler();
}
}
```

3.12.2.3 HAL MSP initialization process

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

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

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

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

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

The HAL MSP callback is implemented inside the *stm32f0xx_hal_msp.c* file in the user folders. An *stm32f0xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32f0xx_hal_msp.c file contains the following functions:

Table 14. MSP functions

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

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

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

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

3.12.3

HAL I/O operation process

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

- Polling mode
- Interrupt mode
- DMA mode

3.12.3.1

Polling mode

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

The example below shows the typical Polling mode processing sequence :

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

3.12.3.2

Interrupt mode

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

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

- *HAL_PPP_Process_IT()*: launch the process
- *HAL_PPP_IRQHandler()*: the global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: the callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: the 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 *stm32f0xx_it.c*.

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

An example of use is illustrated below:

main.c file:

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

stm32f0xx_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 *stm32f0xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
    PPP_TypeDef *Instance; /* Register base address */
    PPP_InitTypeDef Init; /* PPP communication parameters */
    HAL_StateTypeDef State; /* PPP communication state */
    (...)

    DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(...)

void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
    static DMA_HandleTypeDef hdma_tx;
    static DMA_HandleTypeDef hdma_rx;
    (...)

    __HAL_LINKDMA(UartHandle, DMA_HandleTypeDef_tx, hdma_tx);
    __HAL_LINKDMA(UartHandle, DMA_HandleTypeDef_rx, hdma_rx);
    (...)

}
```

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

An example of use is illustrated below:

main.c file:

```
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_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

stm32f0xx_it.c file:

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

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

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

3.12.4 Timeout and error management

3.12.4.1 Timeout management

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

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

The timeout possible value are the following:

Table 15. Timeout values

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

1. *HAL_MAX_DELAY* is defined in the *stm32f0xx_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)
{
(...)
timeout = HAL_GetTick() + LOCAL_PROCESS_TIMEOUT;
(...)
while(ProcessOngoing)
{
(...)
if(HAL_GetTick() >= timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return HAL_PPP_STATE_TIMEOUT;
}
}
(...)
```

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

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
(...)
timeout = HAL_GetTick() + Timeout;
(...)
while(ProcessOngoing)
{
(...)
if(Timeout != HAL_MAX_DELAY)
{
if(HAL_GetTick() >= timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return hppp->State;
}
}
(...)
```

3.12.4.2

Error management

The HAL drivers implement a check on the following items:

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

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32 Size)

{
if ((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
```

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

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

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

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

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

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

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

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

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

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

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

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

3.12.4.3 Run-time checking

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

To enable the run-time checking, use the `assert_param` macro, and leave the define `USE_FULL_ASSERT` uncommented in `stm32f0xx_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 stm32f0xx_hal_conf.h:

```
/* Exported macro -----*/
#ifndef 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);
#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)
{
}
```

Note:

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

4 Overview of low-layer drivers

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

The LL drivers feature:

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

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

4.1 Low-layer files

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

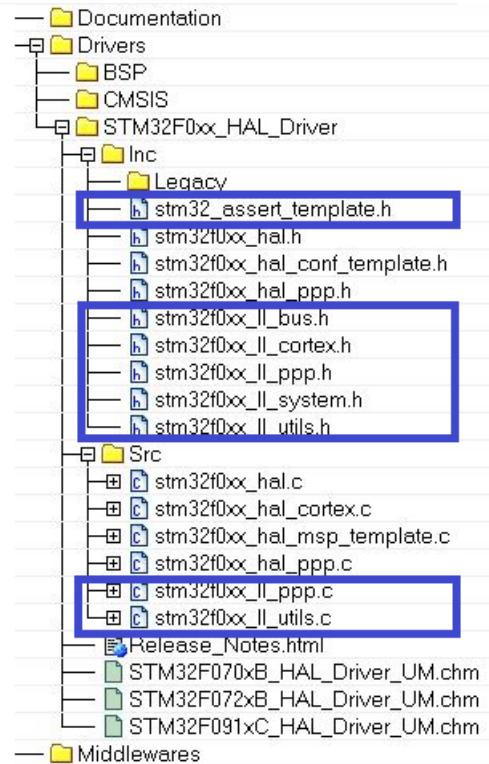
Table 16. LL driver files

File	Description
<code>stm32f0xx_ll_bus.h</code>	This is the h-source file for core bus control and peripheral clock activation and deactivation <i>Example: LL_AHB1_GRP1_EnableClock</i>
<code>stm32f0xx_ll_ppp.h/c</code>	<code>stm32f0xx_ll_ppp.c</code> provides peripheral initialization functions such as <code>LL_PPP_Init()</code> , <code>LL_PPP_StructInit()</code> , <code>LL_PPP_DelInit()</code> . All the other APIs are defined within <code>stm32f0xx_ll_ppp.h</code> file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the <code>stm32f0xx_ll_ppp.h</code> file.
<code>stm32f0xx_ll_cortex.h</code>	Cortex-M related register operation APIs including the Systick, Low power (<code>LL_SYSTICK_xxxxx</code> , <code>LL_LPM_xxxxx</code> "Low Power Mode" ...)
<code>stm32f0xx_ll_utils.h/c</code>	This file covers the generic APIs: <ul style="list-style-type: none">• Read of device unique ID and electronic signature• Timebase and delay management• System clock configuration.
<code>stm32f0xx_ll_system.h</code>	System related operations. <i>Example: LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx</i>
<code>stm32_assert_template.h</code>	Template file allowing to define the <code>assert_param</code> macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to <code>stm32_assert.h</code> .

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

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

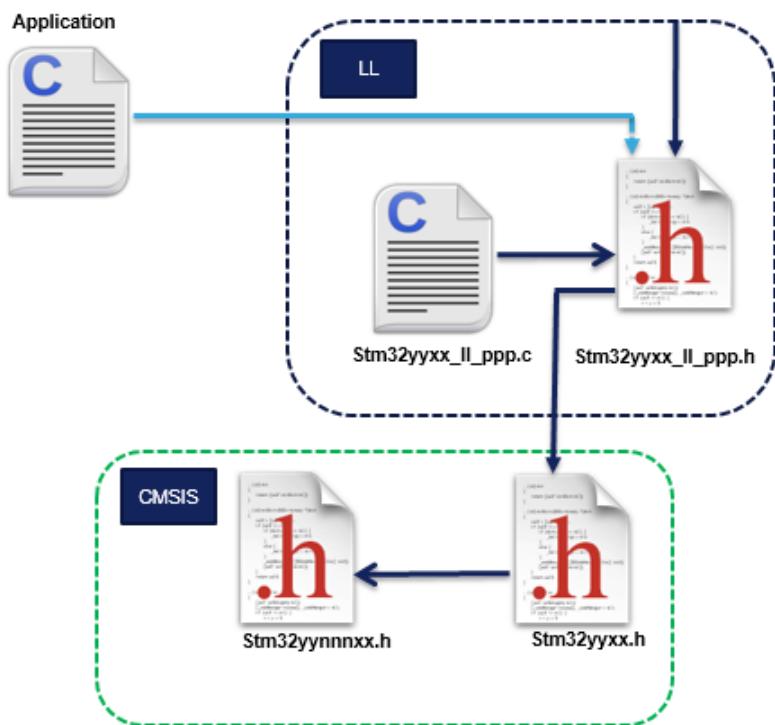
Figure 8. Low-layer driver folders



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

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

Figure 9. 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 `stm32f0xx_ll_ppp.c` file:

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

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

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

Table 17. Common peripheral initialization functions

Functions	Return Type	Parameters	Description
<code>LL_PPP_Init</code>	<code>ErrorStatus</code>	<ul style="list-style-type: none"> • <code>PPP_TypeDef* PPPx</code> • <code>LL_PPP_InitTypeDef* PPP_InitStruct</code> 	<p>Initializes the peripheral main features according to the parameters specified in <code>PPP_InitStruct</code>.</p> <p>Example: <code>LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)</code></p>
<code>LL_PPP_StructInit</code>	<code>void</code>	<ul style="list-style-type: none"> • <code>LL_PPP_InitTypeDef* PPP_InitStruct</code> 	<p>Fills each <code>PPP_InitStruct</code> member with its default value.</p> <p>Example. <code>LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)</code></p>

Functions	Return Type	Parameters	Description
LL_PPP_Delnit	ErrorStatus	• <i>PPP_TypeDef* PPPx</i>	<p>De-initializes the peripheral registers, that is restore them to their default reset values.</p> <p>Example. LL_USART_Delnit(USART_TypeDef *USARTx)</p>

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

4.2.1.1 Run-time checking

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

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

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

Note:

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

4.2.2 Peripheral register-level configuration functions

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

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

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

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

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

Name	Examples
<code>LL_PPP_{CATEGORY}_ActionItem_BITNAME</code>	<ul style="list-style-type: none">• <code>LL_RCC_IsActiveFlag_LSIRDY</code>• <code>LL_RCC_IsActiveFlag_FWRST()</code>• <code>LL_ADC_ClearFlag_EOC(ADC1)</code>• <code>LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)</code>
<code>LL_PPP_{CATEGORY}_IsItem_BITNAME_Action</code>	

Table 20. Available function formats

Item	Action	Format
Flag	Get	<code>LL_PPP_IsActiveFlag_BITNAME</code>
	Clear	<code>LL_PPP_ClearFlag_BITNAME</code>
Interrupts	Enable	<code>LL_PPP_EnableIT_BITNAME</code>
	Disable	<code>LL_PPP_DisableIT_BITNAME</code>
	Get	<code>LL_PPP_IsEnabledIT_BITNAME</code>
DMA	Enable	<code>LL_PPP_EnableDMAReq_BITNAME</code>
	Disable	<code>LL_PPP_DisableDMAReq_BITNAME</code>
	Get	<code>LL_PPP_IsEnabledDMAReq_BITNAME</code>

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
<code>LL_BUS_GRPx_ActionClock{Mode}</code>	<ul style="list-style-type: none">• <code>LL_AHB1_GRP1_EnableClock (LL_AHB1_GRP1_PERIPH_GPIOA)</code>• <code>LL_AHB1_GRP1_PERIPH_GPIOB)</code>

Name	Examples
	<ul style="list-style-type: none">• <code>LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)</code>

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
<code>LL_PPP_{CATEGORY}_Action{Item}</code> <code>LL_PPP_{CATEGORY}_IsItemAction</code>	<ul style="list-style-type: none">• <code>LL_ADC_Enable ()</code>• <code>LL_ADC_StartCalibration();</code>• <code>LL_ADC_IsCalibrationOnGoing;</code>• <code>LL_RCC_HSI_Enable ()</code>• <code>LL_RCC_HSI_IsReady()</code>

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

Table 23. Peripheral configuration management

Name	Examples
<code>LL_PPP_{CATEGORY}_Set{ or Get}ConfigItem</code>	<code>LL_USART_SetBaudRate (USART2, 16000000,LL_USART_OVERSAMPLING_16, 9600)</code>

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

Table 24. Peripheral register management

Name
<code>LL_PPP_WriteReg(_INSTANCE_, _REG_, _VALUE_)</code>
<code>LL_PPP_ReadReg(_INSTANCE_, _REG_)</code>
<code>LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel}, {uint32_t Proprietary})</code>

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

5 Cohabiting of HAL and LL

The low-layer 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 `stm32f0xx_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 [STM32CubeF0](#) 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 `stm32f0` 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:

- HAL Initialization and de-initialization functions
- HAL Control functions

6.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [**HAL_Init**](#)
- [**HAL_DelInit**](#)
- [**HAL_MspInit**](#)
- [**HAL_MspDelInit**](#)
- [**HAL_InitTick**](#)

6.1.3 HAL Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

- `HAL_IncTick`
- `HAL_GetTick`
- `HAL_Delay`
- `HAL_SuspendTick`
- `HAL_ResumeTick`
- `HAL_GetHalVersion`
- `HAL_GetREVID`
- `HAL_GetDEVID`
- `HAL_GetUIDw0`
- `HAL_GetUIDw1`
- `HAL_GetUIDw2`
- `HAL_DBGMCU_EnableDBGStopMode`
- `HAL_DBGMCU_DisableDBGStopMode`
- `HAL_DBGMCU_EnableDBGStandbyMode`
- `HAL_DBGMCU_DisableDBGStandbyMode`

6.1.4 Detailed description of functions

`HAL_Init`

Function name	<code>HAL_StatusTypeDef HAL_Init (void)</code>
Function description	This function configures the Flash prefetch, Configures time base source, NVIC and Low level hardware.
Return values	<ul style="list-style-type: none">• <code>HAL:</code> status
Notes	<ul style="list-style-type: none">• This function is called at the beginning of program after reset and before the clock configuration• The time base configuration is based on HSI clock when exiting from Reset. Once done, time base tick start incrementing. In the default implementation,Systick is used as source of time base. The tick variable is incremented each 1ms in its ISR.

`HAL_DelInit`

Function name	<code>HAL_StatusTypeDef HAL_DelInit (void)</code>
Function description	This function de-Initializes common part of the HAL and stops the source of time base.
Return values	<ul style="list-style-type: none">• <code>HAL:</code> status
Notes	<ul style="list-style-type: none">• This function is optional.

`HAL_MspInit`

Function name	<code>void HAL_MspInit (void)</code>
Function description	Initializes the MSP.
Return values	<ul style="list-style-type: none">• <code>None:</code>

HAL_MspDeInit

Function name `void HAL_MspDeInit (void)`

Function description DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name `HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)`

Function description This function configures the source of the time base.

Parameters

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

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

HAL_IncTick

Function name `void HAL_IncTick (void)`

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

Return values

- **None:**

Notes

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

HAL_Delay

Function name `void HAL_Delay (__IO uint32_t Delay)`

Function description This function provides accurate 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_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 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 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** This method returns the HAL revision.**Return values**

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

HAL_GetREVID

Function name `uint32_t HAL_GetREVID (void)`

Function description Returns the device revision identifier.

Return values • **Device:** revision identifier

HAL_GetDEVID

Function name `uint32_t HAL_GetDEVID (void)`

Function description Returns the device identifier.

Return values • **Device:** identifier

HAL_GetUIDw0

Function name `uint32_t HAL_GetUIDw0 (void)`

Function description Returns first word of the unique device identifier (UID based on 96 bits)

Return values • **Device:** identifier

HAL_GetUIDw1

Function name `uint32_t HAL_GetUIDw1 (void)`

Function description Returns second word of the unique device identifier (UID based on 96 bits)

Return values • **Device:** identifier

HAL_GetUIDw2

Function name `uint32_t HAL_GetUIDw2 (void)`

Function description Returns third word of the unique device identifier (UID based on 96 bits)

Return values • **Device:** identifier

HAL_DBGMCU_EnableDBGStopMode

Function name `void HAL_DBGMCU_EnableDBGStopMode (void)`

Function description Enable the Debug Module during STOP mode.

Return values • **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name `void HAL_DBGMCU_DisableDBGStopMode (void)`

Function description Disable the Debug Module during STOP mode.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name **void HAL_DBGMCU_EnableDBGStandbyMode (void)**

Function description Enable the Debug Module during STANDBY mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name **void HAL_DBGMCU_DisableDBGStandbyMode (void)**

Function description Disable the Debug Module during STANDBY mode.

Return values

- **None:**

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_FASTMO DEPLUS_ENABLE **Description:**

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

Parameters:

- **_FASTMODEPLUS_**: This parameter can be a value of

_HAL_SYSCFG_FASTMO DEPLUS_DISABLE

HAL Freeze Unfreeze Peripherals

_HAL_FREEZE_CAN_DB GMCU

_HAL_UNFREEZE_CAN_D BGMCU

_HAL_DBGMCU_FREEZE RTC

_HAL_DBGMCU_UNFREEZE_RTC

_HAL_DBGMCU_FREEZE_I2C1_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT

__HAL_DBGMCU_FREEZE_IWDG

__HAL_DBGMCU_UNFREEZE_IWDG

__HAL_DBGMCU_FREEZE_WWDG

__HAL_DBGMCU_UNFREEZE_WWDG

__HAL_DBGMCU_FREEZE_TIM2

__HAL_DBGMCU_UNFREEZE_TIM2

__HAL_DBGMCU_FREEZE_TIM3

__HAL_DBGMCU_UNFREEZE_TIM3

__HAL_DBGMCU_FREEZE_TIM6

__HAL_DBGMCU_UNFREEZE_TIM6

__HAL_DBGMCU_FREEZE_TIM7

__HAL_DBGMCU_UNFREEZE_TIM7

__HAL_DBGMCU_FREEZE_TIM14

__HAL_DBGMCU_UNFREEZE_TIM14

__HAL_DBGMCU_FREEZE_TIM1

__HAL_DBGMCU_UNFREEZE_TIM1

__HAL_DBGMCU_FREEZE_TIM15

`__HAL_DBGMCU_UNFREEZE_TIM15`

`__HAL_DBGMCU_FREEZE_TIM16`

`__HAL_DBGMCU_UNFREEZE_TIM16`

`__HAL_DBGMCU_FREEZE_TIM17`

`__HAL_DBGMCU_UNFREEZE_TIM17`

HAL IRDA Enveloppe Selection

`HAL_SYSCFG_IRDA_ENV_SEL_TIM16`

`HAL_SYSCFG_IRDA_ENV_SEL_USART1`

`HAL_SYSCFG_IRDA_ENV_SEL_USART4`

HAL ISR Wrapper

`HAL_SYSCFG_ITLINE0` Internal define for macro handling

`HAL_SYSCFG_ITLINE1` Internal define for macro handling

`HAL_SYSCFG_ITLINE2` Internal define for macro handling

`HAL_SYSCFG_ITLINE3` Internal define for macro handling

`HAL_SYSCFG_ITLINE4` Internal define for macro handling

`HAL_SYSCFG_ITLINE5` Internal define for macro handling

`HAL_SYSCFG_ITLINE6` Internal define for macro handling

`HAL_SYSCFG_ITLINE7` Internal define for macro handling

`HAL_SYSCFG_ITLINE8` Internal define for macro handling

`HAL_SYSCFG_ITLINE9` Internal define for macro handling

`HAL_SYSCFG_ITLINE10` Internal define for macro handling

`HAL_SYSCFG_ITLINE11` Internal define for macro handling

HAL_SYSCFG_ITLINE12	Internal define for macro handling
HAL_SYSCFG_ITLINE13	Internal define for macro handling
HAL_SYSCFG_ITLINE14	Internal define for macro handling
HAL_SYSCFG_ITLINE15	Internal define for macro handling
HAL_SYSCFG_ITLINE16	Internal define for macro handling
HAL_SYSCFG_ITLINE17	Internal define for macro handling
HAL_SYSCFG_ITLINE18	Internal define for macro handling
HAL_SYSCFG_ITLINE19	Internal define for macro handling
HAL_SYSCFG_ITLINE20	Internal define for macro handling
HAL_SYSCFG_ITLINE21	Internal define for macro handling
HAL_SYSCFG_ITLINE22	Internal define for macro handling
HAL_SYSCFG_ITLINE23	Internal define for macro handling
HAL_SYSCFG_ITLINE24	Internal define for macro handling
HAL_SYSCFG_ITLINE25	Internal define for macro handling
HAL_SYSCFG_ITLINE26	Internal define for macro handling
HAL_SYSCFG_ITLINE27	Internal define for macro handling
HAL_SYSCFG_ITLINE28	Internal define for macro handling
HAL_SYSCFG_ITLINE29	Internal define for macro handling
HAL_SYSCFG_ITLINE30	Internal define for macro handling
HAL_SYSCFG_ITLINE31	Internal define for macro handling
HAL_ITLINE_EWDG	EWDG has expired
HAL_ITLINE_PVDOUT	Power voltage detection Interrupt
HAL_ITLINE_VDDIO2	VDDIO2 Interrupt
HAL_ITLINE_RTC_WAKEU	RTC WAKEUP -> exti[20] Interrupt P
HAL_ITLINE_RTC_TSTAMP	RTC Time Stamp -> exti[19] interrupt

HAL_ITLINE_RTC_ALRA	RTC Alarm -> exti[17] interrupt
HAL_ITLINE_FLASH_ITF	Flash ITF Interrupt
HAL_ITLINE_CRS	CRS Interrupt
HAL_ITLINE_CLK_CTRL	CLK Control Interrupt
HAL_ITLINE_EXTI0	External Interrupt 0
HAL_ITLINE_EXTI1	External Interrupt 1
HAL_ITLINE_EXTI2	External Interrupt 2
HAL_ITLINE_EXTI3	External Interrupt 3
HAL_ITLINE_EXTI4	EXTI4 Interrupt
HAL_ITLINE_EXTI5	EXTI5 Interrupt
HAL_ITLINE_EXTI6	EXTI6 Interrupt
HAL_ITLINE_EXTI7	EXTI7 Interrupt
HAL_ITLINE_EXTI8	EXTI8 Interrupt
HAL_ITLINE_EXTI9	EXTI9 Interrupt
HAL_ITLINE_EXTI10	EXTI10 Interrupt
HAL_ITLINE_EXTI11	EXTI11 Interrupt
HAL_ITLINE_EXTI12	EXTI12 Interrupt
HAL_ITLINE_EXTI13	EXTI13 Interrupt
HAL_ITLINE_EXTI14	EXTI14 Interrupt
HAL_ITLINE_EXTI15	EXTI15 Interrupt
HAL_ITLINE_TSC_EOA	Touch control EOA Interrupt
HAL_ITLINE_TSC_MCE	Touch control MCE Interrupt
HAL_ITLINE_DMA1_CH1	DMA1 Channel 1 Interrupt
HAL_ITLINE_DMA1_CH2	DMA1 Channel 2 Interrupt
HAL_ITLINE_DMA1_CH3	DMA1 Channel 3 Interrupt

HAL_ITLINE_DMA2_CH1	DMA2 Channel 1 Interrupt
HAL_ITLINE_DMA2_CH2	DMA2 Channel 2 Interrupt
HAL_ITLINE_DMA1_CH4	DMA1 Channel 4 Interrupt
HAL_ITLINE_DMA1_CH5	DMA1 Channel 5 Interrupt
HAL_ITLINE_DMA1_CH6	DMA1 Channel 6 Interrupt
HAL_ITLINE_DMA1_CH7	DMA1 Channel 7 Interrupt
HAL_ITLINE_DMA2_CH3	DMA2 Channel 3 Interrupt
HAL_ITLINE_DMA2_CH4	DMA2 Channel 4 Interrupt
HAL_ITLINE_DMA2_CH5	DMA2 Channel 5 Interrupt
HAL_ITLINE_ADC	ADC Interrupt
HAL_ITLINE_COMP1	COMP1 Interrupt -> exti[21]
HAL_ITLINE_COMP2	COMP2 Interrupt -> exti[21]
HAL_ITLINE_TIM1_BRK	TIM1 BRK Interrupt
HAL_ITLINE_TIM1_UPD	TIM1 UPD Interrupt
HAL_ITLINE_TIM1_TRG	TIM1 TRG Interrupt
HAL_ITLINE_TIM1_CCU	TIM1 CCU Interrupt
HAL_ITLINE_TIM1_CC	TIM1 CC Interrupt
HAL_ITLINE_TIM2	TIM2 Interrupt
HAL_ITLINE_TIM3	TIM3 Interrupt
HAL_ITLINE_DAC	DAC Interrupt
HAL_ITLINE_TIM6	TIM6 Interrupt
HAL_ITLINE_TIM7	TIM7 Interrupt
HAL_ITLINE_TIM14	TIM14 Interrupt
HAL_ITLINE_TIM15	TIM15 Interrupt
HAL_ITLINE_TIM16	TIM16 Interrupt

<code>HAL_ITLINE_TIM17</code>	TIM17 Interrupt
<code>HAL_ITLINE_I2C1</code>	I2C1 Interrupt -> exti[23]
<code>HAL_ITLINE_I2C2</code>	I2C2 Interrupt
<code>HAL_ITLINE_SPI1</code>	I2C1 Interrupt -> exti[23]
<code>HAL_ITLINE_SPI2</code>	SPI1 Interrupt
<code>HAL_ITLINE_USART1</code>	USART1 GLB Interrupt -> exti[25]
<code>HAL_ITLINE_USART2</code>	USART2 GLB Interrupt -> exti[26]
<code>HAL_ITLINE_USART3</code>	USART3 Interrupt
<code>HAL_ITLINE_USART4</code>	USART4 Interrupt
<code>HAL_ITLINE_USART5</code>	USART5 Interrupt
<code>HAL_ITLINE_USART6</code>	USART6 Interrupt
<code>HAL_ITLINE_USART7</code>	USART7 Interrupt
<code>HAL_ITLINE_USART8</code>	USART8 Interrupt
<code>HAL_ITLINE_CAN</code>	CAN Interrupt
<code>HAL_ITLINE_CEC</code>	CEC Interrupt -> exti[27]

HAL ISR wrapper check

`_HAL_GET_PENDING_IT`

HAL state definition

`HAL_SMBUS_STATE_RES` SMBUS not yet initialized or disabled
`ET`

`HAL_SMBUS_STATE_REA` SMBUS initialized and ready for use
`DY`

`HAL_SMBUS_STATE_BUS` SMBUS internal process is ongoing
`Y`

`HAL_SMBUS_STATE_MAS` Master Data Transmission process is ongoing
`TER_BUSY_TX`

`HAL_SMBUS_STATE_MAS` Master Data Reception process is ongoing
`TER_BUSY_RX`

HAL_SMBUS_STATE_SLAV Slave Data Transmission process is ongoing
E_BUSY_TX

HAL_SMBUS_STATE_SLAV Slave Data Reception process is ongoing
E_BUSY_RX

HAL_SMBUS_STATE_TIME Timeout state
OUT

HAL_SMBUS_STATE_ERR Reception process is ongoing
OR

HAL_SMBUS_STATE_LIST Address Listen Mode is ongoing
EN

HAL SYSCFG IRDA modulation envelope selection

_HAL_SYSCFG_IRDA_EN
V_SELECTION

_HAL_SYSCFG_GET_IRD
A_ENV_SELECTION

HAL SYSCFG Parity check on RAM

_HAL_SYSCFG_RAM_PA
RITYCHECK_DISABLE

Fast-mode Plus on GPIO

SYSCFG_FASTMODEPLUS Enable Fast-mode Plus on PA9
_PA9

SYSCFG_FASTMODEPLUS Enable Fast-mode Plus on PA10
_PA10

SYSCFG_FASTMODEPLUS Enable Fast-mode Plus on PB6
_PB6

SYSCFG_FASTMODEPLUS Enable Fast-mode Plus on PB7
_PB7

SYSCFG_FASTMODEPLUS Enable Fast-mode Plus on PB8
_PB8

SYSCFG_FASTMODEPLUS Enable Fast-mode Plus on PB9
_PB9

7 HAL ADC Generic Driver

7.1 ADC Firmware driver registers structures

7.1.1 ADC_InitTypeDef

`ADC_InitTypeDef` is defined in the `stm32f0xx_hal_adc.h`

Data Fields

- `uint32_t ClockPrescaler`
- `uint32_t Resolution`
- `uint32_t DataAlign`
- `uint32_t ScanConvMode`
- `uint32_t EOCSelection`
- `uint32_t LowPowerAutoWait`
- `uint32_t LowPowerAutoPowerOff`
- `uint32_t ContinuousConvMode`
- `uint32_t DiscontinuousConvMode`
- `uint32_t ExternalTrigConv`
- `uint32_t ExternalTrigConvEdge`
- `uint32_t DMAContinuousRequests`
- `uint32_t Overrun`
- `uint32_t SamplingTimeCommon`

Field Documentation

- `uint32_t ADC_InitTypeDef::ClockPrescaler`

Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from ADC dedicated HSI RC oscillator 14MHz) and clock prescaler. This parameter can be a value of `ADC_ClockPrescaler`. Note: In case of usage of the ADC dedicated HSI RC oscillator, it must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if the ADC is disabled

- `uint32_t ADC_InitTypeDef::Resolution`

Configures the ADC resolution. This parameter can be a value of `ADC_Resolution`

- `uint32_t ADC_InitTypeDef::DataAlign`

Specifies whether the ADC data alignment is left or right. This parameter can be a value of `ADC_Data_align`

- `uint32_t ADC_InitTypeDef::ScanConvMode`

Configures the sequencer of regular group. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. Sequencer is automatically enabled if several channels are set (sequencer cannot be disabled, as it can be the case on other STM32 devices): If only 1 channel is set: Conversion is performed in single mode. If several channels are set: Conversions are performed in sequence mode (ranks defined by each channel number: channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Scan direction can be set to forward (from channel 0 to channel 18) or backward (from channel 18 to channel 0). This parameter can be a value of `ADC_Scan_mode`

- `uint32_t ADC_InitTypeDef::EOCSelection`

Specifies what EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of conversion of each rank or complete sequence. This parameter can be a value of `ADC_EOCSelection`.

- **`uint32_t ADC_InitTypeDef::LowPowerAutoWait`**
Selects the dynamic low power Auto Delay: new conversion start only when the previous conversion (for regular group) has been treated by user software, using function `HAL_ADC_GetValue()`. This feature automatically adapts the ADC conversions trigs to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: Do not use with interruption or DMA (`HAL_ADC_Start_IT()`, `HAL_ADC_Start_DMA()`) since they have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion with `HAL_ADC_Start()`, 2. Later on, when conversion data is needed: use `HAL_ADC_PollForConversion()` to ensure that conversion is completed and use `HAL_ADC_GetValue()` to retrieve conversion result and trig another conversion.
- **`uint32_t ADC_InitTypeDef::LowPowerAutoPowerOff`**
Selects the auto-off mode: the ADC automatically powers-off after a conversion and automatically wakes-up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with automatic wait mode (parameter 'LowPowerAutoWait'). This parameter can be set to ENABLE or DISABLE. Note: If enabled, this feature also turns off the ADC dedicated 14 MHz RC oscillator (HSI14)
- **`uint32_t ADC_InitTypeDef::ContinuousConvMode`**
Specifies whether the conversion is performed in single mode (one conversion) or continuous mode for regular group, after the selected trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- **`uint32_t ADC_InitTypeDef::DiscontinuousConvMode`**
Specifies whether the conversions sequence of regular group is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE Note: Number of discontinuous ranks increment is fixed to one-by-one.
- **`uint32_t ADC_InitTypeDef::ExternalTrigConv`**
Selects the external event used to trigger the conversion start of regular group. If set to `ADC_SOFTWARE_START`, external triggers are disabled. This parameter can be a value of `ADC_External_trigger_source-Regular`
- **`uint32_t ADC_InitTypeDef::ExternalTrigConvEdge`**
Selects the external trigger edge of regular group. If trigger is set to `ADC_SOFTWARE_START`, this parameter is discarded. This parameter can be a value of `ADC_External_trigger_edge-Regular`
- **`uint32_t ADC_InitTypeDef::DMAContinuousRequests`**
Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be set to ENABLE or DISABLE.
- **`uint32_t ADC_InitTypeDef::Overrun`**
Select the behaviour in case of overrun: data preserved or overwritten This parameter has an effect on regular group only, including in DMA mode. This parameter can be a value of `ADC_Overrun`
- **`uint32_t ADC_InitTypeDef::SamplingTimeCommon`**
Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). Note: On STM32F0 devices, the sampling time setting is common to all channels. On some other STM32 devices, this parameter in channel wise and is located into ADC channel initialization structure. This parameter can be a value of `ADC_sampling_times` Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS_vrefint, TS_vbat, TS_temp (values rough order: 5us to 17us).

7.1.2

`ADC_ChannelConfTypeDef`

`ADC_ChannelConfTypeDef` is defined in the `stm32f0xx_hal_adc.h`

Data Fields

- `uint32_t Channel`
- `uint32_t Rank`
- `uint32_t SamplingTime`

Field Documentation

- `uint32_t ADC_ChannelConfTypeDef::Channel`

Specifies the channel to configure into ADC regular group. This parameter can be a value of [ADC_channels](#)
Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability.

- `uint32_t ADC_ChannelConfTypeDef::Rank`

Add or remove the channel from ADC regular group sequencer. On STM32F0 devices, number of ranks in the sequence is defined by number of channels enabled, rank of each channel is defined by channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Despite the channel rank is fixed, this parameter allow an additional possibility: to remove the selected rank (selected channel) from sequencer. This parameter can be a value of [ADC_rank](#)

- `uint32_t ADC_ChannelConfTypeDef::SamplingTime`

Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of [ADC_sampling_times](#)
Caution: this setting impacts the entire regular group. Therefore, call of `HAL_ADC_ConfigChannel()` to configure a channel can impact the configuration of other channels previously set. Caution: Obsolete parameter. Use parameter "SamplingTimeCommon" in ADC initialization structure. If parameter "SamplingTimeCommon" is set to a valid sampling time, parameter "SamplingTime" is discarded. Note: In case of usage of internal measurement channels (Vrefint/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting)
Refer to device datasheet for timings values, parameters TS_vrefint, TS_vbat, TS_temp (values rough order: 5us to 17us).

7.1.3 ADC_AnalogWDGConfTypeDef

`ADC_AnalogWDGConfTypeDef` is defined in the `stm32f0xx_hal_adc.h`

Data Fields

- `uint32_t WatchdogMode`
- `uint32_t Channel`
- `uint32_t ITMode`
- `uint32_t HighThreshold`
- `uint32_t LowThreshold`

Field Documentation

- `uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode`

Configures the ADC analog watchdog mode: single/all/none channels. This parameter can be a value of [ADC_analog_watchdog_mode](#).

- `uint32_t ADC_AnalogWDGConfTypeDef::Channel`

Selects which ADC channel to monitor by analog watchdog. This parameter has an effect only if parameter 'WatchdogMode' is configured on single channel. Only 1 channel can be monitored. This parameter can be a value of [ADC_channels](#).

- `uint32_t ADC_AnalogWDGConfTypeDef::ITMode`

Specifies whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE

- `uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold`

Configures the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.

- `uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold`

Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.

7.1.4 ADC_HandleTypeDef

`ADC_HandleTypeDef` is defined in the `stm32f0xx_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`

Field Documentation

- **`ADC_TypeDef* ADC_HandleTypeDef::Instance`**
Register base address
- **`ADC_InitTypeDef ADC_HandleTypeDef::Init`**
ADC required parameters
- **`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

7.2 ADC Firmware driver API description

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

7.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (common for all channels)
- ADC conversion of regular group.
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- ADC calibration
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

7.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
 - As prerequisite, ADC clock must be configured at RCC top level. Caution: On STM32F0, ADC clock frequency max is 14MHz (refer to device datasheet). Therefore, ADC clock prescaler must be configured in function of ADC clock source frequency to remain below this maximum frequency.
 - Two clock settings are mandatory:
 - ADC clock (core clock, also possibly conversion clock).
 - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from APB clock or asynchronous clock derived from ADC dedicated HSI RC oscillator 14MHz. If asynchronous clock is selected, parameter "HSI14State" must be set either: - to "...HSI14State = RCC_HSI14_ADC_CONTROL" to let the ADC control the HSI14 oscillator enable/disable (if not used to supply the main system clock): feature used if ADC mode LowPowerAutoPowerOff is enabled. - to "...HSI14State = RCC_HSI14_ON" to maintain the HSI14 oscillator always enabled: can be used to supply the main system clock.
 - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
 - __HAL_RCC_ADC1_CLK_ENABLE(); (mandatory) HSI14 enable or let under control of ADC: (optional: if asynchronous clock selected)
 - RCC_OsclInitTypeDef RCC_OsclInitStructure;
 - RCC_OsclInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI14;
 - RCC_OsclInitStructure.HSI14CalibrationValue = RCC_HSI14CALIBRATION_DEFAULT;
 - RCC_OsclInitStructure.HSI14State = RCC_HSI14_ADC_CONTROL;
 - RCC_OsclInitStructure.PLL... (optional if used for system clock)
 - HAL_RCC_OscConfig(&RCC_OsclInitStructure);
 - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
 - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
 - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
 - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
 - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
4. Optionally, in case of usage of DMA:
 - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
 - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
 - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

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

Execution of ADC conversions

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

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

Note: *Callback functions must be implemented in user program:*

- HAL_ADC_ErrorCallback()
- HAL_ADC_LevelOutOfWindowCallback() (*callback of analog watchdog*)
- HAL_ADC_ConvCpltCallback()
- HAL_ADC_ConvHalfCpltCallback()

Deinitialization of ADC

1. Disable the ADC interface
 - ADC clock can be hard reset and disabled at RCC top level.
 - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
 - ADC clock disable using the equivalent macro/functions as configuration step.
 - Example: Into HAL_ADC_MspDelInit() (recommended code location) or with other device clock parameters configuration:
 - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI14;
 - RCC_OscInitStructure.HSI14State = RCC_HSI14_OFF; (if not used for system clock)
 - HAL_RCC_OscConfig(&RCC_OscInitStructure);
2. ADC pins configuration
 - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
 - Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
 - Deinitialize the DMA using function HAL_DMA_Init().
 - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

7.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC

This section contains the following APIs:

- [**HAL_ADC_Init**](#)

- [*HAL_ADC_DelInit*](#)
- [*HAL_ADC_MspInit*](#)
- [*HAL_ADC_MspDelInit*](#)

7.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular group.
- Stop conversion of regular group.
- Poll for conversion complete on regular group.
- Poll for conversion event.
- Get result of regular channel conversion.
- Start conversion of regular group and enable interruptions.
- Stop conversion of regular group and disable interruptions.
- Handle ADC interrupt request
- Start conversion of regular group and enable DMA transfer.
- Stop conversion of regular group and disable ADC DMA transfer.

This section contains the following APIs:

- [*HAL_ADC_Start*](#)
- [*HAL_ADC_Stop*](#)
- [*HAL_ADC_PollForConversion*](#)
- [*HAL_ADC_PollForEvent*](#)
- [*HAL_ADC_Start_IT*](#)
- [*HAL_ADC_Stop_IT*](#)
- [*HAL_ADC_Start_DMA*](#)
- [*HAL_ADC_Stop_DMA*](#)
- [*HAL_ADC_GetValue*](#)
- [*HAL_ADC_IRQHandler*](#)
- [*HAL_ADC_ConvCpltCallback*](#)
- [*HAL_ADC_ConvHalfCpltCallback*](#)
- [*HAL_ADC_LevelOutOfWindowCallback*](#)
- [*HAL_ADC_ErrorCallback*](#)

7.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- [*HAL_ADC_ConfigChannel*](#)
- [*HAL_ADC_AnalogWDGConfig*](#)

7.2.6 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*](#)

7.2.7 Detailed description of functions

HAL_ADC_Init

Function name	HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)
Function description	Initializes the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".
Parameters	<ul style="list-style-type: none">hadc: ADC handle
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">As prerequisite, ADC clock must be configured at RCC top level depending on both possible clock sources: APB clock or HSI clock. See commented example code below that can be copied and uncommented into HAL_ADC_MspInit().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_Delinit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".

HAL_ADC_Delinit

Function name	HAL_StatusTypeDef HAL_ADC_Delinit (ADC_HandleTypeDef * hadc)
Function description	Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.
Parameters	<ul style="list-style-type: none">hadc: ADC handle
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behaviour in case of reset of a single ADC while the other ADCs sharing the same common group is still running.

HAL_ADC_MspInit

Function name	void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)
Function description	Initializes the ADC MSP.
Parameters	<ul style="list-style-type: none">hadc: ADC handle
Return values	<ul style="list-style-type: none">None:

HAL_ADC_MspDeInit

Function name `void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)`

Function description Deinitializes the ADC MSP.

Parameters • `hadc`: ADC handle

Return values • `None`:

HAL_ADC_Start

Function name `HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)`

Function description Enables ADC, starts conversion of regular group.

Parameters • `hadc`: ADC handle

Return values • `HAL`: status

HAL_ADC_Stop

Function name `HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)`

Function description Stop ADC conversion of regular group, disable ADC peripheral.

Parameters • `hadc`: ADC handle

Return values • `HAL`: status.

HAL_ADC_PollForConversion

Function name `HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)`

Function description Wait for regular group conversion to be completed.

Parameters • `hadc`: ADC handle

• `Timeout`: Timeout value in millisecond.

Return values • `HAL`: status

Notes

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

HAL_ADC_PollForEvent

Function name	<code>HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)</code>
Function description	Poll for conversion event.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle• EventType: the ADC event type. This parameter can be one of the following values:<ul style="list-style-type: none">– ADC_AWD_EVENT: ADC Analog watchdog event– ADC_OVR_EVENT: ADC Overrun event• Timeout: Timeout value in millisecond.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_ADC_Start_IT

Function name	<code>HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)</code>
Function description	Enables ADC, starts conversion of regular group with interruption.

HAL_ADC_Stop_IT

Function name	<code>HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)</code>
Function description	Stop ADC conversion of regular group, disable interruption of end-of-conversion, disable ADC peripheral.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• HAL: status.

HAL_ADC_Start_DMA

Function name	<code>HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)</code>
Function description	Enables ADC, starts conversion of regular group and transfers result through DMA.

HAL_ADC_Stop_DMA

Function name	<code>HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)</code>
Function description	Stop ADC conversion of regular group, disable ADC DMA transfer, disable ADC peripheral.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• HAL: status.

HAL_ADC_GetValue

Function name	<code>uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)</code>
Function description	Get ADC regular group conversion result.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• ADC: group regular conversion data
Notes	<ul style="list-style-type: none">• Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion).• This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC. If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: <code>HAL_ADC_IRQHandler()</code>, in programming model polling: <code>HAL_ADC_PollForConversion()</code> or <code>_HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS)</code>.

HAL_ADC_IRQHandler

Function name	<code>void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)</code>
Function description	Handles ADC interrupt request.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None:

HAL_ADC_ConvCpltCallback

Function name	<code>void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)</code>
Function description	Conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None:

HAL_ADC_ConvHalfCpltCallback

Function name	<code>void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)</code>
Function description	Conversion DMA half-transfer callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None:

HAL_ADC_LevelOutOfWindowCallback

Function name	void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)
Function description	Analog watchdog callback in non blocking mode.
Parameters	<ul style="list-style-type: none">hadc: ADC handle
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hadc: ADC handle
Return values	<ul style="list-style-type: none">None:

HAL_ADC_ConfigChannel

Function name	HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)
Function description	Configures the the selected channel to be linked to the regular group.
Parameters	<ul style="list-style-type: none">hadc: ADC handlesConfig: Structure of ADC channel for regular group.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">In case of usage of internal measurement channels: VrefInt/Vbat/TempSensor. Sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values, parameters TS_vrefint, TS_vbat, TS_temp (values rough order: 5us to 17us). These internal paths can be disabled using function HAL_ADC_Delnit().Possibility to update parameters on the fly: This function initializes channel into regular group, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name	HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)
Function description	Configures the analog watchdog.

Parameters	<ul style="list-style-type: none">• hadc: ADC handle• AnalogWDGConfig: Structure of ADC analog watchdog configuration
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• Possibility to update parameters on the fly: This function initializes the selected analog watchdog, following calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".

HAL_ADC_GetState

Function name	uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)
Function description	Return the ADC state.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• HAL: state
Notes	<ul style="list-style-type: none">• ADC state machine is managed by bitfields, ADC status must be compared with states bits. For example: " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_REG_BUSY)) " " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_AWD1)) "

HAL_ADC_GetError

Function name	uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)
Function description	Return the ADC error code.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• ADC: Error Code

7.3 ADC Firmware driver defines

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

7.3.1 ADC

ADC

ADC analog watchdog mode

**ADC_ANALOGWATCHDOG
_NONE**

**ADC_ANALOGWATCHDOG
_SINGLE_REG**

**ADC_ANALOGWATCHDOG
_ALL_REG**

ADC channels

ADC_CHANNEL_0

ADC_CHANNEL_1

ADC_CHANNEL_2

ADC_CHANNEL_3

ADC_CHANNEL_4

ADC_CHANNEL_5

ADC_CHANNEL_6

ADC_CHANNEL_7

ADC_CHANNEL_8

ADC_CHANNEL_9

ADC_CHANNEL_10

ADC_CHANNEL_11

ADC_CHANNEL_12

ADC_CHANNEL_13

ADC_CHANNEL_14

ADC_CHANNEL_15

ADC_CHANNEL_16

ADC_CHANNEL_17

ADC_CHANNEL_TEMPSEN
SOR

ADC_CHANNEL_VREFINT

ADC_CHANNEL_18

ADC_CHANNEL_VBAT

ADC ClockPrescaler

ADC_CLOCK_ASYNC_DIV1 ADC asynchronous clock derived from ADC dedicated HSI

ADC_CLOCK_SYNC_PCLK_DIV2 ADC synchronous clock derived from AHB clock divided by a prescaler of 2

ADC_CLOCK_SYNC_PCLK_DIV4 ADC synchronous clock derived from AHB clock divided by a prescaler of 4

ADC Data_align

ADC_DATAALIGN_RIGHT

ADC_DATAALIGN_LEFT

ADC EOCSelection

ADC_EOC_SINGLE_CONV

ADC_EOC_SEQ_CONV

ADC Error Code

HAL_ADC_ERROR_NONE No error

HAL_ADC_ERROR_INTER_NAL ADC IP internal error: if problem of clocking, enable/disable, erroneous state NAL

HAL_ADC_ERROR_OVR Overrun error

HAL_ADC_ERROR_DMA DMA transfer error

ADC Event type

ADC_AWD_EVENT ADC Analog watchdog 1 event

ADC_OVR_EVENT ADC overrun event

ADC Exported Constants

ADC_CCR_ALL

ADC Exported Macros

_HAL_ADC_ENABLE

Description:

- Enable the ADC peripheral.

Parameters:

- **_HANDLE_**: ADC handle

Return value:

- None

_HAL_ADC_DISABLE

Description:

- Disable the ADC peripheral.

Parameters:

- **_HANDLE_**: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the following values:
 - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
 - ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt source
 - ADC_IT_AWD: ADC Analog watchdog interrupt source
 - ADC_IT_OVR: ADC overrun interrupt source
 - ADC_IT_EOSMP: ADC End of Sampling interrupt source
 - ADC_IT_RDY: ADC Ready interrupt source

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the following values:
 - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
 - ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt source
 - ADC_IT_AWD: ADC Analog watchdog interrupt source
 - ADC_IT_OVR: ADC overrun interrupt source
 - ADC_IT_EOSMP: ADC End of Sampling interrupt source
 - ADC_IT_RDY: ADC Ready interrupt source

Return value:

- None

__HAL_ADC_GET_IT_SOU

RCE

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 any combination of the following values:
 - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
 - ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt source
 - ADC_IT_AWD: ADC Analog watchdog interrupt source
 - ADC_IT_OVR: ADC overrun interrupt source
 - ADC_IT_EOSMP: ADC End of Sampling interrupt source
 - ADC_IT_RDY: ADC Ready interrupt source

Return value:

- State: of interruption (SET or RESET)

<u>__HAL_ADC_GET_FLAG</u>	Description: <ul style="list-style-type: none">Get the selected ADC's flag status. Parameters: <ul style="list-style-type: none"><code>__HANDLE__</code>: ADC handle<code>__FLAG__</code>: ADC flag This parameter can be any combination of the following values:<ul style="list-style-type: none">- <code>ADC_FLAG_EOC</code>: ADC End of Regular conversion flag- <code>ADC_FLAG_EOS</code>: ADC End of Regular sequence of Conversions flag- <code>ADC_FLAG_AWD</code>: ADC Analog watchdog flag- <code>ADC_FLAG_OVR</code>: ADC overrun flag- <code>ADC_FLAG_EOSMP</code>: ADC End of Sampling flag- <code>ADC_FLAG_RDY</code>: ADC Ready flag Return value: <ul style="list-style-type: none">None
<u>__HAL_ADC_CLEAR_FLAG</u>	Description: <ul style="list-style-type: none">Clear the ADC's pending flags. Parameters: <ul style="list-style-type: none"><code>__HANDLE__</code>: ADC handle<code>__FLAG__</code>: ADC flag This parameter can be any combination of the following values:<ul style="list-style-type: none">- <code>ADC_FLAG_EOC</code>: ADC End of Regular conversion flag- <code>ADC_FLAG_EOS</code>: ADC End of Regular sequence of Conversions flag- <code>ADC_FLAG_AWD</code>: ADC Analog watchdog flag- <code>ADC_FLAG_OVR</code>: ADC overrun flag- <code>ADC_FLAG_EOSMP</code>: ADC End of Sampling flag- <code>ADC_FLAG_RDY</code>: ADC Ready flag Return value: <ul style="list-style-type: none">None
<u>__HAL_ADC_RESET_HANDLE_STATE</u>	Description: <ul style="list-style-type: none">Reset ADC handle state. Parameters: <ul style="list-style-type: none"><code>__HANDLE__</code>: ADC handle Return value: <ul style="list-style-type: none">None
<i>ADC Exported Types</i>	
<u>HAL_ADC_STATE_RESET</u>	Notes: <ul style="list-style-type: none">ADC state machine is managed by bitfields, state must be compared with bit by bit. For example: " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_REG_BUSY)) " " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_AWD1)) " ADC not yet initialized or disabled
<u>HAL_ADC_STATE_READY</u>	ADC peripheral ready for use
<u>HAL_ADC_STATE_BUSY_INTERNAL</u>	ADC is busy to internal process (initialization, calibration)

HAL_ADC_STATE_TIMEOU TimeOut occurrence
T

HAL_ADC_STATE_ERROR Internal error occurrence
_INTERNAL

HAL_ADC_STATE_ERROR Configuration error occurrence
_CONFIG

HAL_ADC_STATE_ERROR DMA error occurrence
_DMA

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

HAL_ADC_STATE_REG_E Conversion data available on group regular
OC

HAL_ADC_STATE_REG_O Overrun occurrence
VR

HAL_ADC_STATE_REG_E Not available on STM32F0 device: End Of Sampling flag raised
OSMP

HAL_ADC_STATE_INJ_BU Not available on STM32F0 device: A conversion on group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on, multimode ADC master control)
SY

HAL_ADC_STATE_INJ_EO Not available on STM32F0 device: Conversion data available on group injected
C

HAL_ADC_STATE_INJ_JQ Not available on STM32F0 device: Not available on STM32F0 device: Injected queue overflow
OVF

HAL_ADC_STATE_AWD1 Out-of-window occurrence of analog watchdog 1

HAL_ADC_STATE_AWD2 Not available on STM32F0 device: Out-of-window occurrence of analog watchdog 2

HAL_ADC_STATE_AWD3 Not available on STM32F0 device: Out-of-window occurrence of analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE Not available on STM32F0 device: ADC in multimode slave state, controlled by another ADC master (

ADC External trigger edge Regular

ADC_EXTERNALTRIGCON_VEDGE_NONE

ADC_EXTERNALTRIGCON_VEDGE_RISING

ADC_EXTERNALTRIGCON_VEDGE_FALLING

ADC_EXTERNALTRIGCON
VEDGE_RISINGFALLING

ADC External trigger source Regular

ADC_EXTERNALTRIGCON
V_T1_TRGO

ADC_EXTERNALTRIGCON
V_T1_CC4

ADC_EXTERNALTRIGCON
V_T3_TRGO

ADC_SOFTWARE_START

ADC_EXTERNALTRIGCON
V_T2_TRGO

ADC_EXTERNALTRIGCON
V_T15_TRGO

ADC flags definition

ADC_FLAG_AWD ADC Analog watchdog flag

ADC_FLAG_OVR ADC overrun flag

ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag

ADC_FLAG_EOC ADC End of Regular Conversion flag

ADC_FLAG_EOSMP ADC End of Sampling flag

ADC_FLAG_RDY ADC Ready flag

ADC Internal HAL driver Ext trig src Regular

ADC1_2_EXTERNALTRIG_
T1_TRGO

ADC1_2_EXTERNALTRIG_
T1_CC4

ADC1_2_EXTERNALTRIG_
T2_TRGO

ADC1_2_EXTERNALTRIG_
T3_TRGO

ADC1_2_EXTERNALTRIG_
T15_TRGO

ADC interrupts definition

ADC_IT_AWD	ADC Analog watchdog interrupt source
ADC_IT_OVR	ADC overrun interrupt source
ADC_IT_EOS	ADC End of Regular sequence of Conversions interrupt source
ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_EOSMP	ADC End of Sampling interrupt source
ADC_IT_RDY	ADC Ready interrupt source

ADC Overrun

ADC_OVR_DATA_OVERWR
ITTEN

ADC_OVR_DATA_PRESER
VED

ADC range verification

IS_ADC_RANGE

ADC rank

ADC_RANK_CHANNEL_NU MBER Enable the rank of the selected channels. Number of ranks in the sequence is defined by number of channels enabled, rank of each channel is defined by channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...)

ADC_RANK_NONE Disable the selected rank (selected channel) from sequencer

ADC regular rank verification

IS_ADC_REGULAR_RANK

ADC Resolution

ADC_RESOLUTION_12B ADC 12-bit resolution

ADC_RESOLUTION_10B ADC 10-bit resolution

ADC_RESOLUTION_8B ADC 8-bit resolution

ADC_RESOLUTION_6B ADC 6-bit resolution

ADC sampling times

ADC_SAMPLETIME_1CYCL E_5 Sampling time 1.5 ADC clock cycle

ADC_SAMPLETIME_7CYCL ES_5 Sampling time 7.5 ADC clock cycles

ADC_SAMPLETIME_13CYC Sampling time 13.5 ADC clock cycles
LES_5

ADC_SAMPLETIME_28CYC Sampling time 28.5 ADC clock cycles
LES_5

ADC_SAMPLETIME_41CYC Sampling time 41.5 ADC clock cycles
LES_5

ADC_SAMPLETIME_55CYC Sampling time 55.5 ADC clock cycles
LES_5

ADC_SAMPLETIME_71CYC Sampling time 71.5 ADC clock cycles
LES_5

ADC_SAMPLETIME_239CY Sampling time 239.5 ADC clock cycles
CLES_5

ADC Scan mode

ADC_SCAN_DIRECTION_F Scan direction forward: from channel 0 to channel 18
FORWARD

ADC_SCAN_DIRECTION_B Scan direction backward: from channel 18 to channel 0
ACKWARD

ADC_SCAN_ENABLE

8 HAL ADC Extension Driver

8.1 ADCEx Firmware driver API description

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

8.1.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC calibration.

This section contains the following APIs:

- [*HAL_ADCEx_Calibration_Start*](#)

8.1.2 Detailed description of functions

[*HAL_ADCEx_Calibration_Start*](#)

Function name [*HAL_StatusTypeDef HAL_ADCEx_Calibration_Start \(ADC_HandleTypeDef * hadc\)*](#)

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

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

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

8.2 ADCEx Firmware driver defines

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

8.2.1 ADCEx

ADCEX

9 HAL CAN Generic Driver

9.1 CAN Firmware driver registers structures

9.1.1 CAN_InitTypeDef

`CAN_InitTypeDef` is defined in the `stm32f0xx_hal_can.h`

Data Fields

- `uint32_t Prescaler`
- `uint32_t Mode`
- `uint32_t SJW`
- `uint32_t BS1`
- `uint32_t BS2`
- `uint32_t TTCM`
- `uint32_t ABOM`
- `uint32_t AWUM`
- `uint32_t NART`
- `uint32_t RFLM`
- `uint32_t TXFP`

Field Documentation

- `uint32_t CAN_InitTypeDef::Prescaler`
Specifies the length of a time quantum. This parameter must be a number between Min_Data = 1 and Max_Data = 1024.
- `uint32_t CAN_InitTypeDef::Mode`
Specifies the CAN operating mode. This parameter can be a value of `CAN_operating_mode`
- `uint32_t CAN_InitTypeDef::SJW`
Specifies the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter can be a value of `CAN_synchronisation_jump_width`
- `uint32_t CAN_InitTypeDef::BS1`
Specifies the number of time quanta in Bit Segment 1. This parameter can be a value of `CAN_time_quantum_in_bit_segment_1`
- `uint32_t CAN_InitTypeDef::BS2`
Specifies the number of time quanta in Bit Segment 2. This parameter can be a value of `CAN_time_quantum_in_bit_segment_2`
- `uint32_t CAN_InitTypeDef::TTCM`
Enable or disable the time triggered communication mode. This parameter can be set to ENABLE or DISABLE.
- `uint32_t CAN_InitTypeDef::ABOM`
Enable or disable the automatic bus-off management. This parameter can be set to ENABLE or DISABLE.
- `uint32_t CAN_InitTypeDef::AWUM`
Enable or disable the automatic wake-up mode. This parameter can be set to ENABLE or DISABLE.
- `uint32_t CAN_InitTypeDef::NART`
Enable or disable the non-automatic retransmission mode. This parameter can be set to ENABLE or DISABLE.
- `uint32_t CAN_InitTypeDef::RFLM`
Enable or disable the Receive FIFO Locked mode. This parameter can be set to ENABLE or DISABLE.
- `uint32_t CAN_InitTypeDef::TXFP`
Enable or disable the transmit FIFO priority. This parameter can be set to ENABLE or DISABLE.

9.1.2 CAN_FilterTypeDef

CAN_FilterTypeDef is defined in the `stm32f0xx_hal_can.h`

Data Fields

- `uint32_t FilterIdHigh`
- `uint32_t FilterIdLow`
- `uint32_t FilterMaskIdHigh`
- `uint32_t FilterMaskIdLow`
- `uint32_t FilterFIFOAssignment`
- `uint32_t FilterNumber`
- `uint32_t FilterMode`
- `uint32_t FilterScale`
- `uint32_t FilterActivation`
- `uint32_t BankNumber`

Field Documentation

- **`uint32_t CAN_FilterTypeDef::FilterIdHigh`**

Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

- **`uint32_t CAN_FilterTypeDef::FilterIdLow`**

Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

- **`uint32_t CAN_FilterTypeDef::FilterMaskIdHigh`**

Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

- **`uint32_t CAN_FilterTypeDef::FilterMaskIdLow`**

Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

- **`uint32_t CAN_FilterTypeDef::FilterFIFOAssignment`**

Specifies the FIFO (0 or 1U) which will be assigned to the filter. This parameter can be a value of `CAN_filter_FIFO`

- **`uint32_t CAN_FilterTypeDef::FilterNumber`**

Specifies the filter which will be initialized. This parameter must be a number between Min_Data = 0 and Max_Data = 27.

- **`uint32_t CAN_FilterTypeDef::FilterMode`**

Specifies the filter mode to be initialized. This parameter can be a value of `CAN_filter_mode`

- **`uint32_t CAN_FilterTypeDef::FilterScale`**

Specifies the filter scale. This parameter can be a value of `CAN_filter_scale`

- **`uint32_t CAN_FilterTypeDef::FilterActivation`**

Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.

- **`uint32_t CAN_FilterTypeDef::BankNumber`**

Select the start slave bank filter This parameter must be a number between Min_Data = 0 and Max_Data = 28.

9.1.3 CanTxMsgTypeDef

CanTxMsgTypeDef is defined in the `stm32f0xx_hal_can.h`

Data Fields

- `uint32_t StdId`
- `uint32_t ExtId`
- `uint32_t IDE`
- `uint32_t RTR`

- `uint32_t DLC`
- `uint8_t Data`

Field Documentation

- `uint32_t CanTxMsgTypeDef::StdId`

Specifies the standard identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x7FF.

- `uint32_t CanTxMsgTypeDef::ExtId`

Specifies the extended identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x1FFFFFFF.

- `uint32_t CanTxMsgTypeDef::IDE`

Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of `CAN_identifier_type`

- `uint32_t CanTxMsgTypeDef::RTR`

Specifies the type of frame for the message that will be transmitted. This parameter can be a value of `CAN_remote_transmission_request`

- `uint32_t CanTxMsgTypeDef::DLC`

Specifies the length of the frame that will be transmitted. This parameter must be a number between Min_Data = 0 and Max_Data = 8.

- `uint8_t CanTxMsgTypeDef::Data[8]`

Contains the data to be transmitted. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF.

9.1.4 CanRxMsgTypeDef

`CanRxMsgTypeDef` is defined in the `stm32f0xx_hal_can.h`

Data Fields

- `uint32_t StdId`
- `uint32_t ExtId`
- `uint32_t IDE`
- `uint32_t RTR`
- `uint32_t DLC`
- `uint8_t Data`
- `uint32_t FMI`
- `uint32_t FIFONumber`

Field Documentation

- `uint32_t CanRxMsgTypeDef::StdId`

Specifies the standard identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x7FF.

- `uint32_t CanRxMsgTypeDef::ExtId`

Specifies the extended identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x1FFFFFFF.

- `uint32_t CanRxMsgTypeDef::IDE`

Specifies the type of identifier for the message that will be received. This parameter can be a value of `CAN_identifier_type`

- `uint32_t CanRxMsgTypeDef::RTR`

Specifies the type of frame for the received message. This parameter can be a value of `CAN_remote_transmission_request`

- `uint32_t CanRxMsgTypeDef::DLC`

Specifies the length of the frame that will be received. This parameter must be a number between Min_Data = 0 and Max_Data = 8.

- **`uint8_t CanRxMsgTypeDef::Data[8]`**
Contains the data to be received. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF.
- **`uint32_t CanRxMsgTypeDef::FMI`**
Specifies the index of the filter the message stored in the mailbox passes through. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF.
- **`uint32_t CanRxMsgTypeDef::FIFONumber`**
Specifies the receive FIFO number. This parameter can be CAN_FIFO0 or CAN_FIFO1

9.1.5 CAN_HandleTypeDef

`CAN_HandleTypeDef` is defined in the `stm32f0xx_hal_can.h`

Data Fields

- `CAN_TypeDef * Instance`
- `CAN_InitTypeDef Init`
- `CanTxMsgTypeDef * pTxMsg`
- `CanRxMsgTypeDef * pRxMsg`
- `CanRxMsgTypeDef * pRx1Msg`
- `HAL_LockTypeDef Lock`
- `_IO HAL_CAN_StateTypeDef State`
- `_IO uint32_t ErrorCode`

Field Documentation

- **`CAN_TypeDef* CAN_HandleTypeDef::Instance`**
Register base address
- **`CAN_InitTypeDef CAN_HandleTypeDef::Init`**
CAN required parameters
- **`CanTxMsgTypeDef* CAN_HandleTypeDef::pTxMsg`**
Pointer to transmit structure
- **`CanRxMsgTypeDef* CAN_HandleTypeDef::pRxMsg`**
Pointer to reception structure for RX FIFO0 msg
- **`CanRxMsgTypeDef* CAN_HandleTypeDef::pRx1Msg`**
Pointer to reception structure for RX FIFO1 msg
- **`HAL_LockTypeDef CAN_HandleTypeDef::Lock`**
CAN locking object
- **`_IO HAL_CAN_StateTypeDef CAN_HandleTypeDef::State`**
CAN communication state
- **`_IO uint32_t CAN_HandleTypeDef::ErrorCode`**
CAN Error code This parameter can be a value of `CAN_Error_Code`

9.2 CAN Firmware driver API description

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

9.2.1 How to use this driver

1. Enable the CAN controller interface clock using `__HAL_RCC_CAN1_CLK_ENABLE()`;
2. CAN pins configuration
 - Enable the clock for the CAN GPIOs using the following function:
`__HAL_RCC_GPIOx_CLK_ENABLE();`
 - Connect and configure the involved CAN pins to AF9 using the following function `HAL_GPIO_Init()`;
3. Initialise and configure the CAN using `HAL_CAN_Init()` function.
4. Transmit the desired CAN frame using `HAL_CAN_Transmit()` function.
5. Or transmit the desired CAN frame using `HAL_CAN_Transmit_IT()` function.

6. Receive a CAN frame using HAL_CAN_Receive() function.
7. Or receive a CAN frame using HAL_CAN_Receive_IT() function.

Polling mode IO operation

- Start the CAN peripheral transmission and wait the end of this operation using HAL_CAN_Transmit(), at this stage user can specify the value of timeout according to his end application
- Start the CAN peripheral reception and wait the end of this operation using HAL_CAN_Receive(), at this stage user can specify the value of timeout according to his end application

Interrupt mode IO operation

- Start the CAN peripheral transmission using HAL_CAN_Transmit_IT()
- Start the CAN peripheral reception using HAL_CAN_Receive_IT()
- Use HAL_CAN_IRQHandler() called under the used CAN Interrupt subroutine
- At CAN end of transmission HAL_CAN_TxCpltCallback() function is executed and user can add his own code by customization of function pointer HAL_CAN_TxCpltCallback
- In case of CAN Error, HAL_CAN_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_CAN_ErrorCallback

CAN HAL driver macros list

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

- __HAL_CAN_ENABLE_IT: Enable the specified CAN interrupts
- __HAL_CAN_DISABLE_IT: Disable the specified CAN interrupts
- __HAL_CAN_GET_IT_SOURCE: Check if the specified CAN interrupt source is enabled or disabled
- __HAL_CAN_CLEAR_FLAG: Clear the CAN's pending flags
- __HAL_CAN_GET_FLAG: Get the selected CAN's flag status

Note:

You can refer to the CAN HAL driver header file for more useful macros

9.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the CAN.
- De-initialize the CAN.

This section contains the following APIs:

- [**HAL_CAN_Init**](#)
- [**HAL_CAN_ConfigFilter**](#)
- [**HAL_CAN_DeInit**](#)
- [**HAL_CAN_MspInit**](#)
- [**HAL_CAN_MspDeInit**](#)

9.2.3 Peripheral State and Error functions

This subsection provides functions allowing to :

- Check the CAN state.
- Check CAN Errors detected during interrupt process

This section contains the following APIs:

- [**HAL_CAN_GetState**](#)
- [**HAL_CAN_GetError**](#)

9.2.4 Detailed description of functions

[**HAL_CAN_Init**](#)

Function name

[**HAL_StatusTypeDef HAL_CAN_Init \(CAN_HandleTypeDef * hcan\)**](#)

Function description Initializes the CAN peripheral according to the specified parameters in the CAN_InitStruct.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- **HAL:** status

HAL_CAN_ConfigFilter

Function name **HAL_StatusTypeDef HAL_CAN_ConfigFilter (CAN_HandleTypeDef * hcan, CAN_FilterConfTypeDef * sFilterConfig)**

Function description Configures the CAN reception filter according to the specified parameters in the CAN_FilterInitStruct.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **sFilterConfig:** pointer to a CAN_FilterConfTypeDef structure that contains the filter configuration information.

Return values

- **None:**

HAL_CAN_DeInit

Function name **HAL_StatusTypeDef HAL_CAN_DeInit (CAN_HandleTypeDef * hcan)**

Function description Deinitializes the CANx peripheral registers to their default reset values.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- **HAL:** status

HAL_CAN_MspInit

Function name **void HAL_CAN_MspInit (CAN_HandleTypeDef * hcan)**

Function description Initializes the CAN MSP.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- **None:**

HAL_CAN_MspDeInit

Function name **void HAL_CAN_MspDeInit (CAN_HandleTypeDef * hcan)**

Function description Deinitializes the CAN MSP.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values	<ul style="list-style-type: none">• None:
HAL_CAN_Transmit	
Function name	HAL_StatusTypeDef HAL_CAN_Transmit (CAN_HandleTypeDef * hcan, uint32_t Timeout)
Function description Initiates and transmits a CAN frame message.	
Parameters	<ul style="list-style-type: none">• hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.• Timeout: Timeout duration.
Return values	<ul style="list-style-type: none">• HAL: status
HAL_CAN_Transmit_IT	
Function name	HAL_StatusTypeDef HAL_CAN_Transmit_IT (CAN_HandleTypeDef * hcan)
Function description Initiates and transmits a CAN frame message.	
Parameters	<ul style="list-style-type: none">• hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	<ul style="list-style-type: none">• HAL: status
HAL_CAN_Receive	
Function name	HAL_StatusTypeDef HAL_CAN_Receive (CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t Timeout)
Function description Receives a correct CAN frame.	
Parameters	<ul style="list-style-type: none">• hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.• FIFONumber: FIFO number.• Timeout: Timeout duration.
Return values	<ul style="list-style-type: none">• HAL: status
HAL_CAN_Receive_IT	
Function name	HAL_StatusTypeDef HAL_CAN_Receive_IT (CAN_HandleTypeDef * hcan, uint8_t FIFONumber)
Function description Receives a correct CAN frame.	
Parameters	<ul style="list-style-type: none">• hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.• FIFONumber: FIFO number.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_CAN_Sleep

Function name `HAL_StatusTypeDef HAL_CAN_Sleep (CAN_HandleTypeDef * hcan)`

Function description Enters the Sleep (low power) mode.

Parameters

- `hcan`: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- `HAL`: status.

HAL_CAN_WakeUp

Function name `HAL_StatusTypeDef HAL_CAN_WakeUp (CAN_HandleTypeDef * hcan)`

Function description Wakes up the CAN peripheral from sleep mode, after that the CAN peripheral is in the normal mode.

Parameters

- `hcan`: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- `HAL`: status.

HAL_CAN_IRQHandler

Function name `void HAL_CAN_IRQHandler (CAN_HandleTypeDef * hcan)`

Function description Handles CAN interrupt request.

Parameters

- `hcan`: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- `None`:

HAL_CAN_TxCpltCallback

Function name `void HAL_CAN_TxCpltCallback (CAN_HandleTypeDef * hcan)`

Function description Transmission complete callback in non blocking mode.

Parameters

- `hcan`: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- `None`:

HAL_CAN_RxCpltCallback

Function name `void HAL_CAN_RxCpltCallback (CAN_HandleTypeDef * hcan)`

Function description Transmission complete callback in non blocking mode.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- **None:**

HAL_CAN_ErrorCallback

Function name

void HAL_CAN_ErrorCallback (CAN_HandleTypeDef * hcan)

Function description

Error CAN callback.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- **None:**

HAL_CAN_GetError

Function name

uint32_t HAL_CAN_GetError (CAN_HandleTypeDef * hcan)

Function description

Return the CAN error code.

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- **CAN: Error Code**

HAL_CAN_GetState

Function name

HAL_CAN_StateTypeDef HAL_CAN_GetState (CAN_HandleTypeDef * hcan)

Function description

return the CAN state

Parameters

- **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- **HAL: state**

9.3 CAN Firmware driver defines

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

9.3.1 CAN

CAN

CAN Error Code

HAL_CAN_ERROR_NONE No error

HAL_CAN_ERROR_EWG EWG error

HAL_CAN_ERROR_EPV EPV error

HAL_CAN_ERROR_BOF BOF error

HAL_CAN_ERROR_STF	Stuff error
HAL_CAN_ERROR_FOR	Form error
HAL_CAN_ERROR_ACK	Acknowledgment error
HAL_CAN_ERROR_BR	Bit recessive
HAL_CAN_ERROR_BD	LEC dominant
HAL_CAN_ERROR_CRC	LEC transfer error
HAL_CAN_ERROR_FOV0	FIFO0 overrun error
HAL_CAN_ERROR_FOV1	FIFO1 overrun error
HAL_CAN_ERROR_TXFAIL	Transmit failure

CAN Exported Macros

_HAL_CAN_RESET_HANDLE_STATE	Description: <ul style="list-style-type: none">Reset CAN handle state. Parameters: <ul style="list-style-type: none">_HANDLE_: CAN handle. Return value: <ul style="list-style-type: none">None
_HAL_CAN_ENABLE_IT	Description: <ul style="list-style-type: none">Enable the specified CAN interrupts. Parameters: <ul style="list-style-type: none">_HANDLE_: CAN handle._INTERRUPT_: CAN Interrupt Return value: <ul style="list-style-type: none">None
_HAL_CAN_DISABLE_IT	Description: <ul style="list-style-type: none">Disable the specified CAN interrupts. Parameters: <ul style="list-style-type: none">_HANDLE_: CAN handle._INTERRUPT_: CAN Interrupt Return value: <ul style="list-style-type: none">None

__HAL_CAN_MSG_PENDIN Description:

G

- Return the number of pending received messages.

Parameters:

- __HANDLE__: CAN handle.
- __FIFONUMBER__: Receive FIFO number, CAN_FIFO0 or CAN_FIFO1.

Return value:

- The: number of pending message.

__HAL_CAN_GET_FLAG Description:

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

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - CAN_TSR_RQCP0: Request MailBox0 Flag
 - CAN_TSR_RQCP1: Request MailBox1 Flag
 - CAN_TSR_RQCP2: Request MailBox2 Flag
 - CAN_FLAG_TXOK0: Transmission OK MailBox0 Flag
 - CAN_FLAG_TXOK1: Transmission OK MailBox1 Flag
 - CAN_FLAG_TXOK2: Transmission OK MailBox2 Flag
 - CAN_FLAG_TME0: Transmit mailbox 0 empty Flag
 - CAN_FLAG_TME1: Transmit mailbox 1 empty Flag
 - CAN_FLAG_TME2: Transmit mailbox 2 empty Flag
 - CAN_FLAG_FMP0: FIFO 0 Message Pending Flag
 - CAN_FLAG_FF0: FIFO 0 Full Flag
 - CAN_FLAG_FOV0: FIFO 0 Overrun Flag
 - CAN_FLAG_FMP1: FIFO 1 Message Pending Flag
 - CAN_FLAG_FF1: FIFO 1 Full Flag
 - CAN_FLAG_FOV1: FIFO 1 Overrun Flag
 - CAN_FLAG_WKU: Wake up Flag
 - CAN_FLAG_SLAK: Sleep acknowledge Flag
 - CAN_FLAG_SLAKE: Sleep acknowledge Flag
 - CAN_FLAG_EWG: Error Warning Flag
 - CAN_FLAG_EPV: Error Passive Flag
 - CAN_FLAG_BOF: Bus-Off Flag

Return value:

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

__HAL_CAN_CLEAR_FLAG **Description:**

G

- Clear the specified CAN pending flag.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - CAN_TSR_RQCP0: Request MailBox0 Flag
 - CAN_TSR_RQCP1: Request MailBox1 Flag
 - CAN_TSR_RQCP2: Request MailBox2 Flag
 - CAN_FLAG_TXOK0: Transmission OK MailBox0 Flag
 - CAN_FLAG_TXOK1: Transmission OK MailBox1 Flag
 - CAN_FLAG_TXOK2: Transmission OK MailBox2 Flag
 - CAN_FLAG_TME0: Transmit mailbox 0 empty Flag
 - CAN_FLAG_TME1: Transmit mailbox 1 empty Flag
 - CAN_FLAG_TME2: Transmit mailbox 2 empty Flag
 - CAN_FLAG_FMP0: FIFO 0 Message Pending Flag
 - CAN_FLAG_FF0: FIFO 0 Full Flag
 - CAN_FLAG_FOV0: FIFO 0 Overrun Flag
 - CAN_FLAG_FMP1: FIFO 1 Message Pending Flag
 - CAN_FLAG_FF1: FIFO 1 Full Flag
 - CAN_FLAG_FOV1: FIFO 1 Overrun Flag
 - CAN_FLAG_WKU: Wake up Flag
 - CAN_FLAG_SLAKI: Sleep acknowledge Flag
 - CAN_FLAG_EWG: Error Warning Flag
 - CAN_FLAG_EPV: Error Passive Flag
 - CAN_FLAG_BOF: Bus-Off Flag

Return value:

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

__HAL_CAN_GET_IT_SOURCE **Description:**

RCE

- Check if the specified CAN interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __INTERRUPT__: specifies the CAN interrupt source to check. This parameter can be one of the following values:
 - CAN_IT_TME: Transmit mailbox empty interrupt enable
 - CAN_IT_FMP0: FIFO0 message pending interrupt enable
 - CAN_IT_FMP1: FIFO1 message pending interrupt enable

Return value:

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

__HAL_CAN_TRANSMIT_STATUS **Description:**

TATUS

- Check the transmission status of a CAN Frame.

Parameters:

- __HANDLE__: CAN handle.
- __TRANSMITMAILBOX__: the number of the mailbox that is used for transmission.

Return value:

- The: new status of transmission (TRUE or FALSE).

__HAL_CAN_FIFO_RELEASE **Description:**
SE

- Release the specified receive FIFO.

Parameters:

- **__HANDLE__**: CAN handle.
- **__FIFONUMBER__**: Receive FIFO number, CAN_FIFO0 or CAN_FIFO1.

Return value:

- None

__HAL_CAN_CANCEL_TRANSMIT **Description:**
ANSMIT

- Cancel a transmit request.

Parameters:

- **__HANDLE__**: specifies the CAN Handle.
- **__TRANSMITMAILBOX__**: the number of the mailbox that is used for transmission.

Return value:

- None

__HAL_CAN_DBG_FREEZE **Description:**
E

- Enable or disables the DBG Freeze for CAN.

Parameters:

- **__HANDLE__**: specifies the CAN Handle.
- **__NEWSTATE__**: new state of the CAN peripheral. This parameter can be: ENABLE (CAN reception/transmission is frozen during debug. Reception FIFOs can still be accessed/controlled normally) or DISABLE (CAN is working during debug).

Return value:

- None

CAN Filter FIFO

CAN_FILTER_FIFO0 Filter FIFO 0 assignment for filter x

CAN_FILTER_FIFO1 Filter FIFO 1 assignment for filter x

CAN Filter Mode

CAN_FILTERMODE_IDMASK Identifier mask mode
K

CAN_FILTERMODE_IDLIST Identifier list mode

CAN Filter Scale

CAN_FILTERSCALE_16BIT Two 16-bit filters

CAN_FILTERSCALE_32BIT One 32-bit filter

CAN Flags

CAN_FLAG_RQCP0 Request MailBox0 flag

CAN_FLAG_RQCP1 Request MailBox1 flag

CAN_FLAG_RQCP2	Request MailBox2 flag
CAN_FLAG_TXOK0	Transmission OK MailBox0 flag
CAN_FLAG_TXOK1	Transmission OK MailBox1 flag
CAN_FLAG_TXOK2	Transmission OK MailBox2 flag
CAN_FLAG_TME0	Transmit mailbox 0 empty flag
CAN_FLAG_TME1	Transmit mailbox 0 empty flag
CAN_FLAG_TME2	Transmit mailbox 0 empty flag
CAN_FLAG_FF0	FIFO 0 Full flag
CAN_FLAG_FOV0	FIFO 0 Overrun flag
CAN_FLAG_FF1	FIFO 1 Full flag
CAN_FLAG_FOV1	FIFO 1 Overrun flag
CAN_FLAG_INAK	Initialization acknowledge flag
CAN_FLAG_SLAK	Sleep acknowledge flag
CAN_FLAG_ERRI	Error flag
CAN_FLAG_WKU	Wake up flag
CAN_FLAG_SLAKI	Sleep acknowledge flag
CAN_FLAG_EWG	Error warning flag
CAN_FLAG_EPV	Error passive flag
CAN_FLAG_BOF	Bus-Off flag

CAN Identifier Type

CAN_ID_STD	Standard Id
CAN_ID_EXT	Extended Id

CAN InitStatus

CAN_INITSTATUS_FAILED	CAN initialization failed
CAN_INITSTATUS_SUCCESS	CAN initialization OK

CAN Interrupts

CAN_IT_TME	Transmit mailbox empty interrupt
CAN_IT_FMP0	FIFO 0 message pending interrupt
CAN_IT_FF0	FIFO 0 full interrupt
CAN_IT_FOV0	FIFO 0 overrun interrupt
CAN_IT_FMP1	FIFO 1 message pending interrupt
CAN_IT_FF1	FIFO 1 full interrupt
CAN_IT_FOV1	FIFO 1 overrun interrupt
CAN_IT_WKU	Wake-up interrupt
CAN_IT_SLK	Sleep acknowledge interrupt
CAN_IT_EWG	Error warning interrupt
CAN_IT_EPV	Error passive interrupt
CAN_IT_BOF	Bus-off interrupt
CAN_IT_LEC	Last error code interrupt
CAN_IT_ERR	Error Interrupt

CAN Mailboxes

CAN_TXMAILBOX_0
CAN_TXMAILBOX_1
CAN_TXMAILBOX_2

CAN Operating Mode

CAN_MODE_NORMAL	Normal mode
CAN_MODE_LOOPBACK	Loopback mode
CAN_MODE_SILENT	Silent mode
CAN_MODE_SILENT_LOOPBACK	Loopback combined with silent mode

CAN Receive FIFO Number

CAN_FIFO0	CAN FIFO 0 used to receive
CAN_FIFO1	CAN FIFO 1 used to receive

CAN Remote Transmission Request**CAN_RTR_DATA** Data frame**CAN_RTR_REMOTE** Remote frame**CAN Synchronization Jump Width****CAN_SJW_1TQ** 1 time quantum**CAN_SJW_2TQ** 2 time quantum**CAN_SJW_3TQ** 3 time quantum**CAN_SJW_4TQ** 4 time quantum**CAN Time Quantum in Bit Segment 1****CAN_BS1_1TQ** 1 time quantum**CAN_BS1_2TQ** 2 time quantum**CAN_BS1_3TQ** 3 time quantum**CAN_BS1_4TQ** 4 time quantum**CAN_BS1_5TQ** 5 time quantum**CAN_BS1_6TQ** 6 time quantum**CAN_BS1_7TQ** 7 time quantum**CAN_BS1_8TQ** 8 time quantum**CAN_BS1_9TQ** 9 time quantum**CAN_BS1_10TQ** 10 time quantum**CAN_BS1_11TQ** 11 time quantum**CAN_BS1_12TQ** 12 time quantum**CAN_BS1_13TQ** 13 time quantum**CAN_BS1_14TQ** 14 time quantum**CAN_BS1_15TQ** 15 time quantum**CAN_BS1_16TQ** 16 time quantum**CAN Time Quantum in Bit Segment 2**

CAN_BS2_1TQ	1 time quantum
CAN_BS2_2TQ	2 time quantum
CAN_BS2_3TQ	3 time quantum
CAN_BS2_4TQ	4 time quantum
CAN_BS2_5TQ	5 time quantum
CAN_BS2_6TQ	6 time quantum
CAN_BS2_7TQ	7 time quantum
CAN_BS2_8TQ	8 time quantum

10 HAL CEC Generic Driver

10.1 CEC Firmware driver registers structures

10.1.1 CEC_InitTypeDef

CEC_InitTypeDef is defined in the `stm32f0xx_hal_cec.h`

Data Fields

- `uint32_t SignalFreeTime`
- `uint32_t Tolerance`
- `uint32_t BRERxStop`
- `uint32_t BREErrorBitGen`
- `uint32_t LBPEErrorBitGen`
- `uint32_t BroadcastMsgNoErrorBitGen`
- `uint32_t SignalFreeTimeOption`
- `uint32_t ListenMode`
- `uint16_t OwnAddress`
- `uint8_t * RxBuffer`

Field Documentation

- **`uint32_t CEC_InitTypeDef::SignalFreeTime`**

Set SFT field, specifies the Signal Free Time. It can be one of **CEC_Signal_Free_Time** and belongs to the set {0,...,7} where 0x0 is the default configuration else means 0.5 + (SignalFreeTime - 1) nominal data bit periods

- **`uint32_t CEC_InitTypeDef::Tolerance`**

Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of **CEC_Tolerance** : it is either CEC_STANDARD_TOLERANCE or CEC_EXTENDED_TOLERANCE

- **`uint32_t CEC_InitTypeDef::BRERxStop`**

Set BRESTOP bit **CEC_BRERxStop** : specifies whether or not a Bit Rising Error stops the reception. CEC_NO_RX_STOP_ON_BRE: reception is not stopped. CEC_RX_STOP_ON_BRE: reception is stopped.

- **`uint32_t CEC_InitTypeDef::BREErrorBitGen`**

Set BREGEN bit **CEC_BREErrorBitGen** : specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection. CEC_BRE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_BRE_ERRORBIT_GENERATION: error-bit generation if BRESTOP is set.

- **`uint32_t CEC_InitTypeDef::LBPEErrorBitGen`**

Set LBPEGEN bit **CEC_LBPEErrorBitGen** : specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection. CEC_LBPE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_LBPE_ERRORBIT_GENERATION: error-bit generation.

- **`uint32_t CEC_InitTypeDef::BroadcastMsgNoErrorBitGen`**

Set BRDNOGEN bit **CEC_BroadCastMsgErrorBitGen** : allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message. It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values:1)

CEC_BROADCASTERROR_ERRORBIT_GENERATION. a) BRE detection: error-bit generation on the CEC line if BRESTOP=CEC_RX_STOP_ON_BRE and BREGEN=CEC_BRE_ERRORBIT_NO_GENERATION. b) LBPE detection: error-bit generation on the CEC line if LBPGEN=CEC_LBPE_ERRORBIT_NO_GENERATION.2)

CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.

- **`uint32_t CEC_InitTypeDef::SignalFreeTimeOption`**
Set SFTOP bit **`CEC_SFT_Option`** : specifies when SFT timer starts. CEC_SFT_START_ON_TXSOM SFT: timer starts when TXSOM is set by software. CEC_SFT_START_ON_TX_RX_END: SFT timer starts automatically at the end of message transmission/reception.
- **`uint32_t CEC_InitTypeDef::ListenMode`**
Set LSTN bit **`CEC_Listening_Mode`** : specifies device listening mode. It can take two values: CEC_REDUCED_LISTENING_MODE: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received. CEC_FULL_LISTENING_MODE: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
- **`uint16_t CEC_InitTypeDef::OwnAddress`**
Own addresses configuration This parameter can be a value of **`CEC_OWN_ADDRESS`**
- **`uint8_t* CEC_InitTypeDef::RxBuffer`**
CEC Rx buffer pointeur

10.1.2 CEC_HandleTypeDef

`CEC_HandleTypeDef` is defined in the `stm32f0xx_hal_cec.h`

Data Fields

- **`CEC_TypeDef * Instance`**
- **`CEC_InitTypeDef Init`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferCount`**
- **`uint16_t RxXferSize`**
- **`HAL_LockTypeDef Lock`**
- **`HAL_CEC_StateTypeDef gState`**
- **`HAL_CEC_StateTypeDef RxState`**
- **`uint32_t ErrorCode`**

Field Documentation

- **`CEC_TypeDef* CEC_HandleTypeDef::Instance`**
CEC registers base address
- **`CEC_InitTypeDef CEC_HandleTypeDef::Init`**
CEC communication parameters
- **`uint8_t* CEC_HandleTypeDef::pTxBuffPtr`**
Pointer to CEC Tx transfer Buffer
- **`uint16_t CEC_HandleTypeDef::TxXferCount`**
CEC Tx Transfer Counter
- **`uint16_t CEC_HandleTypeDef::RxXferSize`**
CEC Rx Transfer size, 0: header received only
- **`HAL_LockTypeDef CEC_HandleTypeDef::Lock`**
Locking object
- **`HAL_CEC_StateTypeDef CEC_HandleTypeDef::gState`**
CEC state information related to global Handle management and also related to Tx operations. This parameter can be a value of **`HAL_CEC_StateTypeDef`**
- **`HAL_CEC_StateTypeDef CEC_HandleTypeDef::RxState`**
CEC state information related to Rx operations. This parameter can be a value of **`HAL_CEC_StateTypeDef`**
- **`uint32_t CEC_HandleTypeDef::ErrorCode`**
For errors handling purposes, copy of ISR register in case error is reported

10.2 CEC Firmware driver API description

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

10.2.1 How to use this driver

Note: The specific CEC interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros `_HAL_CEC_ENABLE_IT()` and `_HAL_CEC_DISABLE_IT()` inside the transmit and receive process. (#) Program the Signal Free Time (SFT) and SFT option, Tolerance, reception stop in case of Bit Rising Error, Error-Bit generation conditions, device logical address and Listen mode in the hcec Init structure. (#) Initialize the CEC registers by calling the `HAL_CEC_Init()` API.

Note: This API (`HAL_CEC_Init()`) configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized `HAL_CEC_MspInit()` API.

10.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the CEC

- The following parameters need to be configured:
 - SignalFreeTime
 - Tolerance
 - BRERxStop (RX stopped or not upon Bit Rising Error)
 - BREErrorBitGen (Error-Bit generation in case of Bit Rising Error)
 - LBPErrorBitGen (Error-Bit generation in case of Long Bit Period Error)
 - BroadcastMsgNoErrorBitGen (Error-bit generation in case of broadcast message error)
 - SignalFreeTimeOption (SFT Timer start definition)
 - OwnAddress (CEC device address)
 - ListenMode

This section contains the following APIs:

- [`HAL_CEC_Init`](#)
- [`HAL_CEC_DelInit`](#)
- [`HAL_CEC_SetDeviceAddress`](#)
- [`HAL_CEC_MspInit`](#)
- [`HAL_CEC_MspDelInit`](#)

10.2.3 IO operation functions

This section contains the following APIs:

- [`HAL_CEC_Transmit_IT`](#)
- [`HAL_CEC_GetLastReceivedFrameSize`](#)
- [`HAL_CEC_ChangeRxBuffer`](#)
- [`HAL_CEC_IRQHandler`](#)
- [`HAL_CEC_TxCpltCallback`](#)
- [`HAL_CEC_RxCpltCallback`](#)
- [`HAL_CEC_ErrorCallback`](#)

10.2.4 Peripheral Control function

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

- `HAL_CEC_GetState()` API can be helpful to check in run-time the state of the CEC peripheral.
- `HAL_CEC_GetError()` API can be helpful to check in run-time the error of the CEC peripheral.

This section contains the following APIs:

- [`HAL_CEC_GetState`](#)
- [`HAL_CEC_GetError`](#)

10.2.5 Detailed description of functions

`HAL_CEC_Init`

Function name

`HAL_StatusTypeDef HAL_CEC_Init (CEC_HandleTypeDef * hcec)`

Function description Initializes the CEC mode according to the specified parameters in the CEC_InitTypeDef and creates the associated handle .

Parameters • **hcec:** CEC handle

Return values • **HAL:** status

HAL_CEC_DelInit

Function name **HAL_StatusTypeDef HAL_CEC_DelInit (CEC_HandleTypeDef * hcec)**

Function description Delinitializes the CEC peripheral.

Parameters • **hcec:** CEC handle

Return values • **HAL:** status

HAL_CEC_SetDeviceAddress

Function name **HAL_StatusTypeDef HAL_CEC_SetDeviceAddress (CEC_HandleTypeDef * hcec, uint16_t CEC_OwnAddress)**

Function description Initializes the Own Address of the CEC device.

Parameters • **hcec:** CEC handle
 • **CEC_OwnAddress:** The CEC own address.

Return values • **HAL:** status

HAL_CEC_MspInit

Function name **void HAL_CEC_MspInit (CEC_HandleTypeDef * hcec)**

Function description CEC MSP Init.

Parameters • **hcec:** CEC handle

Return values • **None:**

HAL_CEC_MspDelInit

Function name **void HAL_CEC_MspDelInit (CEC_HandleTypeDef * hcec)**

Function description CEC MSP DelInit.

Parameters • **hcec:** CEC handle

Return values • **None:**

HAL_CEC_Transmit_IT

Function name	<code>HAL_StatusTypeDef HAL_CEC_Transmit_IT (CEC_HandleTypeDef * hcec, uint8_t_t InitiatorAddress, uint8_t_t DestinationAddress, uint8_t_t * pData, uint32_t Size)</code>
Function description	Send data in interrupt mode.
Parameters	<ul style="list-style-type: none">hcec: CEC handleInitiatorAddress: Initiator addressDestinationAddress: destination logical addresspData: pointer to input byte data bufferSize: amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).
Return values	<ul style="list-style-type: none">HAL: status

HAL_CEC_GetLastReceivedFrameSize

Function name	<code>uint32_t HAL_CEC_GetLastReceivedFrameSize (CEC_HandleTypeDef * hcec)</code>
Function description	Get size of the received frame.
Parameters	<ul style="list-style-type: none">hcec: CEC handle
Return values	<ul style="list-style-type: none">Frame: size

HAL_CEC_ChangeRxBuffer

Function name	<code>void HAL_CEC_ChangeRxBuffer (CEC_HandleTypeDef * hcec, uint8_t_t * Rxbuffer)</code>
Function description	Change Rx Buffer.
Parameters	<ul style="list-style-type: none">hcec: CEC handleRxbuffer: Rx Buffer
Return values	<ul style="list-style-type: none">Frame: size
Notes	<ul style="list-style-type: none">This function can be called only inside the HAL_CEC_RxCpltCallback()

HAL_CEC_IRQHandler

Function name	<code>void HAL_CEC_IRQHandler (CEC_HandleTypeDef * hcec)</code>
Function description	This function handles CEC interrupt requests.
Parameters	<ul style="list-style-type: none">hcec: CEC handle
Return values	<ul style="list-style-type: none">None:

HAL_CEC_TxCpltCallback

Function name `void HAL_CEC_TxCpltCallback (CEC_HandleTypeDef * hcec)`

Function description Tx Transfer completed callback.

Parameters

- `hcec`: CEC handle

Return values

- `None`:

HAL_CEC_RxCpltCallback

Function name `void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec, uint32_t RxFrameSize)`

Function description Rx Transfer completed callback.

Parameters

- `hcec`: CEC handle
- `RxFrameSize`: Size of frame

Return values

- `None`:

HAL_CEC_ErrorCallback

Function name `void HAL_CEC_ErrorCallback (CEC_HandleTypeDef * hcec)`

Function description CEC error callbacks.

Parameters

- `hcec`: CEC handle

Return values

- `None`:

HAL_CEC_GetState

Function name `HAL_CEC_StateTypeDef HAL_CEC_GetState (CEC_HandleTypeDef * hcec)`

Function description return the CEC state

Parameters

- `hcec`: pointer to a `CEC_HandleTypeDef` structure that contains the configuration information for the specified CEC module.

Return values

- `HAL`: state

HAL_CEC_GetError

Function name `uint32_t HAL_CEC_GetError (CEC_HandleTypeDef * hcec)`

Function description Return the CEC error code.

Parameters

- `hcec`: pointer to a `CEC_HandleTypeDef` structure that contains the configuration information for the specified CEC.

Return values • CEC: Error Code

10.3 CEC Firmware driver defines

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

10.3.1 CEC

CEC

CEC all RX or TX errors flags

CEC_ISR_ALL_ERROR

CEC Error Bit Generation if Bit Rise Error reported

**CEC_BRE_ERRORBIT_NO
_GENERATION**

**CEC_BRE_ERRORBIT_GE
NERATION**

CEC Reception Stop on Error

**CEC_NO_RX_STOP_ON_B
RE**

CEC_RX_STOP_ON_BRE

CEC Error Bit Generation on Broadcast message

**CEC_BROADCASTERROR
_ERRORBIT_GENERATION**

**CEC_BROADCASTERROR
_NO_ERRORBIT_GENERA
TION**

CEC Error Code

HAL_CEC_ERROR_NONE no error

HAL_CEC_ERROR_RXOVR CEC Rx-Overrun

HAL_CEC_ERROR_BRE CEC Rx Bit Rising Error

HAL_CEC_ERROR_SBPE CEC Rx Short Bit period Error

HAL_CEC_ERROR_LBPE CEC Rx Long Bit period Error

HAL_CEC_ERROR_RXACK CEC Rx Missing Acknowledge
E

HAL_CEC_ERROR_ARBLS CEC Arbitration Lost
T

HAL_CEC_ERROR_TXUDR CEC Tx-Buffer Underrun

HAL_CEC_ERROR_TXERR CEC Tx-Error

HAL_CEC_ERROR_TXACK CEC Tx Missing Acknowledge

CEC Exported Macros

_HAL_CEC_RESET_HANDLE_STATE **Description:**

- Reset CEC handle gstate & RxState.

Parameters:

- **_HANDLE_**: CEC handle.

Return value:

- None

_HAL_CEC_GET_FLAG **Description:**

- Checks whether or not the specified CEC interrupt flag is set.

Parameters:

- **_HANDLE_**: specifies the CEC Handle.
- **_FLAG_**: specifies the flag to check.
 - **CEC_FLAG_TXACKE**: Tx Missing acknowledge Error
 - **CEC_FLAG_TXERR**: Tx Error.
 - **CEC_FLAG_RXUDR**: Tx-Buffer Underrun.
 - **CEC_FLAG_TXEND**: End of transmission (successful transmission of the last byte).
 - **CEC_FLAG_TXBR**: Tx-Byte Request.
 - **CEC_FLAG_ARBLST**: Arbitration Lost
 - **CEC_FLAG_RXACKE**: Rx-Missing Acknowledge
 - **CEC_FLAG_LBPE**: Rx Long period Error
 - **CEC_FLAG_SBPE**: Rx Short period Error
 - **CEC_FLAG_BRE**: Rx Bit Rising Error
 - **CEC_FLAG_RXOVR**: Rx Overrun.
 - **CEC_FLAG_RXEND**: End Of Reception.
 - **CEC_FLAG_RXBR**: Rx-Byte Received.

Return value:

- ITStatus

__HAL_CEC_CLEAR_FLAG Description:

- Clears the interrupt or status flag when raised (write at 1)

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
 - CEC_FLAG_TXACKE: Tx Missing acknowledge Error
 - CEC_FLAG_TXERR: Tx Error.
 - CEC_FLAG_RXUDR: Tx-Buffer Underrun.
 - CEC_FLAG_TXEND: End of transmission (successful transmission of the last byte).
 - CEC_FLAG_TXBR: Tx-Byte Request.
 - CEC_FLAG_ARBLST: Arbitration Lost
 - CEC_FLAG_RXACKE: Rx-Missing Acknowledge
 - CEC_FLAG_LBPE: Rx Long period Error
 - CEC_FLAG_SBPE: Rx Short period Error
 - CEC_FLAG_BRE: Rx Bit Rising Error
 - CEC_FLAG_RXOVR: Rx Overrun.
 - CEC_FLAG_RXEND: End Of Reception.
 - CEC_FLAG_RXBR: Rx-Byte Received.

Return value:

- none

__HAL_CEC_ENABLE_IT Description:

- Enables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to enable. This parameter can be one of the following values:
 - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
 - CEC_IT_TXERR: Tx Error IT Enable
 - CEC_IT_RXUDR: Tx-Buffer Underrun IT Enable
 - CEC_IT_TXEND: End of transmission IT Enable
 - CEC_IT_TXBR: Tx-Byte Request IT Enable
 - CEC_IT_ARBLST: Arbitration Lost IT Enable
 - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
 - CEC_IT_LBPE: Rx Long period Error IT Enable
 - CEC_IT_SBPE: Rx Short period Error IT Enable
 - CEC_IT_BRE: Rx Bit Rising Error IT Enable
 - CEC_IT_RXOVR: Rx Overrun IT Enable
 - CEC_IT_RXEND: End Of Reception IT Enable
 - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_DISABLE_IT

Description:

- Disables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to disable. This parameter can be one of the following values:
 - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
 - CEC_IT_RXERR: Tx Error IT Enable
 - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
 - CEC_IT_TXEND: End of transmission IT Enable
 - CEC_IT_TXBR: Tx-Byte Request IT Enable
 - CEC_IT_ARBLST: Arbitration Lost IT Enable
 - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
 - CEC_IT_LBPE: Rx Long period Error IT Enable
 - CEC_IT_SBPE: Rx Short period Error IT Enable
 - CEC_IT_BRE: Rx Bit Rising Error IT Enable
 - CEC_IT_RXOVR: Rx Overrun IT Enable
 - CEC_IT_RXEND: End Of Reception IT Enable
 - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_GET_IT_SOU

RCE

Description:

- Checks whether or not the specified CEC interrupt is enabled.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to check. This parameter can be one of the following values:
 - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
 - CEC_IT_RXERR: Tx Error IT Enable
 - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
 - CEC_IT_TXEND: End of transmission IT Enable
 - CEC_IT_TXBR: Tx-Byte Request IT Enable
 - CEC_IT_ARBLST: Arbitration Lost IT Enable
 - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
 - CEC_IT_LBPE: Rx Long period Error IT Enable
 - CEC_IT_SBPE: Rx Short period Error IT Enable
 - CEC_IT_BRE: Rx Bit Rising Error IT Enable
 - CEC_IT_RXOVR: Rx Overrun IT Enable
 - CEC_IT_RXEND: End Of Reception IT Enable
 - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- FlagStatus

<u>__HAL_CEC_ENABLE</u>	Description: <ul style="list-style-type: none">Enables the CEC device. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the CEC Handle. Return value: <ul style="list-style-type: none">none
<u>__HAL_CEC_DISABLE</u>	Description: <ul style="list-style-type: none">Disables the CEC device. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the CEC Handle. Return value: <ul style="list-style-type: none">none
<u>__HAL_CEC_FIRST_BYTE_TX_SET</u>	Description: <ul style="list-style-type: none">Set Transmission Start flag. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the CEC Handle. Return value: <ul style="list-style-type: none">none
<u>__HAL_CEC_LAST_BYTE_TX_SET</u>	Description: <ul style="list-style-type: none">Set Transmission End flag. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the CEC Handle. Return value: <ul style="list-style-type: none">none: If the CEC message consists of only one byte, TXEOM must be set before of TXSOM.
<u>__HAL_CEC_GET_TRANS_MISISON_START_FLAG</u>	Description: <ul style="list-style-type: none">Get Transmission Start flag. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the CEC Handle. Return value: <ul style="list-style-type: none">FlagStatus
<u>__HAL_CEC_GET_TRANS_MISISON_END_FLAG</u>	Description: <ul style="list-style-type: none">Get Transmission End flag. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the CEC Handle. Return value: <ul style="list-style-type: none">FlagStatus

__HAL_CEC_CLEAR_OAR **Description:**

- Clear OAR register.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_SET_OAR **Description:**

- Set OAR register (without resetting previously set address in case of multi-address mode)
To reset OAR,

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __ADDRESS__: Own Address value (CEC logical address is identified by bit position)

Return value:

- none

CEC Flags definition**CEC_FLAG_TXACKE****CEC_FLAG_TXERR****CEC_FLAG_RXUDR****CEC_FLAG_RXEND****CEC_FLAG_TXBR****CEC_FLAG_ARBLST****CEC_FLAG_RXACKE****CEC_FLAG_LBPE****CEC_FLAG_SBPE****CEC_FLAG_BRE****CEC_FLAG_RXOVR****CEC_FLAG_RXEND****CEC_FLAG_RXBR*****CEC all RX errors interrupts enabling flag*****CEC_IER_RX_ALL_ERR*****CEC all TX errors interrupts enabling flag*****CEC_IER_TX_ALL_ERR**

CEC Initiator logical address position in message header

CEC_INITIATOR_LSB_POS

CEC Interrupts definition

CEC_IT_TXACKE

CEC_IT_TXERR

CEC_IT_TXUDR

CEC_IT_TXEND

CEC_IT_TXBR

CEC_IT_ARBLST

CEC_IT_RXACKE

CEC_IT_LBPE

CEC_IT_SBPE

CEC_IT_BRE

CEC_IT_RXOVR

CEC_IT_RXEND

CEC_IT_RXBR

CEC Error Bit Generation if Long Bit Period Error reported

**CEC_LBPE_ERRORBIT_N
O_GENERATION**

**CEC_LBPE_ERRORBIT_GE
NERATION**

CEC Listening mode option

**CEC_REDUCED_LISTENIN
G_MODE**

**CEC_FULL_LISTENING_M
ODE**

CEC Device Own Address position in CEC CFGR register

**CEC_CFGR_OAR_LSB_PO
S**

CEC Own Address

CEC_OWN_ADDRESS_NO
NE

CEC_OWN_ADDRESS_0

CEC_OWN_ADDRESS_1

CEC_OWN_ADDRESS_2

CEC_OWN_ADDRESS_3

CEC_OWN_ADDRESS_4

CEC_OWN_ADDRESS_5

CEC_OWN_ADDRESS_6

CEC_OWN_ADDRESS_7

CEC_OWN_ADDRESS_8

CEC_OWN_ADDRESS_9

CEC_OWN_ADDRESS_10

CEC_OWN_ADDRESS_11

CEC_OWN_ADDRESS_12

CEC_OWN_ADDRESS_13

CEC_OWN_ADDRESS_14

CEC Signal Free Time start option

CEC_SFT_START_ON_TXS
OM

CEC_SFT_START_ON_TX_
RX_END

CEC Signal Free Time setting parameter

CEC_DEFAULT_SFT

CEC_0_5_BITPERIOD_SFT

CEC_1_5_BITPERIOD_SFT

CEC_2_5_BITPERIOD_SFT

CEC_3_5_BITPERIOD_SFT

CEC_4_5_BITPERIOD_SFT

CEC_5_5_BITPERIOD_SFT

CEC_6_5_BITPERIOD_SFT

CEC Receiver Tolerance

CEC_STANDARD_TOLERA
NCE

CEC_EXTENDED_TOLERA
NCE

11 HAL COMP Generic Driver

11.1 COMP Firmware driver registers structures

11.1.1 COMP_InitTypeDef

`COMP_InitTypeDef` is defined in the `stm32f0xx_hal_comp.h`

Data Fields

- `uint32_t InvertingInput`
- `uint32_t NonInvertingInput`
- `uint32_t Output`
- `uint32_t OutputPol`
- `uint32_t Hysteresis`
- `uint32_t Mode`
- `uint32_t WindowMode`
- `uint32_t TriggerMode`

Field Documentation

- `uint32_t COMP_InitTypeDef::InvertingInput`
Selects the inverting input of the comparator. This parameter can be a value of `COMP_InvertingInput`
- `uint32_t COMP_InitTypeDef::NonInvertingInput`
Selects the non inverting input of the comparator. This parameter can be a value of `COMP_NonInvertingInput`
- `uint32_t COMP_InitTypeDef::Output`
Selects the output redirection of the comparator. This parameter can be a value of `COMP_Output`
- `uint32_t COMP_InitTypeDef::OutputPol`
Selects the output polarity of the comparator. This parameter can be a value of `COMP_OutputPolarity`
- `uint32_t COMP_InitTypeDef::Hysteresis`
Selects the hysteresis voltage of the comparator. This parameter can be a value of `COMP_Hysteresis`
- `uint32_t COMP_InitTypeDef::Mode`
Selects the operating consumption mode of the comparator to adjust the speed/consumption. This parameter can be a value of `COMP_Mode`
- `uint32_t COMP_InitTypeDef::WindowMode`
Selects the window mode of the comparator 1 & 2. This parameter can be a value of `COMP_WindowMode`
- `uint32_t COMP_InitTypeDef::TriggerMode`
Selects the trigger mode of the comparator (interrupt mode). This parameter can be a value of `COMP_TriggerMode`

11.1.2 COMP_HandleTypeDef

`COMP_HandleTypeDef` is defined in the `stm32f0xx_hal_comp.h`

Data Fields

- `COMP_TypeDef * Instance`
- `COMP_InitTypeDef Init`
- `HAL_LockTypeDef Lock`
- `_IO uint32_t State`

Field Documentation

- `COMP_TypeDef* COMP_HandleTypeDef::Instance`
Register base address

- **`COMP_InitTypeDef COMP_HandleTypeDef::Init`**
COMP required parameters
- **`HAL_LockTypeDef COMP_HandleTypeDef::Lock`**
Locking object
- **`_IO uint32_t COMP_HandleTypeDef::State`**
COMP communication state This parameter can be a value of `COMP_State`

11.2 COMP Firmware driver API description

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

11.2.1 COMP Peripheral features

The STM32F0xx device family integrates up to 2 analog comparators COMP1 and COMP2:

- The non inverting input and inverting input can be set to GPIO pins.
- The COMP output is available using `HAL_COMP_GetOutputLevel()` and can be set on GPIO pins.
- The COMP output can be redirected to embedded timers (TIM1, TIM2 and TIM3).
- The comparators COMP1 and COMP2 can be combined in window mode.
- The comparators have interrupt capability with wake-up from Sleep and Stop modes (through the EXTI controller):
 - COMP1 is internally connected to EXTI Line 21
 - COMP2 is internally connected to EXTI Line 22
- From the corresponding IRQ handler, the right interrupt source can be retrieved with the macros `__HAL_COMP_COMP1_EXTI_GET_FLAG()` and `__HAL_COMP_COMP2_EXTI_GET_FLAG()`.

11.2.2 How to use this driver

This driver provides functions to configure and program the Comparators of STM32F05x, STM32F07x and STM32F09x devices. To use the comparator, perform the following steps:

1. Fill in the `HAL_COMP_MspInit()` to
 - Configure the comparator input in analog mode using `HAL_GPIO_Init()`
 - Configure the comparator output in alternate function mode using `HAL_GPIO_Init()` to map the comparator output to the GPIO pin
 - If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using `HAL_GPIO_Init()` function. After that enable the comparator interrupt vector using `HAL_NVIC_EnableIRQ()` function.
2. Configure the comparator using `HAL_COMP_Init()` function:
 - Select the inverting input (input minus)
 - Select the non inverting input (input plus)
 - Select the output polarity
 - Select the output redirection
 - Select the hysteresis level
 - Select the power mode
 - Select the event/interrupt mode
 - Select the window mode

Note: `HAL_COMP_Init()` calls internally `__HAL_RCC_SYSCFG_CLK_ENABLE()` in order to access the comparator(s) registers.

3. Enable the comparator using `HAL_COMP_Start()` function or `HAL_COMP_Start_IT()` function for interrupt mode.
4. Use `HAL_COMP_TriggerCallback()` and/or `HAL_COMP_GetOutputLevel()` functions to manage comparator outputs (event/interrupt triggered and output level).
5. Disable the comparator using `HAL_COMP_Stop()` or `HAL_COMP_Stop_IT()` function.
6. De-initialize the comparator using `HAL_COMP_DeInit()` function.

7. For safety purposes comparator(s) can be locked using HAL_COMP_Lock() function. Only a MCU reset can reset that protection.

11.2.3 Initialization and Configuration functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- [*HAL_COMP_Init*](#)
- [*HAL_COMP_DelInit*](#)
- [*HAL_COMP_MspInit*](#)
- [*HAL_COMP_MspDelInit*](#)

11.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the COMP data transfers.

This section contains the following APIs:

- [*HAL_COMP_Start*](#)
- [*HAL_COMP_Stop*](#)
- [*HAL_COMP_Start_IT*](#)
- [*HAL_COMP_Stop_IT*](#)
- [*HAL_COMP_IRQHandler*](#)

11.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the COMP data transfers.

This section contains the following APIs:

- [*HAL_COMP_Lock*](#)
- [*HAL_COMP_GetOutputLevel*](#)
- [*HAL_COMP_TriggerCallback*](#)

11.2.6 Peripheral State 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_COMP_GetState*](#)

11.2.7 Detailed description of functions

[*HAL_COMP_Init*](#)

Function name [*HAL_StatusTypeDef HAL_COMP_Init \(COMP_HandleTypeDef * hcomp\)*](#)

Function description Initializes the COMP according to the specified parameters in the COMP_InitTypeDef and create the associated handle.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

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

[*HAL_COMP_DelInit*](#)

Function name [*HAL_StatusTypeDef HAL_COMP_DelInit \(COMP_HandleTypeDef * hcomp\)*](#)

Function description	Deinitializes the COMP peripheral.
Parameters	<ul style="list-style-type: none">• hcomp: COMP handle
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• Deinitialization can't be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_COMP_MspInit

Function name	void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)
----------------------	---

Function description	Initializes the COMP MSP.
-----------------------------	---------------------------

Parameters	<ul style="list-style-type: none">• hcomp: COMP handle
-------------------	---

Return values	<ul style="list-style-type: none">• None:
----------------------	--

HAL_COMP_MspDeInit

Function name	void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)
----------------------	---

Function description	Deinitializes COMP MSP.
-----------------------------	-------------------------

Parameters	<ul style="list-style-type: none">• hcomp: COMP handle
-------------------	---

Return values	<ul style="list-style-type: none">• None:
----------------------	--

HAL_COMP_Start

Function name	HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)
----------------------	--

Function description	Start the comparator.
-----------------------------	-----------------------

Parameters	<ul style="list-style-type: none">• hcomp: COMP handle
-------------------	---

Return values	<ul style="list-style-type: none">• HAL: status
----------------------	--

HAL_COMP_Stop

Function name	HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)
----------------------	---

Function description	Stop the comparator.
-----------------------------	----------------------

Parameters	<ul style="list-style-type: none">• hcomp: COMP handle
-------------------	---

Return values	<ul style="list-style-type: none">• HAL: status
----------------------	--

HAL_COMP_Start_IT

Function name	HAL_StatusTypeDef HAL_COMP_Start_IT (COMP_HandleTypeDef * hcomp)
----------------------	---

Function description Enables the interrupt and starts the comparator.

Parameters • **hcomp:** COMP handle

Return values • **HAL:** status.

HAL_COMP_Stop_IT

Function name **HAL_StatusTypeDef HAL_COMP_Stop_IT (COMP_HandleTypeDef * hcomp)**

Function description Disable the interrupt and Stop the comparator.

Parameters • **hcomp:** COMP handle

Return values • **HAL:** status

HAL_COMP_IRQHandler

Function name **void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)**

Function description Comparator IRQ Handler.

Parameters • **hcomp:** COMP handle

Return values • **HAL:** status

HAL_COMP_Lock

Function name **HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)**

Function description Lock the selected comparator configuration.

Parameters • **hcomp:** COMP handle

Return values • **HAL:** status

HAL_COMP_GetOutputLevel

Function name **uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)**

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

HAL_COMP_TriggerCallback

Function name **void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)**

Function description Comparator callback.

Parameters • **hcomp:** COMP handle

Return values • **None:**

HAL_COMP_GetState

Function name	<code>uint32_t HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)</code>
Function description	Return the COMP state.
Parameters	<ul style="list-style-type: none">• <code>hcomp</code>: COMP handle
Return values	<ul style="list-style-type: none">• <code>HAL</code>: state

11.3 COMP Firmware driver defines

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

11.3.1 COMP

COMP

COMP Exported Macros

<code>_HAL_COMP_RESET_HANDLE_STATE</code>	Description: <ul style="list-style-type: none">• Reset COMP handle state. Parameters: <ul style="list-style-type: none">• <code>_HANDLE_</code>: COMP handle. Return value: <ul style="list-style-type: none">• None
<code>_HAL_COMP_ENABLE</code>	Description: <ul style="list-style-type: none">• Enable the specified comparator. Parameters: <ul style="list-style-type: none">• <code>_HANDLE_</code>: COMP handle. Return value: <ul style="list-style-type: none">• None
<code>_HAL_COMP_DISABLE</code>	Description: <ul style="list-style-type: none">• Disable the specified comparator. Parameters: <ul style="list-style-type: none">• <code>_HANDLE_</code>: COMP handle. Return value: <ul style="list-style-type: none">• None
<code>_HAL_COMP_LOCK</code>	Description: <ul style="list-style-type: none">• Lock the specified comparator configuration. Parameters: <ul style="list-style-type: none">• <code>_HANDLE_</code>: COMP handle. Return value: <ul style="list-style-type: none">• None

_HAL_COMP_COMP1_EX Description:

- TI_ENABLE_RISING_EDGE** • Enable the COMP1 EXTI line rising edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EX Description:

- TI_DISABLE_RISING_EDGE** • Disable the COMP1 EXTI line rising edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EX Description:

- TI_ENABLE_FALLING_EDGE** • Enable the COMP1 EXTI line falling edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EX Description:

- TI_DISABLE_FALLING_EDGE** • Disable the COMP1 EXTI line falling edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EX Description:

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

Return value:

- None

_HAL_COMP_COMP1_EX Description:

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

Return value:

- None

_HAL_COMP_COMP1_EX Description:

- TI_ENABLE_IT** • Enable the COMP1 EXTI line in interrupt mode.

Return value:

- None

_HAL_COMP_COMP1_EX Description:

- TI_DISABLE_IT** • Disable the COMP1 EXTI line in interrupt mode.

Return value:

- None

_HAL_COMP_COMP1_EX Description:

- TI_GENERATE_SWIT** • Generate a software interrupt on the COMP1 EXTI line.

Return value:

- None

- __HAL_COMP_COMP1_EX** **Description:**
TI_ENABLE_EVENT • Enable the COMP1 EXTI Line in event mode.
- Return value:**
• None
- __HAL_COMP_COMP1_EX** **Description:**
TI_DISABLE_EVENT • Disable the COMP1 EXTI Line in event mode.
- Return value:**
• None
- __HAL_COMP_COMP1_EX** **Description:**
TI_GET_FLAG • Check whether the COMP1 EXTI line flag is set or not.
- Return value:**
• RESET: or SET
- __HAL_COMP_COMP1_EX** **Description:**
TI_CLEAR_FLAG • Clear the COMP1 EXTI flag.
- Return value:**
• None
- __HAL_COMP_COMP2_EX** **Description:**
TI_ENABLE_RISING_EDGE • Enable the COMP2 EXTI line rising edge trigger.
- Return value:**
• None
- __HAL_COMP_COMP2_EX** **Description:**
TI_DISABLE_RISING_EDGE • Disable the COMP2 EXTI line rising edge trigger.
- Return value:**
• None
- __HAL_COMP_COMP2_EX** **Description:**
TI_ENABLE_FALLING_EDGE • Enable the COMP2 EXTI line falling edge trigger.
- Return value:**
• None
- __HAL_COMP_COMP2_EX** **Description:**
TI_DISABLE_FALLING_EDGE • Disable the COMP2 EXTI line falling edge trigger.
- Return value:**
• None
- __HAL_COMP_COMP2_EX** **Description:**
TI_ENABLE_RISING_FALLING_EDGE • Enable the COMP2 EXTI line rising & falling edge trigger.
- Return value:**
• None

__HAL_COMP_COMP2_EX **Description:**

- TI_DISABLE_RISING_FALL** • Disable the COMP2 EXTI line rising & falling edge trigger.
ING_EDGE

Return value:

- None

__HAL_COMP_COMP2_EX **Description:**

- TI_ENABLE_IT** • Enable the COMP2 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP2_EX **Description:**

- TI_DISABLE_IT** • Disable the COMP2 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP2_EX **Description:**

- TI_GENERATE_SWIT** • Generate a software interrupt on the COMP2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EX **Description:**

- TI_ENABLE_EVENT** • Enable the COMP2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EX **Description:**

- TI_DISABLE_EVENT** • Disable the COMP2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EX **Description:**

- TI_GET_FLAG** • Check whether the COMP2 EXTI line flag is set or not.

Return value:

- RESET: or SET

__HAL_COMP_COMP2_EX **Description:**

- TI_CLEAR_FLAG** • Clear the COMP2 EXTI flag.

Return value:

- None

__HAL_COMP_GET_FLAG **Description:**

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

Parameters:

- __HANDLE__: specifies the COMP Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - COMP_FLAG_LOCK: lock flag

Return value:

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

COMP EXTI Lines

COMP_EXTI_LINE_COMP1 EXTI line 21 connected to COMP1 output

COMP_EXTI_LINE_COMP2 EXTI line 22 connected to COMP2 output

COMP Flag

COMP_FLAG_LOCK Lock flag

COMP Private macros to get EXTI line associated with Comparators**COMP_GET_EXTI_LINE** **Description:**

- Get the specified EXTI line for a comparator instance.

Parameters:

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

COMP Hysteresis

COMP_HYSTERESIS_NON No hysteresis
E

COMP_HYSTERESIS_LOW Hysteresis level low

COMP_HYSTERESIS_MEDI Hysteresis level medium
UM

COMP_HYSTERESIS_HIGH Hysteresis level high

COMP InvertingInput

COMP_INVERTINGINPUT_1 1/4 VREFINT connected to comparator inverting input
_4VREFINT

COMP_INVERTINGINPUT_1 1/2 VREFINT connected to comparator inverting input
_2VREFINT

COMP_INVERTINGINPUT_3 3/4 VREFINT connected to comparator inverting input
_4VREFINT

COMP_INVERTINGINPUT_VREFINT VREFINT connected to comparator inverting input
VREFINT

COMP_INVERTINGINPUT_DAC1 DAC_OUT1 (PA4) connected to comparator inverting input
DAC1

COMP_INVERTINGINPUT_DAC1SWITCHCLOSED DAC_OUT1 (PA4) connected to comparator inverting input and close switch (PA0 for COMP1 only)

COMP_INVERTINGINPUT_DAC2 DAC_OUT2 (PA5) connected to comparator inverting input
DAC2

COMP_INVERTINGINPUT_I01 IO (PA0 for COMP1 and PA2 for COMP2) connected to comparator inverting input
I01

COMP Private macros to check input parameters

IS_COMP_OUTPUTPOL

IS_COMP_HYSTERESIS

IS_COMP_MODE

IS_COMP_INVERTINGINPUT_T

IS_COMP_NONINVERTINGINPUT

IS_COMP_OUTPUT

IS_COMP_WINDOWMODE

IS_COMP_TRIGGERMODE

COMP Lock

COMP_LOCK_DISABLE

COMP_LOCK_ENABLE

COMP_STATE_BIT_LOCK

COMP Mode

COMP_MODE_HIGHSPEED High Speed

COMP_MODE_MEDIUMSP_EED Medium Speed
EED

COMP_MODE_LOWPOWER_R Low power mode
R

COMP_MODE_ULTRALOW_POWER Ultra-low power mode

COMP NonInvertingInput

COMP_NONINVERTINGINP_UT_IO1 I/O1 (PA1 for COMP1, PA3 for COMP2) connected to comparator non inverting input

COMP_NONINVERTINGINP_UT_DAC1SWITCHCLOSED DAC ouput connected to comparator COMP1 non inverting input

COMP Output

COMP_OUTPUT_NONE COMP output isn't connected to other peripherals

COMP_OUTPUT_TIM1BKIN COMP output connected to TIM1 Break Input (BKIN)

COMP_OUTPUT_TIM1IC1 COMP output connected to TIM1 Input Capture 1

COMP_OUTPUT_TIM1OCR_EFCLR COMP output connected to TIM1 OCREF Clear

COMP_OUTPUT_TIM2IC4 COMP output connected to TIM2 Input Capture 4

COMP_OUTPUT_TIM2OCR_EFCLR COMP output connected to TIM2 OCREF Clear

COMP_OUTPUT_TIM3IC1 COMP output connected to TIM3 Input Capture 1

COMP_OUTPUT_TIM3OCR_EFCLR COMP output connected to TIM3 OCREF Clear

COMP OutputLevel

COMP_OUTPUTLEVEL_LO_W

COMP_OUTPUTLEVEL_HI_GH

COMP OutputPolarity

COMP_OUTPUTPOL_NONI_NVERTED COMP output on GPIO isn't inverted

COMP_OUTPUTPOL_INVE_RTED COMP output on GPIO is inverted

COMP State

HAL_COMP_STATE_RESE_T COMP not yet initialized or disabled

HAL_COMP_STATE_READ_Y COMP initialized and ready for use

HAL_COMP_STATE_READ COMP initialized but the configuration is locked
Y_LOCKED

HAL_COMP_STATE_BUSY COMP is running

HAL_COMP_STATE_BUSY_LOCKED COMP is running and the configuration is locked

COMP TriggerMode

COMP_TRIGGERMODE_N No External Interrupt trigger detection
ONE

COMP_TRIGGERMODE_IT_RISING External Interrupt Mode with Rising edge trigger detection

COMP_TRIGGERMODE_IT_FALLING External Interrupt Mode with Falling edge trigger detection

COMP_TRIGGERMODE_IT_RISING_FALLING External Interrupt Mode with Rising/Falling edge trigger detection

COMP_TRIGGERMODE_EV_ENT_RISING Event Mode with Rising edge trigger detection

COMP_TRIGGERMODE_EV_ENT_FALLING Event Mode with Falling edge trigger detection

COMP_TRIGGERMODE_EV_ENT_RISING_FALLING Event Mode with Rising/Falling edge trigger detection

COMP WindowMode

COMP_WINDOWMODE_DI_SABLE Window mode disabled

COMP_WINDOWMODE_EN Window mode enabled: non inverting input of comparator 2 is connected to the non inverting input of comparator 1 (PA1)

12 HAL CORTEX Generic Driver

12.1 CORTEX Firmware driver API description

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

12.1.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M0 exceptions are managed by CMSIS functions.

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

Note: Lower priority values gives higher priority.

Note: Priority Order:

1. Lowest priority.
2. Lowest hardware priority (IRQn position).
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority()
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ()

Note: Negative value of IRQn_Type are not allowed.

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The HAL_SYSTICK_Config() function calls the SysTick_Config() function which is a CMSIS function that:
 - Configures the SysTick Reload register with value passed as function parameter.
 - Configures the SysTick IRQ priority to the lowest value (0x03).
 - Resets the SysTick Counter register.
 - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
 - Enables the SysTick Interrupt.
 - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The HAL_SYSTICK_CLKSourceConfig() macro is defined inside the stm32f0xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
 - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
 - Reload Value should not exceed 0xFFFFFFF

12.1.2 Initialization and de-initialization 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*](#)

12.1.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK) functionalities.

This section contains the following APIs:

- [*HAL_NVIC_GetPriority*](#)
- [*HAL_NVIC_SetPendingIRQ*](#)
- [*HAL_NVIC_GetPendingIRQ*](#)
- [*HAL_NVIC_ClearPendingIRQ*](#)
- [*HAL_SYSTICK_CLKSourceConfig*](#)
- [*HAL_SYSTICK_IRQHandler*](#)
- [*HAL_SYSTICK_Callback*](#)

12.1.4 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 `stm32f0xx.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 `stm32f0xx` 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 Enables 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 (`stm32f0xxxx.h`))

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the `NVIC_PriorityGroupConfig()` function should be called before.

[**HAL_NVIC_DisableIRQ**](#)

Function name **void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)**

Function description Disables 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 (stm32f0xxxx.h))

- Return values**
- **None:**

HAL_NVIC_SystemReset

Function name **void HAL_NVIC_SystemReset (void)**

Function description Initiates 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 Initializes the System Timer and its interrupt, and starts the System Tick Timer.

- 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 Gets 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 (stm32f0xxxx.h))

- Return values**
- **None:**

HAL_NVIC_GetPendingIRQ

Function name **uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)**

Function description Gets Pending Interrupt (reads the pending register in the NVIC and returns 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 (stm32f0xxxx.h))

- Return values**
- **status:** - 0 Interrupt status is not pending.
– 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name	void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)
Function description	Sets Pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none">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 (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none">None:

HAL_NVIC_ClearPendingIRQ

Function name	void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)
Function description	Clears the pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none">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 (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none">None:

HAL_SYSTICK_CLKSourceConfig

Function name	void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)
Function description	Configures the SysTick clock source.
Parameters	<ul style="list-style-type: none">CLKSource: specifies the SysTick clock source. This parameter can be one of the following values:<ul style="list-style-type: none">SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
Return values	<ul style="list-style-type: none">None:

HAL_SYSTICK_IRQHandler

Function name	void HAL_SYSTICK_IRQHandler (void)
Function description	This function handles SYSTICK interrupt request.
Return values	<ul style="list-style-type: none">None:

HAL_SYSTICK_Callback

Function name	void HAL_SYSTICK_Callback (void)
Function description	SYSTICK callback.

Return values

- **None:**

12.2 CORTEX Firmware driver defines

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

12.2.1 CORTEX

CORTEX

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_H

CLK_DIV8

SYSTICK_CLKSOURCE_H

CLK

13 HAL CRC Generic Driver

13.1 CRC Firmware driver registers structures

13.1.1 CRC_InitTypeDef

`CRC_InitTypeDef` is defined in the `stm32f0xx_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. 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 `CRCEx_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, or `CRC_OUTPUTDATA_INVERSION_ENABLE` CRC `0x11223344` is converted into `0x22CC4488`

13.1.2 CRC_HandleTypeDef

`CRC_HandleTypeDef` is defined in the `stm32f0xx_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-bits 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

13.2

CRC Firmware driver API description

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

13.2.1

How to use this driver

- Enable CRC AHB clock using `__HAL_RCC_CRC_CLK_ENABLE()`;
- Initialize CRC calculator
 - specify generating polynomial (IP default or non-default one)
 - specify initialization value (IP 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

13.2.2

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_Delinit`](#)
- [`HAL_CRC_MspInit`](#)
- [`HAL_CRC_MspDelinit`](#)

13.2.3 Peripheral Control functions

This section provides functions allowing to:

- compute the 7U, 8U, 16 or 32-bit CRC value of an 8U, 16 or 32-bit data buffer using the combination of the previous CRC value and the new one
- or
- compute the 7U, 8U, 16 or 32-bit CRC value of an 8U, 16 or 32-bit data buffer independently of the previous CRC value.

This section contains the following APIs:

- [**HAL_CRC_Accumulate**](#)
- [**HAL_CRC_Calculate**](#)

13.2.4 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**](#)

13.2.5 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 initialize 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 (CRC_HandleTypeDef * hcrc)`

Function description Return the CRC handle state.

Parameters	<ul style="list-style-type: none">• <code>hcrc</code>: CRC handle
Return values	<ul style="list-style-type: none">• <code>HAL</code>: state

13.3 CRC Firmware driver defines

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

13.3.1 CRC

CRC

Aliases for inter STM32 series compatibility

`HAL_CRC_Input_Data_Reverse`

`HAL_CRC_Output_Data_Reversed`

Default CRC computation initialization value

`DEFAULT_CRC_INITVALUE`

Indicates whether or not default init value is used

`DEFAULT_INIT_VALUE_ENABLE`

`DEFAULT_INIT_VALUE_DISABLE`

`IS_DEFAULT_INIT_VALUE`

Indicates whether or not default polynomial is used

`DEFAULT_POLYNOMIAL_ENABLE`

`DEFAULT_POLYNOMIAL_DISABLE`

`IS_DEFAULT_POLYNOMIAL`

Default CRC generating polynomial

`DEFAULT_CRC32_POLY`

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:**

- Stores a 8-bit data in the Independent Data(ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: 8-bit value to be stored in the ID register

Return value:

- None

__HAL_CRC_GET_IDR **Description:**

- Returns the 8-bit data stored in the Independent Data(ID) register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- 8-bit: value of the ID register

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED

CRC_INPUTDATA_FORMAT_BYTES

CRC_INPUTDATA_FORMAT_HALFWORDS

CRC_INPUTDATA_FORMAT
_WORDS

IS_CRC_INPUTDATA_FOR
MAT

14 HAL CRC Extension Driver

14.1 CRCEEx Firmware driver API description

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

14.1.1 How to use this driver

- Extended initialization
- Set or not user-defined generating polynomial other than default one

14.1.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize the CRC generating polynomial: if programmable polynomial feature is applicable to device, set default or non-default generating polynomial according to hcrc->Init.DefaultPolynomialUse parameter. If feature is non-applicable to device in use, HAL_CRCEEx_Init straight away reports HAL_OK.
- Set the generating polynomial

This section contains the following APIs:

- [**HAL_CRCEEx_Init**](#)
- [**HAL_CRCEEx_Input_Data_Reverse**](#)
- [**HAL_CRCEEx_Output_Data_Reverse**](#)
- [**HAL_CRCEEx_Polynomial_Set**](#)

14.1.3 Detailed description of functions

[**HAL_CRCEEx_Init**](#)

Function name [**HAL_StatusTypeDef HAL_CRCEEx_Init \(CRC_HandleTypeDef * hcrc\)**](#)

Function description Extended initialization to set generating polynomial.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

[**HAL_CRCEEx_Input_Data_Reverse**](#)

Function name [**HAL_StatusTypeDef HAL_CRCEEx_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_NOINVERSION:** 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_CRCEEx_Output_Data_Reverse

Function name	HAL_StatusTypeDef HAL_CRCEEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)
Function description	Set the Reverse Output data mode.
Parameters	<ul style="list-style-type: none">• hcrc: CRC handle• OutputReverseMode: Output Data inversion mode This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status

HAL_CRCEEx_Polynomial_Set

Function name	HAL_StatusTypeDef HAL_CRCEEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)
Function description	Initializes the CRC polynomial if different from default one.
Parameters	<ul style="list-style-type: none">• 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:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status

14.2 CRCEEx Firmware driver defines

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

14.2.1 CRCEEx

CRCEEx

CRCEEx Exported Macros

<u>__HAL_CRC_OUTPUTREV</u>	Description:
<u>ERSAL_ENABLE</u>	<ul style="list-style-type: none">• Set CRC output reversal.
	Parameters:
	<ul style="list-style-type: none">• <u>__HANDLE__</u>: CRC handle
	Return value:
	<ul style="list-style-type: none">• None.

`__HAL_CRC_OUTPUTREV` **Description:**
`ERSAL_DISABLE`

- Unset CRC output reversal.

Parameters:

- `__HANDLE__`: CRC handle

Return value:

- None.

`__HAL_CRC_POLYNOMIAL` **Description:**
`_CONFIG`

- 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_INVERS`
`ON_NONE`

`CRC_INPUTDATA_INVERS`
`ON_BYTE`

`CRC_INPUTDATA_INVERS`
`ON_HALFWORD`

`CRC_INPUTDATA_INVERS`
`ON_WORD`

`IS_CRC_INPUTDATA_INVE`
`RSION_MODE`

Output Data Inversion Modes

`CRC_OUTPUTDATA_INVER`
`SION_DISABLE`

`CRC_OUTPUTDATA_INVER`
`SION_ENABLE`

`IS_CRC_OUTPUTDATA_IN`
`VERSION_MODE`

Polynomial sizes to configure the IP

`CRC_POLYLENGTH_32B`

`CRC_POLYLENGTH_16B`

`CRC_POLYLENGTH_8B`

`CRC_POLYLENGTH_7B`

IS_CRC_POL_LENGTH

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

HAL_CRC_LENGTH_16B

HAL_CRC_LENGTH_8B

HAL_CRC_LENGTH_7B

15 HAL DAC Generic Driver

15.1 DAC Firmware driver registers structures

15.1.1 DAC_HandleTypeDef

DAC_HandleTypeDef is defined in the `stm32f0xx_hal_dac.h`

Data Fields

- *DAC_TypeDef * Instance*
- *__IO HAL_DAC_StateTypeDef State*
- *HAL_LockTypeDef Lock*
- *DMA_HandleTypeDef * DMA_Handle1*
- *DMA_HandleTypeDef * DMA_Handle2*
- *__IO uint32_t ErrorCode*

Field Documentation

- ***DAC_TypeDef* DAC_HandleTypeDef::Instance***
Register base address
- ***__IO HAL_DAC_StateTypeDef DAC_HandleTypeDef::State***
DAC communication state
- ***HAL_LockTypeDef DAC_HandleTypeDef::Lock***
DAC locking object
- ***DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle1***
Pointer DMA handler for channel 1
- ***DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle2***
Pointer DMA handler for channel 2
- ***__IO uint32_t DAC_HandleTypeDef::ErrorCode***
DAC Error code

15.1.2 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the `stm32f0xx_hal_dac.h`

Data Fields

- *uint32_t DAC_Trigger*
- *uint32_t DAC_OutputBuffer*

Field Documentation

- ***uint32_t DAC_ChannelConfTypeDef::DAC_Trigger***
Specifies the external trigger for the selected DAC channel. This parameter can be a value of [*DAC_trigger_selection*](#)
- ***uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer***
Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of [*DAC_output_buffer*](#)

15.2 DAC Firmware driver API description

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

15.2.1 DAC Peripheral features

DAC Channels

STM32F0 devices integrates no, one or two 12-bit Digital Analog Converters. STM32F05x devices have one converter (channel1) STM32F07x & STM32F09x devices have two converters (i.e. channel1 & channel2) When 2 converters are present (i.e. channel1 & channel2) they can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output
2. DAC channel2 with DAC_OUT2 (PA5) as output

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM3, TIM6, and TIM15 (DAC_TRIGGER_T2_TRGO, DAC_TRIGGER_T3_TRGO...)
3. Software using DAC_TRIGGER_SOFTWARE

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Note: Refer to the device datasheet for more details about output impedance value with and without output buffer.

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel1 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle wave

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondance

The analog output voltage on each DAC channel pin is determined by the following equation:

$$\text{DAC_OUTx} = \text{VREF+} * \text{DOR} / 4095$$

- with DOR is the Data Output Register

VREF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using HAL_DAC_Start_DMA()

Note: For Dual mode and specific signal (Triangle and noise) generation please refer to Extended Features Driver description STM32F0 devices with one channel (one converting capability) does not support Dual mode and specific signal (Triangle and noise) generation.

15.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA() functions.

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEEx_ConvHalfCpltCallbackCh2()
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEEx_ConvHalfCpltCallbackCh2()
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEEx_DMAUnderrunCallbackCh2() and add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1()
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

DAC HAL driver macros list

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

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

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

15.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.

This section contains the following APIs:

- [**HAL_DAC_Init**](#)
- [**HAL_DAC_DelInit**](#)
- [**HAL_DAC_MspInit**](#)
- [**HAL_DAC_MspDelInit**](#)

15.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- [*HAL_DAC_Start*](#)
- [*HAL_DAC_Stop*](#)
- [*HAL_DAC_Start_DMA*](#)
- [*HAL_DAC_Stop_DMA*](#)
- [*HAL_DAC_IRQHandler*](#)
- [*HAL_DAC_SetValue*](#)
- [*HAL_DAC_ConvCpltCallbackCh1*](#)
- [*HAL_DAC_ConvHalfCpltCallbackCh1*](#)
- [*HAL_DAC_ErrorCallbackCh1*](#)
- [*HAL_DAC_DMADebugCallbackCh1*](#)

15.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Get result of conversion.

This section contains the following APIs:

- [*HAL_DAC_GetValue*](#)
- [*HAL_DAC_ConfigChannel*](#)

15.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- [*HAL_DAC_GetState*](#)
- [*HAL_DAC_GetError*](#)

15.2.7 Detailed description of functions

[*HAL_DAC_Init*](#)

Function name [*HAL_StatusTypeDef HAL_DAC_Init \(DAC_HandleTypeDef * hdac\)*](#)

Function description Initialize the DAC peripheral according to the specified parameters in the *DAC_InitStruct* and initialize the associated handle.

Parameters

- **hdac:** pointer to a *DAC_HandleTypeDef* structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_DelInit

Function name `HAL_StatusTypeDef HAL_DAC_DelInit (DAC_HandleTypeDef * hdac)`

Function description Deinitialize the DAC peripheral registers to their default reset values.

Parameters

- **hdac:** pointer to a `DAC_HandleTypeDef` structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_MspInit

Function name `void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)`

Function description Initialize the DAC MSP.

Parameters

- **hdac:** pointer to a `DAC_HandleTypeDef` structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_MspDeInit

Function name `void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)`

Function description Deinitialize the DAC MSP.

Parameters

- **hdac:** pointer to a `DAC_HandleTypeDef` structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_Start

Function name `HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)`

Function description Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a `DAC_HandleTypeDef` structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - `DAC_CHANNEL_1`: DAC Channel1 selected
 - `DAC_CHANNEL_2`: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Stop

Function name `HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)`

Function description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.• Channel: The selected DAC channel. This parameter can be one of the following values:<ul style="list-style-type: none">– DAC_CHANNEL_1: DAC Channel1 selected– DAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none">• HAL: status
HAL_DAC_Start_DMA	
Function name	HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)
Function description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.• Channel: The selected DAC channel. This parameter can be one of the following values:<ul style="list-style-type: none">– DAC_CHANNEL_1: DAC Channel1 selected– DAC_CHANNEL_2: DAC Channel2 selected• pData: The destination peripheral Buffer address.• Length: The length of data to be transferred from memory to DAC peripheral• Alignment: Specifies the data alignment for DAC channel. This parameter can be one of the following values:<ul style="list-style-type: none">– DAC_ALIGN_8B_R: 8bit right data alignment selected– DAC_ALIGN_12B_L: 12bit left data alignment selected– DAC_ALIGN_12B_R: 12bit right data alignment selected
Return values	<ul style="list-style-type: none">• HAL: status
HAL_DAC_Stop_DMA	
Function name	HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.• Channel: The selected DAC channel. This parameter can be one of the following values:<ul style="list-style-type: none">– DAC_CHANNEL_1: DAC Channel1 selected– DAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none">• HAL: status
HAL_DAC_IRQHandler	
Function name	void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)
Function description	Handles DAC interrupt request.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_SetValue

Function name

HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

Function description

Set the specified data holding register value for DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected
- **Alignment:** Specifies the data alignment. This parameter can be one of the following values:
 - DAC_ALIGN_8B_R: 8bit right data alignment selected
 - DAC_ALIGN_12B_L: 12bit left data alignment selected
 - DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data:** Data to be loaded in the selected data holding register.

Return values

- **HAL:** status

HAL_DAC_ConvCpltCallbackCh1

Function name

void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name

void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ErrorCallbackCh1

Function name

void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description	Error DAC callback for Channel1.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none">• None:
HAL_DAC_DMAUnderrunCallbackCh1	
Function name	void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)
Function description	DMA underrun DAC callback for channel1.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none">• None:
HAL_DAC_GetValue	
Function name	uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.• Channel: The selected DAC channel. This parameter can be one of the following values:<ul style="list-style-type: none">– DAC_CHANNEL_1: DAC Channel1 selected– DAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none">• The: selected DAC channel data output value.
HAL_DAC_ConfigChannel	
Function name	HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)
Function description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.• sConfig: DAC configuration structure.• Channel: The selected DAC channel. This parameter can be one of the following values:<ul style="list-style-type: none">– DAC_CHANNEL_1: DAC Channel1 selected– DAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none">• HAL: status
HAL_DAC_GetState	
Function name	HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)

Function description	return the DAC handle state
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none">• HAL: state

HAL_DAC_GetError

Function name	uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)
---------------	---

Function description	Return the DAC error code.
----------------------	----------------------------

Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
------------	--

Return values	<ul style="list-style-type: none">• DAC: Error Code
---------------	--

15.3 DAC Firmware driver defines

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

15.3.1 DAC

DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE No error

HAL_DAC_ERROR_DMAU DAC channel1 DMA underrun error
NDERRUNCH1

HAL_DAC_ERROR_DMAU DAC channel2 DMA underrun error
NDERRUNCH2

HAL_DAC_ERROR_DMA DMA error

DAC Exported Macros

<u>__HAL_DAC_RESET_HANDLE_STATE</u>	Description: <ul style="list-style-type: none">Reset DAC handle state. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the DAC handle. Return value: <ul style="list-style-type: none">None
<u>__HAL_DAC_ENABLE</u>	Description: <ul style="list-style-type: none">Enable the DAC channel. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the DAC handle.<u>__DAC_Channel__</u>: specifies the DAC channel Return value: <ul style="list-style-type: none">None
<u>__HAL_DAC_DISABLE</u>	Description: <ul style="list-style-type: none">Disable the DAC channel. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the DAC handle<u>__DAC_Channel__</u>: specifies the DAC channel. Return value: <ul style="list-style-type: none">None
<u>__HAL_DAC_ENABLE_IT</u>	Description: <ul style="list-style-type: none">Enable the DAC interrupt. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the DAC handle<u>__INTERRUPT__</u>: specifies the DAC interrupt. This parameter can be any combination of the following values:<ul style="list-style-type: none">DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interruptDAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt Return value: <ul style="list-style-type: none">None
<u>__HAL_DAC_DISABLE_IT</u>	Description: <ul style="list-style-type: none">Disable the DAC interrupt. Parameters: <ul style="list-style-type: none"><u>__HANDLE__</u>: specifies the DAC handle<u>__INTERRUPT__</u>: specifies the DAC interrupt. This parameter can be any combination of the following values:<ul style="list-style-type: none">DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt Return value: <ul style="list-style-type: none">None

__HAL_DAC_GET_IT_SOURECE

- Check whether the specified DAC interrupt source is enabled or not.

Parameters:

- __HANDLE__: DAC handle
- __INTERRUPT__: DAC interrupt source to check This parameter can be any combination of the following values:
 - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
 - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

__HAL_DAC_GET_FLAG**Description:**

- Get the selected DAC's flag status.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to get. This parameter can be any combination of the following values:
 - DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag

Return value:

- None

__HAL_DAC_CLEAR_FLAG

G

Description:

- Clear the DAC's flag.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to clear. This parameter can be any combination of the following values:
 - DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag

Return value:

- None

DAC flags definition**DAC_FLAG_DMAUDR1****DAC_FLAG_DMAUDR2*****DAC IT definition*****DAC_IT_DMAUDR1****DAC_IT_DMAUDR2*****DAC output buffer*****DAC_OUTPUTBUFFER_ENABLE****DAC_OUTPUTBUFFER_DISABLE**

DAC trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
DAC_TRIGGER_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T3_TRGO	TIM3 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T15_TRGO	TIM15 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_EXT_IT9	EXTI Line9 event selected as external conversion trigger for DAC channel
DAC_TRIGGER_SOFTWAR	Conversion started by software trigger for DAC channel

E

16 HAL DAC Extension Driver

16.1 DACEx Firmware driver API description

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

16.1.1 How to use this driver

- When Dual mode is enabled (i.e. DAC Channel1 and Channel2 are used simultaneously) : Use `HAL_DACEx_DualGetValue()` to get digital data to be converted and use `HAL_DACEx_DualSetValue()` to set digital value to converted simultaneously in Channel 1 and Channel 2.
- Use `HAL_DACEx_TriangleWaveGenerate()` to generate Triangle signal.
- Use `HAL_DACEx_NoiseWaveGenerate()` to generate Noise signal.

16.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.

This section contains the following APIs:

- `HAL_DACEx_DualGetValue`
- `HAL_DACEx_TriangleWaveGenerate`
- `HAL_DACEx_NoiseWaveGenerate`
- `HAL_DACEx_DualSetValue`
- `HAL_DACEx_ConvCpltCallbackCh2`
- `HAL_DACEx_ConvHalfCpltCallbackCh2`
- `HAL_DACEx_ErrorCallbackCh2`
- `HAL_DACEx_DMAUnderrunCallbackCh2`

16.1.3 Detailed description of functions

`HAL_DACEx_TriangleWaveGenerate`

Function name `HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)`

Function description Enables or disables the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
 - DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
 - DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
 - DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
 - DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
 - DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
 - DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
 - DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
 - DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
 - DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
 - DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
 - DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047
 - DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values

- **HAL:** status

HAL_DACEEx_NoiseWaveGenerate**Function name**

**HAL_StatusTypeDef HAL_DACEEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac,
uint32_t Channel, uint32_t Amplitude)**

Function description

Enables or disables the selected DAC channel wave generation.

Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.• Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2• Amplitude: Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:<ul style="list-style-type: none">– DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation– DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation– DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation– DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation– DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation– DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation– DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation– DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation– DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation– DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation– DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation– DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation
Return values	<ul style="list-style-type: none">• HAL: status
	HAL_DACE_DualSetValue
Function name	HAL_StatusTypeDef HAL_DACE_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)
Function description	Set the specified data holding register value for dual DAC channel.
Parameters	<ul style="list-style-type: none">• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.• Alignment: Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected• Data1: Data for DAC Channel2 to be loaded in the selected data holding register.• Data2: Data for DAC Channel1 to be loaded in the selected data holding register.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• In dual mode, a unique register access is required to write in both DAC channels at the same time.

HAL_DACEx_ConvCpltCallbackCh2

Function name **void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function description Conversion complete callback in non blocking mode for Channel2.

Parameters • **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values • **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name **void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function description Conversion half DMA transfer callback in non blocking mode for Channel2.

Parameters • **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values • **None:**

HAL_DACEx_ErrorCallbackCh2

Function name **void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function description Error DAC callback for Channel2.

Parameters • **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values • **None:**

HAL_DACEx_DMAUnderrunCallbackCh2

Function name **void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function description DMA underrun DAC callback for channel2.

Parameters • **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values • **None:**

HAL_DACEx_DualGetValue

Function name **uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)**

Function description Returns the last data output value of the selected DAC channel.

- Parameters**
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

- Return values**
- **The:** selected DAC channel data output value.

16.2 DACEEx Firmware driver defines

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

16.2.1 DACEEx

DACEEx

DACEEx_lfsrunmask_triangleamplitude

DAC_LFSRUNMASK_BIT0 Unmask DAC channel LFSR bit0 for noise wave generation

DAC_LFSRUNMASK_BITS1 Unmask DAC channel LFSR bit[1:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS2 Unmask DAC channel LFSR bit[2:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS3 Unmask DAC channel LFSR bit[3:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS4 Unmask DAC channel LFSR bit[4:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS5 Unmask DAC channel LFSR bit[5:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS6 Unmask DAC channel LFSR bit[6:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS7 Unmask DAC channel LFSR bit[7:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS8 Unmask DAC channel LFSR bit[8:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS9 Unmask DAC channel LFSR bit[9:0] for noise wave generation
_0

DAC_LFSRUNMASK_BITS10 Unmask DAC channel LFSR bit[10:0] for noise wave generation
_0_0

DAC_LFSRUNMASK_BITS11 Unmask DAC channel LFSR bit[11:0] for noise wave generation
_1_0

DAC_TRIANGLEAMPLITUD_E_1 Select max triangle amplitude of 1

DAC_TRIANGLEAMPLITUD_E_3 Select max triangle amplitude of 3

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 7
E_7

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 15
E_15

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 31
E_31

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 63
E_63

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 127
E_127

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 255
E_255

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 511
E_511

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 1023
E_1023

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 2047
E_2047

DAC_TRIANGLEAMPLITUD Select max triangle amplitude of 4095
E_4095

17 HAL DMA Generic Driver

17.1 DMA Firmware driver registers structures

17.1.1 DMA_InitTypeDef

`DMA_InitTypeDef` is defined in the `stm32f0xx_hal_dma.h`

Data Fields

- `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::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`

17.1.2 DMA_HandleTypeDef

`DMA_HandleTypeDef` is defined in the `stm32f0xx_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`

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

17.2 DMA Firmware driver API description

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

17.2.1 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 Reference manual for connection between peripherals and DMA requests .
2. For a given Channel, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode, using `HAL_DMA_Init()` function.
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_Channel_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_DMA_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of DMA handle structure).

DMA HAL driver macros list

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

Note:

You can refer to the DMA HAL driver header file for more useful macros

17.2.2 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_DelInit**](#)

17.2.3 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**](#)

17.2.4 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**](#)

17.2.5 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	<ul style="list-style-type: none">• HAL: status
HAL_DMA_Abort	
Function name	HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)
Function description	Abort the DMA Transfer.
Parameters	<ul style="list-style-type: none">• hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none">• HAL: status
HAL_DMA_Abort_IT	
Function name	HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)
Function description	Abort the DMA Transfer in Interrupt mode.
Parameters	<ul style="list-style-type: none">• hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
Return values	<ul style="list-style-type: none">• HAL: status
HAL_DMA_PollForTransfer	
Function name	HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)
Function description	Polling for transfer complete.
Parameters	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• HAL: status
HAL_DMA_IRQHandler	
Function name	void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)
Function description	Handle DMA interrupt request.
Parameters	<ul style="list-style-type: none">• hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none">• None:

HAL_DMA_RegisterCallback

Function name	<code>HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *_hdma) pCallback)</code>
Function description	Register callbacks.
Parameters	<ul style="list-style-type: none">hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.CallbackID: User Callback identifier a HAL_DMA_CallbackIDTypeDef ENUM as parameter.pCallback: pointer to private callback function which has pointer to a DMA_HandleTypeDef structure as parameter.
Return values	<ul style="list-style-type: none">HAL: status

HAL_DMA_UnRegisterCallback

Function name	<code>HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID)</code>
Function description	UnRegister callbacks.
Parameters	<ul style="list-style-type: none">hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.CallbackID: User Callback identifier a HAL_DMA_CallbackIDTypeDef ENUM as parameter.
Return values	<ul style="list-style-type: none">HAL: status

HAL_DMA_GetState

Function name	<code>HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)</code>
Function description	Returns the DMA state.
Parameters	<ul style="list-style-type: none">hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none">HAL: state

HAL_DMA_GetError

Function name	<code>uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)</code>
Function description	Return the DMA error code.
Parameters	<ul style="list-style-type: none">hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none">DMA: Error Code

17.3 DMA Firmware driver defines

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

17.3.1 DMA

DMA

DMA Data transfer direction

DMA_PERIPH_TO_MEMOR Peripheral to memory direction
Y

DMA_MEMORY_TO_PERIP Memory to peripheral direction
H

DMA_MEMORY_TO_MEMO Memory to memory direction
RY

DMA Error Code

HAL_DMA_ERROR_NONE No error

HAL_DMA_ERROR_TE Transfer error

HAL_DMA_ERROR_NO_XF no ongoing transfer
ER

HAL_DMA_ERROR_TIMEO Timeout error
UT

HAL_DMA_ERROR_NOT_S Not supported mode
UPPORTED

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_ENABLE_IT](#)

Description:

- Enables 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:

- Disables 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_SOURE](#)

Description:

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

Parameters:

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

Return value:

- The state of DMA_IT (SET or RESET).

__HAL_DMA_GET_COUNT **Description:**

ER

- Returns the number of remaining data units in the current DMAy Channelx 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 interrupt 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 interrupt 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 interrupt 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 interrupt 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 interrupt 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 interrupt 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 interrupt 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 interrupt enable definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory data size

DMA_MDATAALIGN_BYTE Memory data alignment : Byte

DMA_MDATAALIGN_HALF WORD Memory data alignment : HalfWord WORD

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_HALF WORD Peripheral data alignment : HalfWord WORD

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_HIG Priority level : Very_High
H

18 HAL DMA Extension Driver

18.1 DMAEx Firmware driver defines

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

18.1.1 DMAEx

DMAEx

DMAEx Exported Constants

DMA1_CHANNEL1_RMP Internal define for remapping on STM32F09x/30xC

DMA1_CHANNEL2_RMP Internal define for remapping on STM32F09x/30xC

DMA1_CHANNEL3_RMP Internal define for remapping on STM32F09x/30xC

DMA1_CHANNEL4_RMP Internal define for remapping on STM32F09x/30xC

DMA1_CHANNEL5_RMP Internal define for remapping on STM32F09x/30xC

DMA1_CHANNEL6_RMP Internal define for remapping on STM32F09x/30xC

DMA1_CHANNEL7_RMP Internal define for remapping on STM32F09x/30xC

DMA2_CHANNEL1_RMP Internal define for remapping on STM32F09x/30xC

DMA2_CHANNEL2_RMP Internal define for remapping on STM32F09x/30xC

DMA2_CHANNEL3_RMP Internal define for remapping on STM32F09x/30xC

DMA2_CHANNEL4_RMP Internal define for remapping on STM32F09x/30xC

DMA2_CHANNEL5_RMP Internal define for remapping on STM32F09x/30xC

HAL_DMA1_CH1_DEFAULT Default remap position for DMA1

HAL_DMA1_CH1_ADC Remap ADC on DMA1 Channel 1

HAL_DMA1_CH1_TIM17_C Remap TIM17 channel 1 on DMA1 channel 1
H1

HAL_DMA1_CH1_TIM17_U Remap TIM17 up on DMA1 channel 1
P

HAL_DMA1_CH1_USART1_ Remap USART1 Rx on DMA1 channel 1
RX

HAL_DMA1_CH1_USART2_ Remap USART2 Rx on DMA1 channel 1
RX

HAL_DMA1_CH1_USART3_ Remap USART3 Rx on DMA1 channel 1
RX

HAL_DMA1_CH1_USART4_ Remap USART4 Rx on DMA1 channel 1
RX

HAL_DMA1_CH1_USART5_ Remap USART5 Rx on DMA1 channel 1
RX

HAL_DMA1_CH1_USART6_ Remap USART6 Rx on DMA1 channel 1
RX

HAL_DMA1_CH1_USART7_ Remap USART7 Rx on DMA1 channel 1
RX

HAL_DMA1_CH1_USART8_ Remap USART8 Rx on DMA1 channel 1
RX

HAL_DMA1_CH2_DEFAULT Default remap position for DMA1

HAL_DMA1_CH2_ADC Remap ADC on DMA1 channel 2

HAL_DMA1_CH2_I2C1_TX Remap I2C1 Tx on DMA1 channel 2

HAL_DMA1_CH2_SPI1_RX Remap SPI1 Rx on DMA1 channel 2

HAL_DMA1_CH2_TIM1_CH Remap TIM1 channel 1 on DMA1 channel 2
1

HAL_DMA1_CH2_TIM17_C Remap TIM17 channel 1 on DMA1 channel 2
H1

HAL_DMA1_CH2_TIM17_U Remap TIM17 up on DMA1 channel 2
P

HAL_DMA1_CH2_USART1_ Remap USART1 Tx on DMA1 channel 2
TX

HAL_DMA1_CH2_USART2_ Remap USART2 Tx on DMA1 channel 2
TX

HAL_DMA1_CH2_USART3_ Remap USART3 Tx on DMA1 channel 2
TX

HAL_DMA1_CH2_USART4_ Remap USART4 Tx on DMA1 channel 2
TX

HAL_DMA1_CH2_USART5_ Remap USART5 Tx on DMA1 channel 2
TX

HAL_DMA1_CH2_USART6_ Remap USART6 Tx on DMA1 channel 2
TX

HAL_DMA1_CH2_USART7_ Remap USART7 Tx on DMA1 channel 2
TX

HAL_DMA1_CH2_USART8_ Remap USART8 Tx on DMA1 channel 2
TX

HAL_DMA1_CH3_DEFAULT Default remap position for DMA1

HAL_DMA1_CH3_TIM6_UP Remap TIM6 up on DMA1 channel 3

HAL_DMA1_CH3_DAC_CH_1 Remap DAC Channel 1 on DMA1 channel 3
1

HAL_DMA1_CH3_I2C1_RX Remap I2C1 Rx on DMA1 channel 3

HAL_DMA1_CH3_SPI1_TX Remap SPI1 Tx on DMA1 channel 3

HAL_DMA1_CH3_TIM1_CH_2 Remap TIM1 channel 2 on DMA1 channel 3
2

HAL_DMA1_CH3_TIM2_CH_2 Remap TIM2 channel 2 on DMA1 channel 3
2

HAL_DMA1_CH3_TIM16_C_H1 Remap TIM16 channel 1 on DMA1 channel 3
H1

HAL_DMA1_CH3_TIM16_U_P Remap TIM16 up on DMA1 channel 3
P

HAL_DMA1_CH3_USART1_RX Remap USART1 Rx on DMA1 channel 3
RX

HAL_DMA1_CH3_USART2_RX Remap USART2 Rx on DMA1 channel 3
RX

HAL_DMA1_CH3_USART3_RX Remap USART3 Rx on DMA1 channel 3
RX

HAL_DMA1_CH3_USART4_RX Remap USART4 Rx on DMA1 channel 3
RX

HAL_DMA1_CH3_USART5_RX Remap USART5 Rx on DMA1 channel 3
RX

HAL_DMA1_CH3_USART6_RX Remap USART6 Rx on DMA1 channel 3
RX

HAL_DMA1_CH3_USART7_RX Remap USART7 Rx on DMA1 channel 3
RX

HAL_DMA1_CH3_USART8_RX Remap USART8 Rx on DMA1 channel 3
RX

HAL_DMA1_CH4_DEFAULT Default remap position for DMA1

HAL_DMA1_CH4_TIM7_UP Remap TIM7 up on DMA1 channel 4

HAL_DMA1_CH4_DAC_CH2 Remap DAC Channel 2 on DMA1 channel 4

HAL_DMA1_CH4_I2C2_TX Remap I2C2 Tx on DMA1 channel 4

HAL_DMA1_CH4_SPI2_RX Remap SPI2 Rx on DMA1 channel 4

HAL_DMA1_CH4_TIM2_CH4 Remap TIM2 channel 4 on DMA1 channel 4

HAL_DMA1_CH4_TIM3_CH1 Remap TIM3 channel 1 on DMA1 channel 4

HAL_DMA1_CH4_TIM3_TRI_G Remap TIM3 Trig on DMA1 channel 4

HAL_DMA1_CH4_TIM16_CH1 Remap TIM16 channel 1 on DMA1 channel 4

HAL_DMA1_CH4_TIM16_UP_P Remap TIM16 up on DMA1 channel 4

HAL_DMA1_CH4_USART1_TX Remap USART1 Tx on DMA1 channel 4

HAL_DMA1_CH4_USART2_TX Remap USART2 Tx on DMA1 channel 4

HAL_DMA1_CH4_USART3_TX Remap USART3 Tx on DMA1 channel 4

HAL_DMA1_CH4_USART4_TX Remap USART4 Tx on DMA1 channel 4

HAL_DMA1_CH4_USART5_TX Remap USART5 Tx on DMA1 channel 4

HAL_DMA1_CH4_USART6_TX Remap USART6 Tx on DMA1 channel 4

HAL_DMA1_CH4_USART7_TX Remap USART7 Tx on DMA1 channel 4

HAL_DMA1_CH4_USART8_TX Remap USART8 Tx on DMA1 channel 4

HAL_DMA1_CH5_DEFAULT Default remap position for DMA1

HAL_DMA1_CH5_I2C2_RX Remap I2C2 Rx on DMA1 channel 5

HAL_DMA1_CH5_SPI2_TX Remap SPI1 Tx on DMA1 channel 5

HAL_DMA1_CH5_TIM1_CH_3 Remap TIM1 channel 3 on DMA1 channel 5

HAL_DMA1_CH5_USART1_RX Remap USART1 Rx on DMA1 channel 5

HAL_DMA1_CH5_USART2_RX Remap USART2 Rx on DMA1 channel 5

HAL_DMA1_CH5_USART3_RX Remap USART3 Rx on DMA1 channel 5

HAL_DMA1_CH5_USART4_RX Remap USART4 Rx on DMA1 channel 5

HAL_DMA1_CH5_USART5_RX Remap USART5 Rx on DMA1 channel 5

HAL_DMA1_CH5_USART6_RX Remap USART6 Rx on DMA1 channel 5

HAL_DMA1_CH5_USART7_RX Remap USART7 Rx on DMA1 channel 5

HAL_DMA1_CH5_USART8_RX Remap USART8 Rx on DMA1 channel 5

HAL_DMA1_CH6_DEFAULT Default remap position for DMA1

HAL_DMA1_CH6_I2C1_TX Remap I2C1 Tx on DMA1 channel 6

HAL_DMA1_CH6_SPI2_RX Remap SPI2 Rx on DMA1 channel 6

HAL_DMA1_CH6_TIM1_CH_1 Remap TIM1 channel 1 on DMA1 channel 6

HAL_DMA1_CH6_TIM1_CH_2 Remap TIM1 channel 2 on DMA1 channel 6

HAL_DMA1_CH6_TIM1_CH_3 Remap TIM1 channel 3 on DMA1 channel 6

HAL_DMA1_CH6_TIM3_CH_1 Remap TIM3 channel 1 on DMA1 channel 6

HAL_DMA1_CH6_TIM3_TRI_G Remap TIM3 Trig on DMA1 channel 6

HAL_DMA1_CH6_TIM16_C Remap TIM16 channel 1 on DMA1 channel 6
H1

HAL_DMA1_CH6_TIM16_U Remap TIM16 up on DMA1 channel 6
P

HAL_DMA1_CH6_USART1_ Remap USART1 Rx on DMA1 channel 6
RX

HAL_DMA1_CH6_USART2_ Remap USART2 Rx on DMA1 channel 6
RX

HAL_DMA1_CH6_USART3_ Remap USART3 Rx on DMA1 channel 6
RX

HAL_DMA1_CH6_USART4_ Remap USART4 Rx on DMA1 channel 6
RX

HAL_DMA1_CH6_USART5_ Remap USART5 Rx on DMA1 channel 6
RX

HAL_DMA1_CH6_USART6_ Remap USART6 Rx on DMA1 channel 6
RX

HAL_DMA1_CH6_USART7_ Remap USART7 Rx on DMA1 channel 6
RX

HAL_DMA1_CH6_USART8_ Remap USART8 Rx on DMA1 channel 6
RX

HAL_DMA1_CH7_DEFAULT Default remap position for DMA1

HAL_DMA1_CH7_I2C1_RX Remap I2C1 Rx on DMA1 channel 7

HAL_DMA1_CH7_SPI2_TX Remap SPI2 Tx on DMA1 channel 7

HAL_DMA1_CH7_TIM2_CH Remap TIM2 channel 2 on DMA1 channel 7
2

HAL_DMA1_CH7_TIM2_CH Remap TIM2 channel 4 on DMA1 channel 7
4

HAL_DMA1_CH7_TIM17_C Remap TIM17 channel 1 on DMA1 channel 7
H1

HAL_DMA1_CH7_TIM17_U Remap TIM17 up on DMA1 channel 7
P

HAL_DMA1_CH7_USART1_ Remap USART1 Tx on DMA1 channel 7
TX

HAL_DMA1_CH7_USART2_ Remap USART2 Tx on DMA1 channel 7
TX

HAL_DMA1_CH7_USART3_ Remap USART3 Tx on DMA1 channel 7
TX

HAL_DMA1_CH7_USART4_ Remap USART4 Tx on DMA1 channel 7
TX

HAL_DMA1_CH7_USART5_ Remap USART5 Tx on DMA1 channel 7
TX

HAL_DMA1_CH7_USART6_ Remap USART6 Tx on DMA1 channel 7
TX

HAL_DMA1_CH7_USART7_ Remap USART7 Tx on DMA1 channel 7
TX

HAL_DMA1_CH7_USART8_ Remap USART8 Tx on DMA1 channel 7
TX

HAL_DMA2_CH1_DEFAULT Default remap position for DMA2

HAL_DMA2_CH1_I2C2_TX Remap I2C2 TX on DMA2 channel 1

HAL_DMA2_CH1_USART1_ Remap USART1 Tx on DMA2 channel 1
TX

HAL_DMA2_CH1_USART2_ Remap USART2 Tx on DMA2 channel 1
TX

HAL_DMA2_CH1_USART3_ Remap USART3 Tx on DMA2 channel 1
TX

HAL_DMA2_CH1_USART4_ Remap USART4 Tx on DMA2 channel 1
TX

HAL_DMA2_CH1_USART5_ Remap USART5 Tx on DMA2 channel 1
TX

HAL_DMA2_CH1_USART6_ Remap USART6 Tx on DMA2 channel 1
TX

HAL_DMA2_CH1_USART7_ Remap USART7 Tx on DMA2 channel 1
TX

HAL_DMA2_CH1_USART8_ Remap USART8 Tx on DMA2 channel 1
TX

HAL_DMA2_CH2_DEFAULT Default remap position for DMA2

HAL_DMA2_CH2_I2C2_RX Remap I2C2 Rx on DMA2 channel 2

HAL_DMA2_CH2_USART1_ Remap USART1 Rx on DMA2 channel 2
RX

HAL_DMA2_CH2_USART2_ Remap USART2 Rx on DMA2 channel 2
RX

HAL_DMA2_CH2_USART3_ Remap USART3 Rx on DMA2 channel 2
RX

HAL_DMA2_CH2_USART4_ Remap USART4 Rx on DMA2 channel 2
RX

HAL_DMA2_CH2_USART5_ Remap USART5 Rx on DMA2 channel 2
RX

HAL_DMA2_CH2_USART6_ Remap USART6 Rx on DMA2 channel 2
RX

HAL_DMA2_CH2_USART7_ Remap USART7 Rx on DMA2 channel 2
RX

HAL_DMA2_CH2_USART8_ Remap USART8 Rx on DMA2 channel 2
RX

HAL_DMA2_CH3_DEFAULT Default remap position for DMA2

HAL_DMA2_CH3_TIM6_UP Remap TIM6 up on DMA2 channel 3

HAL_DMA2_CH3_DAC_CH Remap DAC channel 1 on DMA2 channel 3
1

HAL_DMA2_CH3_SPI1_RX Remap SPI1 Rx on DMA2 channel 3

HAL_DMA2_CH3_USART1_ Remap USART1 Rx on DMA2 channel 3
RX

HAL_DMA2_CH3_USART2_ Remap USART2 Rx on DMA2 channel 3
RX

HAL_DMA2_CH3_USART3_ Remap USART3 Rx on DMA2 channel 3
RX

HAL_DMA2_CH3_USART4_ Remap USART4 Rx on DMA2 channel 3
RX

HAL_DMA2_CH3_USART5_ Remap USART5 Rx on DMA2 channel 3
RX

HAL_DMA2_CH3_USART6_ Remap USART6 Rx on DMA2 channel 3
RX

HAL_DMA2_CH3_USART7_ Remap USART7 Rx on DMA2 channel 3
RX

HAL_DMA2_CH3_USART8_ Remap USART8 Rx on DMA2 channel 3
RX

HAL_DMA2_CH4_DEFAULT Default remap position for DMA2

HAL_DMA2_CH4_TIM7_UP Remap TIM7 up on DMA2 channel 4

HAL_DMA2_CH4_DAC_CH2 Remap DAC channel 2 on DMA2 channel 4

HAL_DMA2_CH4_SPI1_TX Remap SPI1 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART1_TX Remap USART1 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART2_TX Remap USART2 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART3_TX Remap USART3 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART4_TX Remap USART4 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART5_TX Remap USART5 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART6_TX Remap USART6 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART7_TX Remap USART7 Tx on DMA2 channel 4

HAL_DMA2_CH4_USART8_TX Remap USART8 Tx on DMA2 channel 4

HAL_DMA2_CH5_DEFAULT Default remap position for DMA2

HAL_DMA2_CH5_ADC Remap ADC on DMA2 channel 5

HAL_DMA2_CH5_USART1_TX Remap USART1 Tx on DMA2 channel 5

HAL_DMA2_CH5_USART2_TX Remap USART2 Tx on DMA2 channel 5

HAL_DMA2_CH5_USART3_TX Remap USART3 Tx on DMA2 channel 5

HAL_DMA2_CH5_USART4_TX Remap USART4 Tx on DMA2 channel 5

HAL_DMA2_CH5_USART5_TX Remap USART5 Tx on DMA2 channel 5

HAL_DMA2_CH5_USART6_ Remap USART6 Tx on DMA2 channel 5
TX

HAL_DMA2_CH5_USART7_ Remap USART7 Tx on DMA2 channel 5
TX

HAL_DMA2_CH5_USART8_ Remap USART8 Tx on DMA2 channel 5
TX

IS_HAL_DMA1_REMAP

IS_HAL_DMA2_REMAP

DMAEx Exported Macros

_HAL_DMA_GET_TC_FLA **Description:**

G_INDEX • Returns the current DMA Channel transfer complete flag.

Parameters:

- **_HANDLE_**: DMA handle

Return value:

- The: specified transfer complete flag index.

_HAL_DMA_GET_HT_FLA **Description:**

G_INDEX • 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_FLA **Description:**

G_INDEX • Returns the current DMA Channel transfer error flag.

Parameters:

- **_HANDLE_**: DMA handle

Return value:

- The: specified transfer error flag index.

_HAL_DMA_GET_GI_FLA **Description:**

G_INDEX • Return the current DMA Channel Global interrupt flag.

Parameters:

- **_HANDLE_**: DMA handle

Return value:

- The: specified transfer error flag index.

[__HAL_DMA_GET_FLAG](#)

Description:

- Get the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
 - DMA_FLAG_TCx: Transfer complete flag
 - DMA_FLAG_HTx: Half transfer complete flag
 - DMA_FLAG_TEx: Transfer error flag Where x can be 0_4, 1_5, 2_6 or 3_7 to select the DMA Channel flag.

Return value:

- The state of FLAG (SET or RESET).

[__HAL_DMA_CLEAR_FLAG](#)

G

Description:

- Clears 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 Where x can be 0_4, 1_5, 2_6 or 3_7 to select the DMA Channel flag.

Return value:

- None

[__HAL_DMA1_REMAP](#)

[__HAL_DMA2_REMAP](#)

19 HAL FLASH Generic Driver

19.1 FLASH Firmware driver registers structures

19.1.1 **FLASH_ProcTypeDef**

FLASH_ProcTypeDef is defined in the `stm32f0xx_hal_flash.h`

Data Fields

- `__IO FLASH_ProcedureTypeDef ProcedureOnGoing`
- `__IO uint32_t DataRemaining`
- `__IO uint32_t Address`
- `__IO uint64_t Data`
- `HAL_LockTypeDef Lock`
- `__IO uint32_t ErrorCode`

Field Documentation

- `__IO FLASH_ProcedureTypeDef FLASH_ProcTypeDef::ProcedureOnGoing`
Internal variable to indicate which procedure is ongoing or not in IT context
- `__IO uint32_t FLASH_ProcTypeDef::DataRemaining`
Internal variable to save the remaining pages to erase or half-word to program in IT context
- `__IO uint32_t FLASH_ProcTypeDef::Address`
Internal variable to save address selected for program or erase
- `__IO uint64_t FLASH_ProcTypeDef::Data`
Internal variable to save data to be programmed
- `HAL_LockTypeDef FLASH_ProcTypeDef::Lock`
FLASH locking object
- `__IO uint32_t FLASH_ProcTypeDef::ErrorCode`
FLASH error code This parameter can be a value of `FLASH_Error_Codes`

19.2 FLASH Firmware driver API description

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

19.2.1 **FLASH peripheral features**

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The Flash memory interface accelerates code execution with a system of instruction prefetch.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- Option Bytes programming

19.2.2 **How to use this driver**

This driver provides functions and macros to configure and program the FLASH memory of all STM32F0xx devices.

1. FLASH Memory I/O Programming functions: this group includes all needed functions to erase and program the main memory:
 - Lock and Unlock the FLASH interface
 - Erase function: Erase page, erase all pages
 - Program functions: half word, word and doubleword
2. FLASH Option Bytes Programming functions: this group includes all needed functions to manage the Option Bytes:
 - Lock and Unlock the Option Bytes
 - Set/Reset the write protection
 - Set the Read protection Level
 - Program the user Option Bytes
 - Launch the Option Bytes loader
 - Erase Option Bytes
 - Program the data Option Bytes
 - Get the Write protection.
 - Get the user option bytes.
3. Interrupts and flags management functions : this group includes all needed functions to:
 - Handle FLASH interrupts
 - Wait for last FLASH operation according to its status
 - Get error flag status

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

- Set/Get the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

19.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- [`HAL_FLASH_Unlock`](#)
- [`HAL_FLASH_Lock`](#)
- [`HAL_FLASH_OB_Unlock`](#)
- [`HAL_FLASH_OB_Lock`](#)
- [`HAL_FLASH_OB_Launch`](#)

19.2.4 Peripheral Errors functions

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

This section contains the following APIs:

- [`HAL_FLASH_GetError`](#)

19.2.5 Detailed description of functions

[`HAL_FLASH_Program`](#)

Function name [`HAL_StatusTypeDef HAL_FLASH_Program \(uint32_t TypeProgram, uint32_t Address, uint64_t Data\)`](#)

Function description Program halfword, word or double word at a specified address.

Parameters	<ul style="list-style-type: none">TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type ProgramAddress: Specific the address to be programmed.Data: Specific the data to be programmed
Return values	<ul style="list-style-type: none">HAL_StatusTypeDef: HAL Status
Notes	<ul style="list-style-type: none">The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interfaceIf an erase and a program operations are requested simultaneously, the erase operation is performed before the program one.FLASH should be previously erased before new programmation (only exception to this is when 0x0000 is programmed)

HAL_FLASH_Program_IT

Function name	HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)
Function description	Program halfword, word or double word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none">TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type ProgramAddress: Specific the address to be programmed.Data: Specific the data to be programmed
Return values	<ul style="list-style-type: none">HAL_StatusTypeDef: HAL Status
Notes	<ul style="list-style-type: none">The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interfaceIf an erase and a program operations are requested simultaneously, the erase operation is performed before the program one.

HAL_FLASH_IRQHandler

Function name	void HAL_FLASH_IRQHandler (void)
Function description	This function handles FLASH interrupt request.
Return values	<ul style="list-style-type: none">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: No return value expected
 - Pages Erase: Address of the page which has been erased (if 0xFFFFFFFF, it means that all the selected pages have been erased)
 - Program: Address which was selected for data program

- Return values**
- **none:**

HAL_FLASH_OperationErrorCallback

Function name `void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)`

Function description FLASH operation error interrupt callback.

- Parameters**
- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure
 - Mass Erase: No return value expected
 - Pages Erase: Address of the page which returned an error
 - Program: Address which was selected for data program

- Return values**
- **none:**

HAL_FLASH_Unlock

Function name `HAL_StatusTypeDef HAL_FLASH_Unlock (void)`

Function description Unlock the FLASH control register access.

- Return values**
- **HAL:** Status

HAL_FLASH_Lock

Function name `HAL_StatusTypeDef HAL_FLASH_Lock (void)`

Function description Locks the FLASH control register access.

- Return values**
- **HAL:** Status

HAL_FLASH_OB_Unlock

Function name `HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)`

Function description Unlock the FLASH Option Control Registers access.

- Return values**
- **HAL:** Status

HAL_FLASH_OB_Lock

Function name `HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)`

Function description Lock the FLASH Option Control Registers access.

- Return values**
- **HAL:** Status

HAL_FLASH_OB_Launch

Function name	HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)
Function description	Launch the option byte loading.
Return values	<ul style="list-style-type: none">• HAL: Status
Notes	<ul style="list-style-type: none">• This function will reset automatically the MCU.

HAL_FLASH_GetError

Function name	uint32_t HAL_FLASH_GetError (void)
Function description	Get the specific FLASH error flag.
Return values	<ul style="list-style-type: none">• FLASH_ErrorCode: The returned value can be: FLASH Error Codes

FLASH_WaitForLastOperation

Function name	HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout)
Function description	Wait for a FLASH operation to complete.
Parameters	<ul style="list-style-type: none">• Timeout: maximum flash operation timeout
Return values	<ul style="list-style-type: none">• HAL: Status

19.3 FLASH Firmware driver defines

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

19.3.1 FLASH

FLASH

FLASH Error Codes

HAL_FLASH_ERROR_NON No error
E

HAL_FLASH_ERROR_PRO Programming error
G

HAL_FLASH_ERROR_WRP Write protection error

FLASH Flag definition

FLASH_FLAG_BSY FLASH Busy flag

FLASH_FLAG_PGERR FLASH Programming error flag

FLASH_FLAG_WRPERR FLASH Write protected error flag

FLASH_FLAG_EOP FLASH End of Operation flag

FLASH Interrupts

__HAL_FLASH_ENABLE_I **Description:**

- Enable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
 - FLASH_IT_EOP End of FLASH Operation Interrupt
 - FLASH_IT_ERR Error Interrupt

Return value:

- none

__HAL_FLASH_DISABLE_I **Description:**

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
 - FLASH_IT_EOP End of FLASH Operation Interrupt
 - FLASH_IT_ERR Error Interrupt

Return value:

- none

__HAL_FLASH_GET_FLAG **Description:**

- Get the specified FLASH flag status.

Parameters:

- __FLAG__: specifies the FLASH flag to check. This parameter can be one of the following values:
 - FLASH_FLAG_BSY FLASH Busy flag
 - FLASH_FLAG_EOP FLASH End of Operation flag
 - FLASH_FLAG_WRPERR FLASH Write protected error flag
 - FLASH_FLAG_PGERR FLASH Programming error flag

Return value:

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

__HAL_FLASH_CLEAR_FL **Description:**

- Clear the specified FLASH flag.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
 - FLASH_FLAG_EOP FLASH End of Operation flag
 - FLASH_FLAG_WRPERR FLASH Write protected error flag
 - FLASH_FLAG_PGERR FLASH Programming error flag

Return value:

- none

FLASH Interrupt definition

FLASH_IT_EOP End of FLASH Operation Interrupt source

FLASH_IT_ERR Error Interrupt source

FLASH Latency

FLASH_LATENCY_0 FLASH Zero Latency cycle

FLASH_LATENCY_1 FLASH One Latency cycle

FLASH Prefetch

_HAL_FLASH_PREFETCH **Description:**

_BUFFER_ENABLE • Enable the FLASH prefetch buffer.

Return value:

- None

_HAL_FLASH_PREFETCH **Description:**

_BUFFER_DISABLE • Disable the FLASH prefetch buffer.

Return value:

- None

FLASH Type Program

FLASH_TYPEPROGRAM_H Program a half-word (16-bit) at a specified address.

ALFWORD

FLASH_TYPEPROGRAM_WORD Program a word (32-bit) at a specified address.

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

DOUBLEWORD

20 HAL FLASH Extension Driver

20.1 FLASHEx Firmware driver registers structures

20.1.1 **FLASH_EraseInitTypeDef**

FLASH_EraseInitTypeDef is defined in the `stm32f0xx_hal_flash_ex.h`

Data Fields

- `uint32_t TypeErase`
- `uint32_t PageAddress`
- `uint32_t NbPages`

Field Documentation

- `uint32_t FLASH_EraseInitTypeDef::TypeErase`

TypeErase: Mass erase or page erase. This parameter can be a value of `FLASHEx_Type_Erase`

- `uint32_t FLASH_EraseInitTypeDef::PageAddress`

PageAddress: Initial FLASH page address to erase when mass erase is disabled. This parameter must be a number between Min_Data = FLASH_BASE and Max_Data = FLASH_BANK1_END

- `uint32_t FLASH_EraseInitTypeDef::NbPages`

NbPages: Number of pages to be erased. This parameter must be a value between Min_Data = 1 and Max_Data = (max number of pages - value of initial page)

20.1.2 **FLASH_OBProgramInitTypeDef**

FLASH_OBProgramInitTypeDef is defined in the `stm32f0xx_hal_flash_ex.h`

Data Fields

- `uint32_t OptionType`
- `uint32_t WRPState`
- `uint32_t WRPPage`
- `uint8_t RDPLevel`
- `uint8_t USERConfig`
- `uint32_t DATAAddress`
- `uint8_t DATAData`

Field Documentation

- `uint32_t FLASH_OBProgramInitTypeDef::OptionType`

OptionType: Option byte to be configured. This parameter can be a value of `FLASHEx_OB_Type`

- `uint32_t FLASH_OBProgramInitTypeDef::WRPState`

WRPState: Write protection activation or deactivation. This parameter can be a value of `FLASHEx_OB_WRP_State`

- `uint32_t FLASH_OBProgramInitTypeDef::WRPPage`

WRPPage: specifies the page(s) to be write protected. This parameter can be a value of `FLASHEx_OB_Write_Protection`

- `uint8_t FLASH_OBProgramInitTypeDef::RDPLevel`

RDPLevel: Set the read protection level.. This parameter can be a value of `FLASHEx_OB_Read_Protection`

- `uint8_t FLASH_OBProgramInitTypeDef::USERConfig`

USERConfig: Program the FLASH User Option Byte: IWDG / STOP / STDBY / BOOT1 / VDDA_ANALOG / SRAM_PARITY This parameter can be a combination of `FLASHEx_OB_IWWatchdog`, `FLASHEx_OB_nRST_STOP`, `FLASHEx_OB_nRST_STDBY`, `FLASHEx_OB_BOOT1`, `FLASHEx_OB_VDDA_Analog_Monitoring` and `FLASHEx_OB_RAM_Parity_Check_Enable`

- ***uint32_t FLASH_OBProgramInitTypeDef::DATAAddress***
DATAAddress: Address of the option byte DATA to be programmed This parameter can be a value of ***FLASHEx_OB_Data_Address***
- ***uint8_t FLASH_OBProgramInitTypeDef::DATAData***
DATAData: Data to be stored in the option byte DATA This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF

20.2 FLASHEx Firmware driver API description

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

20.2.1 FLASH Erasing Programming functions

The FLASH Memory Erasing functions, includes the following functions:

- **@ref HAL_FLASHEx_Erase**: return only when erase has been done
- **@ref HAL_FLASHEx_Erase_IT**: end of erase is done when **@ref HAL_FLASH_EndOfOperationCallback** is called with parameter 0xFFFFFFFF

Any operation of erase should follow these steps:

1. Call the **@ref HAL_FLASH_Unlock()** function to enable the flash control register and program memory access.
2. Call the desired function to erase page.
3. Call the **@ref HAL_FLASH_Lock()** to disable the flash program memory access (recommended to protect the FLASH memory against possible unwanted operation).

This section contains the following APIs:

- ***HAL_FLASHEx_Erase***
- ***HAL_FLASHEx_Erase_IT***

20.2.2 Option Bytes Programming functions

This subsection provides a set of functions allowing to control the FLASH option bytes operations.

This section contains the following APIs:

- ***HAL_FLASHEx_OBErase***
- ***HAL_FLASHEx_OBProgram***
- ***HAL_FLASHEx_OBGetConfig***
- ***HAL_FLASHEx_OBGetUserData***

20.2.3 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_StatusTypeDef**: HAL Status

Notes

- To correctly run this function, the **HAL_FLASH_Unlock()** function must be called before. Call the **HAL_FLASH_Lock()** to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

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	<ul style="list-style-type: none">pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
Return values	<ul style="list-style-type: none">HAL_StatusTypeDef: HAL Status
Notes	<ul style="list-style-type: none">To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

HAL_FLASHEx_OBErase

Function name	HAL_StatusTypeDef HAL_FLASHEx_OBErase (void)
Function description	Erases the FLASH option bytes.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">This functions erases all option bytes except the Read protection (RDP). The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock the options bytes The function HAL_FLASH_OB_Launch() should be called after to force the reload of the options bytes (system reset will occur)

HAL_FLASHEx_OBProgram

Function name	HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)
Function description	Program option bytes.
Parameters	<ul style="list-style-type: none">pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none">HAL_StatusTypeDef: HAL Status
Notes	<ul style="list-style-type: none">The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock the options bytes The function HAL_FLASH_OB_Launch() should be called after to force the reload of the options bytes (system reset will occur)

HAL_FLASHEx_OBGetConfig

Function name	void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)
Function description	Get the Option byte configuration.

- Parameters**
- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.

- Return values**
- **None:**

HAL_FLASHEx_OBGetUserData

Function name `uint32_t HAL_FLASHEx_OBGetUserData (uint32_t DATAAddress)`

Function description Get the Option byte user data.

- Parameters**
- **DATAAddress:** Address of the option byte DATA This parameter can be one of the following values:
 - `OB_DATA_ADDRESS_DATA0`
 - `OB_DATA_ADDRESS_DATA1`

- Return values**
- **Value:** programmed in USER data

20.3 FLASHEx Firmware driver defines

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

20.3.1 FLASHEx

`FLASHEx`

`FLASHEx Option Byte BOOT0`

`OB_BOOT0_RESET` BOOT0 Reset

`OB_BOOT0_SET` BOOT0 Set

`Option Byte BOOT1`

`OB_BOOT1_RESET` BOOT1 Reset

`OB_BOOT1_SET` BOOT1 Set

`FLASHEx Option Byte BOOT SEL`

`OB_BOOT_SEL_RESET` BOOT_SEL Reset

`OB_BOOT_SEL_SET` BOOT_SEL Set

`Option Byte Data Address`

`OB_DATA_ADDRESS_DAT
A0`

`OB_DATA_ADDRESS_DAT
A1`

`Option Byte IWatchdog`

`OB_IWDG_SW` Software IWDG selected

OB_IWDG_HW Hardware IWDG selected

Option Byte nRST STDBY

OB_STDBY_NO_RST No reset generated when entering in STANDBY

OB_STDBY_RST Reset generated when entering in STANDBY

Option Byte nRST STOP

OB_STOP_NO_RST No reset generated when entering in STOP

OB_STOP_RST Reset generated when entering in STOP

Option Byte SRAM Parity Check Enable

OB_SRAM_PARITY_SET SRAM parity check enable set

OB_SRAM_PARITY_RESET SRAM parity check enable reset

Option Byte Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

OB_RDP_LEVEL_2 Warning: When enabling read protection level 2 it's no more possible to go back to level 1 or 0

Option Bytes Type

OPTIONBYTE_WRP WRP option byte configuration

OPTIONBYTE_RDP RDP option byte configuration

OPTIONBYTE_USER USER option byte configuration

OPTIONBYTE_DATA DATA option byte configuration

Option Byte VDDA Analog Monitoring

OB_VDDA_ANALOG_ON Analog monitoring on VDDA Power source ON

OB_VDDA_ANALOG_OFF Analog monitoring on VDDA Power source OFF

FLASHEx OB Write Protection

OB_WRP_PAGES0TO1

OB_WRP_PAGES2TO3

OB_WRP_PAGES4TO5

OB_WRP_PAGES6TO7

OB_WRP_PAGES8TO9

OB_WRP_PAGES10TO11

OB_WRP_PAGES12TO13

OB_WRP_PAGES14TO15

OB_WRP_PAGES16TO17

OB_WRP_PAGES18TO19

OB_WRP_PAGES20TO21

OB_WRP_PAGES22TO23

OB_WRP_PAGES24TO25

OB_WRP_PAGES26TO27

OB_WRP_PAGES28TO29

OB_WRP_PAGES30TO31

OB_WRP_PAGES32TO33

OB_WRP_PAGES34TO35

OB_WRP_PAGES36TO37

OB_WRP_PAGES38TO39

OB_WRP_PAGES40TO41

OB_WRP_PAGES42TO43

OB_WRP_PAGES44TO45

OB_WRP_PAGES46TO47

OB_WRP_PAGES48TO49

OB_WRP_PAGES50TO51

OB_WRP_PAGES52TO53

OB_WRP_PAGES54TO55

OB_WRP_PAGES56TO57

OB_WRP_PAGES58TO59

OB_WRP_PAGES60TO61

OB_WRP_PAGES62TO127

**OB_WRP_PAGES0TO15MA
SK**

**OB_WRP_PAGES16TO31M
ASK**

**OB_WRP_PAGES32TO47M
ASK**

**OB_WRP_PAGES48TO127
MASK**

OB_WRP_ALLPAGES Write protection of all pages

Option Byte WRP State

OB_WRPSTATE_DISABLE Disable the write protection of the desired pages

OB_WRPSTATE_ENABLE Enable the write protection of the desired pages

FLASHEx Page Size

FLASH_PAGE_SIZE

FLASH Type Erase

FLASH_TYPEERASE_PAG Pages erase only
ES

FLASH_TYPEERASE_MAS Flash mass erase activation
SERASE

21 HAL GPIO Generic Driver

21.1 GPIO Firmware driver registers structures

21.1.1 **GPIO_InitTypeDef**

GPIO_InitTypeDef is defined in the `stm32f0xx_hal_gpio.h`

Data Fields

- `uint32_t Pin`
- `uint32_t Mode`
- `uint32_t Pull`
- `uint32_t Speed`
- `uint32_t Alternate`

Field Documentation

- `uint32_t GPIO_InitTypeDef::Pin`

Specifies the GPIO pins to be configured. This parameter can be any value of [`GPIO_pins`](#)

- `uint32_t GPIO_InitTypeDef::Mode`

Specifies the operating mode for the selected pins. This parameter can be a value of [`GPIO_mode`](#)

- `uint32_t GPIO_InitTypeDef::Pull`

Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of [`GPIO_pull`](#)

- `uint32_t GPIO_InitTypeDef::Speed`

Specifies the speed for the selected pins. This parameter can be a value of [`GPIO_speed`](#)

- `uint32_t GPIO_InitTypeDef::Alternate`

Peripheral to be connected to the selected pins This parameter can be a value of [`GPIOEx_Alternate_function_selection`](#)

21.2 GPIO Firmware driver API description

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

21.2.1 **GPIO Peripheral features**

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
 - Input mode
 - Analog mode
 - Output mode
 - Alternate function mode
 - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.
- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

- The external interrupt/event controller consists of up to 28 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

21.2.2 How to use this driver

- Enable the GPIO AHB clock using the following function : `__HAL_RCC_GPIOx_CLK_ENABLE()`.
- 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).
- 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()`.
- `HAL_GPIO_DelInit` allows to set register values to their reset value. It's also recommended to use it to unconfigure pin which was used as an external interrupt or in event mode. That's the only way to reset corresponding bit in EXTI & SYSCFG registers.
- To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
- To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin()`/`HAL_GPIO_TogglePin()`.
- To lock pin configuration until next reset use `HAL_GPIO_LockPin()`.
- During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
- The LSE oscillator pins `OSC32_IN` and `OSC32_OUT` can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
- The HSE oscillator pins `OSC_IN`/`OSC_OUT` can be used as general purpose PF0 and PF1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

21.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- `HAL_GPIO_Init`
- `HAL_GPIO_DelInit`

21.2.4 IO operation functions

This section contains the following APIs:

- `HAL_GPIO_ReadPin`
- `HAL_GPIO_WritePin`
- `HAL_GPIO_TogglePin`
- `HAL_GPIO_LockPin`
- `HAL_GPIO_EXTI_IRQHandler`
- `HAL_GPIO_EXTI_Callback`

21.2.5 Detailed description of functions

`HAL_GPIO_Init`

Function name `void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)`

Function description Initialize the GPIOx peripheral according to the specified parameters in the `GPIO_InitStruct`.

Parameters	<ul style="list-style-type: none">• GPIOx: where x can be (A..F) to select the GPIO peripheral for STM32F0 family• GPIO_Init: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• GPIOx: where x can be (A..F) to select the GPIO peripheral for STM32F0 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• GPIOx: where x can be (A..F) to select the GPIO peripheral for STM32F0 family• GPIO_Pin: specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• GPIOx: where x can be (A..H) to select the GPIO peripheral for STM32F0 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:<ul style="list-style-type: none">– GPIO_PIN_RESET: to clear the port pin– GPIO_PIN_SET: to set the port pin
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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_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..F) to select the GPIO peripheral for STM32F0 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 Locks GPIO Pins configuration registers.

Parameters

- **GPIOx:** where x can be (A..F) to select the GPIO peripheral for STM32F0 family
- **GPIO_Pin:** specifies the port bits to be locked. This parameter can be any combination of `GPIO_Pin_x` where x can be (0..15).

Return values

- **None:**

Notes

- The locked registers are `GPIOx_MODER`, `GPIOx_OTYPER`, `GPIOx_OSPEEDR`, `GPIOx_PUPDR`, `GPIOx_AFRL` and `GPIOx_AFRH`.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name `void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)`

Function description Handle EXTI interrupt request.

Parameters

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

Return values

- **None:**

HAL_GPIO_EXTI_Callback

Function name `void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)`

Function description EXTI line detection callback.

Parameters

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

Return values

- **None:**

21.3 GPIO Firmware driver defines

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

21.3.1 GPIO

GPIO

GPIO Exported Macros

[__HAL_GPIO_EXTI_GET_F](#) **Description:**

LAG

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

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

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

[__HAL_GPIO_EXTI_CLEAR](#) **Description:**

FLAG

- Clear the EXTI's line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

Return value:

- None

[__HAL_GPIO_EXTI_GET_IT](#) **Description:**

IT

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

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

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

[__HAL_GPIO_EXTI_CLEAR](#) **Description:**

IT

- Clear the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

Return value:

- None

[__HAL_GPIO_EXTI_GENERATE_SWIT](#) **Description:**

ATE_SWIT

- Generate a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

- None

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_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 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` range up to 2 MHz, please refer to the product datasheet

`GPIO_SPEED_FREQ_MEDI` range 4 MHz to 10 MHz, please refer to the product datasheet
UM

`GPIO_SPEED_FREQ_HIGH` range 10 MHz to 50 MHz, please refer to the product datasheet

22 HAL GPIO Extension Driver

22.1 GPIOEx Firmware driver defines

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

22.1.1 GPIOEx

GPIOEx

GPIOEx Alternate function selection

<code>GPIO_AF0_EVENTOUT</code>	AF0: EVENTOUT Alternate Function mapping
<code>GPIO_AF0_SWDIO</code>	AF0: SWDIO Alternate Function mapping
<code>GPIO_AF0_SWCLK</code>	AF0: SWCLK Alternate Function mapping
<code>GPIO_AF0_MCO</code>	AF0: MCO Alternate Function mapping
<code>GPIO_AF0_CEC</code>	AF0: CEC Alternate Function mapping
<code>GPIO_AF0_CRS</code>	AF0: CRS Alternate Function mapping
<code>GPIO_AF0_IR</code>	AF0: IR Alternate Function mapping
<code>GPIO_AF0_SPI1</code>	AF0: SPI1/I2S1 Alternate Function mapping
<code>GPIO_AF0_SPI2</code>	AF0: SPI2/I2S2 Alternate Function mapping
<code>GPIO_AF0_TIM1</code>	AF0: TIM1 Alternate Function mapping
<code>GPIO_AF0_TIM3</code>	AF0: TIM3 Alternate Function mapping
<code>GPIO_AF0_TIM14</code>	AF0: TIM14 Alternate Function mapping
<code>GPIO_AF0_TIM15</code>	AF0: TIM15 Alternate Function mapping
<code>GPIO_AF0_TIM16</code>	AF0: TIM16 Alternate Function mapping
<code>GPIO_AF0_TIM17</code>	AF0: TIM17 Alternate Function mapping
<code>GPIO_AF0_TSC</code>	AF0: TSC Alternate Function mapping
<code>GPIO_AF0_USART1</code>	AF0: USART1 Alternate Function mapping
<code>GPIO_AF0_USART2</code>	AF0: USART2 Alternate Function mapping
<code>GPIO_AF0_USART3</code>	AF0: USART3 Alternate Function mapping
<code>GPIO_AF0_USART4</code>	AF0: USART4 Alternate Function mapping

GPIO_AF0_USART8	AF0: USART8 Alternate Function mapping
GPIO_AF0_CAN	AF0: CAN Alternate Function mapping
GPIO_AF1_TIM3	AF1: TIM3 Alternate Function mapping
GPIO_AF1_TIM15	AF1: TIM15 Alternate Function mapping
GPIO_AF1_USART1	AF1: USART1 Alternate Function mapping
GPIO_AF1_USART2	AF1: USART2 Alternate Function mapping
GPIO_AF1_USART3	AF1: USART3 Alternate Function mapping
GPIO_AF1_USART4	AF1: USART4 Alternate Function mapping
GPIO_AF1_USART5	AF1: USART5 Alternate Function mapping
GPIO_AF1_USART6	AF1: USART6 Alternate Function mapping
GPIO_AF1_USART7	AF1: USART7 Alternate Function mapping
GPIO_AF1_USART8	AF1: USART8 Alternate Function mapping
GPIO_AF1_IR	AF1: IR Alternate Function mapping
GPIO_AF1_CEC	AF1: CEC Alternate Function mapping
GPIO_AF1_EVENTOUT	AF1: EVENTOUT Alternate Function mapping
GPIO_AF1_I2C1	AF1: I2C1 Alternate Function mapping
GPIO_AF1_I2C2	AF1: I2C2 Alternate Function mapping
GPIO_AF1_TSC	AF1: TSC Alternate Function mapping
GPIO_AF1_SPI1	AF1: SPI1 Alternate Function mapping
GPIO_AF1_SPI2	AF1: SPI2 Alternate Function mapping
GPIO_AF2_TIM1	AF2: TIM1 Alternate Function mapping
GPIO_AF2_TIM2	AF2: TIM2 Alternate Function mapping
GPIO_AF2_TIM16	AF2: TIM16 Alternate Function mapping
GPIO_AF2_TIM17	AF2: TIM17 Alternate Function mapping
GPIO_AF2_EVENTOUT	AF2: EVENTOUT Alternate Function mapping

GPIO_AF2_USART5	AF2: USART5 Alternate Function mapping
GPIO_AF2_USART6	AF2: USART6 Alternate Function mapping
GPIO_AF2_USART7	AF2: USART7 Alternate Function mapping
GPIO_AF2_USART8	AF2: USART8 Alternate Function mapping
GPIO_AF3_EVENTOUT	AF3: EVENTOUT Alternate Function mapping
GPIO_AF3_TSC	AF3: TSC Alternate Function mapping
GPIO_AF3_TIM15	AF3: TIM15 Alternate Function mapping
GPIO_AF3_I2C1	AF3: I2C1 Alternate Function mapping
GPIO_AF4_TIM14	AF4: TIM14 Alternate Function mapping
GPIO_AF4_USART4	AF4: USART4 Alternate Function mapping
GPIO_AF4_USART3	AF4: USART3 Alternate Function mapping
GPIO_AF4_CRS	AF4: CRS Alternate Function mapping
GPIO_AF4_CAN	AF4: CAN Alternate Function mapping
GPIO_AF4_I2C1	AF4: I2C1 Alternate Function mapping
GPIO_AF4_USART5	AF4: USART5 Alternate Function mapping
GPIO_AF5_TIM15	AF5: TIM15 Alternate Function mapping
GPIO_AF5_TIM16	AF5: TIM16 Alternate Function mapping
GPIO_AF5_TIM17	AF5: TIM17 Alternate Function mapping
GPIO_AF5_SPI2	AF5: SPI2 Alternate Function mapping
GPIO_AF5_I2C2	AF5: I2C2 Alternate Function mapping
GPIO_AF5_MCO	AF5: MCO Alternate Function mapping
GPIO_AF5_USART6	AF5: USART6 Alternate Function mapping
GPIO_AF6_EVENTOUT	AF6: EVENTOUT Alternate Function mapping
GPIO_AF7_COMP1	AF7: COMP1 Alternate Function mapping
GPIO_AF7_COMP2	AF7: COMP2 Alternate Function mapping

IS_GPIO_AF

GPIOEx_Get Port Index

GPIO_GET_INDEX

23 HAL I2C Generic Driver

23.1 I2C Firmware driver registers structures

23.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the `stm32f0xx_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*

23.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the `stm32f0xx_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`

Field Documentation

- `I2C_HandleTypeDef* __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

23.2 I2C Firmware driver API description

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

23.2.1 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 user can add his 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 user can add his 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 user can add his 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 user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his 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 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 @ref I2C_XFEROPTIONS and are listed below:
 - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functionnal 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_SequENTIAL_Transmit_IT()` then `HAL_I2C_Master_SequENTIAL_Transmit_IT()`)
 - 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 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 with a final stop condition in both cases

- Differents sequential I2C interfaces are listed below:
 - Sequential transmit in master I2C mode an amount of data in non-blocking mode using `HAL_I2C_Master_Sequential_Transmit_IT()`
 - At transmission end of current frame transfer, `HAL_I2C_MasterTxCpltCallback()` is executed and user can add his 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_Sequential_Receive_IT()`
 - At reception end of current frame transfer, `HAL_I2C_MasterRxCpltCallback()` is executed and user can add his own code by customization of function pointer `HAL_I2C_MasterRxCpltCallback()`
 - Abort a master I2C process communication with Interrupt using `HAL_I2C_Master_Abort_IT()`
 - End of abort process, `HAL_I2C_AbortCpltCallback()` is executed and user can add his 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 user can add his 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 user can add his 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_Sequential_Transmit_IT()`
 - At transmission end of current frame transfer, `HAL_I2C_SlaveTxCpltCallback()` is executed and user can add his 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_Sequential_Receive_IT()`
 - At reception end of current frame transfer, `HAL_I2C_SlaveRxCpltCallback()` is executed and user can add his own code by customization of function pointer `HAL_I2C_SlaveRxCpltCallback()`
 - In case of transfer Error, `HAL_I2C_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_I2C_ErrorCallback()`
 - Abort a master I2C process communication with Interrupt using `HAL_I2C_Master_Abort_IT()`
 - End of abort process, `HAL_I2C_AbortCpltCallback()` is executed and user can add his 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 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 user can add his 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 user can add his own code by customization of function pointer `HAL_I2C_MemRxCpltCallback()`
- In case of transfer Error, `HAL_I2C_ErrorCallback()` function is executed and user can add his 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 user can add his 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 user can add his 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 user can add his 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 user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his 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 user can add his 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 user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his 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

Note:

You can refer to the I2C HAL driver header file for more useful macros

23.2.2 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**](#)

23.2.3 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()
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()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:

- HAL_I2C_MemTxCpltCallback()
- HAL_I2C_MemRxCpltCallback()
- HAL_I2C_MasterTxCpltCallback()
- HAL_I2C_MasterRxCpltCallback()
- HAL_I2C_SlaveTxCpltCallback()
- HAL_I2C_SlaveRxCpltCallback()
- HAL_I2C_ErrorCallback()

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_Sequential_Transmit_IT**](#)
- [**HAL_I2C_Master_Sequential_Receive_IT**](#)
- [**HAL_I2C_Slave_Sequential_Transmit_IT**](#)
- [**HAL_I2C_Slave_Sequential_Receive_IT**](#)
- [**HAL_I2C_EnableListen_IT**](#)
- [**HAL_I2C_DisableListen_IT**](#)
- [**HAL_I2C_Master_Abort_IT**](#)

23.2.4

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**](#)

23.2.5

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_DelInit

Function name **HAL_StatusTypeDef HAL_I2C_DelInit (I2C_HandleTypeDef * hi2c)**

Function description Delinitialize 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_MspDelInit

Function name **void HAL_I2C_MspDelInit (I2C_HandleTypeDef * hi2c)**

Function description Delinitialize 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_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	<ul style="list-style-type: none">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 shift at right before call interfacepData: Pointer to data bufferSize: Amount of data to be sentTimeout: Timeout duration
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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 shift at right before call interfacepData: Pointer to data bufferSize: Amount of data to be sentTimeout: Timeout duration
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.pData: Pointer to data bufferSize: Amount of data to be sentTimeout: Timeout duration
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.pData: Pointer to data bufferSize: Amount of data to be sentTimeout: Timeout duration
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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 shift at right before call interfaceMemAddress: Internal memory addressMemAddSize: Size of internal memory addresspData: Pointer to data bufferSize: Amount of data to be sentTimeout: Timeout duration
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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 shift at right before call interfaceMemAddress: Internal memory addressMemAddSize: Size of internal memory addresspData: Pointer to data bufferSize: Amount of data to be sentTimeout: Timeout duration
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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 shift at right before call interfaceTrials: Number of trialsTimeout: Timeout duration
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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 shift at right before call interfacepData: Pointer to data bufferSize: Amount of data to be sent
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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 shift at right before call interfacepData: Pointer to data bufferSize: Amount of data to be sent
Return values	<ul style="list-style-type: none">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 shift at right before call 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	<ul style="list-style-type: none">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 shift at right before call interfaceMemAddress: Internal memory addressMemAddSize: Size of internal memory addresspData: Pointer to data bufferSize: Amount of data to be sent
Return values	<ul style="list-style-type: none">HAL: status

HAL_I2C_Master_SequENTIAL_Transmit_IT

Function name	HAL_StatusTypeDef HAL_I2C_Master_Sequential_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	<ul style="list-style-type: none">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 shift at right before call interfacepData: Pointer to data bufferSize: Amount of data to be sentXferOptions: Options of Transfer, value of I2C Sequential Transfer Options
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_SequENTIAL_Receive_IT

Function name	HAL_StatusTypeDef HAL_I2C_Master_Sequential_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	<ul style="list-style-type: none">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 shift at right before call interfacepData: Pointer to data bufferSize: Amount of data to be sentXferOptions: Options of Transfer, value of I2C Sequential Transfer Options
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Transmit_IT

Function name	HAL_StatusTypeDef HAL_I2C_Slave_Sequential_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	<ul style="list-style-type: none">hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.pData: Pointer to data bufferSize: Amount of data to be sentXferOptions: Options of Transfer, value of I2C Sequential Transfer Options
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Receive_IT

Function name	HAL_StatusTypeDef HAL_I2C_Slave_Sequential_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	<ul style="list-style-type: none">hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.pData: Pointer to data bufferSize: Amount of data to be sentXferOptions: Options of Transfer, value of I2C Sequential Transfer Options
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">• hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
Return values	<ul style="list-style-type: none">• 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 I2C IT or DMA process communication with Interrupt.
Parameters	<ul style="list-style-type: none">• 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 shift at right before call interface
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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 shift at right before call interface• pData: Pointer to data buffer• Size: Amount of data to be sent
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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 shift at right before call interface• pData: Pointer to data buffer• Size: Amount of data to be sent
Return values	<ul style="list-style-type: none">• 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 shift at right before call 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 shift at right before call 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_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 (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 (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 (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

23.3 I2C Firmware driver defines

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

23.3.1 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

I2C Exported Macros

_HAL_I2C_RESET_HANDLE_STATE	Description: <ul style="list-style-type: none">Reset I2C handle state. Parameters: <ul style="list-style-type: none">_HANDLE_: specifies the I2C Handle. Return value: <ul style="list-style-type: none">None
_HAL_I2C_ENABLE_IT	Description: <ul style="list-style-type: none">Enable the specified I2C interrupt. Parameters: <ul style="list-style-type: none">_HANDLE_: specifies the I2C Handle._INTERRUPT_: specifies the interrupt source to enable. This parameter can be one of the following values:<ul style="list-style-type: none">I2C_IT_ERRI Errors interrupt enableI2C_IT_TCI Transfer complete interrupt enableI2C_IT_STOPI STOP detection interrupt enableI2C_IT_NACKI NACK received interrupt enableI2C_IT_ADDRI Address match interrupt enableI2C_IT_RXI RX interrupt enableI2C_IT_TXI TX interrupt enable Return value: <ul style="list-style-type: none">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_SOUR

CE

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).

__HAL_I2C_GET_FLAG

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_RXIS 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_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_N**ACK****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_DISA
BLE

I2C_GENERALCALL_ENAB
LE

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_R
EAD

I2C_GENERATE_START_W
RITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMI
T

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FR
AME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FR
AME

I2C_LAST_FRAME

24 HAL I2C Extension Driver

24.1 I2CEEx Firmware driver API description

The following section lists the various functions of the I2CEEx library.

24.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32F0xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode

24.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function `HAL_I2CEEx_ConfigAnalogFilter()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEEx_ConfigDigitalFilter()`
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
 - `HAL_I2CEEx_EnableWakeUp()`
 - `HAL_I2CEEx_DisableWakeUp()`
4. Configure the enable or disable of fast mode plus driving capability using the functions :
 - `HAL_I2CEEx_EnableFastModePlus()`
 - `HAL_I2CEEx_DisableFastModePlus()`

24.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters
- Configure Wake Up Feature

This section contains the following APIs:

- `HAL_I2CEEx_ConfigAnalogFilter`
- `HAL_I2CEEx_ConfigDigitalFilter`
- `HAL_I2CEEx_EnableWakeUp`
- `HAL_I2CEEx_DisableWakeUp`
- `HAL_I2CEEx_EnableFastModePlus`
- `HAL_I2CEEx_DisableFastModePlus`

24.1.4 Detailed description of functions

`HAL_I2CEEx_ConfigAnalogFilter`

Function name	<code>HAL_StatusTypeDef HAL_I2CEEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)</code>
Function description	Configure I2C Analog noise filter.
Parameters	<ul style="list-style-type: none">• hi2c: Pointer to a <code>I2C_HandleTypeDef</code> structure that contains the configuration information for the specified I2Cx peripheral.• AnalogFilter: New state of the Analog filter.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_I2CEEx_ConfigDigitalFilter

Function name	HAL_StatusTypeDef HAL_I2CEEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)
Function description	Configure I2C Digital noise filter.
Parameters	<ul style="list-style-type: none">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	<ul style="list-style-type: none">HAL: status

HAL_I2CEEx_EnableWakeUp

Function name	HAL_StatusTypeDef HAL_I2CEEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)
Function description	Enable I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none">hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	<ul style="list-style-type: none">HAL: status

HAL_I2CEEx_DisableWakeUp

Function name	HAL_StatusTypeDef HAL_I2CEEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)
Function description	Disable I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none">hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	<ul style="list-style-type: none">HAL: status

HAL_I2CEEx_EnableFastModePlus

Function name	void HAL_I2CEEx_EnableFastModePlus (uint32_t ConfigFastModePlus)
Function description	Enable the I2C fast mode plus driving capability.
Parameters	<ul style="list-style-type: none">ConfigFastModePlus: Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values
Return values	<ul style="list-style-type: none">None:

Notes

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

HAL_I2CEx_DisableFastModePlus

Function name `void HAL_I2CEx_DisableFastModePlus (uint32_t ConfigFastModePlus)`

Function description Disable the I2C fast mode plus driving capability.

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

Return values • **None:**

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

24.2 I2CEx Firmware driver defines

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

24.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENA
BLE

I2C_ANALOGFILTER_DISA
BLE

I2C Extended Fast Mode Plus

I2C_FMP_NOT_SUPPORT Fast Mode Plus not supported
D

I2C_FASTMODEPLUS_PA9 Enable Fast Mode Plus on PA9

I2C_FASTMODEPLUS_PA10 Enable Fast Mode Plus on PA10
0

I2C_FASTMODEPLUS_PB6 Enable Fast Mode Plus on PB6

I2C_FASTMODEPLUS_PB7 Enable Fast Mode Plus on PB7

I2C_FASTMODEPLUS_PB8 Enable Fast Mode Plus on PB8

I2C_FASTMODEPLUS_PB9 Enable Fast Mode Plus on PB9

I2C_FASTMODEPLUS_I2C1 Enable Fast Mode Plus on I2C1 pins

I2C_FASTMODEPLUS_I2C2 Enable Fast Mode Plus on I2C2 pins

25 HAL I2S Generic Driver

25.1 I2S Firmware driver registers structures

25.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the `stm32f0xx_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`](#)

25.1.2 I2S_HandleTypeDef

I2S_HandleTypeDef is defined in the `stm32f0xx_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](#)

25.2 I2S Firmware driver API description

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

25.2.1 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 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 I2S 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 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 stm32f0xx_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()

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

Note:

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

25.2.2 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_DelInit() to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- [`HAL_I2S_Init`](#)
- [`HAL_I2S_DelInit`](#)
- [`HAL_I2S_MspInit`](#)
- [`HAL_I2S_MspDelInit`](#)

25.2.3 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**](#)

25.2.4 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**](#)

25.2.5 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_DelInit

Function name	HAL_StatusTypeDef HAL_I2S_DelInit (I2S_HandleTypeDef * hi2s)
Function description	Deinitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none">hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">HAL: status

HAL_I2S_MspInit

Function name	void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)
Function description	I2S MSP Init.
Parameters	<ul style="list-style-type: none">hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">None:

HAL_I2S_MspDelInit

Function name	void HAL_I2S_MspDelInit (I2S_HandleTypeDef * hi2s)
Function description	I2S MSP DelInit.
Parameters	<ul style="list-style-type: none">hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S modulepData: a 16-bit pointer to data buffer.Size: number of data sample to be sent:Timeout: Timeout duration
Return values	<ul style="list-style-type: none">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 16-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 16-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 continouse 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 16-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	<code>HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</code>
Function description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none">hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S modulepData: a 16-bit pointer to the Receive data buffer.Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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 16-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-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.

HAL_I2S_IRQHandler

Function name	<code>void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)</code>
Function description	This function handles I2S interrupt request.
Parameters	<ul style="list-style-type: none">hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">None:

HAL_I2S_Transmit_DMA

Function name	<code>HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</code>
Function description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none">hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S modulepData: a 16-bit pointer to the Transmit data buffer.Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none">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 16-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 16-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 stream 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 stream 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	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• HAL: status
HAL_I2S_TxHalfCpltCallback	
Function name	void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function description	Tx Transfer Half completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• None:
HAL_I2S_TxCpltCallback	
Function name	void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)
Function description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• None:
HAL_I2S_RxHalfCpltCallback	
Function name	void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function description	Rx Transfer half completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• None:
HAL_I2S_RxCpltCallback	
Function name	void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)
Function description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• None:

HAL_I2S_ErrorCallback

Function name	void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)
Function description	I2S error callbacks.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• None:
HAL_I2S_GetState	

Function name	HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)
Function description	Return the I2S state.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• HAL: state
HAL_I2S_GetError	

25.3 I2S Firmware driver defines

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

25.3.1	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

IS_I2S_AUDIO_FREQ

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

IS_I2S_CPOL

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

IS_I2S_DATA_FORMAT

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_UNKNOWN Unknow Error error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: I2S handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

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

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

__HAL_I2S_ENABLE_IT

Description:

- Enable or 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_DISABLE_IT

__HAL_I2S_GET_IT_SOURCE

CE

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_OVRF**Description:**

- Clears the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_UDRF**Description:**

- Clears the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

I2S Flag definition**I2S_FLAG_TXE****I2S_FLAG_RXNE****I2S_FLAG_UDR****I2S_FLAG_OVR****I2S_FLAG_FRE****I2S_FLAG_CHSIDE****I2S_FLAG_BSY*****I2S Interrupt configuration definition*****I2S_IT_TXE**

I2S_IT_RXNE

I2S_IT_ERR

I2S MCLK Output

I2S_MCLKOUTPUT_ENABL
E

I2S_MCLKOUTPUT_DISAB
LE

IS_I2S_MCLK_OUTPUT

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

IS_I2S_MODE

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHO
RT

I2S_STANDARD_PCM_LON
G

IS_I2S_STANDARD

26 HAL IRDA Generic Driver

26.1 IRDA Firmware driver registers structures

26.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the `stm32f0xx_hal_irda.h`

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint8_t Prescaler*
- *uint16_t PowerMode*

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 = ((PCLKx) / ((hirda->Init.BaudRate)))

- *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*

26.1.2 IRDA_HandleTypeDef

IRDA_HandleTypeDef is defined in the `stm32f0xx_hal_irda.h`

Data Fields

- *USART_TypeDef * Instance*
- *IRDA_InitTypeDef Init*
- *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`

Field Documentation

- `USART_TypeDef* IRDA_HandleTypeDef::Instance`
IRDA registers base address
- `IRDA_InitTypeDef IRDA_HandleTypeDef::Init`
IRDA communication parameters
- `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`
IRDA 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 This parameter can be a value of `IRDA_Error`

26.2 IRDA Firmware driver API description

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

26.2.1 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

26.2.2

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
 - 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_DelInit`
- `HAL_IRDA_MspInit`
- `HAL_IRDA_MspDelInit`

26.2.3

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 mode 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*](#)

26.2.4 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*](#)

26.2.5 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_MsplInit

Function name

void HAL_IRDA_MsplInit (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_MspDelInit

Function name

void HAL_IRDA_MspDelInit (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_Transmit

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, 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.
- **Size:** Amount of data 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), 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	<ul style="list-style-type: none">hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.pData: Pointer to data buffer.Size: Amount of data to be received.Timeout: Specify timeout value.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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_IRDA_Transmit_IT

Function name	HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none">hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.pData: Pointer to data buffer.Size: Amount of data to be sent.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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_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	<ul style="list-style-type: none">hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.pData: Pointer to data buffer.Size: Amount of data to be received.
Return values	<ul style="list-style-type: none">HAL: status

Notes

- 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_IRDA_Transmit_DMA

Function name	HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none">hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.pData: pointer to data buffer.Size: amount of data to be sent.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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_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	<ul style="list-style-type: none">hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.pData: Pointer to data buffer.Size: Amount of data to be received.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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_IRDA_DMAPause

Function name	HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)
Function description	Pause the DMA Transfer.
Parameters	<ul style="list-style-type: none">hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values	<ul style="list-style-type: none">• HAL: status
HAL_IRDA_DMAResume	
Function name	HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)
Function description Resume the DMA Transfer.	
Parameters	<ul style="list-style-type: none">• hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• HAL: status
HAL_IRDA_DMAStop	
Function name	HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)
Function description Stop the DMA Transfer.	
Parameters	<ul style="list-style-type: none">• hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• HAL: status
HAL_IRDA_Abort	
Function name	HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)
Function description Abort ongoing transfers (blocking mode).	
Parameters	<ul style="list-style-type: none">• hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none">• None:

HAL_IRDA_TxCpltCallback

Function name	void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)
Function description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none">• 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 (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 (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

26.3 IRDA Firmware driver defines

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

26.3.1 IRDA

IRDA

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

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

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE **Description:**

- DLE_STATE • 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_FLA Description:

G

- 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_PEF Description:

LAG

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEF Description:

LAG

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_NEF Description:

LAG

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_OREF Description:

FLAG

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_IDLE_FLAG **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_RXACK 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.
- __IT__: 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__ (TRUE or FALSE).

__HAL_IRDA_GET_IT_SOU **Description:**

RCE

- Check whether the specified IRDA interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __IT__: 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__ (TRUE or FALSE).

__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_SA_MPLE_ENABLE**Description:**

- Enable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SA_MPLE_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_RXACK	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 Framing error
IRDA_FLAG_PE	IRDA Parity error

IRDA interruptions flags mask

IRDA_IT_MASK	IRDA Interruptions flags mask
---------------------	-------------------------------

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 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_NOR IRDA normal power mode
MAL

IRDA_POWERMODE_LOW IRDA low power mode
POWER

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
ST

IRDA_RXDATA_FLUSH_REQ Receive Data flush Request
QUEST

IRDA_TXDATA_FLUSH_REQ Transmit data flush Request
QUEST

IRDA State

IRDA_STATE_DISABLE IRDA disabled

IRDA_STATE_ENABLE IRDA enabled

IRDA Transfer Mode

IRDA_MODE_RX RX mode

IRDA_MODE_TX TX mode

IRDA_MODE_TX_RX RX and TX mode

27 HAL IRDA Extension Driver

27.1 IRDAEx Firmware driver defines

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

27.1.1 IRDAEx

IRDAEx

IRDA Word Length

IRDA_WORDLENGTH_7B 7-bit long frame

IRDA_WORDLENGTH_8B 8-bit long frame

IRDA_WORDLENGTH_9B 9-bit long frame

28 HAL IWDG Generic Driver

28.1 IWDG Firmware driver registers structures

28.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the `stm32f0xx_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 = 0xFFFF

- *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 = 0xFFFF

28.1.2 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the `stm32f0xx_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

28.2 IWDG Firmware driver API description

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

28.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by Low-Speed clock (LSI) and thus stays active even if the main clock fails.
- Once the IWDG is started, the LSI is forced ON and both can not be disabled. The counter starts counting down from the reset value (0xFFFF). 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 in 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 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 @40KHz (LSI): ~0.1ms / ~26.2s The IWDG timeout may vary due to LSI frequency dispersion. STM32F0xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

28.2.2 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 downcounting.
 - Enable write access to configuration register: IWDG_PR, IWDG_RLR & 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.
 - wait for status flags to be reset"
 - 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.
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

28.2.3 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](#)

28.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- [HAL_IWDG_Refresh](#)

28.2.5 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	<ul style="list-style-type: none">• HAL: status
HAL_IWDG_Refresh	
Function name	HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)
Function description	Refresh the IWDG.
Parameters	<ul style="list-style-type: none">• hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none">• HAL: status

28.3 IWDG Firmware driver defines

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

28.3.1 IWDG

IWDG

IWDG Exported Macros

_HAL_IWDG_START	Description: <ul style="list-style-type: none">• Enable the IWDG peripheral. Parameters: <ul style="list-style-type: none">• _HANDLE_: IWDG handle Return value: <ul style="list-style-type: none">• None
_HAL_IWDG_RELOAD_C OUNTER	Description: <ul style="list-style-type: none">• Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR & IWDG_WINR registers disabled). Parameters: <ul style="list-style-type: none">• _HANDLE_: IWDG handle Return value: <ul style="list-style-type: none">• None
<i>IWDG Prescaler</i>	
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

29 HAL PCD Generic Driver

29.1 PCD Firmware driver registers structures

29.1.1 PCD_InitTypeDef

PCD_InitTypeDef is defined in the `stm32f0xx_hal_pcd.h`

Data Fields

- *uint32_t dev_endpoints*
- *uint32_t speed*
- *uint32_t ep0_mps*
- *uint32_t phy_iface*
- *uint32_t Sof_enable*
- *uint32_t low_power_enable*
- *uint32_t lpm_enable*
- *uint32_t battery_charging_enable*

Field Documentation

- *uint32_t PCD_InitTypeDef::dev_endpoints*

Device Endpoints number. This parameter depends on the used USB core. This parameter must be a number between Min_Data = 1 and Max_Data = 15

- *uint32_t PCD_InitTypeDef::speed*

USB Core speed. This parameter can be any value of [PCD_Core_Speed](#)

- *uint32_t PCD_InitTypeDef::ep0_mps*

Set the Endpoint 0 Max Packet size. This parameter can be any value of [PCD_EP0_MPS](#)

- *uint32_t PCD_InitTypeDef::phy_iface*

Select the used PHY interface. This parameter can be any value of [PCD_Core_PHY](#)

- *uint32_t PCD_InitTypeDef::Sof_enable*

Enable or disable the output of the SOF signal. This parameter can be set to ENABLE or DISABLE

- *uint32_t PCD_InitTypeDef::low_power_enable*

Enable or disable Low Power mode. This parameter can be set to ENABLE or DISABLE

- *uint32_t PCD_InitTypeDef::lpm_enable*

Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE

- *uint32_t PCD_InitTypeDef::battery_charging_enable*

Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE

29.1.2 PCD_EPTTypeDef

PCD_EPTTypeDef is defined in the `stm32f0xx_hal_pcd.h`

Data Fields

- *uint8_t num*
- *uint8_t is_in*
- *uint8_t is_stall*
- *uint8_t type*
- *uint16_t pmaaddress*
- *uint16_t pmaaddr0*
- *uint16_t pmaaddr1*
- *uint8_t doublebuffer*
- *uint32_t maxpacket*
- *uint8_t * xfer_buff*

- `uint32_t xfer_len`
- `uint32_t xfer_count`

Field Documentation

- `uint8_t PCD_EPTypeDef::num`
Endpoint number This parameter must be a number between Min_Data = 1 and Max_Data = 15
- `uint8_t PCD_EPTypeDef::is_in`
Endpoint direction This parameter must be a number between Min_Data = 0 and Max_Data = 1
- `uint8_t PCD_EPTypeDef::is_stall`
Endpoint stall condition This parameter must be a number between Min_Data = 0 and Max_Data = 1
- `uint8_t PCD_EPTypeDef::type`
Endpoint type This parameter can be any value of [PCD_EP_Type](#)
- `uint16_t PCD_EPTypeDef::pmaaddress`
PMA Address This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- `uint16_t PCD_EPTypeDef::pmaaddr0`
PMA Address0 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- `uint16_t PCD_EPTypeDef::pmaaddr1`
PMA Address1 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- `uint8_t PCD_EPTypeDef::doublebuffer`
Double buffer enable This parameter can be 0 or 1
- `uint32_t PCD_EPTypeDef::maxpacket`
Endpoint Max packet size This parameter must be a number between Min_Data = 0 and Max_Data = 64KB
- `uint8_t* PCD_EPTypeDef::xfer_buff`
Pointer to transfer buffer
- `uint32_t PCD_EPTypeDef::xfer_len`
Current transfer length
- `uint32_t PCD_EPTypeDef::xfer_count`
Partial transfer length in case of multi packet transfer

29.1.3 PCD_HandleTypeDefDef

`PCD_HandleTypeDefDef` is defined in the `stm32f0xx_hal_pcd.h`

Data Fields

- `PCD_TypeDef * Instance`
- `PCD_InitTypeDef Init`
- `_IO uint8_t USB_Address`
- `PCD_EPTypeDef IN_ep`
- `PCD_EPTypeDef OUT_ep`
- `HAL_LockTypeDef Lock`
- `_IO PCD_StateTypeDef State`
- `uint32_t Setup`
- `void * pData`

Field Documentation

- `PCD_TypeDef* PCD_HandleTypeDefDef::Instance`
Register base address
- `PCD_InitTypeDef PCD_HandleTypeDefDef::Init`
PCD required parameters
- `_IO uint8_t PCD_HandleTypeDefDef::USB_Address`
USB Address
- `PCD_EPTypeDef PCD_HandleTypeDefDef::IN_ep[8]`
IN endpoint parameters

- ***PCD_EPTTypeDef PCD_HandleTypeDef::OUT_ep[8]***
OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock***
PCD peripheral status
- ***_IO PCD_StateTypeDef PCD_HandleTypeDef::State***
PCD communication state
- ***uint32_t PCD_HandleTypeDef::Setup[12]***
Setup packet buffer
- ***void* PCD_HandleTypeDef::pData***
Pointer to upper stack Handler

29.2 PCD Firmware driver API description

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

29.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the HCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
 - a. Enable the PCD/USB Low Level interface clock using
 - __HAL_RCC_USB_CLK_ENABLE();
 - b. Initialize the related GPIO clocks
 - c. Configure PCD pin-out
 - d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
 - a. hpcd.pData = pdev;
6. Enable HCD transmission and reception:
 - a. HAL_PCD_Start();

29.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_PCD_Init***
- ***HAL_PCD_DelInit***
- ***HAL_PCD_MspInit***
- ***HAL_PCD_MspDelInit***

29.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

This section contains the following APIs:

- ***HAL_PCD_Start***
- ***HAL_PCD_Stop***
- ***HAL_PCD_IRQHandler***
- ***HAL_PCD_DataOutStageCallback***
- ***HAL_PCD_DataInStageCallback***
- ***HAL_PCD_SetupStageCallback***
- ***HAL_PCD_SOFCallback***
- ***HAL_PCD_ResetCallback***
- ***HAL_PCD_SuspendCallback***

- [*HAL_PCD_ResumeCallback*](#)
- [*HAL_PCD_ISOOUTIncompleteCallback*](#)
- [*HAL_PCD_ISOINIncompleteCallback*](#)
- [*HAL_PCD_ConnectCallback*](#)
- [*HAL_PCD_DisconnectCallback*](#)

29.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

This section contains the following APIs:

- [*HAL_PCD_DevConnect*](#)
- [*HAL_PCD_DevDisconnect*](#)
- [*HAL_PCD_SetAddress*](#)
- [*HAL_PCD_EP_Open*](#)
- [*HAL_PCD_EP_Close*](#)
- [*HAL_PCD_EP_Receive*](#)
- [*HAL_PCD_EP_GetRxCount*](#)
- [*HAL_PCD_EP_Transmit*](#)
- [*HAL_PCD_EP_SetStall*](#)
- [*HAL_PCD_EP_ClrStall*](#)
- [*HAL_PCD_EP_Flush*](#)
- [*HAL_PCD_ActivateRemoteWakeUp*](#)
- [*HAL_PCD_DeActivateRemoteWakeUp*](#)

29.2.5 Peripheral State functions

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

This section contains the following APIs:

- [*HAL_PCD_GetState*](#)

29.2.6 Detailed description of functions

[*HAL_PCD_Init*](#)

Function name [*HAL_StatusTypeDef HAL_PCD_Init \(PCD_HandleTypeDef * hpcd\)*](#)

Function description Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and create the associated handle.

Parameters • **hpcd:** PCD handle

Return values • **HAL:** status

[*HAL_PCD_DeInit*](#)

Function name [*HAL_StatusTypeDef HAL_PCD_DeInit \(PCD_HandleTypeDef * hpcd\)*](#)

Function description Deinitializes the PCD peripheral.

Parameters • **hpcd:** PCD handle

Return values • **HAL:** status

HAL_PCD_MspInit

Function name `void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)`

Function description Initializes the PCD MSP.

Parameters • `hpcd`: PCD handle

Return values • `None`:

HAL_PCD_MspDeInit

Function name `void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)`

Function description Deinitializes PCD MSP.

Parameters • `hpcd`: PCD handle

Return values • `None`:

HAL_PCD_Start

Function name `HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)`

Function description Start the USB device.

Parameters • `hpcd`: PCD handle

Return values • `HAL`: status

HAL_PCD_Stop

Function name `HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)`

Function description Stop the USB device.

Parameters • `hpcd`: PCD handle

Return values • `HAL`: status

HAL_PCD_IRQHandler

Function name `void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)`

Function description This function handles PCD interrupt request.

Parameters • `hpcd`: PCD handle

Return values • `HAL`: status

HAL_PCD_DataOutStageCallback

Function name `void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)`

Function description Data out stage callbacks.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DataInStageCallback

Function name `void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)`

Function description Data IN stage callbacks.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_SetupStageCallback

Function name `void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)`

Function description Setup stage callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SOFCallback

Function name `void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)`

Function description USB Start Of Frame callbacks.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResetCallback

Function name `void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)`

Function description USB Reset callbacks.

Parameters

- **hpcd:** PCD handle

Return values	<ul style="list-style-type: none">• None:
HAL_PCD_SuspendCallback	
Function name	void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)
Function description	Suspend event callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None:
HAL_PCD_ResumeCallback	
Function name	void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)
Function description	Resume event callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None:
HAL_PCD_ISOOUTIncompleteCallback	
Function name	void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• epnum: endpoint number
Return values	<ul style="list-style-type: none">• None:
HAL_PCD_ISOINIncompleteCallback	
Function name	void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function description	Incomplete ISO IN callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• epnum: endpoint number
Return values	<ul style="list-style-type: none">• None:
HAL_PCD_ConnectCallback	
Function name	void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)
Function description	Connection event callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle

Return values	<ul style="list-style-type: none">None:
	HAL_PCD_DisconnectCallback
Function name	void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)
Function description	Disconnection event callbacks.
Parameters	<ul style="list-style-type: none">hpcd: PCD handle
Return values	<ul style="list-style-type: none">None:
	HAL_PCD_DevConnect
Function name	HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)
Function description	Connect the USB device.
Parameters	<ul style="list-style-type: none">hpcd: PCD handle
Return values	<ul style="list-style-type: none">HAL: status
	HAL_PCD_DevDisconnect
Function name	HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)
Function description	Disconnect the USB device.
Parameters	<ul style="list-style-type: none">hpcd: PCD handle
Return values	<ul style="list-style-type: none">HAL: status
	HAL_PCD_SetAddress
Function name	HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)
Function description	Set the USB Device address.
Parameters	<ul style="list-style-type: none">hpcd: PCD handleaddress: new device address
Return values	<ul style="list-style-type: none">HAL: status
	HAL_PCD_EP_Open
Function name	HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)
Function description	Open and configure an endpoint.

Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address• ep_mps: endpoint max packet size• ep_type: endpoint type
Return values	<ul style="list-style-type: none">• HAL: status

HAL_PCD_EP_Close

Function name	HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function description	Deactivate an endpoint.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address
Return values	<ul style="list-style-type: none">• HAL: status

HAL_PCD_EP_Receive

Function name	HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)
Function description	Receive an amount of data.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address• pBuf: pointer to the reception buffer• len: amount of data to be received
Return values	<ul style="list-style-type: none">• HAL: status

HAL_PCD_EP_Transmit

Function name	HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)
Function description	Send an amount of data.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address• pBuf: pointer to the transmission buffer• len: amount of data to be sent
Return values	<ul style="list-style-type: none">• HAL: status

HAL_PCD_EP_GetRxCount

Function name	uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function description	Get Received Data Size.

Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address
-------------------	--

Return values	<ul style="list-style-type: none">• Data: Size
----------------------	---

HAL_PCD_EP_SetStall

Function name	HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
----------------------	--

Function description	Set a STALL condition over an endpoint.
-----------------------------	---

Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address
-------------------	--

Return values	<ul style="list-style-type: none">• HAL: status
----------------------	--

HAL_PCD_EP_ClrStall

Function name	HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
----------------------	--

Function description	Clear a STALL condition over in an endpoint.
-----------------------------	--

Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address
-------------------	--

Return values	<ul style="list-style-type: none">• HAL: status
----------------------	--

HAL_PCD_EP_Flush

Function name	HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
----------------------	---

Function description	Flush an endpoint.
-----------------------------	--------------------

Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address
-------------------	--

Return values	<ul style="list-style-type: none">• HAL: status
----------------------	--

HAL_PCD_ActivateRemoteWakeup

Function name	HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)
----------------------	--

Function description	HAL_PCD_ActivateRemoteWakeup : active remote wakeup signalling.
-----------------------------	---

Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
-------------------	---

Return values	<ul style="list-style-type: none">• HAL: status
----------------------	--

HAL_PCD_DeActivateRemoteWakeup

Function name	HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)
Function description	HAL_PCD_DeActivateRemoteWakeup : de-active remote wakeup signalling.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• HAL: status

HAL_PCD_GetState

Function name	PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)
Function description	Return the PCD state.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• HAL: state

29.3 PCD Firmware driver defines

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

29.3.1	PCD
	PCD
	<i>PCD Core PHY</i>

PCD_PHY_EMBEDDED

PCD Core Speed

PCD_SPEED_HIGH

PCD_SPEED_FULL

PCD ENDP

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

`PCD_ENDP7`

PCD Endpoint Kind

`PCD_SNG_BUF`

`PCD_DBL_BUF`

PCD EP0 MPS

`DEP0CTL_MPS_64`

`DEP0CTL_MPS_32`

`DEP0CTL_MPS_16`

`DEP0CTL_MPS_8`

`PCD_EP0MPS_64`

`PCD_EP0MPS_32`

`PCD_EP0MPS_16`

`PCD_EP0MPS_08`

PCD EP Type

`PCD_EP_TYPE_CTRL`

`PCD_EP_TYPE_ISOC`

`PCD_EP_TYPE_BULK`

`PCD_EP_TYPE_INTR`

PCD Exported Macros

`_HAL_PCD_GET_FLAG`

`_HAL_PCD_CLEAR_FLAG`

`_HAL_USB_WAKEUP_EX`
`TI_ENABLE_IT`

`_HAL_USB_WAKEUP_EX`
`TI_DISABLE_IT`

`_HAL_USB_EXTI_GENER`
`ATE_SWIT`

PCD Instance definition

IS_PCD_ALL_INSTANCE

30 HAL PCD Extension Driver

30.1 PCDEEx Firmware driver API description

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

30.1.1 Extended Peripheral Control functions

This section provides functions allowing to:

- Update PMA configuration

This section contains the following APIs:

- [*HAL_PCDEEx_PMAConfig*](#)

30.1.2 Detailed description of functions

[*HAL_PCDEEx_PMAConfig*](#)

Function name `HAL_StatusTypeDef HAL_PCDEEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaaddress)`

Function description Configure PMA for EP.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **ep_kind:** endpoint Kind
 - USB_SNG_BUF: Single Buffer used
 - USB_DBL_BUF: Double Buffer used
- **pmaaddress:** EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.

Return values

- :: status

31 HAL PWR Generic Driver

31.1 PWR Firmware driver API description

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

31.1.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, 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.

31.1.2 Peripheral Control functions

WakeUp pin configuration

- WakeUp pin is used to wakeup the system from Standby mode. This pin is forced in input pull down configuration and is active on rising edges.
- There are two WakeUp pins, and up to eight Wakeup pins on STM32F07x & STM32F09x devices.
 - WakeUp Pin 1 on PA.00.
 - WakeUp Pin 2 on PC.13.
 - WakeUp Pin 3 on PE.06.(STM32F07x/STM32F09x)
 - WakeUp Pin 4 on PA.02.(STM32F07x/STM32F09x)
 - WakeUp Pin 5 on PC.05.(STM32F07x/STM32F09x)
 - WakeUp Pin 6 on PB.05.(STM32F07x/STM32F09x)
 - WakeUp Pin 7 on PB.15.(STM32F07x/STM32F09x)
 - WakeUp Pin 8 on PF.02.(STM32F07x/STM32F09x)

Low Power modes configuration

The devices feature 3 low-power modes:

- Sleep mode: Cortex-M0 core stopped, peripherals kept running.
- Stop mode: all clocks are stopped, regulator running, regulator in low power mode
- Standby mode: 1.2V domain powered off (mode not available on STM32F0x8 devices).

Sleep mode

- Entry: The Sleep mode is entered by using the `HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFx)` functions with
 - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
 - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction
- Exit:
 - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

Stop mode

In Stop mode, all clocks in the 1.8V domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON, PWR_STOPENTRY_WFI) function with:
 - Main regulator ON.
 - Low Power regulator ON.
 - PWR_STOPENTRY_WFI: enter STOP mode with WFI instruction
 - PWR_STOPENTRY_WFE: enter STOP mode with WFE instruction
- Exit:
 - Any EXTI Line (Internal or External) configured in Interrupt/Event mode.
 - Some specific communication peripherals (CEC, USART, I2C) interrupts, when programmed in wakeup mode (the peripheral must be programmed in wakeup mode and the corresponding interrupt vector must be enabled in the NVIC)

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M0 deep sleep mode, with the voltage regulator disabled. The 1.8V domain is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers and Standby circuitry. The voltage regulator is OFF.

- Entry:
 - The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
- Exit:
 - WKUP pin rising edge, RTC alarm (Alarm A), RTC wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop and Standby modes
 - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL_RTC_SetAlarm_IT() function.
 - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL_RTC_SetTimeStamp_IT() or HAL_RTC_SetTamper_IT() functions.
 - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTC_SetWakeUpTimer_IT() function.
- Comparator auto-wakeup (AWU) from the Stop mode
 - To wake up from the Stop mode with a comparator wakeup event, it is necessary to:
 - Configure the EXTI Line associated with the comparator (example EXTI Line 22 for comparator 2) to be sensitive to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the EXTI_Init() function.
 - Configure the comparator to generate the event.

This section contains the following APIs:

- [**HAL_PWR_EnableWakeUpPin**](#)
- [**HAL_PWR_DisableWakeUpPin**](#)
- [**HAL_PWR_EnterSLEEPMode**](#)
- [**HAL_PWR_EnterSTOPMode**](#)
- [**HAL_PWR_EnterSTANDBYMode**](#)
- [**HAL_PWR_EnableSleepOnExit**](#)
- [**HAL_PWR_DisableSleepOnExit**](#)
- [**HAL_PWR_EnableSEVOnPend**](#)
- [**HAL_PWR_DisableSEVOnPend**](#)
- [**HAL_PWR_EnableBkUpAccess**](#)

- **HAL_PWR_DisableBkUpAccess**

31.1.3 Detailed description of functions

HAL_PWR_DeInit

Function name **void HAL_PWR_DeInit (void)**

Function description Deinitializes the PWR peripheral registers to their default reset values.

Return values

- **None:**

HAL_PWR_EnableBkUpAccess

Function name **void HAL_PWR_EnableBkUpAccess (void)**

Function description Enables access to the backup domain (RTC registers, RTC backup data registers when present).

Return values

- **None:**

Notes

- If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_DisableBkUpAccess

Function name **void HAL_PWR_DisableBkUpAccess (void)**

Function description Disables access to the backup domain (RTC registers, RTC backup data registers when present).

Return values

- **None:**

Notes

- If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_EnableWakeUpPin

Function name **void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)**

Function description Enables the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** Specifies the Power Wake-Up pin to enable. This parameter can be value of : PWREx Wakeup Pins

Return values

- **None:**

HAL_PWR_DisableWakeUpPin

Function name **void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)**

Function description Disables the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** Specifies the Power Wake-Up pin to disable. This parameter can be values of : PWREx Wakeup Pins

Return values

- **None:**

HAL_PWR_EnterSTOPMode

Function name

void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)

Function description

Enters STOP mode.

Parameters

- **Regulator:** Specifies the regulator state in STOP mode. This parameter can be one of the following values:
 - PWR_MAINREGULATOR_ON: STOP mode with regulator ON
 - PWR_LOWPOWERREGULATOR_ON: STOP mode with low power regulator ON
- **STOPEntry:** specifies if STOP mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
 - PWR_STOPENTRY_WFI: Enter STOP mode with WFI instruction
 - PWR_STOPENTRY_WFE: Enter STOP mode with WFE instruction

Return values

- **None:**

Notes

- In Stop mode, all I/O pins keep the same state as in Run mode.
- When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.
- When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

HAL_PWR_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)

Function description

Enters Sleep mode.

Parameters

- **Regulator:** Specifies the regulator state in SLEEP mode. On STM32F0 devices, this parameter is a dummy value and it is ignored as regulator can't be modified in this mode. Parameter is kept for platform compatibility.
- **SLEEPEntry:** Specifies if SLEEP mode is entered with WFI or WFE instruction. When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source. This parameter can be one of the following values:
 - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
 - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction

Return values

- **None:**

Notes

- In Sleep mode, all I/O pins keep the same state as in Run mode.

HAL_PWR_EnterSTANDBYMode

Function name

void HAL_PWR_EnterSTANDBYMode (void)

Function description

Enters STANDBY mode.

Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• In Standby mode, all I/O pins are high impedance except for: Reset pad (still available)RTC alternate function pins if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out.WKUP pins if enabled. STM32F0x8 devices, the Stop mode is available, but it is meaningless to distinguish between voltage regulator in Low power mode and voltage regulator in Run mode because the regulator is not used and the core is supplied directly from an external source. Consequently, the Standby mode is not available on those devices.

HAL_PWR_EnableSleepOnExit

Function name	void HAL_PWR_EnableSleepOnExit (void)
Function description	Indicates Sleep-On-Exit when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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	Enables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Sets SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_DisableSEVOnPend

Function name	void HAL_PWR_DisableSEVOnPend (void)
Function description	Disables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none">• None:

Notes

- Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

31.2 PWR Firmware driver defines

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

31.2.1 PWR

PWR

PWR Exported Macro

__HAL_PWR_GET_FLAG Description:

- Check PWR flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - PWR_FLAG_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm A), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
 - PWR_FLAG_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
 - PWR_FLAG_PVDO: PVD Output. This flag is valid only if PVD is enabled by the HAL_PWR_EnablePVD() function. The PVD is stopped by Standby mode For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set. Warning: this Flag is not available on STM32F030x8 products
 - PWR_FLAG_VREFINTRDY: This flag indicates that the internal reference voltage VREFINT is ready. Warning: this Flag is not available on STM32F030x8 products

Return value:

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

__HAL_PWR_CLEAR_FLAG Description:

G

- Clear the PWR's pending flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - PWR_FLAG_WU: Wake Up flag
 - PWR_FLAG_SB: StandBy flag

PWR Regulator state in STOP mode

PWR_MAINREGULATOR_ON

PWR_LOWPOWERREGULATOR_ON

IS_PWR_REGULATOR

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

PWR_SLEEPENTRY_WFE

IS_PWR_SLEEP_ENTRY

PWR STOP mode entry

PWR_STOPENTRY_WFI

PWR_STOPENTRY_WFE

IS_PWR_STOP_ENTRY

32 HAL PWR Extension Driver

32.1 PWREx Firmware driver registers structures

32.1.1 PWR_PVDTTypeDef

`PWR_PVDTTypeDef` is defined in the `stm32f0xx_hal_pwr_ex.h`

Data Fields

- `uint32_t PVDLevel`
- `uint32_t Mode`

Field Documentation

- `uint32_t PWR_PVDTTypeDef::PVDLevel`

PVDLevel: Specifies the PVD detection level This parameter can be a value of `PWREx_PVD_detection_level`

- `uint32_t PWR_PVDTTypeDef::Mode`

Mode: Specifies the operating mode for the selected pins. This parameter can be a value of `PWREx_PVD_Mode`

32.2 PWREx Firmware driver API description

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

32.2.1 Peripheral extended control functions

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the PWR_CR).
- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through `HAL_PWR_ConfigPVD()`, `HAL_PWR_EnablePVD()` functions.
- The PVD is stopped in Standby mode.

Note: *PVD is not available on STM32F030x4/x6/x8*

VDDIO2 Monitor Configuration

- VDDIO2 monitor is used to monitor the VDDIO2 power supply by comparing it to VREFInt Voltage
- This monitor is internally connected to the EXTI line31 and can generate an interrupt if enabled. This is done through `HAL_PWREx_EnableVddio2Monitor()` function.

Note: *VDDIO2 is available on STM32F07x/09x/04x*

This section contains the following APIs:

- `HAL_PWR_ConfigPVD`
- `HAL_PWR_EnablePVD`
- `HAL_PWR_DisablePVD`
- `HAL_PWR_PVD_IRQHandler`
- `HAL_PWR_PVDCallback`
- `HAL_PWREx_EnableVddio2Monitor`
- `HAL_PWREx_DisableVddio2Monitor`
- `HAL_PWREx_Vddio2Monitor_IRQHandler`
- `HAL_PWREx_Vddio2MonitorCallback`

32.2.2 Detailed description of functions

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() or PVD_VDDIO2_IRQHandler().

HAL_PWR_PVDCALLBACK

Function name **void HAL_PWR_PVDCALLBACK (void)**

Function description PWR PVD interrupt callback.

Return values • **None:**

HAL_PWREX_Vddio2Monitor_IRQHandler

Function name **void HAL_PWREX_Vddio2Monitor_IRQHandler (void)**

Function description This function handles the PWR Vddio2 monitor interrupt request.

Return values • **None:**

Notes • This API should be called under the VDDIO2_IRQHandler() PVD_VDDIO2_IRQHandler().

HAL_PWREX_Vddio2MonitorCallback

Function name **void HAL_PWREX_Vddio2MonitorCallback (void)**

Function description PWR Vddio2 Monitor interrupt callback.

Return values • **None:**

HAL_PWR_ConfigPVD

Function name **void HAL_PWR_ConfigPVD (PWR_PVDTTypeDef * sConfigPVD)**

Function description Configures the voltage threshold detected by the Power Voltage Detector(PVD).

Parameters • **sConfigPVD:** pointer to an PWR_PVDTTypeDef structure that contains the configuration information for the PVD.

Return values • **None:**

Notes • Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold 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_PWREx_EnableVddio2Monitor

Function name **void HAL_PWREx_EnableVddio2Monitor (void)**

Function description Enable VDDIO2 monitor: enable Exti 31 and falling edge detection.

Return values • **None:**

Notes • If Exti 31 is enable corretly and VDDIO2 voltage goes below Vrefint, an interrupt is generated IRQ line 1. NVIS has to be enable by user.

HAL_PWREx_DisableVddio2Monitor

Function name **void HAL_PWREx_DisableVddio2Monitor (void)**

Function description Disable the Vddio2 Monitor.

Return values • **None:**

32.3 PWREx Firmware driver defines

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

32.3.1 PWREx

PWREx

PWREx Exported Macros

HAL_PWR_PVD_EXTI_E **Description:**
NABLE_IT • Enable interrupt on PVD Exti Line 16.

Return value:

• None.

__HAL_PWR_PVD_EXTI_DI **Description:****SABLE_IT**

- Disable interrupt on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_E **Description:****NABLE_EVENT**

- Enable event on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DI **Description:****SABLE_EVENT**

- Disable event on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DI **Description:****SABLE_RISING_EDGE**

- Disable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DI **Description:****SABLE_FALLING_EDGE**

- Disable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DI **Description:****SABLE_RISING_FALLING_EDGE**

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_E **Description:****NABLE_FALLING_EDGE**

- PVD EXTI line configuration: set falling edge trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_E **Description:****NABLE_RISING_EDGE**

- PVD EXTI line configuration: set rising edge trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_E **Description:****NABLE_RISING_FALLING_EDGE**

- Enable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_G **Description:****ET_FLAG**

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

Return value:

- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTI_C **Description:****LEAR_FLAG**

- Clear the PVD EXTI flag.

Return value:

- None.

__HAL_PWR_PVD_EXTI_G **Description:****ENERATE_SWIT**

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

__HAL_PWR_VDDIO2_EXTI **Description:****ENABLE_IT**

- Enable interrupt on Vddio2 Monitor Exti Line 31.

Return value:

- None.

__HAL_PWR_VDDIO2_EXTI **Description:****DISABLE_IT**

- Disable interrupt on Vddio2 Monitor Exti Line 31.

Return value:

- None.

__HAL_PWR_VDDIO2_EXTI **Description:****DISABLE_FALLING_EDGE**

- Vddio2 Monitor EXTI line configuration: clear falling edge and rising edge trigger.

Return value:

- None.

__HAL_PWR_VDDIO2_EXTI **Description:****ENABLE_FALLING_EDGE**

- Vddio2 Monitor EXTI line configuration: set falling edge trigger.

Return value:

- None.

__HAL_PWR_VDDIO2_EXTI **Description:****GET_FLAG**

- Check whether the specified VDDIO2 monitor EXTI interrupt flag is set or not.

Return value:

- EXTI: VDDIO2 Monitor Line Status.

__HAL_PWR_VDDIO2_EXTI **Description:****CLEAR_FLAG**

- Clear the VDDIO2 Monitor EXTI flag.

Return value:

- None.

_HAL_PWR_VDDIO2_EXTI Description:**_GENERATE_SWIT** • Generate a Software interrupt on selected EXTI line.**Return value:**

- None.

PWREx EXTI Line**PWR_EXTI_LINE_PVD** External interrupt line 16 Connected to the PVD EXTI Line**PWR_EXTI_LINE_VDDIO2** External interrupt line 31 Connected to the Vddio2 Monitor EXTI Line**PWREx Flag****PWR_FLAG_WU****PWR_FLAG_SB****PWR_FLAG_PVDO****PWR_FLAG_VREFINTRDY****PWREx PVD detection level****PWR_PVDEVEL_0****PWR_PVDEVEL_1****PWR_PVDEVEL_2****PWR_PVDEVEL_3****PWR_PVDEVEL_4****PWR_PVDEVEL_5****PWR_PVDEVEL_6****PWR_PVDEVEL_7****IS_PWR_PVD_LEVEL****PWREx PVD Mode****PWR_PVD_MODE_NORMA** basic mode is used
L**PWR_PVD_MODE_IT_RISING** External Interrupt Mode with Rising edge trigger detection
ING**PWR_PVD_MODE_IT_FALLING** External Interrupt Mode with Falling edge trigger detection
ING

PWR_PVD_MODE_IT_RISING External Interrupt Mode with Rising/Falling edge trigger detection
NG_FALLING

PWR_PVD_MODE_EVENT_RISING Event Mode with Rising edge trigger detection

PWR_PVD_MODE_EVENT_FALLING Event Mode with Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING_FALLING Event Mode with Rising/Falling edge trigger detection

IS_PWR_PVD_MODE

PWREx Wakeup Pins

PWR_WAKEUP_PIN1

PWR_WAKEUP_PIN2

PWR_WAKEUP_PIN3

PWR_WAKEUP_PIN4

PWR_WAKEUP_PIN5

PWR_WAKEUP_PIN6

PWR_WAKEUP_PIN7

PWR_WAKEUP_PIN8

IS_PWR_WAKEUP_PIN

33 HAL RCC Generic Driver

33.1 RCC Firmware driver registers structures

33.1.1 RCC_PLLInitTypeDef

`RCC_PLLInitTypeDef` is defined in the `stm32f0xx_hal_rcc.h`

Data Fields

- `uint32_t PLLState`
- `uint32_t PLLSource`
- `uint32_t PLLMUL`
- `uint32_t PREDIV`

Field Documentation

- `uint32_t RCC_PLLInitTypeDef::PLLState`
PLLState: The new state of the PLL. This parameter can be a value of [`RCC_PLL_Config`](#)
- `uint32_t RCC_PLLInitTypeDef::PLLSource`
PLLSource: PLL entry clock source. This parameter must be a value of [`RCC_PLL_Clock_Source`](#)
- `uint32_t RCC_PLLInitTypeDef::PLLMUL`
PLLMUL: Multiplication factor for PLL VCO input clock This parameter must be a value of [`RCC_PLL_Multiplication_Factor`](#)
- `uint32_t RCC_PLLInitTypeDef::PREDIV`
PREDIV: Predivision factor for PLL VCO input clock This parameter must be a value of [`RCC_PLL_Prediv_Factor`](#)

33.1.2 RCC_OscInitTypeDef

`RCC_OscInitTypeDef` is defined in the `stm32f0xx_hal_rcc.h`

Data Fields

- `uint32_t OscillatorType`
- `uint32_t HSEState`
- `uint32_t LSEState`
- `uint32_t HSIState`
- `uint32_t HSICalibrationValue`
- `uint32_t HSI14State`
- `uint32_t HSI14CalibrationValue`
- `uint32_t LSIState`
- `uint32_t HSI48State`
- `RCC_PLLInitTypeDef PLL`

Field Documentation

- `uint32_t RCC_OscInitTypeDef::OscillatorType`
The oscillators to be configured. This parameter can be a value of [`RCC_Oscillator_Type`](#)
- `uint32_t RCC_OscInitTypeDef::HSEState`
The new state of the HSE. This parameter can be a value of [`RCC_HSE_Config`](#)
- `uint32_t RCC_OscInitTypeDef::LSEState`
The new state of the LSE. This parameter can be a value of [`RCC_LSE_Config`](#)
- `uint32_t RCC_OscInitTypeDef::HSIState`
The new state of the HSI. This parameter can be a value of [`RCC_HSI_Config`](#)

- **`uint32_t RCC_OscInitTypeDef::HSICalibrationValue`**
The HSI calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1FU
- **`uint32_t RCC_OscInitTypeDef::HSI14State`**
The new state of the HSI14. This parameter can be a value of [RCC_HSI14_Config](#)
- **`uint32_t RCC_OscInitTypeDef::HSI14CalibrationValue`**
The HSI14 calibration trimming value (default is RCC_HSI14CALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1FU
- **`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::HSI48State`**
The new state of the HSI48. This parameter can be a value of [RCC_HSI48_Config](#)
- **`RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL`**
PLL structure parameters

33.1.3 [RCC_ClkInitTypeDef](#)

`RCC_ClkInitTypeDef` is defined in the `stm32f0xx_hal_rcc.h`

Data Fields

- **`uint32_t ClockType`**
- **`uint32_t SYSCLKSource`**
- **`uint32_t AHBClockDivider`**
- **`uint32_t APB1ClockDivider`**

Field Documentation

- **`uint32_t RCC_ClkInitTypeDef::ClockType`**
The clock to be configured. This parameter can be a value of [RCC_System_Clock_Type](#)
- **`uint32_t RCC_ClkInitTypeDef::SYSCLKSource`**
The clock source (SYSCLKS) used as system clock. This parameter can be a value of [RCC_System_Clock_Source](#)
- **`uint32_t RCC_ClkInitTypeDef::AHBClockDivider`**
The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of [RCC_AHB_Clock_Source](#)
- **`uint32_t RCC_ClkInitTypeDef::APB1ClockDivider`**
The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of [RCC_APB1_Clock_Source](#)

33.2 [RCC Firmware driver API description](#)

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

33.2.1 [RCC specific features](#)

After reset the device is running from Internal High Speed oscillator (HSI 8MHz) with Flash 0 wait state, Flash prefetch buffer is enabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHB) and Low speed (APB) buses; all peripherals mapped on these buses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the JTAG pins which are assigned to be used for debug purpose.

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

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB buses prescalers

- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (RTC, ADC, I2C, USART, TIM, USB FS, etc..)

33.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
 - AHB & APB peripherals, 1 dummy read is necessary

Workarounds:

1. For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each `__HAL_RCC_PPP_CLK_ENABLE()` macro.

33.2.3 Initialization and de-initialization functions

This section provides functions allowing to configure the internal/external oscillators (HSE, HSI, HSI14, HSI48, LSE, LSI, PLL, CSS and MCO) and the System buses clocks (SYSCLK, AHB and APB1).

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 8 MHz factory-trimmed RC used directly or through the PLL as System clock source. The HSI clock can be used also to clock the USART and I2C peripherals.
2. HSI14 (high-speed internal), 14 MHz factory-trimmed RC used directly to clock the ADC peripheral.
3. LSI (low-speed internal), ~40 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 4 to 32 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL (clocked by HSI, HSI48 or HSE), featuring different output clocks:
 - The first output is used to generate the high speed system clock (up to 48 MHz)
 - The second output is used to generate the clock for the USB FS (48 MHz)
 - The third output may be used to generate the clock for the TIM, I2C and USART peripherals (up to 48 MHz)
7. CSS (Clock security system), once enable using the macro `__HAL_RCC_CSS_ENABLE()` and if a HSE clock failure occurs(HSE used directly or through PLL as System clock source), the System clocks automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M0 NMI (Non-Maskable Interrupt) exception vector.
8. MCO (microcontroller clock output), used to output SYSCLK, HSI, HSE, LSI, LSE or PLL clock (divided by 2) output on pin (such as PA8 pin).

System, AHB and APB buses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) clock is derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these buses. You can use "`@ref HAL_RCC_GetSysClockFreq()`" function to retrieve the frequencies of these clocks.
2. All the peripheral clocks are derived from the System clock (SYSCLK) except:
 - The FLASH program/erase clock which is always HSI 8MHz clock.
 - The USB 48 MHz clock which is derived from the PLL VCO clock.
 - The USART clock which can be derived as well from HSI 8MHz, LSI or LSE.
 - The I2C clock which can be derived as well from HSI 8MHz clock.
 - The ADC clock which is derived from PLL output.
 - The RTC clock which is derived from the LSE, LSI or 1 MHz HSE_RTC (HSE divided by a programmable prescaler). The System clock (SYSCLK) frequency must be higher or equal to the RTC clock frequency.
 - IWDG clock which is always the LSI clock.
3. For the STM32F0xx devices, the maximum frequency of the SYSCLK, HCLK and PCLK1 is 48 MHz, Depending on the SYSCLK frequency, the flash latency should be adapted accordingly.

4. After reset, the System clock source is the HSI (8 MHz) with 0 WS and prefetch is disabled.

This section contains the following APIs:

- [*HAL_RCC_DelInit*](#)
- [*HAL_RCC_OscConfig*](#)
- [*HAL_RCC_ClockConfig*](#)

33.2.4 Peripheral Control functions

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

This section contains the following APIs:

- [*HAL_RCC_MCOConfig*](#)
- [*HAL_RCC_EnableCSS*](#)
- [*HAL_RCC_DisableCSS*](#)
- [*HAL_RCC_GetSysClockFreq*](#)
- [*HAL_RCC_GetHCLKFreq*](#)
- [*HAL_RCC_GetPCLK1Freq*](#)
- [*HAL_RCC_GetOscConfig*](#)
- [*HAL_RCC_GetClockConfig*](#)
- [*HAL_RCC_NMI_IRQHandler*](#)
- [*HAL_RCC_CSSCallback*](#)

33.2.5 Detailed description of functions

HAL_RCC_DelInit

Function name **void HAL_RCC_DelInit (void)**

Function description Resets the RCC clock configuration to the default reset state.

Return values

- **None:**

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE and PLL OFFAHB, APB1 prescaler set to 1.CSS and MCO1 OFFALL interrupts disabled
- This function does not modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name **HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)**

Function description Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

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

Return values

- **HAL:** status

Notes

- The PLL is not disabled when used as system clock.
- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

HAL_RCC_ClockConfig

Function name	<code>HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)</code>
Function description	Initializes the CPU, AHB and APB buses clocks according to the specified parameters in the <code>RCC_ClkInitStruct</code> .
Parameters	<ul style="list-style-type: none">• RCC_ClkInitStruct: pointer to an <code>RCC_OsclInitTypeDef</code> structure that contains the configuration information for the RCC peripheral.• FLatency: FLASH Latency The value of this parameter depend on device used within the same series
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• The <code>SystemCoreClock</code> CMSIS variable is used to store System Clock Frequency and updated by <code>HAL_RCC_GetHCLKFreq()</code> function called within this function• The HSI is used (enabled by hardware) as system clock source after start-up from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).• A switch from one clock source to another occurs only if the target clock source is ready (clock stable after start-up delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready. You can use <code>HAL_RCC_GetClockConfig()</code> function to know which clock is currently used as system clock source.

HAL_RCC_MCOConfig

Function name	<code>void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)</code>
Function description	Selects the clock source to output on MCO pin.

- Parameters**
- **RCC_MCOx:** specifies the output direction for the clock source. This parameter can be one of the following values:
 - RCC_MCO1 Clock source to output on MCO1 pin(PA8).
 - **RCC_MCOsource:** specifies the clock source to output. This parameter can be one of the following values:
 - RCC_MCO1SOURCE_NOCLOCK No clock selected
 - RCC_MCO1SOURCE_SYSCLK System Clock selected as MCO clock
 - RCC_MCO1SOURCE_HSI HSI selected as MCO clock
 - RCC_MCO1SOURCE_HSE HSE selected as MCO clock
 - RCC_MCO1SOURCE_LSI LSI selected as MCO clock
 - RCC_MCO1SOURCE_LSE LSE selected as MCO clock
 - RCC_MCO1SOURCE_HSI14 HSI14 selected as MCO clock
 - RCC_MCO1SOURCE_HSI48 HSI48 selected as MCO clock
 - RCC_MCO1SOURCE_PLLCLK PLLCLK selected as MCO clock
 - RCC_MCO1SOURCE_PLLCLK_DIV2 PLLCLK Divided by 2 selected as MCO clock
 - **RCC_MCODiv:** specifies the MCO DIV. This parameter can be one of the following values:
 - RCC_MCODIV_1 no division applied to MCO clock
 - RCC_MCODIV_2 division by 2 applied to MCO clock
 - RCC_MCODIV_4 division by 4 applied to MCO clock
 - RCC_MCODIV_8 division by 8 applied to MCO clock
 - RCC_MCODIV_16 division by 16 applied to MCO clock
 - RCC_MCODIV_32 division by 32 applied to MCO clock
 - RCC_MCODIV_64 division by 64 applied to MCO clock
 - RCC_MCODIV_128 division by 128 applied to MCO clock

- Return values**
- **None:**

- Notes**
- MCO pin should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name **void HAL_RCC_EnableCSS (void)**

Function description Enables the Clock Security System.

- Return values**
- **None:**

- Notes**
- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M0 NMI (Non-Maskable Interrupt) exception vector.

HAL_RCC_NMI_IRQHandler

Function name **void HAL_RCC_NMI_IRQHandler (void)**

Function description This function handles the RCC CSS interrupt request.

- Return values**
- **None:**

Notes

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

HAL_RCC_CSSCallback**Function name** **void HAL_RCC_CSSCallback (void)****Function description** RCC Clock Security System interrupt callback.**Return values**

- **none:**

HAL_RCC_DisableCSS**Function name** **void HAL_RCC_DisableCSS (void)****Function description** Disables the Clock Security System.**Return values**

- **None:**

HAL_RCC_GetSysClockFreq**Function name** **uint32_t HAL_RCC_GetSysClockFreq (void)****Function description** Returns the SYSCLK frequency.**Return values**

- **SYSCLK:** frequency

Notes

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)
- If SYSCLK source is HSE, function returns a value based on HSE_VALUE divided by PREDIV factor(**)
- If SYSCLK source is PLL, function returns a value based on HSE_VALUE divided by PREDIV factor(**) or depending on STM32F0xxxx devices either a value based on HSI_VALUE divided by 2 or HSI_VALUE divided by PREDIV factor(*) multiplied by the PLL factor.
- (*) HSI_VALUE is a constant defined in stm32f0xx_hal_conf.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) HSE_VALUE is a constant defined in stm32f0xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq**Function name** **uint32_t HAL_RCC_GetHCLKFreq (void)****Function description** Returns the HCLK frequency.**Return values**

- **HCLK:** frequency

Notes

- Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect.
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

HAL_RCC_GetPCLK1Freq

Function name `uint32_t HAL_RCC_GetPCLK1Freq (void)`

Function description Returns the PCLK1 frequency.

Return values

- **PCLK1:** frequency

Notes

- Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name `void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)`

Function description Configures 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 Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that contains the current clock configuration.
- **pFLatency:** Pointer on the Flash Latency.

Return values

- **None:**

33.3 RCC Firmware driver defines

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

33.3.1 RCC

RCC

RCC AHB Clock Enable Disable

`_HAL_RCC_GPIOA_CLK_ENABLE`

`_HAL_RCC_GPIOB_CLK_ENABLE`

`__HAL_RCC_GPIOC_CLK_ENABLE`

`__HAL_RCC_GPIOF_CLK_ENABLE`

`__HAL_RCC_CRC_CLK_ENABLE`

`__HAL_RCC_DMA1_CLK_ENABLE`

`__HAL_RCC_SRAM_CLK_ENABLE`

`__HAL_RCC_FLITF_CLK_ENABLE`

`__HAL_RCC_GPIOA_CLK_DISABLE`

`__HAL_RCC_GPIOB_CLK_DISABLE`

`__HAL_RCC_GPIOC_CLK_DISABLE`

`__HAL_RCC_GPIOF_CLK_DISABLE`

`__HAL_RCC_CRC_CLK_DISABLE`

`__HAL_RCC_DMA1_CLK_DISABLE`

`__HAL_RCC_SRAM_CLK_DISABLE`

`__HAL_RCC_FLITF_CLK_DISABLE`

AHB Clock Source

`RCC_SYSCLK_DIV1` SYSCLK not divided

`RCC_SYSCLK_DIV2` SYSCLK divided by 2

`RCC_SYSCLK_DIV4` SYSCLK divided by 4

`RCC_SYSCLK_DIV8` SYSCLK divided by 8

`RCC_SYSCLK_DIV16` SYSCLK divided by 16

`RCC_SYSCLK_DIV64` SYSCLK divided by 64

`RCC_SYSCLK_DIV128` SYSCLK divided by 128

`RCC_SYSCLK_DIV256` SYSCLK divided by 256

`RCC_SYSCLK_DIV512` SYSCLK divided by 512

RCC AHB Force Release Reset

`_HAL_RCC_AHB_FORCE_RESET`

`_HAL_RCC_GPIOA_FORC_E_RESET`

`_HAL_RCC_GPIOB_FORC_E_RESET`

`_HAL_RCC_GPIOC_FORC_E_RESET`

`_HAL_RCC_GPIOF_FORC_E_RESET`

`_HAL_RCC_AHB_RELEASE_RESET`

`_HAL_RCC_GPIOA_RELEASE_RESET`

`_HAL_RCC_GPIOB_RELEASE_RESET`

`_HAL_RCC_GPIOC_RELEASE_RESET`

`_HAL_RCC_GPIOF_RELEASE_RESET`

AHB Peripheral Clock Enable Disable Status

`_HAL_RCC_GPIOA_IS_CLK_ENABLED`

`_HAL_RCC_GPIOB_IS_CLK_ENABLED`

`_HAL_RCC_GPIOC_IS_CLK_ENABLED`

`_HAL_RCC_GPIOF_IS_CLK_ENABLED`

`_HAL_RCC_CRC_IS_CLK
_ENABLED`

`_HAL_RCC_DMA1_IS_CLK
_ENABLED`

`_HAL_RCC_SRAM_IS_CLK
_ENABLED`

`_HAL_RCC_FLITF_IS_CLK
_ENABLED`

`_HAL_RCC_GPIOA_IS_CLK
_DISABLED`

`_HAL_RCC_GPIOB_IS_CLK
_DISABLED`

`_HAL_RCC_GPIOC_IS_CLK
_DISABLED`

`_HAL_RCC_GPIOF_IS_CLK
_DISABLED`

`_HAL_RCC_CRC_IS_CLK
_DISABLED`

`_HAL_RCC_DMA1_IS_CLK
_DISABLED`

`_HAL_RCC_SRAM_IS_CLK
_DISABLED`

`_HAL_RCC_FLITF_IS_CLK
_DISABLED`

RCC APB1 Clock Enable Disable

`_HAL_RCC_TIM3_CLK_ENABLE`

`_HAL_RCC_TIM14_CLK_ENABLE`

`_HAL_RCC_WWDG_CLK_ENABLE`

`_HAL_RCC_I2C1_CLK_ENABLE`

`_HAL_RCC_PWR_CLK_ENABLE`

`_HAL_RCC_TIM3_CLK_DISABLE`

`_HAL_RCC_TIM14_CLK_DISABLE`

`_HAL_RCC_WWDG_CLK_DISABLE`

`_HAL_RCC_I2C1_CLK_DISABLE`

`_HAL_RCC_PWR_CLK_DISABLE`

RCC APB1 Clock Source

`RCC_HCLK_DIV1` HCLK not divided

`RCC_HCLK_DIV2` HCLK divided by 2

`RCC_HCLK_DIV4` HCLK divided by 4

`RCC_HCLK_DIV8` HCLK divided by 8

`RCC_HCLK_DIV16` HCLK divided by 16

RCC APB1 Force Release Reset

`_HAL_RCC_APB1_FORCE_RESET`

`_HAL_RCC_TIM3_FORCE_RESET`

`_HAL_RCC_TIM14_FORCE_RESET`

`_HAL_RCC_WWDG_FORCE_RESET`

`_HAL_RCC_I2C1_FORCE_RESET`

`_HAL_RCC_PWR_FORCE_RESET`

`_HAL_RCC_APB1_RELEASE_RESET`

`_HAL_RCC_TIM3_RELEASE_RESET`

`__HAL_RCC_TIM14_RELEASE_RESET`

`__HAL_RCC_WWDG_RELEASE_RESET`

`__HAL_RCC_I2C1_RELEASE_RESET`

`__HAL_RCC_PWR_RELEASE_RESET`

APB1 Peripheral Clock Enable Disable Status

`__HAL_RCC_TIM3_IS_CLK_ENABLED`

`__HAL_RCC_TIM14_IS_CLK_ENABLED`

`__HAL_RCC_WWDG_IS_CLK_ENABLED`

`__HAL_RCC_I2C1_IS_CLK_ENABLED`

`__HAL_RCC_PWR_IS_CLK_ENABLED`

`__HAL_RCC_TIM3_IS_CLK_DISABLED`

`__HAL_RCC_TIM14_IS_CLK_DISABLED`

`__HAL_RCC_WWDG_IS_CLK_DISABLED`

`__HAL_RCC_I2C1_IS_CLK_DISABLED`

`__HAL_RCC_PWR_IS_CLK_DISABLED`

RCC APB2 Clock Enable Disable

`__HAL_RCC_SYSCFG_CLK_ENABLE`

`__HAL_RCC_ADC1_CLK_ENABLE`

`__HAL_RCC_TIM1_CLK_ENABLE`

`_HAL_RCC_SPI1_CLK_ENABLE`

`_HAL_RCC_TIM16_CLK_ENABLE`

`_HAL_RCC_TIM17_CLK_ENABLE`

`_HAL_RCC_USART1_CLK_ENABLE`

`_HAL_RCC_DBGMCU_CLK_ENABLE`

`_HAL_RCC_SYSCFG_CLK_DISABLE`

`_HAL_RCC_ADC1_CLK_DISABLE`

`_HAL_RCC_TIM1_CLK_DISABLE`

`_HAL_RCC_SPI1_CLK_DISABLE`

`_HAL_RCC_TIM16_CLK_DISABLE`

`_HAL_RCC_TIM17_CLK_DISABLE`

`_HAL_RCC_USART1_CLK_DISABLE`

`_HAL_RCC_DBGMCU_CLK_DISABLE`

RCC APB2 Force Release Reset

`_HAL_RCC_APB2_FORCE_RESET`

`_HAL_RCC_SYSCFG_FORCE_RESET`

`_HAL_RCC_ADC1_FORCE_RESET`

`_HAL_RCC_TIM1_FORCE_RESET`

`_HAL_RCC_SPI1_FORCE_RESET`

`_HAL_RCC_USART1_FORCE_RESET`

`_HAL_RCC_TIM16_FORCE_RESET`

`_HAL_RCC_TIM17_FORCE_RESET`

`_HAL_RCC_DBGMCU_FORCE_RESET`

`_HAL_RCC_APB2_RELEASE_RESET`

`_HAL_RCC_SYSCFG_RELEASE_RESET`

`_HAL_RCC_ADC1_RELEASE_RESET`

`_HAL_RCC_TIM1_RELEASE_RESET`

`_HAL_RCC_SPI1_RELEASE_RESET`

`_HAL_RCC_USART1_RELEASE_RESET`

`_HAL_RCC_TIM16_RELEASE_RESET`

`_HAL_RCC_TIM17_RELEASE_RESET`

`_HAL_RCC_DBGMCU_RELEASE_RESET`

APB2 Peripheral Clock Enable Disable Status

`_HAL_RCC_SYSCFG_IS_CLK_ENABLED`

`_HAL_RCC_ADC1_IS_CLK_ENABLED`

`_HAL_RCC_TIM1_IS_CLK_ENABLED`

`_HAL_RCC_SPI1_IS_CLK
_ENABLED`

`_HAL_RCC_TIM16_IS_CLK
_ENABLED`

`_HAL_RCC_TIM17_IS_CLK
_ENABLED`

`_HAL_RCC_USART1_IS_CLK
_ENABLED`

`_HAL_RCC_DBGMCU_IS_CLK
_ENABLED`

`_HAL_RCC_SYSCFG_IS_CLK
_DISABLED`

`_HAL_RCC_ADC1_IS_CLK
_DISABLED`

`_HAL_RCC_TIM1_IS_CLK
_DISABLED`

`_HAL_RCC_SPI1_IS_CLK
_DISABLED`

`_HAL_RCC_TIM16_IS_CLK
_DISABLED`

`_HAL_RCC_TIM17_IS_CLK
_DISABLED`

`_HAL_RCC_USART1_IS_CLK
_DISABLED`

`_HAL_RCC_DBGMCU_IS_CLK
_DISABLED`

Flags

`RCC_FLAG_HSIRDY`

`RCC_FLAG_HSERDY`

`RCC_FLAG_PLLRDY`

`RCC_FLAG_HSI14RDY`

`RCC_FLAG_LSIRDY`

`RCC_FLAG_V18PWRRST`

RCC_FLAG_OBLRST

RCC_FLAG_PINRST	PIN reset flag
RCC_FLAG_PORRST	POR/PDR reset flag
RCC_FLAG_SFTRST	Software Reset flag
RCC_FLAG_IWDGRST	Independent Watchdog reset flag
RCC_FLAG_WWDGRST	Window watchdog reset flag
RCC_FLAG_LPWRRST	Low-Power reset flag
RCC_FLAG_LSERDY	External Low Speed oscillator Ready

RCC_FLAG_HSI48RDY

Flags Interrupts Management

_HAL_RCC_ENABLE_IT	Description: <ul style="list-style-type: none">Enable RCC interrupt. Parameters: <ul style="list-style-type: none">_INTERRUPT_: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:<ul style="list-style-type: none">RCC_IT_LSIRDY LSI ready interruptRCC_IT_LSERDY LSE ready interruptRCC_IT_HSIRDY HSI ready interruptRCC_IT_HSERDY HSE ready interruptRCC_IT_PLLRDY main PLL ready interruptRCC_IT_HSI14RDY HSI14 ready interruptRCC_IT_HSI48RDY HSI48 ready interrupt
_HAL_RCC_DISABLE_IT	Description: <ul style="list-style-type: none">Disable RCC interrupt. Parameters: <ul style="list-style-type: none">_INTERRUPT_: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:<ul style="list-style-type: none">RCC_IT_LSIRDY LSI ready interruptRCC_IT_LSERDY LSE ready interruptRCC_IT_HSIRDY HSI ready interruptRCC_IT_HSERDY HSE ready interruptRCC_IT_PLLRDY main PLL ready interruptRCC_IT_HSI14RDY HSI14 ready interruptRCC_IT_HSI48RDY HSI48 ready interrupt

[__HAL_RCC_CLEAR_IT](#)

Description:

- Clear the RCC's interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY LSI ready interrupt.
 - RCC_IT_LSERDY LSE ready interrupt.
 - RCC_IT_HSIRDY HSI ready interrupt.
 - RCC_IT_HSERDY HSE ready interrupt.
 - RCC_IT_PLLRDY Main PLL ready interrupt.
 - RCC_IT_CSS Clock Security System interrupt
 - RCC_IT_HSI14RDY HSI14 ready interrupt
 - RCC_IT_HSI48RDY HSI48 ready interrupt

[__HAL_RCC_GET_IT](#)

Description:

- Check the RCC's interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be one of the following values:
 - RCC_IT_LSIRDY LSI ready interrupt.
 - RCC_IT_LSERDY LSE ready interrupt.
 - RCC_IT_HSIRDY HSI ready interrupt.
 - RCC_IT_HSERDY HSE ready interrupt.
 - RCC_IT_PLLRDY Main PLL ready interrupt.
 - RCC_IT_CSS Clock Security System interrupt
 - RCC_IT_HSI14RDY HSI14 ready interrupt enable
 - RCC_IT_HSI48RDY HSI48 ready interrupt

Return value:

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

[__HAL_RCC_CLEAR_RESET_FLAGS](#)

The reset flags are RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_OBLRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRRST

__HAL_RCC_GET_FLAG

Description:

- Check RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - RCC_FLAG_HSIRDY HSI oscillator clock ready.
 - RCC_FLAG_HSERDY HSE oscillator clock ready.
 - RCC_FLAG_PLLRDY Main PLL clock ready.
 - RCC_FLAG_HSI14RDY HSI14 oscillator clock ready
 - RCC_FLAG_HSI48RDY HSI48 oscillator clock ready
 - RCC_FLAG_LSERDY LSE oscillator clock ready.
 - RCC_FLAG_LSIRDY LSI oscillator clock ready.
 - RCC_FLAG_OBLRST Option Byte Load reset
 - RCC_FLAG_PINRST Pin reset.
 - RCC_FLAG_PORRST POR/PDR reset.
 - RCC_FLAG_SFTRST Software reset.
 - RCC_FLAG_IWDGRST Independent Watchdog reset.
 - RCC_FLAG_WWDGRST Window Watchdog reset.
 - RCC_FLAG_LPWRST Low Power reset.

Return value:

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

Get Clock source

__HAL_RCC_SYSCLK_CO

NFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
 - RCC_SYSCLKSOURCE_HSI HSI oscillator is used as system clock source.
 - RCC_SYSCLKSOURCE_HSE HSE oscillator is used as system clock source.
 - RCC_SYSCLKSOURCE_PLLCLK PLL output is used as system clock source.

__HAL_RCC_GET_SYSCLK

SOURCE

Description:

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

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
 - RCC_SYSCLKSOURCE_STATUS_HSI HSI used as system clock
 - RCC_SYSCLKSOURCE_STATUS_HSE HSE used as system clock
 - RCC_SYSCLKSOURCE_STATUS_PLLCLK PLL used as system clock

HSE Config

RCC_HSE_OFF

HSE clock deactivation

RCC_HSE_ON

HSE clock activation

RCC_HSE_BYPASS

External clock source for HSE clock

HSE Configuration

__HAL_RCC_HSE_CONFIG Description:

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
 - RCC_HSE_OFF turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
 - RCC_HSE_ON turn ON the HSE oscillator
 - RCC_HSE_BYPASS HSE oscillator bypassed with external clock

Notes:

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you have to enable it again after calling this function.

__HAL_RCC_HSE_PREDIV_CONFIG Description:

- Macro to configure the External High Speed oscillator (HSE) Predivision factor for PLL.

Parameters:

- __HSE_PREDIV_VALUE__: specifies the division value applied to HSE. This parameter must be a number between RCC_HSE_PREDIV_DIV1 and RCC_HSE_PREDIV_DIV16.

Notes:

- Predivision factor can not be changed if PLL is used as system clock In this case, you have to select another source of the system clock, disable the PLL and then change the HSE predivision factor.

RCC HSI14 Config

RCC_HSI14_OFF

RCC_HSI14_ON

RCC_HSI14_ADC_CONTROL

RCC_HSI14CALIBRATION_DEFAULT

RCC_HSI14_Configuration

__HAL_RCC_HSI14_ENABLE Notes:

- After enabling the HSI14 with __HAL_RCC_HSI14_ENABLE(), the application software should wait on HSI14RDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This is not necessary if HAL_RCC_OscConfig() is used.

__HAL_RCC_HSI14_DISAB Notes: LE

- The HSI14 is stopped by hardware when entering STOP and STANDBY modes. HSI14 can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI14. When the HSI14 is stopped, HSI14RDY flag goes low after 6 HSI14 oscillator clock cycles.

__HAL_RCC_HSI14ADC_E NABLE

__HAL_RCC_HSI14ADC_DI SABLE

__HAL_RCC_HSI14_CALIB Description: RATIONVALUE_ADJUST

- Macro to adjust the Internal 14Mhz High Speed oscillator (HSI) calibration value.

Parameters:

- __HSI14CALIBRATIONVALUE: specifies the calibration trimming value (default is RCC_HSI14CALIBRATION_DEFAULT). This parameter must be a number between 0 and 0x1F.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI14 RC.

HSI48 Config

RCC_HSI48_OFF

RCC_HSI48_ON

HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

RCC_HSICALIBRATION_D

EFAULT

HSI Configuration

__HAL_RCC_HSI_ENABLE Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

[__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST](#)

Description:

- Macro to adjust the Internal High Speed oscillator (HSI) calibration value.

Parameters:

- _HSICALIBRATIONVALUE_: specifies the calibration trimming value. (default is `RCC_HSICALIBRATION_DEFAULT`). This parameter must be a number between 0 and 0x1F.

Notes:

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

RCC I2C1 Clock Source

[RCC_I2C1CLKSOURCE_HS](#)

[RCC_I2C1CLKSOURCE_SY](#)

RCC I2Cx Clock Config

[__HAL_RCC_I2C1_CONFIG](#)

- Macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- _I2C1CLKSOURCE_: specifies the I2C1 clock source. This parameter can be one of the following values:
 - `RCC_I2C1CLKSOURCE_HSI` HSI selected as I2C1 clock
 - `RCC_I2C1CLKSOURCE_SYSCLK` System Clock selected as I2C1 clock

[__HAL_RCC_GET_I2C1_SOURCE](#)

Description:

- Macro to get the I2C1 clock source.

Return value:

- The clock source can be one of the following values:
 - `RCC_I2C1CLKSOURCE_HSI` HSI selected as I2C1 clock
 - `RCC_I2C1CLKSOURCE_SYSCLK` System Clock selected as I2C1 clock

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_HSI14RDY HSI14 Ready Interrupt flag

RCC_IT_HSI48RDY HSI48 Ready Interrupt flag

RCC_IT_CSS Clock Security System Interrupt flag

RCC_IT_HSI48 HSI48 Ready Interrupt flag

LSE Config

RCC_LSE_OFF LSE clock deactivation

RCC_LSE_ON LSE clock activation

RCC_LSE_BYPASS External clock source for LSE clock

LSE Configuration

__HAL_RCC_LSE_CONFIG Description:

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- STATE: specifies the new state of the LSE. This parameter can be one of the following values:
 - RCC_LSE_OFF turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
 - RCC_LSE_ON turn ON the LSE oscillator.
 - RCC_LSE_BYPASS LSE oscillator bypassed with external clock.

Notes:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

LSI Config

RCC_LSI_OFF LSI clock deactivation

RCC_LSI_ON LSI clock activation

LSI Configuration

__HAL_RCC_LSI_ENABLE Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC.

__HAL_RCC_LSI_DISABLE Notes:

- LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

RCC MCO Clock Source

RCC_MCO1SOURCE_NOCLOCK

RCC_MCO1SOURCE_LSI

RCC_MCO1SOURCE_LSE

RCC_MCO1SOURCE_SYS
CLK

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLLC
LK_DIV2

RCC_MCO1SOURCE_HSI1
4

RCC_MCO1SOURCE_PLLC
LK

RCC_MCO1SOURCE_HSI4
8

MCO Index

RCC_MCO1

RCC_MCO MCO1 to be compliant with other families with 2 MCOs

Oscillator Type

RCC_OSCILLATORTYPE_N
ONE

RCC_OSCILLATORTYPE_H
SE

RCC_OSCILLATORTYPE_H
SI

RCC_OSCILLATORTYPE_L
SE

RCC_OSCILLATORTYPE_L
SI

RCC_OSCILLATORTYPE_H
SI14

RCC_OSCILLATORTYPE_H
SI48

PLL Clock Source

RCC_PLLSOURCE_HSE HSE clock selected as PLL entry clock source

RCC_PLLSOURCE_HSI

RCC_PLLSOURCE_HSI48

PLL Config

RCC_PLL_NONE PLL is not configured

RCC_PLL_OFF PLL deactivation

RCC_PLL_ON PLL activation

PLL Configuration

__HAL_RCC_PLL_ENABLE Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE Notes:

- E
- The main PLL can not be disabled if it is used as system clock source

__HAL_RCC_PLL_CONFIG Description:

- Macro to configure the PLL clock source, multiplication and division factors.

Parameters:

- RCC_PLLSOURCE: specifies the PLL entry clock source. This parameter can be one of the following values:
 - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
 - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
- PLLMUL: specifies the multiplication factor for PLL VCO output clock This parameter can be one of the following values: This parameter must be a number between RCC_PLL_MUL2 and RCC_PLL_MUL16.
- PREDIV: specifies the predivider factor for PLL VCO input clock This parameter must be a number between RCC_PREDIV_DIV1 and RCC_PREDIV_DIV16.

Notes:

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

__HAL_RCC_GET_PLL_OS Description:

CSOURCE

- Get oscillator clock selected as PLL input clock.

Return value:

- The: clock source used for PLL entry. The returned value can be one of the following:
 - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL input clock

RCC PLL Multiplication Factor

RCC_PLL_MUL2

RCC_PLL_MUL3

RCC_PLL_MUL4

RCC_PLL_MUL5

RCC_PLL_MUL6

RCC_PLL_MUL7

RCC_PLL_MUL8

RCC_PLL_MUL9

RCC_PLL_MUL10

RCC_PLL_MUL11

RCC_PLL_MUL12

RCC_PLL_MUL13

RCC_PLL_MUL14

RCC_PLL_MUL15

RCC_PLL_MUL16

RCC PLL Prediv Factor

RCC_PREDIV_DIV1

RCC_PREDIV_DIV2

RCC_PREDIV_DIV3

RCC_PREDIV_DIV4

RCC_PREDIV_DIV5

RCC_PREDIV_DIV6

RCC_PREDIV_DIV7

RCC_PREDIV_DIV8

RCC_PREDIV_DIV9

RCC_PREDIV_DIV10

RCC_PREDIV_DIV11

RCC_PREDIV_DIV12

[RCC_PREDIV_DIV13](#)

[RCC_PREDIV_DIV14](#)

[RCC_PREDIV_DIV15](#)

[RCC_PREDIV_DIV16](#)

Register offsets

[RCC_OFFSET](#)

[RCC_CR_OFFSET](#)

[RCC_CFGR_OFFSET](#)

[RCC_CIR_OFFSET](#)

[RCC_BDCR_OFFSET](#)

[RCC_CSR_OFFSET](#)

RCC RTC Clock Configuration

[__HAL_RCC_RTC_CONFIG](#) Description:

- Macro to configure the RTC clock (RTCCLK).

Parameters:

- `__RTC_CLKSOURCE__`: specifies the RTC clock source. This parameter can be one of the following values:
 - `RCC_RTCCLKSOURCE_NO_CLK` No clock selected as RTC clock
 - `RCC_RTCCLKSOURCE_LSE` LSE selected as RTC clock
 - `RCC_RTCCLKSOURCE_LSI` LSI selected as RTC clock
 - `RCC_RTCCLKSOURCE_HSE_DIV32` HSE clock divided by 32

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the Backup domain is reset using `__HAL_RCC_BACKUPRESET_FORCE()` macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the LSI clock and HSE clock divided by 32 is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The system must always be configured so as to get a PCLK frequency greater than or equal to the RTCCLK frequency for a proper operation of the RTC.

__HAL_RCC_GET_RTC_SO Description:

URCE

- Macro to get the RTC clock source.

Return value:

- The clock source can be one of the following values:
 - RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock
 - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock
 - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock
 - RCC_RTCCLKSOURCE_HSE_DIV32 HSE clock divided by 32

__HAL_RCC_RTC_ENABL Notes:

E

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

__HAL_RCC_RTC_DISABL Notes:

E

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

__HAL_RCC_BACKUPRES Notes:

ET_FORCE

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_BDCR register.

__HAL_RCC_BACKUPRES
ET_RELEASE***RTC Clock Source***RCC_RTCCLKSOURCE_N No clock
O_CLKRCC_RTCCLKSOURCE_LS LSE oscillator clock used as RTC clock
ERCC_RTCCLKSOURCE_LS LSI oscillator clock used as RTC clock
IRCC_RTCCLKSOURCE_HS HSE oscillator clock divided by 32 used as RTC clock
E_DIV32***System Clock Source***RCC_SYSCLKSOURCE_HS HSI selected as system clock
IRCC_SYSCLKSOURCE_HS HSE selected as system clock
ERCC_SYSCLKSOURCE_PL PLL selected as system clock
LCLKRCC_SYSCLKSOURCE_HS
I48***System Clock Source Status***

RCC_SYSCLKSOURCE_ST HSI used as system clock
ATUS_HSI

RCC_SYSCLKSOURCE_ST HSE used as system clock
ATUS_HSE

RCC_SYSCLKSOURCE_ST PLL used as system clock
ATUS_PLLCLK

RCC_SYSCLKSOURCE_ST
ATUS_HSI48

System Clock Type

RCC_CLOCKTYPE_SYSCLK SYSCLK to configure
K

RCC_CLOCKTYPE_HCLK HCLK to configure

RCC_CLOCKTYPE_PCLK1 PCLK1 to configure

RCC Timeout

**RCC_DBP_TIMEOUT_VALU
E**

**RCC_LSE_TIMEOUT_VALU
E**

**CLOCKSWITCH_TIMEOUT_
VALUE**

HSE_TIMEOUT_VALUE

HSI_TIMEOUT_VALUE

LSI_TIMEOUT_VALUE

PLL_TIMEOUT_VALUE

HSI14_TIMEOUT_VALUE

HSI48_TIMEOUT_VALUE

RCC USART1 Clock Source

**RCC_USART1CLKSOURCE
_PCLK1**

**RCC_USART1CLKSOURCE
_SYSCLK**

**RCC_USART1CLKSOURCE
_LSE**

RCC_USART1CLKSOURCE _HSI

RCC USARTx Clock Config

__HAL_RCC_USART1_CO_NFIG Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1CLKSOURCE__: specifies the USART1 clock source. This parameter can be one of the following values:
 - RCC_USART1CLKSOURCE_PCLK1 PCLK1 selected as USART1 clock
 - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
 - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__HAL_RCC_GET_USART1_SOURCE Description:

- Macro to get the USART1 clock source.

Return value:

- The clock source can be one of the following values:
 - RCC_USART1CLKSOURCE_PCLK1 PCLK1 selected as USART1 clock
 - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
 - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

34 HAL RCC Extension Driver

34.1 RCCEx Firmware driver registers structures

34.1.1 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the `stm32f0xx_hal_rcc_ex.h`

Data Fields

- *uint32_t PeriphClockSelection*
- *uint32_t RTCClockSelection*
- *uint32_t Usart1ClockSelection*
- *uint32_t Usart2ClockSelection*
- *uint32_t Usart3ClockSelection*
- *uint32_t I2c1ClockSelection*
- *uint32_t CecClockSelection*

Field Documentation

- *uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection*
The Extended Clock to be configured. This parameter can be a value of [RCCEEx_Periph_Clock_Selection](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection*
Specifies RTC Clock Prescalers Selection This parameter can be a value of [RCC_RTC_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection*
USART1 clock source This parameter can be a value of [RCC_USART1_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection*
USART2 clock source This parameter can be a value of [RCCEEx_USART2_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::Usart3ClockSelection*
USART3 clock source This parameter can be a value of [RCCEEx_USART3_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection*
I2C1 clock source This parameter can be a value of [RCC_I2C1_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::CecClockSelection*
HDMI CEC clock source This parameter can be a value of [RCCEEx_CEC_Clock_Source](#)

34.1.2 RCC_CRSInitTypeDef

RCC_CRSInitTypeDef is defined in the `stm32f0xx_hal_rcc_ex.h`

Data Fields

- *uint32_t Prescaler*
- *uint32_t Source*
- *uint32_t Polarity*
- *uint32_t ReloadValue*
- *uint32_t ErrorLimitValue*
- *uint32_t HSI48CalibrationValue*

Field Documentation

- *uint32_t RCC_CRSInitTypeDef::Prescaler*
Specifies the division factor of the SYNC signal. This parameter can be a value of [RCCEEx_CRS_SynchroDivider](#)
- *uint32_t RCC_CRSInitTypeDef::Source*
Specifies the SYNC signal source. This parameter can be a value of [RCCEEx_CRS_SynchroSource](#)

- **`uint32_t RCC_CRSInitTypeDef::Polarity`**
Specifies the input polarity for the SYNC signal source. This parameter can be a value of `RCCEEx_CRS_SynchroPolarity`
- **`uint32_t RCC_CRSInitTypeDef::ReloadValue`**
Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro `_HAL_RCC_CRS_RELOADVALUE_CALCULATE(_FTARGET_, __FSYNC__)` This parameter must be a number between 0 and 0xFFFF or a value of `RCCEEx_CRS_ReloadValueDefault`.
- **`uint32_t RCC_CRSInitTypeDef::ErrorLimitValue`**
Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of `RCCEEx_CRS_ErrorLimitDefault`
- **`uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue`**
Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of `RCCEEx_CRS_HSI48CalibrationDefault`

34.1.3 **RCC_CRSSynchroInfoTypeDef**

`RCC_CRSSynchroInfoTypeDef` is defined in the `stm32f0xx_hal_rcc_ex.h`

Data Fields

- `uint32_t ReloadValue`
- `uint32_t HSI48CalibrationValue`
- `uint32_t FreqErrorCapture`
- `uint32_t FreqErrorDirection`

Field Documentation

- **`uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue`**
Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFFU
- **`uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue`**
Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3FU
- **`uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture`**
Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFFU
- **`uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection`**
Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of `RCCEEx_CRS_FreqErrorDirection`

34.2 RCCEEx Firmware driver API description

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

34.2.1 Extended Peripheral Control functions

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

Note:

Important note: Care must be taken when `HAL_RCCEEx_PeriphCLKConfig()` is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.

This section contains the following APIs:

- `HAL_RCCEEx_PeriphCLKConfig`
- `HAL_RCCEEx_GetPeriphCLKConfig`
- `HAL_RCCEEx_GetPeriphCLKFreq`

34.2.2 Extended Clock Recovery System Control functions

For devices with Clock Recovery System feature (CRS), RCC Extention HAL driver can be used as follows:

1. In System clock config, HSI48 needs to be enabled

2. Enable CRS clock in IP MSP init which will use CRS functions
3. Call CRS functions as follows:
 - a. Prepare synchronization configuration necessary for HSI48 calibration
 - Default values can be set for frequency Error Measurement (reload and error limit) and also HSI48 oscillator smooth trimming.
 - Macro @ref __HAL_RCC_CRS_RELOADVALUE_CALCULATE can be also used to calculate directly reload value with target and synchronization frequencies values
 - b. Call function @ref HAL_RCCEEx_CRSConfig which
 - Reset CRS registers to their default values.
 - Configure CRS registers with synchronization configuration
 - Enable automatic calibration and frequency error counter feature Note: When using USB LPM (Link Power Management) and the device is in Sleep mode, the periodic USB SOF will not be generated by the host. No SYNC signal will therefore be provided to the CRS to calibrate the HSI48 on the run. To guarantee the required clock precision after waking up from Sleep mode, the LSE or reference clock on the GPIOs should be used as SYNC signal.
 - c. A polling function is provided to wait for complete synchronization
 - Call function @ref HAL_RCCEEx_CRSWaitSynchronization()
 - According to CRS status, user can decide to adjust again the calibration or continue application if synchronization is OK
4. User can retrieve information related to synchronization in calling function @ref HAL_RCCEEx_CRSGetSynchronizationInfo()
5. Regarding synchronization status and synchronization information, user can try a new calibration in changing synchronization configuration and call again HAL_RCCEEx_CRSConfig. Note: When the SYNC event is detected during the downcounting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value should be incremented), while when it is detected during the upcounting phase it means that the actual frequency is higher (and that the TRIM value should be decremented).
6. In interrupt mode, user can resort to the available macros (__HAL_RCC_CRS_XXX_IT). Interrupts will go through CRS Handler (RCC IRQn/RCC_IRQHandler)
 - Call function @ref HAL_RCCEEx_CRSConfig()
 - Enable RCC IRQn (thanks to NVIC functions)
 - Enable CRS interrupt (@ref __HAL_RCC_CRS_ENABLE_IT)
 - Implement CRS status management in the following user callbacks called from HAL_RCCEEx_CRS_IRQHandler():
 - @ref HAL_RCCEEx_CRS_SyncOkCallback()
 - @ref HAL_RCCEEx_CRS_SyncWarnCallback()
 - @ref HAL_RCCEEx_CRS_ExpectedSyncCallback()
 - @ref HAL_RCCEEx_CRS_ErrorCallback()
7. To force a SYNC EVENT, user can use the function @ref HAL_RCCEEx_CRSSoftwareSynchronizationGenerate(). This function can be called before calling @ref HAL_RCCEEx_CRSConfig (for instance in Systick handler)

This section contains the following APIs:

- [**HAL_RCCEEx_CRSConfig**](#)
- [**HAL_RCCEEx_CRSSoftwareSynchronizationGenerate**](#)
- [**HAL_RCCEEx_CRSGetSynchronizationInfo**](#)
- [**HAL_RCCEEx_CRSWaitSynchronization**](#)
- [**HAL_RCCEEx_CRS_IRQHandler**](#)
- [**HAL_RCCEEx_CRS_SyncOkCallback**](#)
- [**HAL_RCCEEx_CRS_SyncWarnCallback**](#)
- [**HAL_RCCEEx_CRS_ExpectedSyncCallback**](#)
- [**HAL_RCCEEx_CRS_ErrorCallback**](#)

34.2.3 Detailed description of functions

HAL_RCCEEx_PeriphCLKConfig

Function name	HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
Function description	Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.
Parameters	<ul style="list-style-type: none">PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks (USART, RTC, I2C, CEC and USB).
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">Care must be taken when HAL_RCCEEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.

HAL_RCCEEx_GetPeriphCLKConfig

Function name	void HAL_RCCEEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
Function description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none">PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks (USART, RTC, I2C, CEC and USB).
Return values	<ul style="list-style-type: none">None:

HAL_RCCEEx_GetPeriphCLKFreq

Function name	uint32_t HAL_RCCEEx_GetPeriphCLKFreq (uint32_t PeriphClk)
Function description	Returns the peripheral clock frequency.
Parameters	<ul style="list-style-type: none">PeriphClk: Peripheral clock identifier This parameter can be one of the following values:<ul style="list-style-type: none">RCC_PERIPHCLK_RTC RTC peripheral clockRCC_PERIPHCLK_USART1 USART1 peripheral clockRCC_PERIPHCLK_I2C1 I2C1 peripheral clockRCC_PERIPHCLK_USART2 USART2 peripheral clockRCC_PERIPHCLK_USART3 USART3 peripheral clockRCC_PERIPHCLK_CEC CEC peripheral clock
Return values	<ul style="list-style-type: none">Frequency: in Hz (0: means that no available frequency for the peripheral)
Notes	<ul style="list-style-type: none">Returns 0 if peripheral clock is unknown

HAL_RCCEEx_CRSConfig

Function name `void HAL_RCCEEx_CRSConfig (RCC_CRSInitTypeDef * pInit)`

Function description Start automatic synchronization for polling mode.

Parameters • `pInit`: Pointer on RCC_CRSInitTypeDef structure

Return values • `None`:

HAL_RCCEEx_CRSSoftwareSynchronizationGenerate

Function name `void HAL_RCCEEx_CRSSoftwareSynchronizationGenerate (void)`

Function description Generate the software synchronization event.

Return values • `None`:

HAL_RCCEEx_CRSGetSynchronizationInfo

Function name `void HAL_RCCEEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)`

Function description Return synchronization info.

Parameters • `pSynchroInfo`: Pointer on RCC_CRSSynchroInfoTypeDef structure

Return values • `None`:

HAL_RCCEEx_CRSWaitSynchronization

Function name `uint32_t HAL_RCCEEx_CRSWaitSynchronization (uint32_t Timeout)`

Function description Wait for CRS Synchronization status.

Parameters • `Timeout`: Duration of the timeout

Return values • **Combination:** of Synchronization status This parameter can be a combination of the following values:

- `RCC_CRS_TIMEOUT`
- `RCC_CRS_SYNCOK`
- `RCC_CRS_SYNCWARN`
- `RCC_CRS_SYNCERR`
- `RCC_CRS_SYNCMISS`
- `RCC_CRS_TRIMOVF`

Notes • Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.
• If Timeout set to `HAL_MAX_DELAY`, `HAL_TIMEOUT` will be never returned.

HAL_RCCEEx_CRS_IRQHandler

Function name **void HAL_RCCEEx_CRS_IRQHandler (void)**

Function description Handle the Clock Recovery System interrupt request.

Return values • None:

HAL_RCCEEx_CRS_SyncOkCallback

Function name **void HAL_RCCEEx_CRS_SyncOkCallback (void)**

Function description RCCEEx Clock Recovery System SYNCOK interrupt callback.

Return values • none:

HAL_RCCEEx_CRS_SyncWarnCallback

Function name **void HAL_RCCEEx_CRS_SyncWarnCallback (void)**

Function description RCCEEx Clock Recovery System SYNCWARN interrupt callback.

Return values • none:

HAL_RCCEEx_CRS_ExpectedSyncCallback

Function name **void HAL_RCCEEx_CRS_ExpectedSyncCallback (void)**

Function description RCCEEx Clock Recovery System Expected SYNC interrupt callback.

Return values • none:

HAL_RCCEEx_CRS_ErrorCallback

Function name **void HAL_RCCEEx_CRS_ErrorCallback (uint32_t Error)**

Function description RCCEEx Clock Recovery System Error interrupt callback.

Parameters • **Error:** Combination of Error status. This parameter can be a combination of the following values:
– RCC_CRS_SYNCERR
– RCC_CRS_SYNCMISS
– RCC_CRS_TRIMOVF

Return values • none:

34.3 RCCEEx Firmware driver defines

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

34.3.1 RCCEEx

RCCEEx CEC Clock Source

`RCC_CECCLKSOURCE_HS`

|

`RCC_CECCLKSOURCE_LS`
E

RCCEEx CRS Default Error Limit Value

`RCC_CRS_ERRORLIMIT_D` Default Frequency error limit
EFAULT

RCCEEx CRS Extended Features

`_HAL_RCC_CRS_FREQ_E` Description:

`RROR_COUNTER_ENABLE` • Enable the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFG register becomes write-protected.

`_HAL_RCC_CRS_FREQ_E` Description:

`RROR_COUNTER_DISABLE` • Disable the oscillator clock for frequency error counter.
E

Return value:

- None

`_HAL_RCC_CRS_AUTOM` Description:

`ATIC_CALIB_ENABLE` • Enable the automatic hardware adjustement of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFG register becomes write-protected.

`_HAL_RCC_CRS_AUTOM` Description:

`ATIC_CALIB_DISABLE` • Disable the automatic hardware adjustement of TRIM bits.

Return value:

- None

__HAL_RCC_CRS_RELOAD **Description:****DVALUE_CALCULATE**

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- FTARGET: Target frequency (value in Hz)
- FSYNC: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (FTARGET / fSYNC) -1

RCCEEx CRS Flags**RCC_CRS_FLAG_SYNCOK** SYNC event OK flag**RCC_CRS_FLAG_SYNCWA** SYNC warning flag
RN**RCC_CRS_FLAG_ERR** Error flag**RCC_CRS_FLAG_ESYNC** Expected SYNC flag**RCC_CRS_FLAG_SYNCER** SYNC error
R**RCC_CRS_FLAG_SYNCMI** SYNC missed
SS**RCC_CRS_FLAG_TRIMOV** Trimming overflow or underflow
F***RCCEEx CRS Frequency Error Direction*****RCC_CRS_FREQERRORDI** Upcounting direction, the actual frequency is above the target
R_UP**RCC_CRS_FREQERRORDI** Downcounting direction, the actual frequency is below the target
R_DOWN***RCCEEx CRS Default HSI48 Calibration value*****RCC_CRS_HSI48CALIBRATION_DEFAULT** The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency***RCCEEx CRS Interrupt Sources*****RCC_CRS_IT_SYNCOK** SYNC event OK**RCC_CRS_IT_SYNCWARN** SYNC warning

RCC_CRS_IT_ERR	Error
RCC_CRS_IT_ESYNC	Expected SYNC
RCC_CRS_IT_SYNCERR	SYNC error
RCC_CRS_IT_SYNCMISS	SYNC missed
RCC_CRS_IT_TRIMOVF	Trimming overflow or underflow

RCCEEx CRS Default Reload Value

RCC_CRS_RELOADVALUE_DEFAULT The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEEx CRS Status

RCC_CRS_NONE
RCC_CRS_TIMEOUT
RCC_CRS_SYNCOK
RCC_CRS_SYNCWARN
RCC_CRS_SYNCERR
RCC_CRS_SYNCMISS
RCC_CRS_TRIMOVF

RCCEEx CRS Synchronization Divider

RCC_CRS_SYNC_DIV1	Synchro Signal not divided (default)
RCC_CRS_SYNC_DIV2	Synchro Signal divided by 2
RCC_CRS_SYNC_DIV4	Synchro Signal divided by 4
RCC_CRS_SYNC_DIV8	Synchro Signal divided by 8
RCC_CRS_SYNC_DIV16	Synchro Signal divided by 16
RCC_CRS_SYNC_DIV32	Synchro Signal divided by 32
RCC_CRS_SYNC_DIV64	Synchro Signal divided by 64
RCC_CRS_SYNC_DIV128	Synchro Signal divided by 128

RCCEEx CRS Synchronization Polarity

RCC_CRS_SYNC_POLARIT Synchro Active on rising edge (default)
Y_RISING

RCC_CRS_SYNC_POLARIT Synchro Active on falling edge
Y_FALLING

RCCEEx CRS Synchronization Source

RCC_CRS_SYNC_SOURCE Synchro Signal source GPIO
_GPIO

RCC_CRS_SYNC_SOURCE Synchro Signal source LSE
_LSE

RCC_CRS_SYNC_SOURCE Synchro Signal source USB SOF (default)
_USB

RCCEEx Force Release Peripheral Reset

**_HAL_RCC_GPIOD_FORC
E_RESET**

**_HAL_RCC_GPIOD_RELEASE
RESET**

**_HAL_RCC_GPIOE_FORC
E_RESET**

**_HAL_RCC_GPIOE_RELEASE
RESET**

**_HAL_RCC_TSC_FORCE_
RESET**

**_HAL_RCC_TSC_RELEASE
RESET**

**_HAL_RCC_USART2_FORCE_
RESET**

**_HAL_RCC_SPI2_FORCE
RESET**

**_HAL_RCC_USART2_RELEASE
RESET**

**_HAL_RCC_SPI2_RELEASE
RESET**

**_HAL_RCC_TIM2_FORCE
RESET**

**_HAL_RCC_TIM2_RELEASE
RESET**

_HAL_RCC_TIM6_FORCE_RESET

_HAL_RCC_I2C2_FORCE_RESET

_HAL_RCC_TIM6_RELEASE_RESET

_HAL_RCC_I2C2_RELEASE_RESET

_HAL_RCC_DAC1_FORCE_RESET

_HAL_RCC_DAC1_RELEASE_RESET

_HAL_RCC_CEC_FORCE_RESET

_HAL_RCC_CEC_RELEASE_RESET

_HAL_RCC_TIM7_FORCE_RESET

_HAL_RCC_USART3_FORCE_RESET

_HAL_RCC_USART4_FORCE_RESET

_HAL_RCC_TIM7_RELEASE_RESET

_HAL_RCC_USART3_RELEASE_RESET

_HAL_RCC_USART4_RELEASE_RESET

_HAL_RCC_CAN1_FORCE_RESET

_HAL_RCC_CAN1_RELEASE_RESET

_HAL_RCC_CRS_FORCE_RESET

_HAL_RCC_CRS_RELEASE_RESET

`_HAL_RCC_USART5_FOR
CE_RESET`

`_HAL_RCC_USART5_REL
EASE_RESET`

`_HAL_RCC_TIM15_FORC
E_RESET`

`_HAL_RCC_TIM15_RELEASE
ASE_RESET`

`_HAL_RCC_USART6_FOR
CE_RESET`

`_HAL_RCC_USART6_REL
EASE_RESET`

`_HAL_RCC_USART7_FOR
CE_RESET`

`_HAL_RCC_USART8_FOR
CE_RESET`

`_HAL_RCC_USART7_REL
EASE_RESET`

`_HAL_RCC_USART8_REL
EASE_RESET`

RCCEEx HSI48 Enable Disable

`_HAL_RCC_HSI48_ENAB
LE`

`_HAL_RCC_HSI48_DISAB
LE`

`_HAL_RCC_GET_HSI48_S` **Description:**

- Macro to get the Internal 48Mhz High Speed oscillator (HSI48) state.

Return value:

- The: clock source can be one of the following values:
 - RCC_HSI48_ON HSI48 enabled
 - RCC_HSI48_OFF HSI48 disabled

RCCEEx IT and Flag

__HAL_RCC_CRS_ENABL **Description:****E_IT**

- Enable the specified CRS interrupts.

Parameters:

- **__INTERRUPT__**: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:
 - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
 - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
 - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
 - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_DISABL **Description:****E_IT**

- Disable the specified CRS interrupts.

Parameters:

- **__INTERRUPT__**: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
 - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
 - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
 - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
 - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_GET_IT **Description:****SOURCE**

- Check whether the CRS interrupt has occurred or not.

Parameters:

- **__INTERRUPT__**: specifies the CRS interrupt source to check. This parameter can be one of the following values:
 - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
 - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
 - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
 - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

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

__HAL_RCC_CRS_CLEAR_ **Description:****IT**

- Clear the CRS interrupt pending bits.

Parameters:

- **__INTERRUPT__**: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
 - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
 - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
 - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
 - RCC_CRS_IT_ESYNC Expected SYNC interrupt
 - RCC_CRS_IT_TRIMOVF Trimming overflow or underflow interrupt
 - RCC_CRS_IT_SYNCERR SYNC error interrupt
 - RCC_CRS_IT_SYNCMISS SYNC missed interrupt

_HAL_RCC_CRS_GET_FL Description:

AG

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

Parameters:

- _FLAG_: specifies the flag to check. This parameter can be one of the following values:
 - RCC_CRS_FLAG_SYNCOK SYNC event OK
 - RCC_CRS_FLAG_SYNCWARN SYNC warning
 - RCC_CRS_FLAG_ERR Error
 - RCC_CRS_FLAG_ESYNC Expected SYNC
 - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
 - RCC_CRS_FLAG_SYNCERR SYNC error
 - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

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

_HAL_RCC_CRS_CLEAR_FLAG Description:

FLAG

- Clear the CRS specified FLAG.

Parameters:

- _FLAG_: specifies the flag to clear. This parameter can be one of the following values:
 - RCC_CRS_FLAG_SYNCOK SYNC event OK
 - RCC_CRS_FLAG_SYNCWARN SYNC warning
 - RCC_CRS_FLAG_ERR Error
 - RCC_CRS_FLAG_ESYNC Expected SYNC
 - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
 - RCC_CRS_FLAG_SYNCERR SYNC error
 - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- None

Notes:

- RCC_CRS_FLAG_ERR clears RCC_CRS_FLAG_TRIMOVF, RCC_CRS_FLAG_SYNCERR, RCC_CRS_FLAG_SYNCMISS and consequently RCC_CRS_FLAG_ERR

RCC LSE Drive Configuration**RCC_LSEDRIVE_LOW** Xtal mode lower driving capability**RCC_LSEDRIVE_MEDIUMOW** Xtal mode medium low driving capability**RCC_LSEDRIVE_MEDIUMIGH** Xtal mode medium high driving capability**RCC_LSEDRIVE_HIGH** Xtal mode higher driving capability**LSE Drive Configuration**

__HAL_RCC_LSEDRIVE_C Description:

ONFIG

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- RCC_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

RCC Extended MCOx Clock Config

__HAL_RCC_MCO1_CONF1 Description:

G

- Macro to configure the MCO clock.

Parameters:

- MCOCLKSOURCE: specifies the MCO clock source. This parameter can be one of the following values:
 - RCC_MCO1SOURCE_NOCLOCK No clock selected as MCO clock
 - RCC_MCO1SOURCE_SYSCLK System Clock selected as MCO clock
 - RCC_MCO1SOURCE_HSI HSI oscillator clock selected as MCO clock
 - RCC_MCO1SOURCE_HSE HSE selected as MCO clock
 - RCC_MCO1SOURCE_LSI LSI selected as MCO clock
 - RCC_MCO1SOURCE_LSE LSE selected as MCO clock
 - RCC_MCO1SOURCE_HSI14 HSI14 selected as MCO clock
 - RCC_MCO1SOURCE_HSI48 HSI48 selected as MCO clock
 - RCC_MCO1SOURCE_PLLCLK PLLCLK selected as MCO clock
 - RCC_MCO1SOURCE_PLLCLK_DIV2 PLLCLK Divided by 2 selected as MCO clock
- MCODIV: specifies the MCO clock prescaler. This parameter can be one of the following values:
 - RCC_MCODIV_1 MCO clock source is divided by 1
 - RCC_MCODIV_2 MCO clock source is divided by 2
 - RCC_MCODIV_4 MCO clock source is divided by 4
 - RCC_MCODIV_8 MCO clock source is divided by 8
 - RCC_MCODIV_16 MCO clock source is divided by 16
 - RCC_MCODIV_32 MCO clock source is divided by 32
 - RCC_MCODIV_64 MCO clock source is divided by 64
 - RCC_MCODIV_128 MCO clock source is divided by 128

RCCEEx MCOx Clock Prescaler

RCC_MCODIV_1

RCC_MCODIV_2

RCC_MCODIV_4

RCC_MCODIV_8

[RCC_MCODIV_16](#)

[RCC_MCODIV_32](#)

[RCC_MCODIV_64](#)

[RCC_MCODIV_128](#)

RCCEx_Peripheral_Clock_Enable_Disable

[__HAL_RCC_GPIOD_CLK_ENABLE](#)

[__HAL_RCC_GPIOD_CLK_DISABLE](#)

[__HAL_RCC_GPIOE_CLK_ENABLE](#)

[__HAL_RCC_GPIOE_CLK_DISABLE](#)

[__HAL_RCC_TSC_CLK_ENABLE](#)

[__HAL_RCC_TSC_CLK_DISABLE](#)

[__HAL_RCC_DMA2_CLK_ENABLE](#)

[__HAL_RCC_DMA2_CLK_DISABLE](#)

[__HAL_RCC_USART2_CLK_ENABLE](#)

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

[__HAL_RCC_USART2_CLK_DISABLE](#)

[__HAL_RCC_SPI2_CLK_ENABLE](#)

[__HAL_RCC_SPI2_CLK_DISABLE](#)

[__HAL_RCC_TIM2_CLK_ENABLE](#)

[__HAL_RCC_TIM2_CLK_DISABLE](#)

`_HAL_RCC_TIM6_CLK_ENABLE`

`_HAL_RCC_I2C2_CLK_ENABLE`

`_HAL_RCC_TIM6_CLK_DISABLE`

`_HAL_RCC_I2C2_CLK_DISABLE`

`_HAL_RCC_DAC1_CLK_ENABLE`

`_HAL_RCC_DAC1_CLK_DISABLE`

`_HAL_RCC_CEC_CLK_ENABLE`

`_HAL_RCC_CEC_CLK_DISABLE`

`_HAL_RCC_TIM7_CLK_ENABLE`

`_HAL_RCC_USART3_CLK_ENABLE`

`_HAL_RCC_USART4_CLK_ENABLE`

`_HAL_RCC_TIM7_CLK_DISABLE`

`_HAL_RCC_USART3_CLK_DISABLE`

`_HAL_RCC_USART4_CLK_DISABLE`

`_HAL_RCC_CAN1_CLK_ENABLE`

`_HAL_RCC_CAN1_CLK_DISABLE`

`_HAL_RCC_CRS_CLK_ENABLE`

`_HAL_RCC_CRS_CLK_DISABLE`

`_HAL_RCC_USART5_CLK_ENABLE`

`_HAL_RCC_USART5_CLK_DISABLE`

`_HAL_RCC_TIM15_CLK_E` Notes:

NABLE

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

`_HAL_RCC_TIM15_CLK_DISABLE`

`_HAL_RCC_USART6_CLK_ENABLE`

`_HAL_RCC_USART6_CLK_DISABLE`

`_HAL_RCC_USART7_CLK_ENABLE`

`_HAL_RCC_USART8_CLK_ENABLE`

`_HAL_RCC_USART7_CLK_DISABLE`

`_HAL_RCC_USART8_CLK_DISABLE`

Peripheral Clock Enable Disable Status

`_HAL_RCC_GPIOD_IS_CLK_ENABLED`

`_HAL_RCC_GPIOD_IS_CLK_DISABLED`

`_HAL_RCC_GPIOE_IS_CLK_ENABLED`

`_HAL_RCC_GPIOE_IS_CLK_DISABLED`

`_HAL_RCC_TSC_IS_CLK_ENABLED`

`_HAL_RCC_TSC_IS_CLK_DISABLED`

`_HAL_RCC_DMA2_IS_CLK_ENABLED`

__HAL_RCC_DMA2_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_DAC1_IS_CLK_ENABLED

__HAL_RCC_DAC1_IS_CLK_DISABLED

__HAL_RCC_CEC_IS_CLK_ENABLED

__HAL_RCC_CEC_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_USART4_IS_CLK_ENABLED

_HAL_RCC_TIM7_IS_CLK
_DISABLED

_HAL_RCC_USART3_IS_CLK
_DISABLED

_HAL_RCC_USART4_IS_CLK
_DISABLED

_HAL_RCC_CAN1_IS_CLK
_ENABLED

_HAL_RCC_CAN1_IS_CLK
_DISABLED

_HAL_RCC_CRS_IS_CLK
_ENABLED

_HAL_RCC_CRS_IS_CLK
_DISABLED

_HAL_RCC_USART5_IS_CLK
_ENABLED

_HAL_RCC_USART5_IS_CLK
_DISABLED

_HAL_RCC_TIM15_IS_CLK
_ENABLED

_HAL_RCC_TIM15_IS_CLK
_DISABLED

_HAL_RCC_USART6_IS_CLK
_ENABLED

_HAL_RCC_USART6_IS_CLK
_DISABLED

_HAL_RCC_USART7_IS_CLK
_ENABLED

_HAL_RCC_USART8_IS_CLK
_ENABLED

_HAL_RCC_USART7_IS_CLK
_DISABLED

_HAL_RCC_USART8_IS_CLK
_DISABLED

RCCEEx Peripheral Clock Source Config

_HAL_RCC_CEC_CONFIG Description:

- Macro to configure the CEC clock.

Parameters:

- _CECCLKSOURCE_: specifies the CEC clock source. This parameter can be one of the following values:
 - RCC_CECCLKSOURCE_HSI HSI selected as CEC clock
 - RCC_CECCLKSOURCE_LSE LSE selected as CEC clock

_HAL_RCC_GET_CEC_SOURCE Description:

- Macro to get the HDMI CEC clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_CECCLKSOURCE_HSI HSI selected as CEC clock
 - RCC_CECCLKSOURCE_LSE LSE selected as CEC clock

_HAL_RCC_USART2_CONFIG Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- _USART2CLKSOURCE_: specifies the USART2 clock source. This parameter can be one of the following values:
 - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
 - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
 - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
 - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

_HAL_RCC_GET_USART2_SOURCE Description:

- Macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
 - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
 - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
 - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

_HAL_RCC_USART3_CONFIG Description:

- Macro to configure the USART3 clock (USART3CLK).

Parameters:

- _USART3CLKSOURCE_: specifies the USART3 clock source. This parameter can be one of the following values:
 - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
 - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
 - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

_HAL_RCC_GET_USART3 Description:**_SOURCE**

- Macro to get the USART3 clock source.

Return value:

- The clock source can be one of the following values:
 - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
 - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
 - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

RCCEx Periph Clock Selection**RCC_PERIPHCLK_USART1****RCC_PERIPHCLK_USART2****RCC_PERIPHCLK_I2C1****RCC_PERIPHCLK_CEC****RCC_PERIPHCLK_RTC****RCC_PERIPHCLK_USART3*****RCCEx USART2 Clock Source*****RCC_USART2CLKSOURCE_PCLK1****RCC_USART2CLKSOURCE_SYSCLK****RCC_USART2CLKSOURCE_LSE****RCC_USART2CLKSOURCE_HSI*****RCCEx USART3 Clock Source*****RCC_USART3CLKSOURCE_PCLK1****RCC_USART3CLKSOURCE_SYSCLK****RCC_USART3CLKSOURCE_LSE****RCC_USART3CLKSOURCE_HSI**

35 HAL RTC Generic Driver

35.1 RTC Firmware driver registers structures

35.1.1 RTC_InitTypeDef

`RTC_InitTypeDef` is defined in the `stm32f0xx_hal_rtc.h`

Data Fields

- `uint32_t HourFormat`
- `uint32_t AsynchPrediv`
- `uint32_t SynchPrediv`
- `uint32_t OutPut`
- `uint32_t OutPutPolarity`
- `uint32_t OutPutType`

Field Documentation

- `uint32_t RTC_InitTypeDef::HourFormat`

Specifies the RTC Hour Format. This parameter can be a value of [`RTC_Hour_Formats`](#)

- `uint32_t RTC_InitTypeDef::AsynchPrediv`

Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F

- `uint32_t RTC_InitTypeDef::SynchPrediv`

Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF

- `uint32_t RTC_InitTypeDef::OutPut`

Specifies which signal will be routed to the RTC output. This parameter can be a value of [`RTCEx_Output_selection_Definitions`](#)

- `uint32_t RTC_InitTypeDef::OutPutPolarity`

Specifies the polarity of the output signal. This parameter can be a value of [`RTC_Output_Polarity_Definitions`](#)

- `uint32_t RTC_InitTypeDef::OutPutType`

Specifies the RTC Output Pin mode. This parameter can be a value of [`RTC_Output_Type_ALARM_OUT`](#)

35.1.2 RTC_TimeTypeDef

`RTC_TimeTypeDef` is defined in the `stm32f0xx_hal_rtc.h`

Data Fields

- `uint8_t Hours`
- `uint8_t Minutes`
- `uint8_t Seconds`
- `uint8_t TimeFormat`
- `uint32_t SubSeconds`
- `uint32_t SecondFraction`
- `uint32_t DayLightSaving`
- `uint32_t StoreOperation`

Field Documentation

- `uint8_t RTC_TimeTypeDef::Hours`

Specifies the RTC Time Hour. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the `RTC_HourFormat_12` is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the `RTC_HourFormat_24` is selected

- **`uint8_t RTC_TimeTypeDef::Minutes`**
Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- **`uint8_t RTC_TimeTypeDef::Seconds`**
Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- **`uint8_t RTC_TimeTypeDef::TimeFormat`**
Specifies the RTC AM/PM Time. This parameter can be a value of [RTC_AM_PM_Definitions](#)
- **`uint32_t RTC_TimeTypeDef::SubSeconds`**
Specifies the RTC_SSR RTC Sub Second register content. This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity
- **`uint32_t RTC_TimeTypeDef::SecondFraction`**
Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- **`uint32_t RTC_TimeTypeDef::DayLightSaving`**
Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of [RTC_DayLightSaving_Definitions](#)
- **`uint32_t RTC_TimeTypeDef::StoreOperation`**
Specifies RTC_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of [RTC_StoreOperation_Definitions](#)

35.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the `stm32f0xx_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

35.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the `stm32f0xx_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*](#)

35.1.5 RTC_HandleTypeDef

RTC_HandleTypeDef is defined in `stm32f0xx_hal_rtc.h`

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***

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

35.2 RTC Firmware driver API description

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

35.2.1 How to use RTC Driver

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

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the `HAL_RTC_SetTime()` and `HAL_RTC_SetDate()` functions.
- To read the RTC Calendar, use the `HAL_RTC_GetTime()` and `HAL_RTC_GetDate()` functions.

Alarm configuration

- To configure the RTC Alarm use the `HAL_RTC_SetAlarm()` function. You can also configure the RTC Alarm with interrupt mode using the `HAL_RTC_SetAlarm_IT()` function.
- To read the RTC Alarm, use the `HAL_RTC_GetAlarm()` function.

35.2.2 RTC and low power modes

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

The RTC alternate functions are the RTC alarm (Alarm A), RTC wake-up, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

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

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

35.2.3 Initialization and de-initialization functions

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

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
 - A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
 - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wake-up 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_DelInit`](#)
- [`HAL_RTC_MsInit`](#)
- [`HAL_RTC_MspDelInit`](#)

35.2.4 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- [`HAL_RTC_SetTime`](#)
- [`HAL_RTC_GetTime`](#)
- [`HAL_RTC_SetDate`](#)
- [`HAL_RTC_GetDate`](#)

35.2.5 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- [`HAL_RTC_SetAlarm`](#)
- [`HAL_RTC_SetAlarm_IT`](#)
- [`HAL_RTC_DeactivateAlarm`](#)
- [`HAL_RTC_GetAlarm`](#)
- [`HAL_RTC_AlarmIRQHandler`](#)
- [`HAL_RTC_AlarmAEventCallback`](#)
- [`HAL_RTC_PollForAlarmAEvent`](#)

35.2.6 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- [*HAL_RTC_WaitForSyncro*](#)

35.2.7 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- [*HAL_RTC_GetState*](#)

35.2.8 Detailed description of functions

[*HAL_RTC_Init*](#)

Function name [**HAL_StatusTypeDef HAL_RTC_Init \(RTC_HandleTypeDef * hrtc\)**](#)

Function description Initialize the RTC according to the specified parameters in the RTC_InitTypeDef structure and initialize the associated handle.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

[*HAL_RTC_DeInit*](#)

Function name [**HAL_StatusTypeDef HAL_RTC_DeInit \(RTC_HandleTypeDef * hrtc\)**](#)

Function description Deinitialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This function doesn't reset the RTC Backup Data registers.

[*HAL_RTC_MspInit*](#)

Function name [**void HAL_RTC_MspInit \(RTC_HandleTypeDef * hrtc\)**](#)

Function description Initialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

[*HAL_RTC_MspDeInit*](#)

Function name [**void HAL_RTC_MspDeInit \(RTC_HandleTypeDef * hrtc\)**](#)

Function description Deinitialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_SetTime

Function name **HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)**

Function description Set RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - **RTC_FORMAT_BIN:** Binary data format
 - **RTC_FORMAT_BCD:** BCD data format

Return values

- **HAL:** status

HAL_RTC_GetTime

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

Function description Get RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation.
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - **RTC_FORMAT_BIN:** Binary data format
 - **RTC_FORMAT_BCD:** BCD data format

Return values

- **HAL:** status

Notes

- You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula:
Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit
This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS
- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

HAL_RTC_SetDate

Function name	HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function description	Set RTC current date.
Parameters	<ul style="list-style-type: none">hrtc: RTC handlesDate: Pointer to date structureFormat: specifies the format of the entered parameters. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_FORMAT_BIN: Binary data format– RTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none">HAL: status

HAL_RTC_GetDate

Function name	HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function description	Get RTC current date.
Parameters	<ul style="list-style-type: none">hrtc: RTC handlesDate: Pointer to Date structureFormat: Specifies the format of the entered parameters. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_FORMAT_BIN : Binary data format– RTC_FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_SetAlarm

Function name	HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function description	Set the specified RTC Alarm.
Parameters	<ul style="list-style-type: none">hrtc: RTC handlesAlarm: Pointer to Alarm structureFormat: Specifies the format of the entered parameters. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_FORMAT_BIN: Binary data format– RTC_FORMAT_BCD: BCD data format

Return values	<ul style="list-style-type: none">• HAL: status
HAL_RTC_SetAlarm_IT	
Function name	HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function description	Set the specified RTC Alarm with Interrupt.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• sAlarm: Pointer to Alarm structure• Format: Specifies the format of the entered parameters. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_FORMAT_BIN: Binary data format– RTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	Deactivate the specified RTC Alarm.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• Alarm: Specifies the Alarm. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_ALARM_A: AlarmA
Return values	<ul style="list-style-type: none">• 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	Get the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• sAlarm: Pointer to Date structure• Alarm: Specifies the Alarm. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_ALARM_A: AlarmA• Format: Specifies the format of the entered parameters. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_FORMAT_BIN: Binary data format– RTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTC_AlarmIRQHandler

Function name	<code>void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)</code>
Function description	Handle Alarm interrupt request.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle
Return values	<ul style="list-style-type: none">• <code>None</code>:

HAL_RTC_PollForAlarmAEvent

Function name	<code>HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function description	Handle AlarmA Polling request.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle• <code>Timeout</code>: Timeout duration
Return values	<ul style="list-style-type: none">• <code>HAL</code>: status

HAL_RTC_AlarmAEventCallback

Function name	<code>void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)</code>
Function description	Alarm A callback.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle
Return values	<ul style="list-style-type: none">• <code>None</code>:

HAL_RTC_WaitForSynchro

Function name	<code>HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)</code>
Function description	Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle
Return values	<ul style="list-style-type: none">• <code>HAL</code>: status
Notes	<ul style="list-style-type: none">• The RTC Resynchronization mode is write protected, use the <code>__HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function.• To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState

Function name	HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)
Function description	Return the RTC handle state.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• HAL: state

RTC_EnterInitMode

Function name	HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)
Function description	Enter the RTC Initialization mode.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• The RTC Initialization mode is write protected, use the <code>_HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function.

RTC_ByteToBcd2

Function name	uint8_t RTC_ByteToBcd2 (uint8_t Value)
Function description	Convert a 2 digit decimal to BCD format.
Parameters	<ul style="list-style-type: none">• Value: Byte to be converted
Return values	<ul style="list-style-type: none">• Converted: byte

RTC_Bcd2ToByte

Function name	uint8_t RTC_Bcd2ToByte (uint8_t Value)
Function description	Convert from 2 digit BCD to Binary.
Parameters	<ul style="list-style-type: none">• Value: BCD value to be converted
Return values	<ul style="list-style-type: none">• Converted: word

35.3 RTC Firmware driver defines

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

35.3.1	RTC
	RTC
	RTC Alarm Date WeekDay Definitions

RTC_ALARMDATEWEEKD
AYSEL_DATE

RTC_ALARMDATEWEEKD
AYSEL_WEEKDAY

RTC Alarm Mask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATE
WEEKDAY

RTC_ALARMMASK_HOUR
S

RTC_ALARMMASK_MINUT
ES

RTC_ALARMMASK_SECO
NDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECOND All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
MASK_ALL

RTC_ALARMSUBSECOND SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
MASK_SS14_1

RTC_ALARMSUBSECOND SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
MASK_SS14_2

RTC_ALARMSUBSECOND SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared
MASK_SS14_3

RTC_ALARMSUBSECOND SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
MASK_SS14_4

RTC_ALARMSUBSECOND SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared
MASK_SS14_5

RTC_ALARMSUBSECOND SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
MASK_SS14_6

RTC_ALARMSUBSECOND SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared
MASK_SS14_7

RTC_ALARMSUBSECOND SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
MASK_SS14_8

RTC_ALARMSUBSECOND SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
MASK_SS14_9

RTC_ALARMSUBSECOND SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
MASK_SS14_10

RTC_ALARMSUBSECOND SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
MASK_SS14_11

RTC_ALARMSUBSECOND SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared
MASK_SS14_12

RTC_ALARMSUBSECOND SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
MASK_SS14_13

RTC_ALARMSUBSECOND SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
MASK_SS14

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

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC DayLight Saving Definitions

RTC_DAYLIGHTSAVING_S
UB1H

RTC_DAYLIGHTSAVING_A
DD1H

RTC_DAYLIGHTSAVING_N
ONE

RTC Exported Macros

__HAL_RTC_RESET_HAND **Description:**

LE_STATE • Reset RTC handle state.

Parameters:

• **__HANDLE__**: RTC handle.

Return value:

• None

_HAL_RTC_WRITEPROTE **Description:**
CTION_DISABLE

- Disable the write protection for RTC registers.

Parameters:

- **_HANDLE_**: specifies the RTC handle.

Return value:

- None

_HAL_RTC_WRITEPROTE **Description:**
CTION_ENABLE

- Enable the write protection for RTC registers.

Parameters:

- **_HANDLE_**: specifies the RTC handle.

Return value:

- None

_HAL_RTC_ALARMA **Description:**
ABLE

- Enable the RTC ALARMA peripheral.

Parameters:

- **_HANDLE_**: specifies the RTC handle.

Return value:

- None

_HAL_RTC_ALARMA **Description:**
ABLE

- Disable the RTC ALARMA peripheral.

Parameters:

- **_HANDLE_**: specifies the RTC handle.

Return value:

- None

_HAL_RTC_ALARM **Description:**
BLE_IT

- Enable the RTC Alarm interrupt.

Parameters:

- **_HANDLE_**: specifies the RTC handle.

• **_INTERRUPT_**: specifies the RTC Alarm interrupt sources to be enabled or disabled.
This parameter can be any combination of the following values:
– RTC_IT_ALRA: Alarm A interrupt

Return value:

- None

_HAL_RTC_ALARM **Description:**
BLE_IT

- Disable the RTC Alarm interrupt.

Parameters:

- **_HANDLE_**: specifies the RTC handle.

• **_INTERRUPT_**: specifies the RTC Alarm interrupt sources to be enabled or disabled.
This parameter can be any combination of the following values:
– RTC_IT_ALRA: Alarm A interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT Description:

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

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt to check. This parameter can be:
 - RTC_IT_ALRA: Alarm A interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT_SOURCE Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - RTC_IT_ALRA: Alarm A interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_FLAG Description:

- Get the selected RTC Alarm's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRAWF

Return value:

- None

__HAL_RTC_ALARM_CLE_AR_FLAG Description:

- Clear the RTC Alarm's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
 - RTC_FLAG_ALRAF

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_IT Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_IT Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

_HAL_RTC_ALARM_EXTI **Description:**

- _ENABLE_EVENT**
- Enable event on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _DISABLE_EVENT**
- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _ENABLE_FALLING_EDGE**
- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _DISABLE_FALLING_EDGE**
- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _ENABLE_RISING_EDGE**
- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _DISABLE_RISING_EDGE**
- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _ENABLE_RISING_FALLIN_G_EDGE**
- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _DISABLE_RISING_FALLIN_G_EDGE**
- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

- _GET_FLAG**
- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

_HAL_RTC_ALARM_EXTI **Description:**

CLEAR_FLAG

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None.

_HAL_RTC_ALARM_EXTI **Description:**

GENERATE_SWIT

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None.

RTC Flags Definitions

RTC_FLAG_RECALPF

RTC_FLAG_TAMP3F

RTC_FLAG_TAMP2F

RTC_FLAG_TAMP1F

RTC_FLAG_TSOVF

RTC_FLAG_TSF

RTC_FLAG_WUTF

RTC_FLAG_ALRAF

RTC_FLAG_INITF

RTC_FLAG_RSF

RTC_FLAG_INITS

RTC_FLAG_SHPF

RTC_FLAG_WUTWF

RTC_FLAG_ALRAWF

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input parameter format definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interrupts Definitions

`RTC_IT_TS`

`RTC_IT_WUT`

`RTC_IT_ALRA`

`RTC_IT_TAMP`

`RTC_IT_TAMP1`

`RTC_IT_TAMP2`

`RTC_IT_TAMP3`

RTC Private macros to check input parameters

`IS_RTC_HOUR_FORMAT`

`IS_RTC_OUTPUT_POL`

`IS_RTC_OUTPUT_TYPE`

`IS_RTC_HOUR12`

`IS_RTC_HOUR24`

`IS_RTC_ASYNCH_PREDIV`

`IS_RTC_SYNCH_PREDIV`

`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_WE
EKDAY_DATE

IS_RTC_ALARM_DATE_WE
EKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WE
EKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SEC
OND_VALUE

IS_RTC_ALARM_SUB_SEC
OND_MASK

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_
HIGH

RTC_OUTPUT_POLARITY_
LOW

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_OPE
NDRAIN

RTC_OUTPUT_TYPE_PUS
HPULL

RTC Store Operation Definitions

RTC_STOREOPERATION_
RESET

RTC_STOREOPERATION_S
ET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNES
DAY

RTC_WEEKDAY_THURSDA
Y

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDA
Y

RTC_WEEKDAY_SUNDAY

36 HAL RTC Extension Driver

36.1 RTCE Firmware driver registers structures

36.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the `stm32f0xx_hal_rtc_ex.h`

Data Fields

- *uint32_t Tamper*
- *uint32_t Trigger*
- *uint32_t Filter*
- *uint32_t SamplingFrequency*
- *uint32_t PrechargeDuration*
- *uint32_t TamperPullUp*
- *uint32_t TimeStampOnTamperDetection*

Field Documentation

- *uint32_t RTC_TamperTypeDef::Tamper*
Specifies the Tamper Pin. This parameter can be a value of [*RTCE_Ex_Tamper_Pins_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::Trigger*
Specifies the Tamper Trigger. This parameter can be a value of [*RTCE_Ex_Tamper_Trigger_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::Filter*
Specifies the RTC Filter Tamper. This parameter can be a value of [*RTCE_Ex_Tamper_Filter_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::SamplingFrequency*
Specifies the sampling frequency. This parameter can be a value of [*RTCE_Ex_Tamper_Sampling_Frequencies_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::PrechargeDuration*
Specifies the Precharge Duration . This parameter can be a value of [*RTCE_Ex_Tamper_Pin_Precharge_Duration_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::TamperPullUp*
Specifies the Tamper PullUp . This parameter can be a value of [*RTCE_Ex_Tamper_Pull_UP_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection*
Specifies the TimeStampOnTamperDetection. This parameter can be a value of [*RTCE_Ex_Tamper_TimeStampOnTamperDetection_Definitions*](#)

36.2 RTCE Firmware driver API description

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

36.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the `HAL_RTC_Init()` function.

RTC Wake-up configuration

Note: Not available on F030x4/x6/x8 and F070x6

TimeStamp configuration

- Configure the RTC_AF trigger and enable the RTC TimeStamp using the `HAL_RTCE_SetTimeStamp()` function. You can also configure the RTC TimeStamp with interrupt mode using the `HAL_RTCE_SetTimeStamp_IT()` function.

- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, precharge or discharge and Pull-UP using the HAL_RTCEEx_SetTamper() function. You can configure RTC Tamper in interrupt mode using HAL_RTCEEx_SetTamper_IT() function.

Backup Data Registers configuration

Note: Not available on F030x6/x8/xC and F070x6/xB (F0xx Value Line devices)

36.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- [`HAL_RTCEEx_SetTimeStamp`](#)
- [`HAL_RTCEEx_SetTimeStamp_IT`](#)
- [`HAL_RTCEEx_DeactivateTimeStamp`](#)
- [`HAL_RTCEEx_GetTimeStamp`](#)
- [`HAL_RTCEEx_SetTamper`](#)
- [`HAL_RTCEEx_SetTamper_IT`](#)
- [`HAL_RTCEEx_DeactivateTamper`](#)
- [`HAL_RTCEEx_TamperTimeStampIRQHandler`](#)
- [`HAL_RTCEEx_TimeStampEventCallback`](#)
- [`HAL_RTCEEx_Tamper1EventCallback`](#)
- [`HAL_RTCEEx_Tamper2EventCallback`](#)
- [`HAL_RTCEEx_Tamper3EventCallback`](#)
- [`HAL_RTCEEx_PollForTimeStampEvent`](#)
- [`HAL_RTCEEx_PollForTamper1Event`](#)
- [`HAL_RTCEEx_PollForTamper2Event`](#)
- [`HAL_RTCEEx_PollForTamper3Event`](#)

36.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- [`HAL_RTCEEx_SetWakeUpTimer`](#)
- [`HAL_RTCEEx_SetWakeUpTimer_IT`](#)
- [`HAL_RTCEEx_DeactivateWakeUpTimer`](#)
- [`HAL_RTCEEx_GetWakeUpTimer`](#)
- [`HAL_RTCEEx_WakeUpTimerIRQHandler`](#)
- [`HAL_RTCEEx_WakeUpTimerEventCallback`](#)
- [`HAL_RTCEEx_PollForWakeUpTimerEvent`](#)

36.2.4 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- 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_RTCEEx_BKUPWrite`](#)
- [`HAL_RTCEEx_BKUPRead`](#)
- [`HAL_RTCEEx_SetSmoothCalib`](#)
- [`HAL_RTCEEx_SetSynchroShift`](#)
- [`HAL_RTCEEx_SetCalibrationOutPut`](#)
- [`HAL_RTCEEx_DeactivateCalibrationOutPut`](#)
- [`HAL_RTCEEx_SetRefClock`](#)
- [`HAL_RTCEEx_DeactivateRefClock`](#)
- [`HAL_RTCEEx_EnableBypassShadow`](#)
- [`HAL_RTCEEx_DisableBypassShadow`](#)

36.2.5 Detailed description of functions

`HAL_RTCEEx_SetTimeStamp`

Function name	<code>HAL_StatusTypeDef HAL_RTCEEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</code>
Function description	Set TimeStamp.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• TimeStampEdge: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:<ul style="list-style-type: none">– <code>RTC_TIMESTAMPEDGE_RISING</code>: the Time stamp event occurs on the rising edge of the related pin.– <code>RTC_TIMESTAMPEDGE_FALLING</code>: the Time stamp event occurs on the falling edge of the related pin.• RTC_TimeStampPin: specifies the RTC TimeStamp Pin. This parameter can be one of the following values:<ul style="list-style-type: none">– <code>RTC_TIMESTAMPPIN_DEFAULT</code>: PC13 is selected as RTC TimeStamp Pin.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• This API must be called before enabling the TimeStamp feature.

`HAL_RTCEEx_SetTimeStamp_IT`

Function name	<code>HAL_StatusTypeDef HAL_RTCEEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</code>
Function description	Set TimeStamp with Interrupt.

Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• TimeStampEdge: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.– RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.• RTC_TimeStampPin: Specifies the RTC TimeStamp Pin. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• This API must be called before enabling the TimeStamp feature.

HAL_RTCEEx_DeactivateTimeStamp

Function name	HAL_StatusTypeDef HAL_RTCEEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)
Function description	Deactivate TimeStamp.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTCEEx_GetTimeStamp

Function name	HAL_StatusTypeDef HAL_RTCEEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)
Function description	Get the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• sTimeStamp: Pointer to Time structure• sTimeStampDate: Pointer to Date structure• Format: specifies the format of the entered parameters. This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_FORMAT_BIN: Binary data format– RTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTCEEx_SetTamper

Function name	HAL_StatusTypeDef HAL_RTCEEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function description	Set Tamper.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• sTamper: Pointer to Tamper Structure.

Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• By calling this API we disable the tamper interrupt for all tampers.

HAL_RTCEEx_SetTamper_IT

Function name	HAL_StatusTypeDef HAL_RTCEEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.• sTamper: Pointer to RTC Tamper.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• By calling this API we force the tamper interrupt for all tampers.

HAL_RTCEEx_DeactivateTamper

Function name	HAL_StatusTypeDef HAL_RTCEEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)
Function description	Deactivate Tamper.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• Tamper: Selected tamper pin. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTCEEx_TamperTimeStampIRQHandler

Function name	void HAL_RTCEEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)
Function description	Handle TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None:

HAL_RTCEEx_Tamper1EventCallback

Function name	void HAL_RTCEEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)
Function description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None:

HAL_RTCEEx_Tamper2EventCallback

Function name	<code>void HAL_RTCEEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)</code>
Function description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle
Return values	<ul style="list-style-type: none">• None:

HAL_RTCEEx_Tamper3EventCallback

Function name	<code>void HAL_RTCEEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)</code>
Function description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle
Return values	<ul style="list-style-type: none">• None:

HAL_RTCEEx_TimeStampEventCallback

Function name	<code>void HAL_RTCEEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)</code>
Function description	TimeStamp callback.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle
Return values	<ul style="list-style-type: none">• None:

HAL_RTCEEx_PollForTimeStampEvent

Function name	<code>HAL_StatusTypeDef HAL_RTCEEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function description	Handle TimeStamp polling request.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle• <code>Timeout</code>: Timeout duration
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTCEEx_PollForTamper1Event

Function name	<code>HAL_StatusTypeDef HAL_RTCEEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function description	Handle Tamper 1 Polling.
Parameters	<ul style="list-style-type: none">• <code>hrtc</code>: RTC handle• <code>Timeout</code>: Timeout duration

Return values	<ul style="list-style-type: none">• HAL: status
HAL_RTCEx_PollForTamper2Event	
Function name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function description	Handle Tamper 2 Polling.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• Timeout: Timeout duration
Return values	<ul style="list-style-type: none">• HAL: status
HAL_RTCEx_PollForTamper3Event	
Function name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function description	Handle Tamper 3 Polling.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• Timeout: Timeout duration
Return values	<ul style="list-style-type: none">• HAL: status
HAL_RTCEx_SetWakeUpTimer	
Function name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function description	Set wake up timer.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• WakeUpCounter: Wake up counter• WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none">• 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	Set wake up timer with interrupt.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• WakeUpCounter: Wake up counter• WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTCEx_DeactivateWakeUpTimer

Function name	<code>uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)</code>
Function description	Deactivate wake up timer counter.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTCEx_GetWakeUpTimer

Function name	<code>uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)</code>
Function description	Get wake up timer counter.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• Counter: value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name	<code>void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)</code>
Function description	Handle Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None:

HAL_RTCEx_WakeUpTimerEventCallback

Function name	<code>void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)</code>
Function description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None:

HAL_RTCEx_PollForWakeUpTimerEvent

Function name	<code>HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function description	Handle Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• Timeout: Timeout duration
Return values	<ul style="list-style-type: none">• HAL: status

HAL_RTCEEx_BKUPWrite

Function name `void HAL_RTCEEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)`

Function description Write a data in a specified RTC Backup data register.

- Parameters**
- **hrtc:** RTC handle
 - **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 4 to specify the register.
 - **Data:** Data to be written in the specified RTC Backup data register.
- Return values**
- **None:**

HAL_RTCEEx_BKUPRead

Function name `uint32_t HAL_RTCEEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)`

Function description Reads data from the specified RTC Backup data Register.

- Parameters**
- **hrtc:** RTC handle
 - **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 4 to specify the register.
- Return values**
- **Read:** value

HAL_RTCEEx_SetSmoothCalib

Function name `HAL_StatusTypeDef HAL_RTCEEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)`

Function description Set the Smooth calibration parameters.

- Parameters**
- **hrtc:** RTC handle
 - **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be one of the following values :
 - RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s.
 - RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s.
 - RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.
 - **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values:
 - RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses.
 - RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.
 - **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
- Return values**
- **HAL:** status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

HAL_RTCEx_SetSynchroShift

Function name	HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)
Function description	Configure the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none">hrtc: RTC handleShiftAdd1S: Select to add or not 1 second to the time calendar. This parameter can be one of the following values :<ul style="list-style-type: none">– 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 0xFFFF.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name	HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)
Function description	Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none">hrtc: RTC handleCalibOutput: Select the Calibration output Selection . This parameter can be one of the following values:<ul style="list-style-type: none">– RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.– RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	<ul style="list-style-type: none">HAL: status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name	HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)
Function description	Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none">hrtc: RTC handle
Return values	<ul style="list-style-type: none">HAL: status

HAL_RTCEx_SetRefClock

Function name	HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)
----------------------	---

Function description Enable the RTC reference clock detection.

Parameters • **hrtc:** RTC handle

Return values • **HAL:** status

HAL_RTCEEx_DeactivateRefClock

Function name **HAL_StatusTypeDef HAL_RTCEEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)**

Function description Disable the RTC reference clock detection.

Parameters • **hrtc:** RTC handle

Return values • **HAL:** status

HAL_RTCEEx_EnableBypassShadow

Function name **HAL_StatusTypeDef HAL_RTCEEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)**

Function description Enable the Bypass Shadow feature.

Parameters • **hrtc:** RTC handle

Return values • **HAL:** status

Notes • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEEx_DisableBypassShadow

Function name **HAL_StatusTypeDef HAL_RTCEEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)**

Function description Disable the Bypass Shadow feature.

Parameters • **hrtc:** RTC handle

Return values • **HAL:** status

Notes • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

36.3 RTCEEx Firmware driver defines

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

36.3.1 RTCEEx

RTCEEx

RTCEEx Add 1 Second Parameter Definition

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTCEEx Backup Registers Definition

[RTC_BKP_DR0](#)

[RTC_BKP_DR1](#)

[RTC_BKP_DR2](#)

[RTC_BKP_DR3](#)

[RTC_BKP_DR4](#)

RTC Calibration

[__HAL_RTC_CALIBRATION_OUTPUT_ENABLE](#) **Description:**

- Enable the RTC calibration output.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_CALIBRATION_OUTPUT_DISABLE](#) **Description:**

- Disable the calibration output.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_CLOCKREF_DETECTION_ENABLE](#) **Description:**

- Enable the clock reference detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_CLOCKREF_DETECTION_DISABLE](#) **Description:**

- Disable the clock reference detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

__HAL_RTC_SHIFT_GET_F Description:

LAG

- 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

Return value:

- None

RTCEEx Calib Output selection Definitions**RTC_CALIBOUTPUT_512H**

Z

RTC_CALIBOUTPUT_1HZ***Private macros to check input parameters*****IS_RTC_OUTPUT****IS_RTC_BKP****IS_TIMESTAMP_EDGE****IS_RTC_TAMPER****IS_RTC_TIMESTAMP_PIN****IS_RTC_TAMPER_TRIGGER****IS_RTC_TAMPER_FILTER****IS_RTC_TAMPER_SAMPLING_FREQ****IS_RTC_TAMPER_PRECHARGE_DURATION****IS_RTC_TAMPER_TIMESTAMPON_TAMPER_DETECTION****IS_RTC_TAMPER_PULLUP_STATE****IS_RTC_WAKEUP_CLOCK****IS_RTC_WAKEUP_COUNT**

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

RTCEEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARM

RTC_OUTPUT_WAKEUP

RTCEEx Smooth calib period Definition

RTC_SMOOTHCALIB_PERI If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else $2^{\text{exp}20}$ RTCCLK seconds
OD_32SEC

RTC_SMOOTHCALIB_PERI If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else $2^{\text{exp}19}$ RTCCLK seconds
OD_16SEC

RTC_SMOOTHCALIB_PERI If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else $2^{\text{exp}18}$ RTCCLK seconds
OD_8SEC

RTCEEx Smooth calib Plus pulses Definition

RTC_SMOOTHCALIB_PLU The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y =
SPULSES_SET 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLU The number of RTCCLK pulses substituted during a 32-second window = CALM[8:0]
SPULSES_RESET

RTC Tamper

HAL_RTC_TAMPER1_EN **Description:**

ABLE

- Enable the RTC Tamper1 input detection.

Parameters:

- **_HANDLE_**: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER1_DI **Description:**

- SABLE • Disable the RTC Tamper1 input detection.

Parameters:

- HANDLE: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_EN **Description:**

- ABLE • Enable the RTC Tamper2 input detection.

Parameters:

- HANDLE: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_DI **Description:**

- SABLE • Disable the RTC Tamper2 input detection.

Parameters:

- HANDLE: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_EN **Description:**

- ABLE • Enable the RTC Tamper3 input detection.

Parameters:

- HANDLE: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_DI **Description:**

- SABLE • Disable the RTC Tamper3 input detection.

Parameters:

- HANDLE: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER_EN **Description:**

- ABLE_IT • Enable the RTC Tamper interrupt.

Parameters:

- HANDLE: specifies the RTC handle.
- INTERRUPT: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
 - RTC_IT_TAMP: Tamper interrupt

Return value:

- None

__HAL_RTC_TAMPER_DIS **Description:****ABLE_IT**

- Disable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
 - RTC_IT_TAMP: Tamper interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET **Description:****_IT**

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

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
 - RTC_IT_TAMP1: Tamper1 interrupt
 - RTC_IT_TAMP2: Tamper2 interrupt
 - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET **Description:****_IT_SOURCE**

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
 - RTC_IT_TAMP: Tamper interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET **Description:****_FLAG**

- Get the selected RTC Tamper's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
 - RTC_FLAG_TAMP1F
 - RTC_FLAG_TAMP2F
 - RTC_FLAG_TAMP3F

Return value:

- None

__HAL_RTC_TAMPER_CLE Description:**AR_FLAG**

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
 - RTC_FLAG_TAMP1F
 - RTC_FLAG_TAMP2F
 - RTC_FLAG_TAMP3F

Return value:

- None

RTCEEx Tamper Filter Definition**RTC_TAMPERFILTER_DISA** Tamper filter is disabled
BLE**RTC_TAMPERFILTER_2SA** Tamper is activated after 2 consecutive samples at the active level
MPLE**RTC_TAMPERFILTER_4SA** Tamper is activated after 4 consecutive samples at the active level
MPLE**RTC_TAMPERFILTER_8SA** Tamper is activated after 8 consecutive samples at the active level.
MPLE***RTCEEx Tamper Pins Definition*****RTC_TAMPER_1****RTC_TAMPER_2****RTC_TAMPER_3*****RTCEEx Tamper Pin Precharge Duration Definition*****RTC_TAMPERPRECHARG** Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
EDURATION_1RTCCLK**RTC_TAMPERPRECHARG** Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
EDURATION_2RTCCLK**RTC_TAMPERPRECHARG** Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
EDURATION_4RTCCLK**RTC_TAMPERPRECHARG** Tamper pins are pre-charged before sampling during 8 RTCCLK cycles
EDURATION_8RTCCLK***RTCEEx Tamper Pull UP Definition*****RTC_TAMPER_PULLUP_E** Tamper pins are pre-charged before sampling
NABLE

RTC_TAMPER_PULLUP_DI Tamper pins are not pre-charged before sampling
SABLE

RTCEEx Tamper Sampling Frequencies Definition

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
REQ_RTCCLK_DIV32768

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
REQ_RTCCLK_DIV16384

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
REQ_RTCCLK_DIV8192

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
REQ_RTCCLK_DIV4096

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
REQ_RTCCLK_DIV2048

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
REQ_RTCCLK_DIV1024

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
REQ_RTCCLK_DIV512

RTC_TAMPERSAMPLINGF Each of the tamper inputs are sampled with a frequency = RTCCLK / 256
REQ_RTCCLK_DIV256

EXTI RTC Tamper Timestamp EXTI

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_ENABLE_IT • Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_DISABLE_I • Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

T

Return value:

- None

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_ENABLE_E • Enable event on the RTC Tamper and Timestamp associated Exti line.
VENT

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_DISABLE_E • Disable event on the RTC Tamper and Timestamp associated Exti line.
VENT

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_ENABLE_F • Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.
ALLING_EDGE

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_DISABLE_F • Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.
ALLING_EDGE

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_ENABLE_RI • Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.
SING_EDGE

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_DISABLE_R • Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.
ISING_EDGE

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_ENABLE_RI • Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.
SING_FALLING_EDGE

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_DISABLE_R • Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.
ISING_FALLING_EDGE

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_GET_FLAG • Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_CLEAR_FL • Clear the RTC Tamper and Timestamp associated Exti line flag.
AG

Return value:

- None.

_HAL_RTC_TAMPER_TIM Description:

ESTAMP_EXTI_GENERATE_SWIT • Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

RTCEEx TamperTimeStampOnTamperDetection Definition

RTC_TIMESTAMPONTAMP_ERDETECTION_ENABLE TimeStamp on Tamper Detection event saved

RTC_TIMESTAMPONTAMP_ERDETECTION_DISABLE TimeStamp on Tamper Detection event is not saved

RTCEEx Tamper Trigger Definition

RTC_TAMPERTRIGGER_RI_SINGEDGE

RTC_TAMPERTRIGGER_F_ALLINGEDGE

RTC_TAMPERTRIGGER_L_OWLEVEL

RTC_TAMPERTRIGGER_HI_GHLEVEL

RTC Timestamp

__HAL_RTC_TIMESTAMP_ENABLE **Description:**

- Enable the RTC TimeStamp peripheral.

Parameters:

- **__HANDLE__**: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE **Description:**

- Disable the RTC TimeStamp peripheral.

Parameters:

- **__HANDLE__**: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE_IT **Description:**

- Enable the RTC TimeStamp interrupt.

Parameters:

- **__HANDLE__**: specifies the RTC handle.
- **__INTERRUPT__**: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
 - **RTC_IT_TS**: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

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

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt to check. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

- Get the selected RTC TimeStamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
 - RTC_FLAG_TSF
 - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_TIMESTAMP_ **Description:****CLEAR_FLAG**

- Clear the RTC Time Stamp's pending flags.

Parameters:

- HANDLE: specifies the RTC handle.
- FLAG: specifies the RTC Alarm Flag to clear. This parameter can be:
 - RTC_FLAG_TSF

Return value:

- None

RTCEEx TimeStamp Pin Selection**RTC_TIMESTAMPPIN_DEF**
AULT***RTCEEx Time Stamp Edges definition*****RTC_TIMESTAMPEDGE_RISING****RTC_TIMESTAMPEDGE_FALLING*****RTC WakeUp Timer*****__HAL_RTC_WAKEUPTIME** **Description:****R_ENABLE**

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- HANDLE: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIME **Description:****R_DISABLE**

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- HANDLE: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIME **Description:****R_ENABLE_IT**

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- HANDLE: specifies the RTC handle.
- INTERRUPT: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIME Description:**R_DISABLE_IT**

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIME Description:**R_GET_IT**

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

Parameters:

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

Return value:

- None

__HAL_RTC_WAKEUPTIME Description:**R_GET_IT_SOURCE**

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

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

Return value:

- None

__HAL_RTC_WAKEUPTIME Description:**R_GET_FLAG**

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
 - RTC_FLAG_WUTF
 - RTC_FLAG_WUTWF

Return value:

- None

_HAL_RTC_WAKEUPTIME Description:**R_CLEAR_FLAG**

- Clear the RTC Wake Up timer's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
 - RTC_FLAG_WUTF

Return value:

- None

_HAL_RTC_WAKEUPTIME Description:**R_EXTI_ENABLE_IT**

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

_HAL_RTC_WAKEUPTIME Description:**R_EXTI_DISABLE_IT**

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

_HAL_RTC_WAKEUPTIME Description:**R_EXTI_ENABLE_EVENT**

- Enable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:**R_EXTI_DISABLE_EVENT**

- Disable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:**R_EXTI_ENABLE_FALLING_EDGE**

- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:**R_EXTI_DISABLE_FALLING_EDGE**

- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:**R_EXTI_ENABLE_RISING_EDGE**

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:

R_EXTI_DISABLE_RISING_EDGE • Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:

R_EXTI_ENABLE_RISING_FALLING_EDGE • Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:

R_EXTI_DISABLE_RISING_FALLING_EDGE • Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:

R_EXTI_GET_FLAG • Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

_HAL_RTC_WAKEUPTIME Description:

R_EXTI_CLEAR_FLAG • Clear the RTC WakeUp Timer associated Exti line flag.

Return value:

- None.

_HAL_RTC_WAKEUPTIME Description:

R_EXTI_GENERATE_SWIT • Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

RTCEEx Wakeup Timer Definition

RTC_WAKEUPCLOCK_RT_CCLK_DIV16

RTC_WAKEUPCLOCK_RT_CCLK_DIV8

RTC_WAKEUPCLOCK_RT_CCLK_DIV4

RTC_WAKEUPCLOCK_RT_CCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

37 HAL SMARTCARD Generic Driver

37.1 SMARTCARD Firmware driver registers structures

37.1.1 SMARTCARD_InitTypeDef

`SMARTCARD_InitTypeDef` is defined in the `stm32f0xx_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`

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 = ((PCLKx) / ((hsmartcard->Init.BaudRate)))

- `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.
- **`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)

37.1.2 SMARTCARD_AdvFeatureInitTypeDef

`SMARTCARD_AdvFeatureInitTypeDef` is defined in the `stm32f0xx_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`**

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 [`SMARTCARD_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`

37.1.3 SMARTCARD_HandleTypeDef

`SMARTCARD_HandleTypeDef` is defined in the `stm32f0xx_hal_smartcard.h`

Data Fields

- `USART_TypeDef * Instance`
- `SMARTCARD_InitTypeDef Init`
- `SMARTCARD_AdvFeatureInitTypeDef AdvancedInit`
- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `_IO uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `_IO uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `_IO HAL_SMARTCARD_StateTypeDef gState`
- `_IO HAL_SMARTCARD_StateTypeDef RxState`
- `_IO uint32_t ErrorCode`

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
- **`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

- **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 This parameter can be a value of **SMARTCARD_Error**

37.2

SMARTCARD Firmware driver API description

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

37.2.1

How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.
3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
 - Enable the USARTx interface clock.
 - USART pins configuration:
 - Enable the clock for the USART GPIOs.
 - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
 - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
 - This API configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

Note:

The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros **_HAL_SMARTCARD_ENABLE_IT()** and **_HAL_SMARTCARD_DISABLE_IT()** inside the transmit and receive process.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

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

- __HAL_SMARTCARD_GET_FLAG : Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

Note:

You can refer to the SMARTCARD HAL driver header file for more useful macros

37.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

- These parameters can be configured:
 - Baud Rate
 - Parity: should be enabled
 - 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_DelInit`](#)
- [`HAL_SMARTCARD_MspInit`](#)
- [`HAL_SMARTCARD_MspDelInit`](#)

37.2.3 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:
 - 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, 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.
 - 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 :
 - `HAL_SMARTCARD_Transmit()`
 - `HAL_SMARTCARD_Receive()`
 3. Non Blocking mode APIs with Interrupt are :
 - `HAL_SMARTCARD_Transmit_IT()`
 - `HAL_SMARTCARD_Receive_IT()`
 - `HAL_SMARTCARD_IRQHandler()`
 4. Non Blocking mode functions with DMA are :
 - `HAL_SMARTCARD_Transmit_DMA()`
 - `HAL_SMARTCARD_Receive_DMA()`
 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - `HAL_SMARTCARD_TxCpltCallback()`
 - `HAL_SMARTCARD_RxCpltCallback()`
 - `HAL_SMARTCARD_ErrorCallback()`

6. Non-Blocking mode transfers could be aborted using Abort API's :
 - HAL_SMARTCARD_Abort()
 - HAL_SMARTCARD_AbortTransmit()
 - HAL_SMARTCARD_AbortReceive()
 - HAL_SMARTCARD_Abort_IT()
 - HAL_SMARTCARD_AbortTransmit_IT()
 - HAL_SMARTCARD_AbortReceive_IT()
7. For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
 - HAL_SMARTCARD_AbortCpltCallback()
 - HAL_SMARTCARD_AbortTransmitCpltCallback()
 - HAL_SMARTCARD_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_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.
 - 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**](#)

37.2.4 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`

37.2.5 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_DelInit`

Function name `HAL_StatusTypeDef HAL_SMARTCARD_DelInit (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description Delinitialize 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_MspDelInit`

Function name `void HAL_SMARTCARD_MspDelInit (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description Delinitialize 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_Transmit

Function name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none">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	<ul style="list-style-type: none">HAL: status

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	<ul style="list-style-type: none">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	<ul style="list-style-type: none">HAL: status

HAL_SMARTCARD_Transmit_IT

Function name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)
Function description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none">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	<ul style="list-style-type: none">HAL: status

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

HAL_SMARTCARD_Transmit_DMA

- Function name**
- HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef *
hsmartcard, 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	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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 * hsmcard)`

Function description Handle SMARTCARD interrupt requests.

Parameters

- **hsmcard:** 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 * hsmcard)`

Function description Tx Transfer completed callback.

Parameters

- **hsmcard:** 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 * hsmcard)`

Function description Rx Transfer completed callback.

Parameters

- **hsmcard:** 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 * hsmcard)`

Function description SMARTCARD error callback.

Parameters

- **hsmcard:** 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 * hsmcard)`

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 (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 (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

37.3 SMARTCARD Firmware driver defines

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

37.3.1 SMARTCARD

SMARTCARD

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATUR No advanced feature initialization
E_NO_INIT

SMARTCARD_ADVFEATUR TX pin active level inversion
E_TXINVERT_INIT

SMARTCARD_ADVFEATUR RX pin active level inversion
E_RXINVERT_INIT

SMARTCARD_ADVFEATUR Binary data inversion
E_DATAINVERT_INIT

SMARTCARD_ADVFEATUR TX/RX pins swap
E_SWAP_INIT

SMARTCARD_ADVFEATUR RX overrun disable
E_RXOVERRUNDISABLE_INIT

SMARTCARD_ADVFEATUR DMA disable on Reception Error
E_DMADISABLEONERROR_INIT

SMARTCARD_ADVFEATUR Most significant bit sent/received first
E_MSBFIRST_INIT

SMARTCARD Clock Phase

SMARTCARD_PHASE_1ED SMARTCARD frame phase on first clock transition
GE

SMARTCARD_PHASE_2ED SMARTCARD frame phase on second clock transition
GE

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW SMARTCARD frame low polarity
LOW

SMARTCARD_POLARITY_HIGH SMARTCARD frame high polarity
HIGH

SMARTCARD auto retry counter LSB position in CR3 register

SMARTCARD_CR3_SCARC SMARTCARD auto retry counter LSB position in CR3 register
NT_LSB_POS

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATUR Binary data inversion disable
E_DATAINV_DISABLE

SMARTCARD_ADVFEATUR Binary data inversion enable
E_DATAINV_ENABLE

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATUR DMA enable on Reception Error
E_DMA_ENABLEONRXER
ROR

SMARTCARD_ADVFEATUR DMA disable on Reception Error
E_DMA_DISABLEONRXER
ROR

SMARTCARD Error

HAL_SMARTCARD_ERRO No error
R_NONE

HAL_SMARTCARD_ERRO Parity error
R_PE

HAL_SMARTCARD_ERRO Noise error
R_NE

HAL_SMARTCARD_ERRO frame error
R_FE

HAL_SMARTCARD_ERRO Overrun error
R_ORE

HAL_SMARTCARD_ERRO DMA transfer error
R_DMA

HAL_SMARTCARD_ERRO Receiver TimeOut error
R_RTO

SMARTCARD Exported Macros

_HAL_SMARTCARD_RES **Description:**
ET_HANDLE_STATE • Reset SMARTCARD handle states.
Parameters:
 • **_HANDLE_**: SMARTCARD handle.
Return value:
 • None

_HAL_SMARTCARD_FLU **Description:****SH_DRREGISTER**

- Flush the Smartcard Data registers.

Parameters:

- HANDLE: specifies the SMARTCARD Handle.

Return value:

- None

_HAL_SMARTCARD_CLE **Description:****AR_FLAG**

- 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_RTOF Receiver timeout clear flag
 - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

_HAL_SMARTCARD_CLE **Description:****AR_PEFLAG**

- Clear the SMARTCARD PE pending flag.

Parameters:

- HANDLE: specifies the SMARTCARD Handle.

Return value:

- None

_HAL_SMARTCARD_CLE **Description:****AR_FEFLAG**

- Clear the SMARTCARD FE pending flag.

Parameters:

- HANDLE: specifies the SMARTCARD Handle.

Return value:

- None

_HAL_SMARTCARD_CLE **Description:****AR_NEFLAG**

- Clear the SMARTCARD NE pending flag.

Parameters:

- HANDLE: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLE_AR_OREFLAG **Description:**

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLE_AR_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_RXACK 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

Return value:

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

__HAL_SMARTCARD_ENA **Description:****BLE_IT**

- 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_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)

Return value:

- None

__HAL_SMARTCARD_DISA **Description:****BLE_IT**

- 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_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)

Return value:

- None

__HAL_SMARTCARD_GET_IT Description:

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

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT__: 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_RXNE Receive data register not empty interrupt
 - SMARTCARD_IT_IDLE Idle line detection interrupt
 - SMARTCARD_IT_ORE Overrun error interrupt
 - SMARTCARD_IT_NE Noise error interrupt
 - SMARTCARD_IT_FE Framing error interrupt
 - SMARTCARD_IT_PE Parity error interrupt

Return value:

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

__HAL_SMARTCARD_GET_IT_SOURCE Description:

- Check whether the specified SmartCard interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT__: 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_RXNE Receive data register not empty interrupt
 - SMARTCARD_IT_IDLE Idle line detection interrupt
 - SMARTCARD_IT_ERR Framing, overrun or noise error interrupt
 - SMARTCARD_IT_PE Parity error interrupt

Return value:

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

__HAL_SMARTCARD_CLE **Description:****AR_IT**

- 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_TCF Transmission complete clear flag
 - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
 - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_SEN **Description:****D_REQ**

- 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 **Description:****BIT_SAMPLE_ENABLE**

- Enable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ONE **Description:****BIT_SAMPLE_DISABLE**

- Disable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

_HAL_SMARTCARD_ENA Description:

BLE

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- HANDLE: specifies the SMARTCARD Handle.

Return value:

- None

_HAL_SMARTCARD_DISA Description:

BLE

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- HANDLE: specifies the SMARTCARD Handle.

Return value:

- None

SMARTCARD Flags**SMARTCARD_FLAG_REAC** SMARTCARD receive enable acknowledge flag

K

SMARTCARD_FLAG_TEAC SMARTCARD transmit enable acknowledge flag

K

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_TC** SMARTCARD transmission complete**SMARTCARD_FLAG_RXNE** SMARTCARD read data register 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 guard time value LSB position in GTPR register****SMARTCARD_GTPR_GT_L** SMARTCARD guard time value LSB position in GTPR register
SB_POS**SMARTCARD interruptions flags mask**

SMARTCARD_IT_MASK SMARTCARD interruptions flags mask

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE SMARTCARD parity error interruption

SMARTCARD_IT_TXE SMARTCARD transmit data register empty interruption

SMARTCARD_IT_TC SMARTCARD transmission complete interruption

SMARTCARD_IT_RXNE SMARTCARD read data register 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 Interruption Clear Flags

SMARTCARD_CLEAR_PEF SMARTCARD parity error clear flag

SMARTCARD_CLEAR_FEF SMARTCARD framing error clear flag

SMARTCARD_CLEAR_NEF SMARTCARD noise detected clear flag

**SMARTCARD_CLEAR_OR
EF** SMARTCARD overrun error clear flag

**SMARTCARD_CLEAR_IDL
EF** SMARTCARD idle line detected clear flag

SMARTCARD_CLEAR_TCF SMARTCARD transmission complete clear flag

**SMARTCARD_CLEAR_RTO
F** SMARTCARD receiver time out clear flag

**SMARTCARD_CLEAR_EO
BF** SMARTCARD end of block clear flag

SMARTCARD Last Bit

**SMARTCARD_LASTBIT_DI
SABLE** SMARTCARD frame last data bit clock pulse not output to SCLK pin

SMARTCARD_LASTBIT_E SMARTCARD frame last data bit clock pulse output to SCLK pin
NABLE

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX SMARTCARD RX mode

SMARTCARD_MODE_TX SMARTCARD TX mode

SMARTCARD_MODE_TX_R SMARTCARD RX and TX mode
X

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATUR Most significant bit sent/received first disable
E_MSBFIRST_DISABLE

SMARTCARD_ADVFEATUR Most significant bit sent/received first enable
E_MSBFIRST_ENABLE

SMARTCARD NACK Enable

SMARTCARD_NACK_ENA SMARTCARD NACK transmission disabled
BLE

SMARTCARD_NACK_DISA SMARTCARD NACK transmission enabled
BLE

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_S SMARTCARD frame one-bit sample disabled
AMPLE_DISABLE

SMARTCARD_ONE_BIT_S SMARTCARD frame one-bit sample enabled
AMPLE_ENABLE

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATUR RX overrun enable
E_OVERRUN_ENABLE

SMARTCARD_ADVFEATUR RX overrun disable
E_OVERRUN_DISABLE

SMARTCARD Parity

SMARTCARD_PARITY_EV SMARTCARD frame even parity
EN

SMARTCARD_PARITY_OD SMARTCARD frame odd parity
D

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FL Receive data flush request
USH_REQUEST

SMARTCARD_TXDATA_FL Transmit data flush request
USH_REQUEST

SMARTCARD block length LSB position in RTOR register

SMARTCARD_RTOR_BLEN SMARTCARD block length LSB position in RTOR register
_LSB_POS

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATUR RX pin active level inversion disable
E_RXINV_DISABLE

SMARTCARD_ADVFEATUR RX pin active level inversion enable
E_RXINV_ENABLE

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATUR TX/RX pins swap disable
E_SWAP_DISABLE

SMARTCARD_ADVFEATUR TX/RX pins swap enable
E_SWAP_ENABLE

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0 SMARTCARD frame with 0.5 stop bit
_5

SMARTCARD_STOPBITS_1 SMARTCARD frame with 1.5 stop bits
_5

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DI SMARTCARD receiver timeout disabled
SABLE

SMARTCARD_TIMEOUT_E SMARTCARD receiver timeout enabled
NABLE

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATUR TX pin active level inversion disable
E_TXINV_DISABLE

SMARTCARD_ADVFEATUR TX pin active level inversion enable
E_TXINV_ENABLE

SMARTCARD Word Length

SMARTCARD_WORDLENG SMARTCARD frame length
TH_9B

38 HAL SMARTCARD Extension Driver

38.1 SMARTCARDEX Firmware driver API description

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

38.1.1 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 `hsmcard AdvancedInit` structure.

38.1.2 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`

38.1.3 Detailed description of functions

`HAL_SMARTCARDEX_BlockLength_Config`

Function name `void HAL_SMARTCARDEX_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmcard, uint8_t BlockLength)`

Function description Update on the fly the SMARTCARD block length in RTOR register.

Parameters

- **hsmcard:** 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 * hsmcard, uint32_t TimeOutValue)`

Function description Update on the fly the receiver timeout value in RTOR register.

Parameters

- **hsmcard:** 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 0xFFFFFFFF.

Return values	<ul style="list-style-type: none">None:
	HAL_SMARTCARDEX_EnableReceiverTimeOut
Function name	HAL_StatusTypeDef HAL_SMARTCARDEX_EnableReceiverTimeOut(SMARTCARD_HandleTypeDef * hsmartcard)
Function description	Enable the SMARTCARD receiver timeout feature.
Parameters	<ul style="list-style-type: none">hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">HAL: status
	HAL_SMARTCARDEX_DisableReceiverTimeOut
Function name	HAL_StatusTypeDef HAL_SMARTCARDEX_DisableReceiverTimeOut(SMARTCARD_HandleTypeDef * hsmartcard)
Function description	Disable the SMARTCARD receiver timeout feature.
Parameters	<ul style="list-style-type: none">hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">HAL: status

39 HAL SMBUS Generic Driver

39.1 SMBUS Firmware driver registers structures

39.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the `stm32f0xx_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 or 10-bit address.

- `uint32_t SMBUS_InitTypeDef::AddressingMode`

Specifies if 7-bit or 10-bit addressing mode for master is 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

39.1.2 **SMBUS_HandleTypeDef**

`SMBUS_HandleTypeDef` is defined in the `stm32f0xx_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`

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

39.2 **SMBUS Firmware driver API description**

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

39.2.1 **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 user can add his 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 user can add his 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 user can add his 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 user can add his 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 user can add his 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 user can add his own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT() HAL_SMBUS_DisableAlert_IT()
 - When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and user can add his 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 user can add his 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

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

39.2.2 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 Filter 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_DelInit() 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_DelInit`](#)
- [`HAL_SMBUS_MspInit`](#)
- [`HAL_SMBUS_MspDelInit`](#)
- [`HAL_SMBUS_ConfigAnalogFilter`](#)
- [`HAL_SMBUS_ConfigDigitalFilter`](#)

39.2.3 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**](#)

39.2.4 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**](#)

39.2.5 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_DelInit

Function name `HAL_StatusTypeDef HAL_SMBUS_DelInit (SMBUS_HandleTypeDef * hsmbus)`

Function description Delinitialize 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_MspDelInit

Function name `void HAL_SMBUS_MspDelInit (SMBUS_HandleTypeDef * hsmbus)`

Function description Delinitialize 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_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 shift at right before call 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 shift at right before call 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 shift at right before call 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 shift at right before call 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmbus: Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmbus: Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmbus: Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hsmbus: Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none">• 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 (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 (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

39.3 SMBUS Firmware driver defines

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

39.3.1 SMBUS

SMBUS

SMBUS addressing mode

**SMBUS_ADDRESSINGMO
DE_7BIT**

**SMBUS_ADDRESSINGMO
DE_10BIT**

SMBUS Analog Filter

**SMBUS_ANALOGFILTER_
ENABLE**

**SMBUS_ANALOGFILTER_
DISABLE**

SMBUS dual addressing mode

**SMBUS_DUALADDRESS_D
ISABLE**

**SMBUS_DUALADDRESS_E
NABLE**

SMBUS Error Code definition

HAL_SMBUS_ERROR_NO No error
NE

HAL_SMBUS_ERROR_BER BERR error
R

HAL_SMBUS_ERROR_ARL ARLO error
O

HAL_SMBUS_ERROR_ACK ACKF error
F

HAL_SMBUS_ERROR_OVR OVR error

HAL_SMBUS_ERROR_HAL Timeout error
TIMEOUT

HAL_SMBUS_ERROR_BUS Bus Timeout error
TIMEOUT

HAL_SMBUS_ERROR_ALE Alert error
RT

HAL_SMBUS_ERROR_PEC PEC error
ERR

SMBUS Exported Macros

_HAL_SMBUS_RESET_H **Description:**
ANDLE_STATE

- Reset SMBUS handle state.

Parameters:

- **_HANDLE_**: specifies the SMBUS Handle.

Return value:

- None

_HAL_SMBUS_ENABLE_I **Description:**

T

- 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:

T

- 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:

T

- 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__ (TRUE or FALSE).

`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__` (TRUE or FALSE).

`__HAL_SMBUS_GET_FLAG`

`__HAL_SMBUS_CLEAR_FL` Description:

AG

- 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_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
DE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE
DE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE
DE_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_STAR
T_READ

SMBUS_GENERATE_STAR
T_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST
_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO
_PEC

SMBUS_FIRST_AND_LAST
_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WI
TH_PEC

SMBUS_OTHER_FRAME_N
O_PEC

SMBUS_OTHER_FRAME_
WITH_PEC

SMBUS_OTHER_AND_LAS
T_FRAME_NO_PEC

SMBUS_OTHER_AND_LAS
T_FRAME_WITH_PEC

40 HAL SPI Generic Driver

40.1 SPI Firmware driver registers structures

40.1.1 SPI_InitTypeDef

`SPI_InitTypeDef` is defined in the `stm32f0xx_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)..

40.1.2 [**__SPI_HandleTypeDef**](#)

[**__SPI_HandleTypeDef**](#) is defined in the `stm32f0xx_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***

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

40.2 SPI Firmware driver API description

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

40.2.1 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 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=0) 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_DelInit()
 - b. HAL_SPI_Init()

The HAL drivers do not allow reaching all supported SPI frequencies in the different SPI modes. Refer to the source code (stm32xxxx_hal_spi.c header) to get a summary of the maximum SPI frequency that can be reached with a data size of 8 or 16 bits, depending on the APBx peripheral clock frequency (fPCLK) used by the SPI instance.

Data buffer address alignment restriction:

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

40.2.2

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_DelInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- [**HAL_SPI_Init**](#)
- [**HAL_SPI_DelInit**](#)
- [**HAL_SPI_MspInit**](#)
- [**HAL_SPI_MspDelInit**](#)

40.2.3

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_TxRxCPtCallback() 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`](#)

40.2.4

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`](#)

40.2.5

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_DelInit

Function name

HAL_StatusTypeDef HAL_SPI_DelInit (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_MspDelInit

Function name

void HAL_SPI_MspDelInit (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_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	<code>HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none">hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.pData: pointer to data bufferSize: amount of data to be receivedTimeout: Timeout duration
Return values	<ul style="list-style-type: none">HAL: status

HAL_SPI_TransmitReceive

Function name	<code>HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</code>
Function description	Transmit and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none">hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.pTxData: pointer to transmission data bufferpRxData: pointer to reception data bufferSize: amount of data to be sent and receivedTimeout: Timeout duration
Return values	<ul style="list-style-type: none">HAL: status

HAL_SPI_Transmit_IT

Function name	<code>HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</code>
Function description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none">hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.pData: pointer to data bufferSize: amount of data to be sent
Return values	<ul style="list-style-type: none">HAL: status

HAL_SPI_Receive_IT

Function name	<code>HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</code>
Function description	Receive an amount of data in non-blocking mode with Interrupt.

Parameters	<ul style="list-style-type: none">hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.pData: pointer to data bufferSize: amount of data to be sent
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.pTxData: pointer to transmission data bufferpRxData: pointer to reception data bufferSize: amount of data to be sent and received
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.pData: pointer to data bufferSize: amount of data to be sent
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.pData: pointer to data bufferSize: amount of data to be sent
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">• HAL: status
HAL_SPI_Abort	
Function name	HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)
Function description	Abort ongoing transfer (blocking mode).
Parameters	<ul style="list-style-type: none">• hspi: SPI handle.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hspi: SPI handle.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none">• None:
HAL_SPI_TxCpltCallback	
Function name	void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

Function description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None:
HAL_SPI_RxCpltCallback	
Function name	void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)
Function description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None:
HAL_SPI_TxRxCpltCallback	
Function name	void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
Function description	Tx and Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None:
HAL_SPI_TxHalfCpltCallback	
Function name	void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function description	Tx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None:
HAL_SPI_RxHalfCpltCallback	
Function name	void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function description	Rx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• 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

40.3 SPI Firmware driver defines

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

40.3.1 SPI

SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCAL

ER_2

SPI_BAUDRATEPRESCAL

ER_4

SPI_BAUDRATEPRESCAL

ER_8

SPI_BAUDRATEPRESCAL

ER_16

SPI_BAUDRATEPRESCAL

ER_32

SPI_BAUDRATEPRESCAL

ER_64

SPI_BAUDRATEPRESCAL

ER_128

SPI_BAUDRATEPRESCAL

ER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCALCULATION_DISABLE

SPI_CRCALCULATION_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

SPI Exported Macros

_HAL_SPI_RESET_HANDLE_STATE	Description: <ul style="list-style-type: none">Reset SPI handle state. Parameters: <ul style="list-style-type: none">_HANDLE_: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral. Return value: <ul style="list-style-type: none">None
_HAL_SPI_ENABLE_IT	Description: <ul style="list-style-type: none">Enable the specified SPI interrupts. Parameters: <ul style="list-style-type: none">_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:<ul style="list-style-type: none">SPI_IT_TXE: Tx buffer empty interrupt enableSPI_IT_RXNE: RX buffer not empty interrupt enableSPI_IT_ERR: Error interrupt enable Return value: <ul style="list-style-type: none">None
_HAL_SPI_DISABLE_IT	Description: <ul style="list-style-type: none">Disable the specified SPI interrupts. Parameters: <ul style="list-style-type: none">_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:<ul style="list-style-type: none">SPI_IT_TXE: Tx buffer empty interrupt enableSPI_IT_RXNE: RX buffer not empty interrupt enableSPI_IT_ERR: Error interrupt enable Return value: <ul style="list-style-type: none">None

__HAL_SPI_GET_IT_SOUR Description:

CE

- 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_CRC Description:

RRFLAG

- 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_MODF Description:

FLAG

- 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_OVRF **Description:**

LAG

- 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_FREFL **Description:**

AG

- 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 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_FU
LL**

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

41 HAL SPI Extension Driver

41.1 SPIEx Firmware driver API description

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

41.1.1 IO operation functions

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

1. Rx data flush function:

- `HAL_SPIExFlushRxFifo()`

This section contains the following APIs:

- `HAL_SPIExFlushRxFifo`

41.1.2 Detailed description of functions

`HAL_SPIExFlushRxFifo`

Function name `HAL_StatusTypeDef HAL_SPIExFlushRxFifo (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

42 HAL TIM Generic Driver

42.1 TIM Firmware driver registers structures

42.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the `stm32f0xx_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 This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

Note:

- This parameter is valid only for TIM1 and TIM8.

- *uint32_t TIM_Base_InitTypeDef::AutoReloadPreload*

Specifies the auto-reload preload. This parameter can be a value of [*TIM_AutoReloadPreload*](#)

42.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the `stm32f0xx_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 TIM1 and TIM8.
- **`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 TIM1 and TIM8.
- **`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 TIM1 and TIM8.

42.1.3 `TIM_OnePulse_InitTypeDef`

`TIM_OnePulse_InitTypeDef` is defined in the `stm32f0xx_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 IC_Polarity`
- `uint32_t IC_Selection`
- `uint32_t IC_Filter`

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 TIM1 and TIM8.
- ***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 TIM1 and TIM8.
- ***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 TIM1 and TIM8.
- ***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

42.1.4 **TIM_IC_InitTypeDef**

TIM_IC_InitTypeDef is defined in the `stm32f0xx_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

42.1.5 **TIM_Encoder_InitTypeDef**

TIM_Encoder_InitTypeDef is defined in the `stm32f0xx_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_Input_Capture_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_Input_Capture_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

42.1.6**TIM_ClockConfigTypeDef**

`TIM_ClockConfigTypeDef` is defined in the `stm32f0xx_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

42.1.7**TIM_ClearInputConfigTypeDef**

`TIM_ClearInputConfigTypeDef` is defined in the `stm32f0xx_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 [`TIMEx_Clock_Clear_Input_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 can be a value of [`TIM_ClearInput_Prescaler`](#)
- **`uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter`**
TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

42.1.8 **TIM_SlaveConfigTypeDef**

`TIM_SlaveConfigTypeDef` is defined in the `stm32f0xx_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

42.1.9 **TIM_HandleTypeDef**

`TIM_HandleTypeDef` is defined in the `stm32f0xx_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`**

Field Documentation

- **`TIM_TypeDef* TIM_HandleTypeDef::Instance`**
Register base address
- **`TIM_Base_InitTypeDef TIM_HandleTypeDef::Init`**
TIM Time Base required parameters
- **`HAL_TIM_ActiveChannel TIM_HandleTypeDef::Channel`**
Active channel
- **`DMA_HandleTypeDef* TIM_HandleTypeDef::hdma[7]`**
DMA Handlers array This array is accessed by a [`TIM_DMA_Handle_index`](#)

- **`HAL_LockTypeDef TIM_HandleTypeDef::Lock`**
Locking object
- **`__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State`**
TIM operation state

42.2 TIM Firmware driver API description

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

42.2.1 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

42.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
 - 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()

42.2.3 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_DelInit**](#)
- [**HAL_TIM_Base_MspInit**](#)
- [**HAL_TIM_Base_MspDelInit**](#)
- [**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**](#)

42.2.4 Time Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.

This section contains the following APIs:

- [**HAL_TIM_OC_Init**](#)
- [**HAL_TIM_OC_DelInit**](#)
- [**HAL_TIM_OC_MspInit**](#)
- [**HAL_TIM_OC_MspDelInit**](#)
- [**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*](#)

42.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM OPWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_PWM_Init*](#)
- [*HAL_TIM_PWM_DelInit*](#)
- [*HAL_TIM_PWM_MspInit*](#)
- [*HAL_TIM_PWM_MspDelInit*](#)
- [*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*](#)

42.2.6 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.
- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_IC_Init*](#)
- [*HAL_TIM_IC_DelInit*](#)
- [*HAL_TIM_IC_MspInit*](#)
- [*HAL_TIM_IC_MspDelInit*](#)
- [*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*](#)

42.2.7 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.

This section contains the following APIs:

- [`HAL_TIM_OnePulse_Init`](#)
- [`HAL_TIM_OnePulse_DelInit`](#)
- [`HAL_TIM_OnePulse_MspInit`](#)
- [`HAL_TIM_OnePulse_MspDelInit`](#)
- [`HAL_TIM_OnePulse_Start`](#)
- [`HAL_TIM_OnePulse_Stop`](#)
- [`HAL_TIM_OnePulse_Start_IT`](#)
- [`HAL_TIM_OnePulse_Stop_IT`](#)

42.2.8 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.
- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.

This section contains the following APIs:

- [`HAL_TIM_Encoder_Init`](#)
- [`HAL_TIM_Encoder_DelInit`](#)
- [`HAL_TIM_Encoder_MspInit`](#)
- [`HAL_TIM_Encoder_MspDelInit`](#)
- [`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`](#)

42.2.9 IRQ handler management

This section provides Timer IRQ handler function.

This section contains the following APIs:

- [`HAL_TIM_IRQHandler`](#)

42.2.10 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.

This section contains the following APIs:

- [`HAL_TIM_OC_ConfigChannel`](#)
- [`HAL_TIM_IC_ConfigChannel`](#)
- [`HAL_TIM_PWM_ConfigChannel`](#)
- [`HAL_TIM_OnePulse_ConfigChannel`](#)
- [`HAL_TIM_DMABurst_WriteStart`](#)
- [`HAL_TIM_DMABurst_MultiWriteStart`](#)
- [`HAL_TIM_DMABurst_WriteStop`](#)
- [`HAL_TIM_DMABurst_ReadStart`](#)
- [`HAL_TIM_DMABurst_MultiReadStart`](#)
- [`HAL_TIM_DMABurst_ReadStop`](#)
- [`HAL_TIM_GenerateEvent`](#)
- [`HAL_TIM_ConfigOCrefClear`](#)
- [`HAL_TIM_ConfigClockSource`](#)
- [`HAL_TIM_ConfigT1Input`](#)
- [`HAL_TIM_SlaveConfigSynchronization`](#)
- [`HAL_TIM_SlaveConfigSynchronization_IT`](#)
- [`HAL_TIM_ReadCapturedValue`](#)

42.2.11

TIM Callbacks functions

This section provides TIM callback functions:

- Timer Period elapsed callback
- Timer Output Compare callback
- Timer Input capture callback
- Timer Trigger callback
- Timer Error callback

This section contains the following APIs:

- [`HAL_TIM_PeriodElapsedCallback`](#)
- [`HAL_TIM_OC_DelayElapsedCallback`](#)
- [`HAL_TIM_IC_CaptureCallback`](#)
- [`HAL_TIM_PWM_PulseFinishedCallback`](#)
- [`HAL_TIM_TriggerCallback`](#)
- [`HAL_TIM_ErrorCallback`](#)

42.2.12

Peripheral State 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_TIM_Base_GetState`](#)
- [`HAL_TIM_OC_GetState`](#)
- [`HAL_TIM_PWM_GetState`](#)
- [`HAL_TIM_IC_GetState`](#)
- [`HAL_TIM_OnePulse_GetState`](#)
- [`HAL_TIM_Encoder_GetState`](#)

42.2.13 Detailed description of functions

TIM_Base_SetConfig

Function name `void TIM_Base_SetConfig (TIM_TypeDef * TIMx, TIM_Base_InitTypeDef * Structure)`

Function description Time Base configuration.

Parameters

- **TIMx:** TIM peripheral
- **Structure:** TIM Base configuration structure

Return values

- **None:**

TIM_TI1_SetConfig

Function name `void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)`

Function description Configure the TI1 as Input.

Parameters

- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** The Input Polarity. This parameter can be one of the following values:
 - `TIM_ICPOLARITY_RISING`
 - `TIM_ICPOLARITY_FALLING`
 - `TIM_ICPOLARITY_BOTHEDGE`
- **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values:
 - `TIM_ICSELECTION_DIRECTTI` : TIM Input 1 is selected to be connected to IC1.
 - `TIM_ICSELECTION_INDIRECTTI` : TIM Input 1 is selected to be connected to IC2.
 - `TIM_ICSELECTION_TRC` : TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between `0x00` and `0x0F`.

Return values

- **None:**

Notes

- `TIM_ICFilter` and `TIM_ICPolarity` are not used in INDIRECT mode as `TI2FP1` (on channel2 path) is used as the input signal. Therefore `CCMR1` must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name `void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)`

Function description Time Ouput Compare 2 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The ouput configuration structure

Return values

- **None:**

TIM_DMADelayPulseCplt

Function name	void TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)
Function description	TIM DMA Delay Pulse complete callback.
Parameters	<ul style="list-style-type: none">• hdma: pointer to DMA handle.
Return values	<ul style="list-style-type: none">• None:

TIM_DMAError

Function name	void TIM_DMAError (DMA_HandleTypeDef * hdma)
Function description	TIM DMA error callback.
Parameters	<ul style="list-style-type: none">• hdma: pointer to DMA handle.
Return values	<ul style="list-style-type: none">• None:

TIM_DMACaptureCplt

Function name	void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)
Function description	TIM DMA Capture complete callback.
Parameters	<ul style="list-style-type: none">• hdma: pointer to DMA handle.
Return values	<ul style="list-style-type: none">• None:

TIM_CCxChannelCmd

Function name	void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)
Function description	Enables or disables the TIM Capture Compare Channel x.
Parameters	<ul style="list-style-type: none">• TIMx: to select the TIM peripheral• Channel: specifies the TIM Channel This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1 : TIM Channel 1– TIM_CHANNEL_2 : TIM Channel 2– TIM_CHANNEL_3 : TIM Channel 3– TIM_CHANNEL_4 : TIM Channel 4• ChannelState: specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_ENABLE or TIM_CCx_Disable.
Return values	<ul style="list-style-type: none">• None:

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 create the associated handle.

Parameters • **htim:** TIM Base handle

Return values • **HAL:** status

HAL_TIM_Base_DelInit

Function name **HAL_StatusTypeDef HAL_TIM_Base_DelInit (TIM_HandleTypeDef * htim)**

Function description DelInitializes 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 handle

Return values • **None:**

HAL_TIM_Base_MspDeInit

Function name **void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)**

Function description DelInitializes TIM Base MSP.

Parameters • **htim:** TIM 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 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 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 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 handle

Return values • **HAL:** status

HAL_TIM_Base_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**

Function description Starts the TIM Base generation in DMA mode.

Parameters • **htim:** TIM 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 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 create the associated handle.

Parameters • **htim:** TIM Output Compare handle

Return values • **HAL:** status

HAL_TIM_OC_DelInit

Function name **HAL_StatusTypeDef HAL_TIM_OC_DelInit (TIM_HandleTypeDef * htim)**

Function description DelInitializes 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 handle

Return values • **None:**

HAL_TIM_OC_MspDeInit

Function name **void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)**

Function description DelInitializes TIM Output Compare MSP.

Parameters • **htim:** TIM 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

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	<ul style="list-style-type: none">• htim: TIM handle• Channel: TIM Channel to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM OC handle• Channel: TIM Channel to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Output Compare handle• Channel: TIM Channel to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status

HAL_TIM_OC_Start_DMA

Function name	HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function description	Starts the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none">• htim: TIM Output Compare handle• Channel: TIM Channel to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Output Compare handle• Channel: TIM Channel to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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 create the associated handle.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIM_PWM_DeInit

Function name	HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)
Function description	DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM 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 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 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

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 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM handle• Channel: TIM Channel to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM handle• Channel: TIM Channels to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status
HAL_TIM_PWM_Start_DMA	
Function name	HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function description	Starts the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none">• htim: TIM handle• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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 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 create the associated handle.

Parameters

- `htim`: TIM Input Capture handle

Return values

- `HAL`: status

HAL_TIM_IC_DelInit

Function name `HAL_StatusTypeDef HAL_TIM_IC_DelInit (TIM_HandleTypeDef * htim)`

Function description DelInitializes 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 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	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Input Capture handle• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM handle• Channel: TIM Channels to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Input Capture handle• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM handle• Channel: TIM Channels to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Input Capture handle• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Input Capture handle• Channel: TIM Channels to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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 create the associated handle.

Parameters

- **htim:** TIM OnePulse 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 wil be generated.

Return values

- `HAL`: status

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 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 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	<ul style="list-style-type: none">• htim: TIM One Pulse handle• OutputChannel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1 selected– TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none">• HAL: status
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	<ul style="list-style-type: none">• htim: TIM One Pulse handle• OutputChannel: TIM Channels to be disable This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1 selected– TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none">• HAL: status
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	<ul style="list-style-type: none">• htim: TIM One Pulse handle• OutputChannel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1 selected– TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none">• HAL: status
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	<ul style="list-style-type: none">• htim: TIM One Pulse handle• OutputChannel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1 selected– TIM_CHANNEL_2: TIM Channel 2 selected

Return values	<ul style="list-style-type: none">• HAL: status
HAL_TIM_Encoder_Init	
Function name	HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)
Function description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none">• htim: TIM Encoder Interface handle• sConfig: TIM Encoder Interface configuration structure
Return values	<ul style="list-style-type: none">• HAL: status
HAL_TIM_Encoder_Delinit	
Function name	HAL_StatusTypeDef HAL_TIM_Encoder_Delinit (TIM_HandleTypeDef * htim)
Function description	Deinitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none">• htim: TIM Encoder handle
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None:
HAL_TIM_Encoder_MspDelinit	
Function name	void HAL_TIM_Encoder_MspDelinit (TIM_HandleTypeDef * htim)
Function description	Deinitializes TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Encoder Interface handle• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM Encoder Interface handle• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status
HAL_TIM_IRQHandler	
Function name	void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)
Function description This function handles TIM interrupts requests.	
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None:
HAL_TIM_OC_ConfigChannel	
Function name	HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, 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	<ul style="list-style-type: none">• htim: TIM Output Compare handle• sConfig: TIM Output Compare configuration structure• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status

HAL_TIM_PWM_ConfigChannel

Function name	HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, 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	<ul style="list-style-type: none">• htim: TIM handle• sConfig: TIM PWM configuration structure• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status

HAL_TIM_IC_ConfigChannel

Function name	HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, 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	<ul style="list-style-type: none">• htim: TIM IC handle• sConfig: TIM Input Capture configuration structure• Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM One Pulse handle• sConfig: TIM One Pulse configuration structure• OutputChannel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1 selected– TIM_CHANNEL_2: TIM Channel 2 selected• InputChannel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1 selected– TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIM_ConfigOCrefClear

Function name	HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)
---------------	---

Function description	Configures the OCRef clear feature.
----------------------	-------------------------------------

Parameters	<ul style="list-style-type: none">• 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:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1– TIM_CHANNEL_2: TIM Channel 2– TIM_CHANNEL_3: TIM Channel 3– TIM_CHANNEL_4: TIM Channel 4• 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:<ul style="list-style-type: none">– 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
Return values	<ul style="list-style-type: none">• HAL: status• None:

HAL_TIM_ConfigClockSource

Function name	HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)
---------------	--

Function description	Configures the clock source to be used.
----------------------	---

Parameters	<ul style="list-style-type: none">• htim: TIM handle• sClockSourceConfig: pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status
HAL_TIM_SlaveConfigSynchronization	
Function name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function description	Configures the TIM in Slave mode.
Parameters	<ul style="list-style-type: none">• 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) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	<ul style="list-style-type: none">• HAL: status
HAL_TIM_SlaveConfigSynchronization_IT	
Function name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function description	Configures the TIM in Slave mode in interrupt mode.
Parameters	<ul style="list-style-type: none">• 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) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIM_DMABurst_WriteStart

Function name	<code>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)</code>
Function description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.
Parameters	<ul style="list-style-type: none">• 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:<ul style="list-style-type: none">– <code>TIM_DMABASE_CR1</code>– <code>TIM_DMABASE_CR2</code>– <code>TIM_DMABASE_SMCR</code>– <code>TIM_DMABASE_DIER</code>– <code>TIM_DMABASE_SR</code>– <code>TIM_DMABASE_EGR</code>– <code>TIM_DMABASE_CCMR1</code>– <code>TIM_DMABASE_CCMR2</code>– <code>TIM_DMABASE_CCER</code>– <code>TIM_DMABASE_CNT</code>– <code>TIM_DMABASE_PSC</code>– <code>TIM_DMABASE_ARR</code>– <code>TIM_DMABASE_RCR</code>– <code>TIM_DMABASE_CCR1</code>– <code>TIM_DMABASE_CCR2</code>– <code>TIM_DMABASE_CCR3</code>– <code>TIM_DMABASE_CCR4</code>– <code>TIM_DMABASE_BDTR</code>– <code>TIM_DMABASE_DCR</code>• BurstRequestSrc: TIM DMA Request sources This parameter can be one of the following values:<ul style="list-style-type: none">– <code>TIM_DMA_UPDATE</code>: TIM update Interrupt source– <code>TIM_DMA_CC1</code>: TIM Capture Compare 1 DMA source– <code>TIM_DMA_CC2</code>: TIM Capture Compare 2 DMA source– <code>TIM_DMA_CC3</code>: TIM Capture Compare 3 DMA source– <code>TIM_DMA_CC4</code>: TIM Capture Compare 4 DMA source– <code>TIM_DMA_COM</code>: TIM Commutation DMA source– <code>TIM_DMA_TRIGGER</code>: TIM Trigger DMA source• BurstBuffer: The Buffer address.• BurstLength: DMA Burst length. This parameter can be one value between: <code>TIM_DMABURSTLENGTH_1TRANSFER</code> and <code>TIM_DMABURSTLENGTH_18TRANSFERS</code>.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIM_DMABurst_MultiWriteStart

Function name	<code>HAL_StatusTypeDef HAL_TIM_DMABurst_MultiWriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)</code>
Function description	Configure the DMA Burst to transfer multiple Data from the memory to the TIM peripheral.
Parameters	<ul style="list-style-type: none">• 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:<ul style="list-style-type: none">– <code>TIM_DMABASE_CR1</code>– <code>TIM_DMABASE_CR2</code>– <code>TIM_DMABASE_SMCR</code>– <code>TIM_DMABASE_DIER</code>– <code>TIM_DMABASE_SR</code>– <code>TIM_DMABASE_EGR</code>– <code>TIM_DMABASE_CCMR1</code>– <code>TIM_DMABASE_CCMR2</code>– <code>TIM_DMABASE_CCER</code>– <code>TIM_DMABASE_CNT</code>– <code>TIM_DMABASE_PSC</code>– <code>TIM_DMABASE_ARR</code>– <code>TIM_DMABASE_RCR</code>– <code>TIM_DMABASE_CCR1</code>– <code>TIM_DMABASE_CCR2</code>– <code>TIM_DMABASE_CCR3</code>– <code>TIM_DMABASE_CCR4</code>– <code>TIM_DMABASE_BDTR</code>– <code>TIM_DMABASE_DCR</code>• BurstRequestSrc: TIM DMA Request sources This parameter can be one of the following values:<ul style="list-style-type: none">– <code>TIM_DMA_UPDATE</code>: TIM update Interrupt source– <code>TIM_DMA_CC1</code>: TIM Capture Compare 1 DMA source– <code>TIM_DMA_CC2</code>: TIM Capture Compare 2 DMA source– <code>TIM_DMA_CC3</code>: TIM Capture Compare 3 DMA source– <code>TIM_DMA_CC4</code>: TIM Capture Compare 4 DMA source– <code>TIM_DMA_COM</code>: TIM Commutation DMA source– <code>TIM_DMA_TRIGGER</code>: TIM Trigger DMA source• BurstBuffer: The Buffer address.• BurstLength: DMA Burst length. This parameter can be one value between: <code>TIM_DMABURSTLENGTH_1TRANSFER</code> and <code>TIM_DMABURSTLENGTH_18TRANSFERS</code>.• DataLength: Data length. This parameter can be one value between 1 and <code>0xFFFF</code>.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIM_DMABurst_WriteStop

Function name	HAL_StatusTypeDef HAL_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)
Function description	Stops the TIM DMA Burst mode.
Parameters	<ul style="list-style-type: none">htim: TIM handleBurstRequestSrc: TIM DMA Request sources to disable
Return values	<ul style="list-style-type: none">HAL: status

HAL_TIM_DMABurst_ReadStart

Function name	HAL_StatusTypeDef HAL_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 starts 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_DCR
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
 - TIM_DMA_UPDATE: TIM update Interrupt source
 - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
 - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
 - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
 - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
 - TIM_DMA_COM: TIM Commutation DMA source
 - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between:
`TIM_DMABURSTLENGTH_1TRANSFER` and
`TIM_DMABURSTLENGTH_18TRANSFERS`.

Return values

- **HAL:** status

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 multiple Data from the TIM peripheral to the memory.

Parameters	<ul style="list-style-type: none">• htim: TIM handle• BurstBaseAddress: TIM Base address from where the DMA will starts the Data read This parameter can be one of the following values:<ul style="list-style-type: none">- 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• BurstRequestSrc: TIM DMA Request sources This parameter can be one of the following values:<ul style="list-style-type: none">- TIM_DMA_UPDATE: TIM update Interrupt source- TIM_DMA_CC1: TIM Capture Compare 1 DMA source- TIM_DMA_CC2: TIM Capture Compare 2 DMA source- TIM_DMA_CC3: TIM Capture Compare 3 DMA source- TIM_DMA_CC4: TIM Capture Compare 4 DMA source- TIM_DMA_COM: TIM Commutation DMA source- TIM_DMA_TRIGGER: TIM Trigger DMA source• BurstBuffer: The Buffer address.• BurstLength: DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.• DataLength: Data length. This parameter can be one value between 1 and 0xFFFF.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIM_DMABurst_ReadStop

Function name	HAL_StatusTypeDef HAL_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)
Function description	Stop the DMA burst reading.
Parameters	<ul style="list-style-type: none">• htim: TIM handle• BurstRequestSrc: TIM DMA Request sources to disable.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIM_GenerateEvent

Function name	<code>HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)</code>
Function description	Generate a software event.
Parameters	<ul style="list-style-type: none">htim: TIM handleEventSource: specifies the event source. This parameter can be one of the following values:<ul style="list-style-type: none">- <code>TIM_EVENTSOURCE_UPDATE</code>: Timer update Event source- <code>TIM_EVENTSOURCE_CC1</code>: Timer Capture Compare 1 Event source- <code>TIM_EVENTSOURCE_CC2</code>: Timer Capture Compare 2 Event source- <code>TIM_EVENTSOURCE_CC3</code>: Timer Capture Compare 3 Event source- <code>TIM_EVENTSOURCE_CC4</code>: Timer Capture Compare 4 Event source- <code>TIM_EVENTSOURCE_COM</code>: Timer COM event source- <code>TIM_EVENTSOURCE_TRIGGER</code>: Timer Trigger Event source- <code>TIM_EVENTSOURCE_BREAK</code>: Timer Break event source
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">TIM6 and TIM7 can only generate an update event.<code>TIM_EVENTSOURCE_COM</code> and <code>TIM_EVENTSOURCE_BREAK</code> are used only with TIM1, TIM15, TIM16 and TIM17.

HAL_TIM_ReadCapturedValue

Function name	<code>uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function description	Read the captured value from Capture Compare unit.
Parameters	<ul style="list-style-type: none">htim: TIM handle.Channel: TIM Channels to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">- <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected- <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected- <code>TIM_CHANNEL_3</code> : TIM Channel 3 selected- <code>TIM_CHANNEL_4</code> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none">Captured: value

HAL_TIM_PeriodElapsedCallback

Function name	<code>void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)</code>
Function description	Period elapsed callback in non blocking mode.
Parameters	<ul style="list-style-type: none">htim: TIM handle
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">• htim: TIM OC handle
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM IC handle
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None:

HAL_TIM_TriggerCallback

Function name	void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)
Function description	Hall Trigger detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None:

HAL_TIM_ErrorCallback

Function name	void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)
Function description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None:

HAL_TIM_Base_GetState

Function name	HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)
Function description	Return the TIM Base state.
Parameters	<ul style="list-style-type: none">• htim: TIM Base handle
Return values	<ul style="list-style-type: none">• HAL: state

HAL_TIM_OC_GetState

Function name	HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)
Function description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none">• htim: TIM Ouput Compare handle
Return values	<ul style="list-style-type: none">• HAL: state

HAL_TIM_PWM_GetState

Function name	HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)
Function description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• HAL: state

HAL_TIM_IC_GetState

Function name	HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)
Function description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none">• htim: TIM IC handle
Return values	<ul style="list-style-type: none">• HAL: state

HAL_TIM_OnePulse_GetState

Function name	HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)
Function description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none">• htim: TIM OPM handle
Return values	<ul style="list-style-type: none">• HAL: state

HAL_TIM_Encoder_GetState

Function name	HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)
Function description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none">• htim: TIM Encoder handle
Return values	<ul style="list-style-type: none">• HAL: state

TIM_ETR_SetConfig

Function name	void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)
Function description	Configures the TIMx External Trigger (ETR).
Parameters	<ul style="list-style-type: none">• TIMx: to select the TIM peripheral• TIM_ExtTRGPrescaler: The external Trigger Prescaler. This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_ETRPRESCALER_DIV1 : ETRP Prescaler OFF.– TIM_ETRPRESCALER_DIV2 : ETRP frequency divided by 2.– TIM_ETRPRESCALER_DIV4 : ETRP frequency divided by 4.– TIM_ETRPRESCALER_DIV8 : ETRP frequency divided by 8.• TIM_ExtTRGPolarity: The external Trigger Polarity. This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_ETRPOLARITY_INVERTED : active low or falling edge active.– TIM_ETRPOLARITY_NONINVERTED : active high or rising edge active.• ExtTRGFilter: External Trigger Filter. This parameter must be a value between 0x00 and 0x0F
Return values	<ul style="list-style-type: none">• None:

42.3 TIM Firmware driver defines

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

42.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

TIM Auto-Reload Preload

TIM_AUTORELOAD_PREL TIMx_ARR register is not buffered
OAD_DISABLE

TIM_AUTORELOAD_PREL TIMx_ARR register is buffered
OAD_ENABLE

TIM Break Input Enable Disable

TIM_BREAK_ENABLE

TIM_BREAK_DISABLE

TIM Break Input Polarity

TIM_BREAKPOLARITY_LO
W

TIM_BREAKPOLARITY_HI
GH

TIM Channel

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_ALL

TIM Capture/Compare Channel State

TIM_CCx_ENABLE

TIM_CCx_DISABLE

TIM_CCxN_ENABLE

TIM_CCxN_DISABLE

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARIT Polarity for ETRx pin
Y_INVERTED

TIM_CLEARINPUTPOLARIT Polarity for ETRx pin
Y_NONINVERTED

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCA No prescaler is used
LER_DIV1

TIM_CLEARINPUTPRESCA Prescaler for External ETR pin: Capture performed once every 2 events.
LER_DIV2

TIM_CLEARINPUTPRESCA Prescaler for External ETR pin: Capture performed once every 4 events.
LER_DIV4

TIM_CLEARINPUTPRESCA Prescaler for External ETR pin: Capture performed once every 8 events.
LER_DIV8

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

TIM_CLOCKDIVISION_DIV2

TIM_CLOCKDIVISION_DIV4

TIM Clock Polarity

TIM_CLOCKPOLARITY_INV Polarity for ETRx clock sources
ERTED

TIM_CLOCKPOLARITY_NO Polarity for ETRx clock sources
NINVERTED

TIM_CLOCKPOLARITY_RIS Polarity for TIx clock sources
ING

TIM_CLOCKPOLARITY_FA Polarity for TIx clock sources
LLING

TIM_CLOCKPOLARITY_BO Polarity for TIx clock sources
THEEDGE

TIM Clock Prescaler

TIM_CLOCKPRESCALER_ No prescaler is used
DIV1

TIM_CLOCKPRESCALER_ Prescaler for External ETR Clock: Capture performed once every 2 events.
DIV2

TIM_CLOCKPRESCALER_ Prescaler for External ETR Clock: Capture performed once every 4 events.
DIV4

TIM_CLOCKPRESCALER_ Prescaler for External ETR Clock: Capture performed once every 8 events.
DIV8

TIM Clock Source

TIM_CLOCKSOURCE_ETR
MODE2

TIM_CLOCKSOURCE_INTE
RNAL

TIM_CLOCKSOURCE_ITR0

TIM_CLOCKSOURCE_ITR1

TIM_CLOCKSOURCE_ITR2

TIM_CLOCKSOURCE_ITR3

TIM_CLOCKSOURCE_TI1E
D

TIM_CLOCKSOURCE_TI1

TIM_CLOCKSOURCE_TI2

TIM_CLOCKSOURCE_ETR
MODE1

TIM Commutation Source

TIM_COMMUTATION_TRGI

TIM_COMMUTATION_SOFT
WARE

TIM Counter Mode

TIM_COUNTERMODE_UP

TIM_COUNTERMODE_DO
WN

TIM_COUNTERMODE_CEN
TERALIGNED1

TIM_COUNTERMODE_CEN
TERALIGNED2

TIM_COUNTERMODE_CEN
TERALIGNED3

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_OR`

TIM DMA Burst Length

`TIM_DMABURSTLENGTH_1TRANSFER`

`TIM_DMABURSTLENGTH_2TRANSFERS`

`TIM_DMABURSTLENGTH_3TRANSFERS`

`TIM_DMABURSTLENGTH_4TRANSFERS`

`TIM_DMABURSTLENGTH_5TRANSFERS`

`TIM_DMABURSTLENGTH_6TRANSFERS`

`TIM_DMABURSTLENGTH_7TRANSFERS`

TIM_DMABURSTLENGTH_8TRANSFERS

TIM_DMABURSTLENGTH_9TRANSFERS

TIM_DMABURSTLENGTH_10TRANSFERS

TIM_DMABURSTLENGTH_11TRANSFERS

TIM_DMABURSTLENGTH_12TRANSFERS

TIM_DMABURSTLENGTH_13TRANSFERS

TIM_DMABURSTLENGTH_14TRANSFERS

TIM_DMABURSTLENGTH_15TRANSFERS

TIM_DMABURSTLENGTH_16TRANSFERS

TIM_DMABURSTLENGTH_17TRANSFERS

TIM_DMABURSTLENGTH_18TRANSFERS

TIM DMA Handle Index

TIM_DMA_ID_UPDATE Index of the DMA handle used for Update DMA requests

TIM_DMA_ID_CC1 Index of the DMA handle used for Capture/Compare 1 DMA requests

TIM_DMA_ID_CC2 Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3 Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4 Index of the DMA handle used for Capture/Compare 4 DMA requests

TIM_DMA_ID_COMMUTATION Index of the DMA handle used for Commutation DMA requests

TIM_DMA_ID_TRIGGER Index of the DMA handle used for Trigger DMA requests

TIM DMA Sources

TIM_DMA_UPDATE

`TIM_DMA_CC1`

`TIM_DMA_CC2`

`TIM_DMA_CC3`

`TIM_DMA_CC4`

`TIM_DMA_COM`

`TIM_DMA_TRIGGER`

TIM Encoder Mode

`TIM_ENCODERMODE_TI1`

`TIM_ENCODERMODE_TI2`

`TIM_ENCODERMODE_TI12`

TIM ETR Polarity

`TIM_ETRPOLARITY_INVER` Polarity for ETR source
TED

`TIM_ETRPOLARITY_NONIN` Polarity for ETR source
VERTED

TIM ETR Prescaler

`TIM_ETRPRESCALER_DIV` No prescaler is used
1

`TIM_ETRPRESCALER_DIV` ETR input source is divided by 2
2

`TIM_ETRPRESCALER_DIV` ETR input source is divided by 4
4

`TIM_ETRPRESCALER_DIV` ETR input source is divided by 8
8

TIM Event Source

`TIM_EVENTSOURCE_UPD`
ATE

`TIM_EVENTSOURCE_CC1`

`TIM_EVENTSOURCE_CC2`

`TIM_EVENTSOURCE_CC3`

`TIM_EVENTSOURCE_CC4`

`TIM_EVENTSOURCE_COM`

`TIM_EVENTSOURCE_TRIG`
`GER`

`TIM_EVENTSOURCE_BRE`
`AK`

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:**

- Enables 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:

- Disables 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:

- Enables the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
 - TIM_DMA_UPDATE: Update DMA request
 - TIM_DMA_CC1: Capture/Compare 1 DMA request
 - TIM_DMA_CC2: Capture/Compare 2 DMA request
 - TIM_DMA_CC3: Capture/Compare 3 DMA request
 - TIM_DMA_CC4: Capture/Compare 4 DMA request
 - TIM_DMA_COM: Commutation DMA request
 - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DM

A

Description:

- Disables the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
 - TIM_DMA_UPDATE: Update DMA request
 - TIM_DMA_CC1: Capture/Compare 1 DMA request
 - TIM_DMA_CC2: Capture/Compare 2 DMA request
 - TIM_DMA_CC3: Capture/Compare 3 DMA request
 - TIM_DMA_CC4: Capture/Compare 4 DMA request
 - TIM_DMA_COM: Commutation DMA request
 - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

[__HAL_TIM_GET_FLAG](#)

Description:

- Checks 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_COM: Commutation interrupt flag
 - TIM_FLAG_TRIGGER: Trigger interrupt flag
 - TIM_FLAG_BREAK: Break interrupt flag
 - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
 - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
 - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
 - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

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

[__HAL_TIM_CLEAR_FLAG](#)

Description:

- Clears 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_COM: Commutation interrupt flag
 - TIM_FLAG_TRIGGER: Trigger interrupt flag
 - TIM_FLAG_BREAK: Break interrupt flag
 - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
 - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
 - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
 - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

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

__HAL_TIM_GET_IT_SOUR **Description:**

CE

- Checks whether the specified TIM interrupt has occurred or not.

Parameters:

- HANDLE: TIM handle
- INTERRUPT: specifies the TIM interrupt source to check.

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.

Return value:

- None

**__HAL_TIM_IS_TIM_COUN
TING_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_PRESCA **Description:**

LER

- Sets the TIM active prescaler register value on update event.

Parameters:

- HANDLE: TIM handle.
- PRES: specifies the active prescaler register new value.

Return value:

- None

__HAL_TIM_SET_COMPARE Description:

- E
- Sets 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
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE Description:

- E
- Gets 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

Return value:

- 16-bit or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_SET_COUNTER Description:

- R
- Sets 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:

- R
- Gets 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:**

- Sets 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:**

- Gets 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:**

- Sets 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:**

- Gets 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_ENABLE_OCx **Description:****Ppreload**

- Sets 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

Return value:

- None

_HAL_TIM_DISABLE_OCx **Description:****Ppreload**

- Resets 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

Return value:

- None

_HAL_TIM_SET_ICPRES **Description:****ALER**

- Sets 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_ICPRES](#) Description:

ALER

- Gets 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_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_CAPTUR` **Description:**
EPOLARITY

- Sets 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

Notes:

- The polarity `TIM_INPUTCHANNELPOLARITY_BOTHEDGE` is not authorized for TIM Channel 4.

TIM Flag Definition

`TIM_FLAG_UPDATE`

`TIM_FLAG_CC1`

`TIM_FLAG_CC2`

`TIM_FLAG_CC3`

`TIM_FLAG_CC4`

`TIM_FLAG_COM`

`TIM_FLAG_TRIGGER`

`TIM_FLAG_BREAK`

`TIM_FLAG_CC1OF`

`TIM_FLAG_CC2OF`

`TIM_FLAG_CC3OF`

`TIM_FLAG_CC4OF`

TIM Input Capture Polarity

`TIM_ICPOLARITY_RISING`

TIM_ICPOLARITY_FALLING

TIM_ICPOLARITY_BOTHEDGE

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_DIRC TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively TTI

TIM_ICSELECTION_INDIRE TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively CTTI

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

TIM_IT_CC1

TIM_IT_CC2

TIM_IT_CC3

TIM_IT_CC4

TIM_IT_COM

TIM_IT_TRIGGER

`TIM_IT_BREAK`

TIM Lock level

`TIM_LOCKLEVEL_OFF`

`TIM_LOCKLEVEL_1`

`TIM_LOCKLEVEL_2`

`TIM_LOCKLEVEL_3`

TIM Master Mode Selection

`TIM_TRGO_RESET`

`TIM_TRGO_ENABLE`

`TIM_TRGO_UPDATE`

`TIM_TRGO_OC1`

`TIM_TRGO_OC1REF`

`TIM_TRGO_OC2REF`

`TIM_TRGO_OC3REF`

`TIM_TRGO_OC4REF`

TIM Master Slave Mode

`TIM_MASTERSLAVEMODE_ENABLE`

`TIM_MASTERSLAVEMODE_DISABLE`

TIM One Pulse Mode

`TIM_OPMODE_SINGLE`

`TIM_OPMODE_REPETITIVE`

TIM OSSI Off State Selection for Idle mode state

`TIM_OSSI_ENABLE`

`TIM_OSSI_DISABLE`

TIM OSSR Off State Selection for Run mode state

`TIM_OSSR_ENABLE`

TIM_OSSR_DISABLE

TIM Output Compare and PWM modes

TIM_OCMODE_TIMING

TIM_OCMODE_ACTIVE

TIM_OCMODE_INACTIVE

TIM_OCMODE_TOGGLE

TIM_OCMODE_PWM1

TIM_OCMODE_PWM2

**TIM_OCMODE_FORCED_A
CTIVE**

**TIM_OCMODE_FORCED_IN
ACTIVE**

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

TIM_OCIDLESTATE_RESET

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

TIM_OCNIDLESTATE_RESET

TIM Complementary Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Fast State

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

TIM Slave Mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

TIM_TI1SELECTION_XORCOMBINATION

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_I Polarity for ETRx trigger sources INVERTED

TIM_TRIGGERPOLARITY_N Polarity for ETRx trigger sources NONINVERTED

TIM_TRIGGERPOLARITY_R Polarity for TIxFPx or TI1_ED trigger sources RISING

TIM_TRIGGERPOLARITY_F Polarity for TIxFPx or TI1_ED trigger sources FALLING

TIM_TRIGGERPOLARITY_B Polarity for TIxFPx or TI1_ED trigger sources BOTHEdge

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1 No prescaler is used

TIM_TRIGGERPRESCALER_DIV2 Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4 Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8 Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

[TIM_TS_ITR1](#)

[TIM_TS_ITR2](#)

[TIM_TS_ITR3](#)

[TIM_TS_TI1F_ED](#)

[TIM_TS_TI1FP1](#)

[TIM_TS_TI2FP2](#)

[TIM_TS_ETRF](#)

[TIM_TS_NONE](#)

43 HAL TIM Extension Driver

43.1 TIME Firmware driver registers structures

43.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the `stm32f0xx_hal_tim_ex.h`

Data Fields

- *uint32_t IC1Polarity*
- *uint32_t IC1Prescaler*
- *uint32_t IC1Filter*
- *uint32_t Commutation_Delay*

Field Documentation

- *uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity*

Specifies the active edge of the input signal. This parameter can be a value of [*TIM_Input_Capture_Polarity*](#)

- *uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler*

Specifies the Input Capture Prescaler. This parameter can be a value of [*TIM_Input_Capture_Prescaler*](#)

- *uint32_t TIM_HallSensor_InitTypeDef::IC1Filter*

Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

- *uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay*

Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

43.1.2 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the `stm32f0xx_hal_tim_ex.h`

Data Fields

- *uint32_t MasterOutputTrigger*
- *uint32_t MasterSlaveMode*

Field Documentation

- *uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger*

Trigger output (TRGO) selection This parameter can be a value of [*TIM_Master_Mode_Selection*](#)

- *uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode*

Master/slave mode selection This parameter can be a value of [*TIM_Master_Slave_Mode*](#)

43.1.3 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the `stm32f0xx_hal_tim_ex.h`

Data Fields

- *uint32_t OffStateRunMode*
- *uint32_t OffStateIDLEMode*
- *uint32_t LockLevel*
- *uint32_t DeadTime*
- *uint32_t BreakState*
- *uint32_t BreakPolarity*
- *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::AutomaticOutput`**
TIM Automatic Output Enable state This parameter can be a value of `TIM_AOE_Bit_Set_Reset`

43.2 TIMEEx Firmware driver API description

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

43.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
 - Output Compare
 - PWM generation (Edge and Center-aligned Mode)
 - One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

43.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
 - Complementary Output Compare : `HAL_TIM_OC_MspInit()`
 - Complementary PWM generation : `HAL_TIM_PWM_MspInit()`
 - Complementary One-pulse mode output : `HAL_TIM_OnePulse_MspInit()`
 - Hall Sensor output : `HAL_TIM_HallSensor_MspInit()`
2. Initialize the TIM low level resources :
 - a. Enable the TIM interface clock using `__HAL_RCC_TIMx_CLK_ENABLE()`;
 - b. TIM pins configuration
 - Enable the clock for the TIM GPIOs using the following function:
`__HAL_RCC_GPIOx_CLK_ENABLE();`
 - Configure these TIM pins in Alternate function mode using `HAL_GPIO_Init()`;
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: `HAL_TIM_ConfigClockSource`, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
 - `HAL_TIMEEx_HallSensor_Init` and `HAL_TIMEEx_ConfigCommutationEvent`: to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).

5. Activate the TIM peripheral using one of the start functions:
 - Complementary Output Compare : `HAL_TIMEx_OCN_Start()`, `HAL_TIMEx_OCN_Start_DMA()`,
`HAL_TIMEx_OCN_Start_IT()`
 - Complementary PWM generation : `HAL_TIMEx_PWMN_Start()`, `HAL_TIMEx_PWMN_Start_DMA()`,
`HAL_TIMEx_PWMN_Start_IT()`
 - Complementary One-pulse mode output : `HAL_TIMEx_OnePulseN_Start()`,
`HAL_TIMEx_OnePulseN_Start_IT()`
 - Hall Sensor output : `HAL_TIMEx_HallSensor_Start()`, `HAL_TIMEx_HallSensor_Start_DMA()`,
`HAL_TIMEx_HallSensor_Start_IT()`.

43.2.3

Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.

This section contains the following APIs:

- [`HAL_TIMEx_HallSensor_Init`](#)
- [`HAL_TIMEx_HallSensor_DelInit`](#)
- [`HAL_TIMEx_HallSensor_MspInit`](#)
- [`HAL_TIMEx_HallSensor_MspDelInit`](#)
- [`HAL_TIMEx_HallSensor_Start`](#)
- [`HAL_TIMEx_HallSensor_Stop`](#)
- [`HAL_TIMEx_HallSensor_Start_IT`](#)
- [`HAL_TIMEx_HallSensor_Stop_IT`](#)
- [`HAL_TIMEx_HallSensor_Start_DMA`](#)
- [`HAL_TIMEx_HallSensor_Stop_DMA`](#)

43.2.4

Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.

This section contains the following APIs:

- [`HAL_TIMEx_OCN_Start`](#)
- [`HAL_TIMEx_OCN_Stop`](#)
- [`HAL_TIMEx_OCN_Start_IT`](#)
- [`HAL_TIMEx_OCN_Stop_IT`](#)
- [`HAL_TIMEx_OCN_Start_DMA`](#)
- [`HAL_TIMEx_OCN_Stop_DMA`](#)

43.2.5

Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- [`HAL_TIMEx_PWMN_Start`](#)
- [`HAL_TIMEx_PWMN_Stop`](#)
- [`HAL_TIMEx_PWMN_Start_IT`](#)
- [`HAL_TIMEx_PWMN_Stop_IT`](#)
- [`HAL_TIMEx_PWMN_Start_DMA`](#)
- [`HAL_TIMEx_PWMN_Stop_DMA`](#)

43.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- [`HAL_TIMEx_OnePulseN_Start`](#)
- [`HAL_TIMEx_OnePulseN_Stop`](#)
- [`HAL_TIMEx_OnePulseN_Start_IT`](#)
- [`HAL_TIMEx_OnePulseN_Stop_IT`](#)

43.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.

This section contains the following APIs:

- [`HAL_TIMEx_ConfigCommutationEvent`](#)
- [`HAL_TIMEx_ConfigCommutationEvent_IT`](#)
- [`HAL_TIMEx_ConfigCommutationEvent_DMA`](#)
- [`HAL_TIMEx_MasterConfigSynchronization`](#)
- [`HAL_TIMEx_ConfigBreakDeadTime`](#)
- [`HAL_TIMEx_RemapConfig`](#)

43.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- [*HAL_TIMEx_CommputationCallback*](#)
- [*HAL_TIMEx_BreakCallback*](#)
- [*TIMEx_DMACommputationCplt*](#)

43.2.9 Extension Peripheral State 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_TIMEx_HallSensor_GetState*](#)

43.2.10 Detailed description of functions

[*HAL_TIMEx_HallSensor_Init*](#)

Function name [*HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init \(TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig\)*](#)

Function description Initializes the TIM Hall Sensor Interface and create the associated handle.

Parameters

- **htim:** TIM Encoder Interface handle
- **sConfig:** TIM Hall Sensor configuration structure

Return values

- **HAL:** status

[*HAL_TIMEx_HallSensor_DelInit*](#)

Function name [*HAL_StatusTypeDef HAL_TIMEx_HallSensor_DelInit \(TIM_HandleTypeDef * htim\)*](#)

Function description Delinitializes the TIM Hall Sensor interface.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** status

[*HAL_TIMEx_HallSensor_MspInit*](#)

Function name [*void HAL_TIMEx_HallSensor_MspInit \(TIM_HandleTypeDef * htim\)*](#)

Function description Initializes the TIM Hall Sensor MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

[*HAL_TIMEx_HallSensor_MspDelInit*](#)

Function name [*void HAL_TIMEx_HallSensor_MspDelInit \(TIM_HandleTypeDef * htim\)*](#)

Function description DeInitializes TIM Hall Sensor MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_Start

Function name **HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)**

Function description Starts the TIM Hall Sensor Interface.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop

Function name **HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)**

Function description Stops the TIM Hall sensor Interface.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name **HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)**

Function description Starts the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)**

Function description Stops the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**

Function description Starts the TIM Hall Sensor Interface in DMA mode.

- Parameters**
- **htim:** TIM Hall Sensor handle
 - **pData:** The destination Buffer address.
 - **Length:** The length of data to be transferred from TIM peripheral to memory.

- Return values**
- **HAL:** status

HAL_TIMEEx_HallSensor_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIMEEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)**

Function description Stops the TIM Hall Sensor Interface in DMA mode.

- Parameters**
- **htim:** TIM handle

- Return values**
- **HAL:** status

HAL_TIMEEx_OCN_Start

Function name **HAL_StatusTypeDef HAL_TIMEEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Output Compare signal generation on the complementary output.

- Parameters**
- **htim:** TIM Output Compare handle
 - **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEEx_OCN_Stop

Function name **HAL_StatusTypeDef HAL_TIMEEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Output Compare signal generation on the complementary output.

- Parameters**
- **htim:** TIM handle
 - **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEEx_OCN_Start_IT

Function name **HAL_StatusTypeDef HAL_TIMEEx_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	<ul style="list-style-type: none">• htim: TIM OC handle• Channel: TIM Channel to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status
HAL_TIMEEx_OCN_Stop_IT	
Function name	HAL_StatusTypeDef HAL_TIMEEx_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	<ul style="list-style-type: none">• htim: TIM Output Compare handle• Channel: TIM Channel to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status
HAL_TIMEEx_OCN_Start_DMA	
Function name	HAL_StatusTypeDef HAL_TIMEEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function description	Starts the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none">• htim: TIM Output Compare handle• Channel: TIM Channel to be enabled This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status
HAL_TIMEEx_OCN_Stop_DMA	
Function name	HAL_StatusTypeDef HAL_TIMEEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

- Parameters**
- **htim:** TIM Output Compare handle
 - **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEx_PWMN_Start

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the PWM signal generation on the complementary output.

- Parameters**
- **htim:** TIM handle
 - **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the PWM signal generation on the complementary output.

- Parameters**
- **htim:** TIM handle
 - **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the PWM signal generation in interrupt mode on the complementary output.

- Parameters**
- **htim:** TIM handle
 - **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the PWM signal generation in interrupt mode on the complementary output.

- Parameters**
- **htim:** TIM handle
 - **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**

Function description Starts the TIM PWM signal generation in DMA mode on the complementary output.

- Parameters**
- **htim:** TIM handle
 - **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
 - **pData:** The source Buffer address.
 - **Length:** The length of data to be transferred from memory to TIM peripheral

- Return values**
- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM PWM signal generation in DMA mode on the complementary output.

- Parameters**
- **htim:** TIM handle
 - **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

- Return values**
- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name **HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Starts the TIM One Pulse signal generation on the complementary output.

- Parameters**
- **htim:** TIM One Pulse handle
 - **OutputChannel:** 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

- Return values**
- **HAL:** status

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:** 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

- Return values**
- **HAL:** status

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:** 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

Return values	<ul style="list-style-type: none">• HAL: status
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	<ul style="list-style-type: none">• htim: TIM One Pulse handle• OutputChannel: TIM Channel to be disabled This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_CHANNEL_1: TIM Channel 1 selected– TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none">• HAL: status
HAL_TIMEx_ConfigCommutationEvent	
Function name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function description	Configure the TIM commutation event sequence.
Parameters	<ul style="list-style-type: none">• htim: TIM handle• InputTrigger: the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_TS_ITR0: Internal trigger 0 selected– TIM_TS_ITR1: Internal trigger 1 selected– TIM_TS_ITR2: Internal trigger 2 selected– TIM_TS_ITR3: Internal trigger 3 selected– TIM_TS_NONE: No trigger is needed• CommutationSource: the Commutation Event source This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
HAL_TIMEx_ConfigCommutationEvent_IT	
Function name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function description	Configure the TIM commutation event sequence with interrupt.

- Parameters**
- **htim:** TIM handle
 - **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
 - TIM_TS_ITR0: Internal trigger 0 selected
 - TIM_TS_ITR1: Internal trigger 1 selected
 - TIM_TS_ITR2: Internal trigger 2 selected
 - TIM_TS_ITR3: Internal trigger 3 selected
 - TIM_TS_NONE: No trigger is needed
 - **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
 - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
 - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
- Return values**
- **HAL:** status
- Notes**
- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEEx_ConfigCommutationEvent_DMA

- Function name**
- HAL_StatusTypeDef HAL_TIMEEx_ConfigCommutationEvent_DMA (TIM_HandleTypeDef *
htim, uint32_t InputTrigger, uint32_t CommutationSource)**
- Function description**
- Configure the TIM commutation event sequence with DMA.
- Parameters**
- **htim:** TIM handle
 - **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
 - TIM_TS_ITR0: Internal trigger 0 selected
 - TIM_TS_ITR1: Internal trigger 1 selected
 - TIM_TS_ITR2: Internal trigger 2 selected
 - TIM_TS_ITR3: Internal trigger 3 selected
 - TIM_TS_NONE: No trigger is needed
 - **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
 - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
 - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
- Return values**
- **HAL:** status

Notes

- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
- : The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEEx_MasterConfigSynchronization

Function name	HAL_StatusTypeDef HAL_TIMEEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)
Function description	Configures the TIM in master mode.
Parameters	<ul style="list-style-type: none">• htim: TIM handle.• sMasterConfig: pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIMEEx_ConfigBreakDeadTime

Function name	HAL_StatusTypeDef HAL_TIMEEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)
Function description	Configures the Break feature, dead time, Lock level, OSS1/OSSR State and the AOE(automatic output enable).
Parameters	<ul style="list-style-type: none">• htim: TIM handle• sBreakDeadTimeConfig: pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIMEEx_RemapConfig

Function name	HAL_StatusTypeDef HAL_TIMEEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)
Function description	Configures the TIM14 Remapping input capabilities.
Parameters	<ul style="list-style-type: none">• htim: TIM handle.• Remap: specifies the TIM remapping source. This parameter can be one of the following values:<ul style="list-style-type: none">– TIM_TIM14_GPIO: TIM14 TI1 is connected to GPIO– TIM_TIM14_RTC: TIM14 TI1 is connected to RTC_clock– TIM_TIM14_HSE: TIM14 TI1 is connected to HSE/32– TIM_TIM14_MCO: TIM14 TI1 is connected to MCO
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TIMEEx_CommuationCallback

Function name	void HAL_TIMEEx_CommuationCallback (TIM_HandleTypeDef * htim)
Function description	Hall commutation changed callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None:

HAL_TIMEEx_BreakCallback

Function name	void HAL_TIMEEx_BreakCallback (TIM_HandleTypeDef * htim)
Function description	Hall Break detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None:

HAL_TIMEEx_HallSensor_GetState

Function name	HAL_TIM_StateTypeDef HAL_TIMEEx_HallSensor_GetState (TIM_HandleTypeDef * htim)
Function description	Return the TIM Hall Sensor interface state.
Parameters	<ul style="list-style-type: none">• htim: TIM Hall Sensor handle
Return values	<ul style="list-style-type: none">• HAL: state

TIMEx_DMACommuationCplt

Function name	void TIMEx_DMACommuationCplt (DMA_HandleTypeDef * hdma)
Function description	TIM DMA Commutation callback.
Parameters	<ul style="list-style-type: none">• hdma: pointer to DMA handle.
Return values	<ul style="list-style-type: none">• None:

43.3 TIMEEx Firmware driver defines

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

43.3.1 TIMEEx

TIMEEx

TIMEEx Clear Input Source

TIM_CLEARINPUTSOURCE
_NONE

**TIM_CLEARINPUTSOURCE
_ETR**

**TIM_CLEARINPUTSOURCE
_OCREFCLR**

TIME_x Remap

TIM_TIM14_GPIO TIM14 TI1 is connected to GPIO

TIM_TIM14_RTC TIM14 TI1 is connected to RTC_clock

TIM_TIM14_HSE TIM14 TI1 is connected to HSE/32

TIM_TIM14_MCO TIM14 TI1 is connected to MCO

44 HAL TSC Generic Driver

44.1 TSC Firmware driver registers structures

44.1.1 TSC_InitTypeDef

`TSC_InitTypeDef` is defined in the `stm32f0xx_hal_tsc.h`

Data Fields

- `uint32_t CTPulseHighLength`
- `uint32_t CTPulseLowLength`
- `uint32_t SpreadSpectrum`
- `uint32_t SpreadSpectrumDeviation`
- `uint32_t SpreadSpectrumPrescaler`
- `uint32_t PulseGeneratorPrescaler`
- `uint32_t MaxCountValue`
- `uint32_t IODefaultMode`
- `uint32_t SynchroPinPolarity`
- `uint32_t AcquisitionMode`
- `uint32_t MaxCountInterrupt`
- `uint32_t ChannelIOs`
- `uint32_t ShieldIOs`
- `uint32_t SamplingIOs`

Field Documentation

- `uint32_t TSC_InitTypeDef::CTPulseHighLength`
Charge-transfer high pulse length
- `uint32_t TSC_InitTypeDef::CTPulseLowLength`
Charge-transfer low pulse length
- `uint32_t TSC_InitTypeDef::SpreadSpectrum`
Spread spectrum activation
- `uint32_t TSC_InitTypeDef::SpreadSpectrumDeviation`
Spread spectrum deviation
- `uint32_t TSC_InitTypeDef::SpreadSpectrumPrescaler`
Spread spectrum prescaler
- `uint32_t TSC_InitTypeDef::PulseGeneratorPrescaler`
Pulse generator prescaler
- `uint32_t TSC_InitTypeDef::MaxCountValue`
Max count value
- `uint32_t TSC_InitTypeDef::IODefaultMode`
IO default mode
- `uint32_t TSC_InitTypeDef::SynchroPinPolarity`
Synchro pin polarity
- `uint32_t TSC_InitTypeDef::AcquisitionMode`
Acquisition mode
- `uint32_t TSC_InitTypeDef::MaxCountInterrupt`
Max count interrupt activation
- `uint32_t TSC_InitTypeDef::ChannelIOs`
Channel IOs mask

- *uint32_t TSC_InitTypeDef::ShieldIOs*
Shield IOs mask
- *uint32_t TSC_InitTypeDef::SamplingIOs*
Sampling IOs mask

44.1.2 **TSC_IOConfigTypeDef**

TSC_IOConfigTypeDef is defined in the `stm32f0xx_hal_tsc.h`

Data Fields

- *uint32_t ChannelIOs*
- *uint32_t ShieldIOs*
- *uint32_t SamplingIOs*

Field Documentation

- *uint32_t TSC_IOConfigTypeDef::ChannelIOs*
Channel IOs mask
- *uint32_t TSC_IOConfigTypeDef::ShieldIOs*
Shield IOs mask
- *uint32_t TSC_IOConfigTypeDef::SamplingIOs*
Sampling IOs mask

44.1.3 **TSC_HandleTypeDef**

TSC_HandleTypeDef is defined in the `stm32f0xx_hal_tsc.h`

Data Fields

- *TSC_TypeDef * Instance*
- *TSC_InitTypeDef Init*
- *__IO HAL_TSC_StateTypeDef State*
- *HAL_LockTypeDef Lock*

Field Documentation

- *TSC_TypeDef* TSC_HandleTypeDef::Instance*
Register base address
- *TSC_InitTypeDef TSC_HandleTypeDef::Init*
Initialization parameters
- *__IO HAL_TSC_StateTypeDef TSC_HandleTypeDef::State*
Peripheral state
- *HAL_LockTypeDef TSC_HandleTypeDef::Lock*
Lock feature

44.2 TSC Firmware driver API description

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

44.2.1 TSC specific features

- Proven and robust surface charge transfer acquisition principle
- Supports up to 3 capacitive sensing channels per group
- Capacitive sensing channels can be acquired in parallel offering a very good response time
- Spread spectrum feature to improve system robustness in noisy environments
- Full hardware management of the charge transfer acquisition sequence
- Programmable charge transfer frequency
- Programmable sampling capacitor I/O pin
- Programmable channel I/O pin
- Programmable max count value to avoid long acquisition when a channel is faulty
- Dedicated end of acquisition and max count error flags with interrupt capability

- One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
- Compatible with proximity, touchkey, linear and rotary touch sensor implementation

44.2.2 How to use this driver

1. Enable the TSC interface clock using `__HAL_RCC_TSC_CLK_ENABLE()` macro.
2. GPIO pins configuration
 - Enable the clock for the TSC GPIOs using `__HAL_RCC_GPIOx_CLK_ENABLE()` macro.
 - Configure the TSC pins used as sampling IOs in alternate function output Open-Drain mode, and TSC pins used as channel/shield IOs in alternate function output Push-Pull mode using `HAL_GPIO_Init()` function.
 - Configure the alternate function on all the TSC pins using `HAL_xxxx()` function.
3. Interrupts configuration
 - Configure the NVIC (if the interrupt model is used) using `HAL_xxx()` function.
4. TSC configuration
 - Configure all TSC parameters and used TSC IOs using `HAL_TSC_Init()` function.

Acquisition sequence

- Discharge all IOs using `HAL_TSC_IODischarge()` function.
- Wait a certain time allowing a good discharge of all capacitors. This delay depends of the sampling capacitor and electrodes design.
- Select the channel IOs to be acquired using `HAL_TSC_IOConfig()` function.
- Launch the acquisition using either `HAL_TSC_Start()` or `HAL_TSC_Start_IT()` function. If the synchronized mode is selected, the acquisition will start as soon as the signal is received on the synchro pin.
- Wait the end of acquisition using either `HAL_TSC_PollForAcquisition()` or `HAL_TSC_GetState()` function or using WFI instruction for example.
- Check the group acquisition status using `HAL_TSC_GroupGetStatus()` function.
- Read the acquisition value using `HAL_TSC_GroupGetValue()` function.

44.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TSC.
- De-initialize the TSC.

This section contains the following APIs:

- [`HAL_TSC_Init`](#)
- [`HAL_TSC_DelInit`](#)
- [`HAL_TSC_MspInit`](#)
- [`HAL_TSC_MspDelInit`](#)

44.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure TSC IOs
- Discharge TSC IOs

This section contains the following APIs:

- [`HAL_TSC_IOConfig`](#)
- [`HAL_TSC_IODischarge`](#)

44.2.5 State functions

This subsection provides functions allowing to

- Get TSC state.
- Poll for acquisition completed.
- Handles TSC interrupt request.

This section contains the following APIs:

- [*HAL_TSC_GetState*](#)
- [*HAL_TSC_PollForAcquisition*](#)
- [*HAL_TSC_IRQHandler*](#)

44.2.6 Detailed description of functions

HAL_TSC_Init

Function name [**HAL_StatusTypeDef HAL_TSC_Init \(TSC_HandleTypeDef * htsc\)**](#)

Function description Initializes the TSC peripheral according to the specified parameters in the TSC_InitTypeDef structure.

Parameters • **htsc:** TSC handle

Return values • **HAL:** status

HAL_TSC_DelInit

Function name [**HAL_StatusTypeDef HAL_TSC_DelInit \(TSC_HandleTypeDef * htsc\)**](#)

Function description Deinitializes the TSC peripheral registers to their default reset values.

Parameters • **htsc:** TSC handle

Return values • **HAL:** status

HAL_TSC_MspInit

Function name [**void HAL_TSC_MspInit \(TSC_HandleTypeDef * htsc\)**](#)

Function description Initializes the TSC MSP.

Parameters • **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values • **None:**

HAL_TSC_MspDelInit

Function name [**void HAL_TSC_MspDelInit \(TSC_HandleTypeDef * htsc\)**](#)

Function description Deinitializes the TSC MSP.

Parameters • **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values • **None:**

HAL_TSC_Start

Function name [**HAL_StatusTypeDef HAL_TSC_Start \(TSC_HandleTypeDef * htsc\)**](#)

Function description	Starts the acquisition.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TSC_Start_IT

Function name	HAL_StatusTypeDef HAL_TSC_Start_IT (TSC_HandleTypeDef * htsc)
----------------------	--

Function description	Enables the interrupt and starts the acquisition.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• HAL: status.

HAL_TSC_Stop

Function name	HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)
----------------------	--

Function description	Stops the acquisition previously launched in polling mode.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TSC_Stop_IT

Function name	HAL_StatusTypeDef HAL_TSC_Stop_IT (TSC_HandleTypeDef * htsc)
----------------------	---

Function description	Stops the acquisition previously launched in interrupt mode.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• HAL: status

HAL_TSC_GroupGetStatus

Function name	TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (TSC_HandleTypeDef * htsc, uint32_t gx_index)
----------------------	--

Function description	Gets the acquisition status for a group.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.• gx_index: Index of the group
Return values	<ul style="list-style-type: none">• Group: status

HAL_TSC_GroupGetValue

Function name	<code>uint32_t HAL_TSC_GroupGetValue (TSC_HandleTypeDef * htsc, uint32_t gx_index)</code>
Function description	Gets the acquisition measure for a group.
Parameters	<ul style="list-style-type: none">htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.gx_index: Index of the group
Return values	<ul style="list-style-type: none">Acquisition: measure

HAL_TSC_IOConfig

Function name	<code>HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)</code>
Function description	Configures TSC IOs.
Parameters	<ul style="list-style-type: none">htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.config: pointer to the configuration structure.
Return values	<ul style="list-style-type: none">HAL: status

HAL_TSC_IODischarge

Function name	<code>HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, uint32_t choice)</code>
Function description	Discharge TSC IOs.
Parameters	<ul style="list-style-type: none">htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.choice: enable or disable
Return values	<ul style="list-style-type: none">HAL: status

HAL_TSC_GetState

Function name	<code>HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)</code>
Function description	Return the TSC state.
Parameters	<ul style="list-style-type: none">htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">HAL: state

HAL_TSC_PollForAcquisition

Function name	HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)
Function description	Start acquisition and wait until completion.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• HAL: state
Notes	<ul style="list-style-type: none">• There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

HAL_TSC_IRQHandler

Function name	void HAL_TSC_IRQHandler (TSC_HandleTypeDef * htsc)
Function description	Handles TSC interrupt request.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• None:

HAL_TSC_ConvCpltCallback

Function name	void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef * htsc)
Function description	Acquisition completed callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• None:

HAL_TSC_ErrorCallback

Function name	void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)
Function description	Error callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none">• None:

44.3 TSC Firmware driver defines

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

44.3.1 TSC
TSC
TSC Acquisition mode

TSC_ACQ_MODE_NORMA
L

TSC_ACQ_MODE_SYNCHR
O

IS_TSC_ACQ_MODE

TSC Charge Transfer Pulse High

TSC_CTPH_1CYCLE

TSC_CTPH_2CYCLES

TSC_CTPH_3CYCLES

TSC_CTPH_4CYCLES

TSC_CTPH_5CYCLES

TSC_CTPH_6CYCLES

TSC_CTPH_7CYCLES

TSC_CTPH_8CYCLES

TSC_CTPH_9CYCLES

TSC_CTPH_10CYCLES

TSC_CTPH_11CYCLES

TSC_CTPH_12CYCLES

TSC_CTPH_13CYCLES

TSC_CTPH_14CYCLES

TSC_CTPH_15CYCLES

TSC_CTPH_16CYCLES

IS_TSC_CTPH

TSC Charge Transfer Pulse Low

TSC_CTPL_1CYCLE

[**TSC_CTPL_2CYCLES**](#)

[**TSC_CTPL_3CYCLES**](#)

[**TSC_CTPL_4CYCLES**](#)

[**TSC_CTPL_5CYCLES**](#)

[**TSC_CTPL_6CYCLES**](#)

[**TSC_CTPL_7CYCLES**](#)

[**TSC_CTPL_8CYCLES**](#)

[**TSC_CTPL_9CYCLES**](#)

[**TSC_CTPL_10CYCLES**](#)

[**TSC_CTPL_11CYCLES**](#)

[**TSC_CTPL_12CYCLES**](#)

[**TSC_CTPL_13CYCLES**](#)

[**TSC_CTPL_14CYCLES**](#)

[**TSC_CTPL_15CYCLES**](#)

[**TSC_CTPL_16CYCLES**](#)

[**IS_TSC_CTPL**](#)

TSC Exported Macros

[**_HAL_TSC_RESET_HAND**](#) **Description:**

- [**LE_STATE**](#) • Reset TSC handle state.

Parameters:

- [**_HANDLE_**](#): TSC handle.

Return value:

- None

[**_HAL_TSC_ENABLE**](#)

Description:

- Enable the TSC peripheral.

Parameters:

- [**_HANDLE_**](#): TSC handle

Return value:

- None

<u>__HAL_TSC_DISABLE</u>	Description: <ul style="list-style-type: none">• Disable the TSC peripheral. Parameters: <ul style="list-style-type: none">• <u>__HANDLE__</u>: TSC handle Return value: <ul style="list-style-type: none">• None
<u>__HAL_TSC_START_ACQ</u>	Description: <ul style="list-style-type: none">• Start acquisition. Parameters: <ul style="list-style-type: none">• <u>__HANDLE__</u>: TSC handle Return value: <ul style="list-style-type: none">• None
<u>__HAL_TSC_STOP_ACQ</u>	Description: <ul style="list-style-type: none">• Stop acquisition. Parameters: <ul style="list-style-type: none">• <u>__HANDLE__</u>: TSC handle Return value: <ul style="list-style-type: none">• None
<u>__HAL_TSC_SET_IODEF_OUTPPLOW</u>	Description: <ul style="list-style-type: none">• Set IO default mode to output push-pull low. Parameters: <ul style="list-style-type: none">• <u>__HANDLE__</u>: TSC handle Return value: <ul style="list-style-type: none">• None
<u>__HAL_TSC_SET_IODEF_INFLOAT</u>	Description: <ul style="list-style-type: none">• Set IO default mode to input floating. Parameters: <ul style="list-style-type: none">• <u>__HANDLE__</u>: TSC handle Return value: <ul style="list-style-type: none">• None
<u>__HAL_TSC_SET_SYNC_POL_FALL</u>	Description: <ul style="list-style-type: none">• Set synchronization polarity to falling edge. Parameters: <ul style="list-style-type: none">• <u>__HANDLE__</u>: TSC handle Return value: <ul style="list-style-type: none">• None

__HAL_TSC_SET_SYNC_P **Description:****OL_RISE_HIGH**

- Set synchronization polarity to rising edge and high level.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_ENABLE_IT**Description:**

- Enable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_DISABLE_IT**Description:**

- Disable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_GET_IT_SOURE**Description:****RCE**

- Check if the specified TSC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: TSC Handle
- __INTERRUPT__: TSC interrupt

Return value:

- SET: or RESET

__HAL_TSC_GET_FLAG**Description:**

- Get the selected TSC's flag status.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- SET: or RESET

__HAL_TSC_CLEAR_FLAG**Description:**

- Clear the TSC's pending flag.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- None

__HAL_TSC_ENABLE_HYS **Description:**

TERESIS

- Enable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_HY **Description:**

STERESIS

- Disable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_OPEN_ANAL **Description:**

OG_SWITCH

- Open analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_CLOSE_ANAL **Description:**

OG_SWITCH

- Close analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_CHA **Description:**

NNEL

- Enable a group of IOs in channel mode.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_CH **Description:**

ANNEL

- Disable a group of channel IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_SA **Description:**

MPLING

- Enable a group of IOs in sampling mode.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_SA **Description:**

MPLING

- Disable a group of sampling IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_GR **Description:**

OUP

- Enable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None

__HAL_TSC_DISABLE_GR **Description:**

OUP

- Disable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None

__HAL_TSC_GET_GROUP_ STATUS

- Gets acquisition group status.

Parameters:

- __HANDLE__: TSC Handle
- __GX_INDEX__: Group index

Return value:

- SET: or RESET

TSC Flags Definition**TSC_FLAG_EOA****TSC_FLAG_MCE*****TSC groups definition*****TSC_NB_OF_GROUPS****TSC_GROUP1****TSC_GROUP2****TSC_GROUP3****TSC_GROUP4****TSC_GROUP5****TSC_GROUP6****TSC_GROUP7****TSC_GROUP8****TSC_ALL_GROUPS****TSC_GROUP1_IDX****TSC_GROUP2_IDX****TSC_GROUP3_IDX****TSC_GROUP4_IDX****TSC_GROUP5_IDX****TSC_GROUP6_IDX****TSC_GROUP7_IDX**

TSC_GROUP8_IDX

IS_GROUP_INDEX

TSC_GROUP1_IO1

TSC_GROUP1_IO2

TSC_GROUP1_IO3

TSC_GROUP1_IO4

TSC_GROUP1_ALL_IOS

TSC_GROUP2_IO1

TSC_GROUP2_IO2

TSC_GROUP2_IO3

TSC_GROUP2_IO4

TSC_GROUP2_ALL_IOS

TSC_GROUP3_IO1

TSC_GROUP3_IO2

TSC_GROUP3_IO3

TSC_GROUP3_IO4

TSC_GROUP3_ALL_IOS

TSC_GROUP4_IO1

TSC_GROUP4_IO2

TSC_GROUP4_IO3

TSC_GROUP4_IO4

TSC_GROUP4_ALL_IOS

TSC_GROUP5_IO1

TSC_GROUP5_IO2

TSC_GROUP5_IO3

TSC_GROUP5_IO4

TSC_GROUP5_ALL_IOS

TSC_GROUP6_IO1

TSC_GROUP6_IO2

TSC_GROUP6_IO3

TSC_GROUP6_IO4

TSC_GROUP6_ALL_IOS

TSC_GROUP7_IO1

TSC_GROUP7_IO2

TSC_GROUP7_IO3

TSC_GROUP7_IO4

TSC_GROUP7_ALL_IOS

TSC_GROUP8_IO1

TSC_GROUP8_IO2

TSC_GROUP8_IO3

TSC_GROUP8_IO4

TSC_GROUP8_ALL_IOS

TSC_ALL_GROUPS_ALL_I
OS

TSC interrupts definition

TSC_IT_EOA

TSC_IT_MCE

IS_TSC_MCE_IT

TSC I/O default mode definition

TSC_IODEF_OUT_PP_LOW

TSC_IODEF_IN_FLOAT

IS_TSC_IODEF

TSC I/O mode definition

TSC_IOMODE_UNUSED**TSC_IOMODE_CHANNEL****TSC_IOMODE_SHIELD****TSC_IOMODE_SAMPLING****IS_TSC_IOMODE**

TSC Max Count Value definition

TSC_MCV_255**TSC_MCV_511****TSC_MCV_1023****TSC_MCV_2047****TSC_MCV_4095****TSC_MCV_8191****TSC_MCV_16383****IS_TSC_MCV**

TSC Pulse Generator prescaler definition

TSC_PG_PRESC_DIV1**TSC_PG_PRESC_DIV2****TSC_PG_PRESC_DIV4****TSC_PG_PRESC_DIV8****TSC_PG_PRESC_DIV16****TSC_PG_PRESC_DIV32****TSC_PG_PRESC_DIV64****TSC_PG_PRESC_DIV128****IS_TSC_PG_PRESC**

TSC Spread Spectrum**IS_TSC_SS****IS_TSC_SSD*****TSC Spread spectrum prescaler definition*****TSC_SS_PRESC_DIV1****TSC_SS_PRESC_DIV2****IS_TSC_SS_PRESC*****TSC Synchronization pin polarity*****TSC_SYNC_POLARITY_FA
LLING****TSC_SYNC_POLARITY_RI
SING****IS_TSC_SYNC_POL**

45 HAL UART Generic Driver

45.1 UART Firmware driver registers structures

45.1.1 **UART_InitTypeDef**

UART_InitTypeDef is defined in the `stm32f0xx_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**

Field Documentation

- **uint32_t UART_InitTypeDef::BaudRate**

This member configures the UART communication baud rate. The baud rate register is computed using the following formula:

- If oversampling is 16 or in LIN mode (LIN mode not available on F030xx devices), Baud Rate Register = $((\text{PCLKx}) / ((\text{uart} \rightarrow \text{Init.BaudRate}))$)
- If oversampling is 8, Baud Rate Register[15:4] = $((2 * \text{PCLKx}) / ((\text{uart} \rightarrow \text{Init.BaudRate})))$ [15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = $((2 * \text{PCLKx}) / ((\text{uart} \rightarrow \text{Init.BaudRate})))$ [3:0] $\gg 1$

- **uint32_t UART_InitTypeDef::WordLength**

Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [**UARTEWordLength**](#).

- **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**](#).

45.1.2 **UART_AdvFeatureInitTypeDef**

UART_AdvFeatureInitTypeDef is defined in the `stm32f0xx_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 [UARTEx_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](#).

45.1.3 `UART_HandleTypeDef`

`UART_HandleTypeDef` is defined in the `stm32f0xx_hal_uart.h`

Data Fields

- `USART_TypeDef * Instance`
- `UART_InitTypeDef Init`
- `UART_AdvFeatureInitTypeDef AdvancedInit`
- `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_UART_StateTypeDef gState`
- `__IO HAL_UART_StateTypeDef RxState`
- `__IO uint32_t ErrorCode`

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
- `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
- `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

45.2 UART Firmware driver API description

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

45.2.1 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 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, 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 Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
 8. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

Note:

These APIs(HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_UART_Transmit()
- Receive an amount of data in blocking mode using HAL_UART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_UART_Transmit_IT()
- At transmission end of half transfer HAL_UART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxHalfCpltCallback
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_UART_Receive_IT()
- At reception end of half transfer HAL_UART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxHalfCpltCallback

- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_UART_Transmit_DMA()
- At transmission end of half transfer HAL_UART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxHalfCpltCallback
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_UART_Receive_DMA()
- At reception end of half transfer HAL_UART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxHalfCpltCallback
- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback
- Pause the DMA Transfer using HAL_UART_DMAPause()
- Resume the DMA Transfer using HAL_UART_DMAResume()
- Stop the DMA Transfer using HAL_UART_DMAStop()

UART HAL driver macros list

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

- __HAL_UART_ENABLE: Enable the UART peripheral
- __HAL_UART_DISABLE: Disable the UART peripheral
- __HAL_UART_GET_FLAG : Check whether the specified UART flag is set or not
- __HAL_UART_CLEAR_FLAG : Clear the specified UART pending flag
- __HAL_UART_ENABLE_IT: Enable the specified UART interrupt
- __HAL_UART_DISABLE_IT: Disable the specified UART interrupt

Note:

You can refer to the *UART HAL driver header file* for more useful macros

45.2.2 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
 - 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() and HAL_MultiProcessor_Init() API follow respectively the UART asynchronous, UART Half duplex and 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_MultiProcessor_Init**](#)
- [**HAL_UART_DelInit**](#)
- [**HAL_UART_MspInit**](#)
- [**HAL_UART_MspDelInit**](#)

45.2.3 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**](#)

45.2.4 Peripheral Control functions

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

- [**HAL_MultiProcessor_EnableMuteMode\(\)**](#) API enables mute mode
- [**HAL_MultiProcessor_DisableMuteMode\(\)**](#) API disables mute mode
- [**HAL_MultiProcessor_EnterMuteMode\(\)**](#) API enters mute mode
- [**HAL_HalfDuplex_EnableTransmitter\(\)**](#) API disables receiver and enables transmitter
- [**HAL_HalfDuplex_EnableReceiver\(\)**](#) API disables transmitter and enables receiver

This section contains the following APIs:

- [**HAL_MultiProcessor_EnableMuteMode**](#)
- [**HAL_MultiProcessor_DisableMuteMode**](#)
- [**HAL_MultiProcessor_EnterMuteMode**](#)
- [**HAL_HalfDuplex_EnableTransmitter**](#)

- [*HAL_HalfDuplex_EnableReceiver*](#)

45.2.5 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*](#)

45.2.6 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_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_DelInit

Function name `HAL_StatusTypeDef HAL_UART_DelInit (UART_HandleTypeDef * huart)`

Function description Delinitialize 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_MspDelInit

Function name `void HAL_UART_MspDelInit (UART_HandleTypeDef * huart)`

Function description Delinitialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_Transmit

Function name `HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, 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.
- **Size:** Amount of data 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), 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.
- Size:** amount of data 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), 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, 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.
- Size:** amount of data 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), 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	<ul style="list-style-type: none">huart: UART handle.pData: pointer to data buffer.Size: amount of data to be received.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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, uint8_t * pData, uint16_t Size)
Function description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none">huart: UART handle.pData: pointer to data buffer.Size: amount of data to be sent.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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	<ul style="list-style-type: none">huart: UART handle.pData: pointer to data buffer.Size: amount of data to be received.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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_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_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_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_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_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	<ul style="list-style-type: none">• None:
HAL_UART_AbortReceiveCpltCallback	
Function name	void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)
Function description	UART Abort Receive Complete callback.
Parameters	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• None:
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	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• HAL: status.
HAL_UART_GetState	
Function name	HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)
Function description	Return the UART handle state.
Parameters	<ul style="list-style-type: none">• huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none">• HAL: state
HAL_UART_GetError	
Function name	uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)
Function description	Return the UART handle error code.
Parameters	<ul style="list-style-type: none">• huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none">• UART: Error Code
UART_AdvFeatureConfig	
Function name	void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)
Function description	Configure the UART peripheral advanced features.
Parameters	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• None:
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_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_Transmit_IT

Function name **HAL_StatusTypeDef** **UART_Transmit_IT** (**UART_HandleTypeDef** * **huart**)

Function description Send an amount of data in interrupt mode.

Parameters • **huart:** UART handle.

Return values • **HAL:** status

Notes • Function is called under interruption only, once interruptions have been enabled by **HAL_UART_Transmit_IT()**.

UART_EndTransmit_IT

Function name **HAL_StatusTypeDef** **UART_EndTransmit_IT** (**UART_HandleTypeDef** * **huart**)

Function description Wrap up transmission in non-blocking mode.

Parameters • **huart:** pointer to a **UART_HandleTypeDef** structure that contains the configuration information for the specified UART module.

Return values • **HAL:** status

UART_Receive_IT

Function name **HAL_StatusTypeDef** **UART_Receive_IT** (**UART_HandleTypeDef** * **huart**)

Function description Receive an amount of data in interrupt mode.

Parameters • **huart:** UART handle.

Return values • **HAL:** status

Notes • Function is called under interruption only, once interruptions have been enabled by **HAL_UART_Receive_IT()**

UART_WaitOnFlagUntilTimeout

Function name	HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)
Function description	Handle UART Communication Timeout.
Parameters	<ul style="list-style-type: none">huart: UART handle.Flag: Specifies the UART flag to checkStatus: Flag status (SET or RESET)Tickstart: Tick start valueTimeout: Timeout duration
Return values	<ul style="list-style-type: none">HAL: status

45.3 UART Firmware driver defines

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

45.3.1 UART

UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_I No advanced feature initialization
NIT

UART_ADVFEATURE_TXIN TX pin active level inversion
VERT_INIT

UART_ADVFEATURE_RXIN RX pin active level inversion
VERT_INIT

UART_ADVFEATURE_DAT Binary data inversion
AINVERT_INIT

UART_ADVFEATURE_SWA TX/RX pins swap
P_INIT

UART_ADVFEATURE_RXO RX overrun disable
VERRUNDISABLE_INIT

UART_ADVFEATURE_DMA DMA disable on Reception Error
DISABLEONERROR_INIT

UART_ADVFEATURE_AUT Auto Baud rate detection initialization
OBAUDRATE_INIT

UART_ADVFEATURE_MSB Most significant bit sent/received first
FIRST_INIT

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUT RX Auto Baud rate detection enable
OBAUDRATE_DISABLE

UART_ADVFEATURE_AUT RX Auto Baud rate detection disable
OBAUDRATE_ENABLE

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRE UART Driver Enable assertion time LSB position in CR1 register
SS_LSB_POS

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRE UART Driver Enable de-assertion time LSB position in CR1 register
SS_LSB_POS

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LS UART address-matching LSB position in CR2 register
B_POS

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DAT Binary data inversion disable
AINV_DISABLE

UART_ADVFEATURE_DAT Binary data inversion enable
AINV_ENABLE

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA DMA enable on Reception Error
_ENABLEONRXERROR

UART_ADVFEATURE_DMA DMA disable on Reception Error
_DISABLEONRXERROR

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_HIG Driver enable signal is active high
H

UART_DE_POLARITY_LO Driver enable signal is active low
W

UART Error

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_BUSY Busy Error

UART Exported Macros

_HAL_UART_RESET_HANDLE **Description:**

DLE_STATE

- Reset UART handle states.

Parameters:

- **_HANDLE_**: UART handle.

Return value:

- None

_HAL_UART_CLEAR_FLAG **Description:**

G

- 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_TCF** Transmission Complete Clear Flag
 - **UART_CLEAR_LBDF** LIN Break Detection Clear Flag (not available on all devices)
 - **UART_CLEAR_CTSF** CTS Interrupt Clear Flag
 - **UART_CLEAR_RTOF** Receiver Time Out Clear Flag
 - **UART_CLEAR_EOBF** End Of Block Clear Flag (not available on all devices)
 - **UART_CLEAR_CMF** Character Match Clear Flag
 - **UART_CLEAR_WUF** Wake Up from stop mode Clear Flag (not available on all devices)

Return value:

- None

__HAL_UART_CLEAR_PEF **Description:****LAG**

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEF **Description:****LAG**

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEF **Description:****LAG**

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OR **Description:****EFLAG**

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_IDL **Description:****EFLAG**

- Clear the UART IDLE pending 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_RXACK Receive enable acknowledge flag
 - UART_FLAG_TEACK Transmit enable acknowledge flag
 - UART_FLAG_WUF Wake up from stop mode flag (not available on F030xx devices)
 - UART_FLAG_RXWU Receiver wake up flag (not available on F030xx devices)
 - 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_EOBF End of block flag (not available on F030xx devices)
 - UART_FLAG_RTOF Receiver timeout flag
 - UART_FLAG_CTS CTS Change flag
 - UART_FLAG_LBDF LIN Break detection flag (not available on F030xx devices)
 - UART_FLAG_TXE Transmit data register empty flag
 - UART_FLAG_TC Transmission Complete flag
 - UART_FLAG_RXNE Receive data register not empty 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_WUF Wakeup from stop mode interrupt (not available on F030xx devices)
 - UART_IT_CM Character match interrupt
 - UART_IT_CTS CTS change interrupt
 - UART_IT_LBD LIN Break detection interrupt (not available on F030xx devices)
 - UART_IT_TXE Transmit Data Register empty interrupt
 - UART_IT_TC Transmission complete interrupt
 - UART_IT_RXNE Receive Data register not empty 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_WUF Wakeup from stop mode interrupt (not available on F030xx devices)
 - UART_IT_CM Character match interrupt
 - UART_IT_CTS CTS change interrupt
 - UART_IT_LBD LIN Break detection interrupt (not available on F030xx devices)
 - UART_IT_TXE Transmit Data Register empty interrupt
 - UART_IT_TC Transmission complete interrupt
 - UART_IT_RXNE Receive Data register not empty 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.
- __IT__: specifies the UART interrupt to check. This parameter can be one of the following values:
 - UART_IT_WUF Wakeup from stop mode interrupt (not available on F030xx devices)
 - UART_IT_CM Character match interrupt
 - UART_IT_CTS CTS change interrupt
 - UART_IT_LBD LIN Break detection interrupt (not available on F030xx devices)
 - UART_IT_TXE Transmit Data Register empty interrupt
 - UART_IT_TC Transmission complete interrupt
 - UART_IT_RXNE Receive Data register not empty interrupt
 - UART_IT_IDLE Idle line detection interrupt
 - UART_IT_ORE Overrun Error interrupt
 - UART_IT_NE Noise Error interrupt
 - UART_IT_FE Framing Error interrupt
 - UART_IT_PE Parity Error interrupt

Return value:

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

[__HAL_UART_GET_IT_SO](#) **Description:**

URCE

- Check whether the specified UART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
 - UART_IT_WUF Wakeup from stop mode interrupt (not available on F030xx devices)
 - UART_IT_CM Character match interrupt
 - UART_IT_CTS CTS change interrupt
 - UART_IT_LBD LIN Break detection interrupt (not available on F030xx devices)
 - UART_IT_TXE Transmit Data Register empty interrupt
 - UART_IT_TC Transmission complete interrupt
 - UART_IT_RXNE Receive Data register not empty interrupt
 - UART_IT_IDLE Idle line detection interrupt
 - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)
 - UART_IT_PE Parity Error interrupt

Return value:

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

[__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_TCF Transmission Complete Clear Flag
 - UART_CLEAR_LBDF LIN Break Detection Clear Flag (not available on F030xx devices)
 - UART_CLEAR_CTSF CTS Interrupt Clear Flag
 - UART_CLEAR_RTOF Receiver Time Out Clear Flag
 - UART_CLEAR_EOBF End Of Block Clear Flag
 - UART_CLEAR_CMF Character Match Clear Flag
 - UART_CLEAR_WUF Wake Up from stop mode Clear Flag (not available on F030xx devices)

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 (not available on F030xx devices)

Return value:

- None

__HAL_UART_ONE_BIT_S **Description:****AMPLE_ENABLE**

- Enable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ONE_BIT_S **Description:****AMPLE_DISABLE**

- 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 Description:**L_CTS_ENABLE**

- 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 Description:**L_CTS_DISABLE**

- 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 Description:**L_RTS_ENABLE**

- 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 Description:**L_RTS_DISABLE**

- 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__)).

UARTEx Status Flags

UART_FLAG_RXACK	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_EOBF	UART end of block flag
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 (not available on F030xx devices)
UART_FLAG_TXE	UART transmit data register empty
UART_FLAG_TC	UART transmission complete

UART_FLAG_RXNE	UART read data register 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
--------------	-------------------------------

UARTEx Interrupts Definition

UART_IT_PE	UART parity error interruption
------------	--------------------------------

UART_IT_TXE	UART transmit data register empty interruption
-------------	--

UART_IT_TC	UART transmission complete interruption
------------	---

UART_IT_RXNE	UART read data register 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

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

UARTEx Interruption Clear Flags

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_TCF Transmission Complete Clear Flag

UART_CLEAR_LBDF LIN Break Detection Clear Flag (not available on F030xx devices)

UART_CLEAR_CTSF CTS Interrupt Clear Flag

UART_CLEAR_RTOF Receiver Time Out Clear Flag

UART_CLEAR_EOBF End Of Block Clear Flag

UART_CLEAR_CMF Character Match Clear Flag

UART_CLEAR_WUF Wake Up from stop mode Clear Flag

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_MSB_FIRST_DISABLE Most significant bit sent/received first disable

UART_ADVFEATURE_MSB FIRST_ENABLE Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUT EMODE_DISABLE UART mute mode disable

UART_ADVFEATURE_MUT EMODE_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_OVE RRUN_ENABLE RX overrun enable

UART_ADVFEATURE_OVE RRUN_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 T_DISABLE UART receiver timeout disable

UART_RECEIVER_TIMEOUT T_ENABLE UART receiver timeout enable

UARTEx Request Parameters

UART_AUTOBAUD_REQEST Auto-Baud Rate Request

UART_SENDBREAK_REQEST Send Break Request

UART_MUTE_MODE_REQ Mute Mode Request
UEST

UART_RXDATA_FLUSH_R Receive Data flush Request
EQUEST

UART_TXDATA_FLUSH_RE Transmit data flush Request
QUEST

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXIN RX pin active level inversion disable
V_DISABLE

UART_ADVFEATURE_RXIN RX pin active level inversion enable
V_ENABLE

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWA TX/RX pins swap disable
P_DISABLE

UART_ADVFEATURE_SWA TX/RX pins swap enable
P_ENABLE

UART State

UART_STATE_DISABLE UART disabled

UART_STATE_ENABLE UART enabled

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

UARTEx Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STO UART stop mode disable
PMODE_DISABLE

UART_ADVFEATURE_STO UART stop mode enable
PMODE_ENABLE

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VAL UART polling-based communications time-out value
UE

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXIN TX pin active level inversion disable
V_DISABLE

UART_ADVFEATURE_TXIN TX pin active level inversion enable
V_ENABLE

UART WakeUp Address Length

UART_ADDRESS_DETECT 4-bit long wake-up address
_4B

UART_ADDRESS_DETECT 7-bit long wake-up address
_7B

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADD UART wake-up on address
RESS

UART_WAKEUP_ON_STAR UART wake-up on start bit
TBIT

UART_WAKEUP_ON_REA UART wake-up on receive data register not empty
DDATA_NONEMPTY

UART WakeUp Methods

UART_WAKEUPMETHOD_I UART wake-up on idle line
DLELINE

UART_WAKEUPMETHOD_ UART wake-up on address mark
ADDRESSMARK

46 HAL UART Extension Driver

46.1 UARTEEx Firmware driver registers structures

46.1.1 **UART_WakeUpTypeDef**

UART_WakeUpTypeDef is defined in the `stm32f0xx_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 [`UART_WakeUp_Address_Length`](#).

- `uint8_t UART_WakeUpTypeDef::Address`

UART/USART node address (7-bit long max).

46.2 UARTEEx Firmware driver API description

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

46.2.1 **UART peripheral extended features**

46.2.2 **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 (Fixed to 8-bits only for LIN mode)
 - Stop Bit
 - Parity
 - 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_LIN_Init()` and `HAL_RS485Ex_Init()` APIs follows respectively the LIN and the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [*HAL_RS485Ex_Init*](#)
- [*HAL_LIN_Init*](#)

46.2.3 IO operation function

This subsection provides function to handle Wake up interrupt call-back.

1. Callback provided in No_Blocking mode:
 - [*HAL_UARTEx_WakeupCallback\(\)*](#)

This section contains the following APIs:

- [*HAL_UARTEx_WakeupCallback*](#)

46.2.4 Peripheral Control functions

This subsection provides extended functions allowing to control the UART.

- [*HAL_UARTEx_StopModeWakeUpSourceConfig\(\)*](#) API sets Wakeup from Stop mode interrupt flag selection
- [*HAL_UARTEx_EnableStopMode\(\)*](#) API allows the UART to wake up the MCU from Stop mode as long as UART clock is HSI or LSE
- [*HAL_UARTEx_DisableStopMode\(\)*](#) API disables the above feature
- [*HAL_MultiProcessorEx_AddressLength_Set\(\)*](#) API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- [*HAL_LIN_SendBreak\(\)*](#) API transmits the break characters

This section contains the following APIs:

- [*HAL_UARTEx_StopModeWakeUpSourceConfig*](#)
- [*HAL_UARTEx_EnableStopMode*](#)
- [*HAL_UARTEx_DisableStopMode*](#)
- [*HAL_MultiProcessorEx_AddressLength_Set*](#)
- [*HAL_LIN_SendBreak*](#)

46.2.5 Detailed description of functions

[*HAL_RS485Ex_Init*](#)

Function name	<code>HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)</code>
Function description	Initialize the RS485 Driver enable feature according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle.
Parameters	<ul style="list-style-type: none">• huart: UART handle.• Polarity: select the driver enable polarity. This parameter can be one of the following values:<ul style="list-style-type: none">– <code>UART_DE_POLARITY_HIGH</code> DE signal is active high– <code>UART_DE_POLARITY_LOW</code> 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">huart: UART handle.BreakDetectLength: specifies the LIN break detection length. This parameter can be one of the following values:<ul style="list-style-type: none">– UART_LINBREAKDETECTLENGTH_10B 10-bit break detection– UART_LINBREAKDETECTLENGTH_11B 11-bit break detection
Return values	<ul style="list-style-type: none">HAL: status

HAL_UARTEx_WakeupCallback

Function name	void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)
Function description	UART wakeup from Stop mode callback.
Parameters	<ul style="list-style-type: none">huart: UART handle.
Return values	<ul style="list-style-type: none">None:

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	<ul style="list-style-type: none">huart: UART handle.AddressLength: this parameter can be one of the following values:<ul style="list-style-type: none">– UART_ADDRESS_DETECT_4B 4-bit long address– UART_ADDRESS_DETECT_7B 6-, 7- or 8-bit long address
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">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_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 bit status. This parameter can be one of the following values:
 - UART_WAKEUP_ON_ADDRESS
 - UART_WAKEUP_ON_STARTBIT
 - UART_WAKEUP_ON_RXNE

- Return values**
- **HAL:** status

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_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

46.3 UARTEx Firmware driver defines

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

46.3.1 UARTEx

UARTEx

UARTEx Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUT Auto Baud rate detection on start bit
OBAUDRATE_ONSTARTBT

UART_ADVFEATURE_AUT Auto Baud rate detection on falling edge
OBAUDRATE_ONFALLING
EDGE

UART_ADVFEATURE_AUT Auto Baud rate detection on 0x7F frame detection
OBAUDRATE_ON0X7FFRA
ME

UART_ADVFEATURE_AUT Auto Baud rate detection on 0x55 frame detection
OBAUDRATE_ON0X55FRA
ME

UARTEx Exported Macros

_HAL_UART_FLUSH_DRR Description:
REGISTER

- Flush the UART Data registers.

Parameters:

- _HANDLE_**: specifies the UART Handle.

Return value:

- None

UARTEx Local Interconnection Network mode

UART_LIN_DISABLE Local Interconnect Network disable

UART_LIN_ENABLE Local Interconnect Network enable

UARTEx LIN Break Detection

UART_LINBREAKDETECTL LIN 10-bit break detection length
ENGTH_10B

UART_LINBREAKDETECTL LIN 11-bit break detection length
ENGTH_11B

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

47 HAL USART Generic Driver

47.1 USART Firmware driver registers structures

47.1.1 USART_InitTypeDef

`USART_InitTypeDef` is defined in the `stm32f0xx_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`

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 = ((PCLKx) / ((hart->Init.BaudRate))).

- `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`.

47.1.2 USART_HandleTypeDef

`USART_HandleTypeDef` is defined in the `stm32f0xx_hal_usart.h`

Data Fields

- `USART_TypeDef * Instance`
- `USART_InitTypeDef Init`
- `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_USART_StateTypeDef State`
- `__IO uint32_t ErrorCode`

Field Documentation

- **`USART_TypeDef* USART_HandleTypeDef::Instance`**
USART registers base address
- **`USART_InitTypeDef USART_HandleTypeDef::Init`**
USART communication parameters
- **`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
- **`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

47.2 USART Firmware driver API description

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

47.2.1 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, Hardware flow control and Mode (Receiver/Transmitter) in the usart 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.
 5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_USART_Transmit()
- Receive an amount of data in blocking mode using HAL_USART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_USART_Transmit_IT()
- At transmission end of half transfer HAL_USART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxHalfCpltCallback
- At transmission end of transfer HAL_USART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_USART_Receive_IT()
- At reception end of half transfer HAL_USART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxHalfCpltCallback
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback

DMA mode IO operation

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

- At transmission end of transfer HAL_USART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_USART_Receive_DMA()
- At reception end of half transfer HAL_USART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxHalfCpltCallback
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback
- Pause the DMA Transfer using HAL_USART_DMAPause()
- Resume the DMA Transfer using HAL_USART_DMAResume()
- Stop the DMA Transfer using HAL_USART_DMAStop()

USART HAL driver macros list

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

- `_HAL_USART_ENABLE`: Enable the USART peripheral
- `_HAL_USART_DISABLE`: Disable the USART peripheral
- `_HAL_USART_GET_FLAG` : Check whether the specified USART flag is set or not
- `_HAL_USART_CLEAR_FLAG` : Clear the specified USART pending flag
- `_HAL_USART_ENABLE_IT`: Enable the specified USART interrupt
- `_HAL_USART_DISABLE_IT`: Disable the specified USART interrupt

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

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_HandleTypeDef`.

47.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity
 - USART polarity
 - USART phase
 - USART LastBit
 - Receiver/transmitter modes

The `HAL_USART_Init()` function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- `HAL_USART_Init`
- `HAL_USART_DelInit`
- `HAL_USART_MspInit`
- `HAL_USART_MspDelInit`

47.2.3 IO operation functions

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

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA. These APIs 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 APIs are :
 - HAL_USART_Transmit() in simplex mode
 - HAL_USART_Receive() in full duplex receive only
 - HAL_USART_TransmitReceive() in full duplex mode
3. No-Blocking mode APIs 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 APIs 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_TxRxCPtCallback*](#)
- [*HAL_USART_ErrorCallback*](#)
- [*HAL_USART_AbortCpltCallback*](#)

47.2.4 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*](#)

47.2.5 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 * huart)`

Function description Initialize the USART MSP.

Parameters • **husart:** USART handle.

Return values • **None:**

HAL_USART_MspDeInit

Function name `void HAL_USART_MspDeInit (USART_HandleTypeDef * huart)`

Function description Deinitialize the USART MSP.

Parameters • **husart:** USART handle.

Return values • **None:**

HAL_USART_Transmit

Function name `HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * huart, 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.
• **Size:** Amount of data 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), 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 * huart, 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.
• **Size:** Amount of data to be received.
• **Timeout:** Timeout duration.

Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• To receive synchronous data, dummy data are simultaneously transmitted.• 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 pRxData.

HAL_USART_TransmitReceive

Function name	HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, 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	<ul style="list-style-type: none">• husart: USART handle.• pTxData: pointer to TX data buffer.• pRxData: pointer to RX data buffer.• Size: amount of data to be sent (same amount to be received).• Timeout: Timeout duration.
------------	---

Return values	<ul style="list-style-type: none">• HAL: status
---------------	--

Notes	<ul style="list-style-type: none">• When UART 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, uint8_t * pTxData, uint16_t Size)
---------------	---

Function description Send an amount of data in interrupt mode.

Parameters	<ul style="list-style-type: none">• husart: USART handle.• pTxData: pointer to data buffer.• Size: amount of data to be sent.
------------	--

Return values	<ul style="list-style-type: none">• HAL: status
---------------	--

Notes	<ul style="list-style-type: none">• 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 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	<ul style="list-style-type: none">husart: USART handle.pRxData: pointer to data buffer.Size: amount of data to be received.
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">To receive synchronous data, dummy data are simultaneously transmitted.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 pRxData.

HAL_USART_TransmitReceive_IT

Function name	HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none">husart: USART handle.pTxData: pointer to TX data buffer.pRxData: pointer to RX data buffer.Size: amount of data to be sent (same amount to be received).
Return values	<ul style="list-style-type: none">HAL: status
Notes	<ul style="list-style-type: none">When UART 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, uint8_t * pTxData, uint16_t Size)
Function description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none">husart: USART handle.pTxData: pointer to data buffer.Size: amount of data to be sent.
Return values	<ul style="list-style-type: none">HAL: status

Notes

- 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 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.
- Size:** amount of data 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), 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.
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.

HAL_USART_TransmitReceive_DMA

Function name `HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, 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.
- pRxData:** pointer to RX data buffer.
- Size:** amount of data to be received/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), 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 * huart)`

Function description Handle USART interrupt request.

Parameters • **husart:** USART handle.

Return values • **None:**

HAL_USART_TxCpltCallback

Function name `void HAL_USART_TxCpltCallback (USART_HandleTypeDef * huart)`

Function description Tx Transfer completed callback.

Parameters • **husart:** USART handle.

Return values • **None:**

HAL_USART_RxCpltCallback

Function name `void HAL_USART_RxCpltCallback (USART_HandleTypeDef * huart)`

Function description Rx Transfer completed callback.

Parameters • **husart:** USART handle.

Return values • **None:**

HAL_USART_TxHalfCpltCallback

Function name `void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * huart)`

Function description Tx Half Transfer completed callback.

Parameters • **husart:** USART handle.

Return values • **None:**

HAL_USART_RxHalfCpltCallback

Function name `void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * huart)`

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 (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 (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

47.3 USART Firmware driver defines

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

47.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLE USART clock disable

USART_CLOCK_ENABLE USART clock enable

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 USART Clock signal is steady Low

USART_POLARITY_HIGH USART Clock signal is steady High

USART Error

HAL_USART_ERROR_NN No error

E

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

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_RXACK Receive enable acknowledge flag
 - USART_FLAG_TEACK Transmit enable acknowledge flag
 - USART_FLAG_BUSY Busy flag
 - USART_FLAG_CTS CTS Change flag
 - USART_FLAG_TXE Transmit data register empty flag
 - USART_FLAG_TC Transmission Complete flag
 - USART_FLAG_RXNE Receive data register not empty 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_FL Description:

AG

- 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
 - USART_CLEAR_FEF
 - USART_CLEAR_NEF
 - USART_CLEAR_OREF
 - USART_CLEAR_IDLEF
 - USART_CLEAR_TCF
 - USART_CLEAR_CTSF

Return value:

- None

__HAL_USART_CLEAR_PE Description:

FLAG

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FE **Description:**
FLAG

- Clear the USART FE pending flag.

Parameters:

- **__HANDLE__**: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NE **Description:**
FLAG

- Clear the USART NE pending flag.

Parameters:

- **__HANDLE__**: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_O **Description:**
REFLAG

- Clear the USART ORE pending flag.

Parameters:

- **__HANDLE__**: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_ID **Description:**
LEFLAG

- Clear the USART IDLE pending flag.

Parameters:

- **__HANDLE__**: specifies the USART Handle.

Return value:

- None

T __HAL_USART_ENABLE_I **Description:**
T

- 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_TXE Transmit Data Register empty interrupt
- USART_IT_TC Transmission complete interrupt
- USART_IT_RXNE Receive Data register 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:

T

- 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_TXE Transmit Data Register empty interrupt
 - USART_IT_TC Transmission complete interrupt
 - USART_IT_RXNE Receive Data register 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.
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - USART_IT_TXE Transmit Data Register empty interrupt
 - USART_IT_TC Transmission complete interrupt
 - USART_IT_RXNE Receive Data register 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 __IT__ (TRUE or FALSE).

__HAL_USART_GET_IT_SOURCE

Description:

- Check whether the specified USART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - USART_IT_TXE Transmit Data Register empty interrupt
 - USART_IT_TC Transmission complete interrupt
 - USART_IT_RXNE Receive Data register 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 __IT__ (TRUE or FALSE).

__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_TCF Transmission Complete Clear Flag
 - USART_CLEAR_CTSF CTS Interrupt Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ **Description:**

Q

- 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:**

Q

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE **Description:**

Q

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE **Description:**

Q

- 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_RXACK USART receive enable acknowledge flag

USART_FLAG_TEACK USART transmit enable acknowledge flag

USART_FLAG_BUSY USART busy flag

USART_FLAG_CTS USART clear to send flag

USART_FLAG_CTSIF USART clear to send interrupt flag

USART_FLAG_TXE USART transmit data register empty

USART_FLAG_TC USART transmission complete

USART_FLAG_RXNE USART read data register 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 Interrupts Definition

USART_IT_PE USART parity error interruption

USART_IT_TXE USART transmit data register empty interruption

USART_IT_TC USART transmission complete interruption

USART_IT_RXNE USART read data register 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 Interruption Clear Flags

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_TCF	Transmission Complete Clear Flag
USART_CLEAR_CTSF	CTS Interrupt Clear Flag

USART Last Bit

USART_LASTBIT_DISABLE	USART frame last data bit clock pulse not output to SCLK pin E
------------------------------	--

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

USARTE Request Parameters

USART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
-----------------------------------	----------------------------

USART_TXDATA_FLUSH_R Transmit data flush Request
EQUEST

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

48 HAL USART Extension Driver

48.1 USARTEx Firmware driver defines

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

48.1.1 USARTEx

USARTEx

USARTEx Exported Macros

_HAL_USART_FLUSH_DR **Description:**

REGISTER

- Flush the USART Data registers.

Parameters:

- _HANDLE_: specifies the USART Handle.

Return value:

- None

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

49 HAL WWDG Generic Driver

49.1 WWDG Firmware driver registers structures

49.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the `stm32f0xx_hal_wwdg.h`

Data Fields

- `uint32_t Prescaler`
- `uint32_t Window`
- `uint32_t Counter`
- `uint32_t EWIMode`

Field Documentation

- `uint32_t WWDG_InitTypeDef::Prescaler`

Specifies the prescaler value of the WWDG. This parameter can be a value of [`WWDG_Prescaler`](#)

- `uint32_t WWDG_InitTypeDef::Window`

Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number
Min_Data = 0x40 and Max_Data = 0x7F

- `uint32_t WWDG_InitTypeDef::Counter`

Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data
= 0x40 and Max_Data = 0x7F

- `uint32_t WWDG_InitTypeDef::EWIMode`

Specifies if WWDG Early Wakeup Interrupt is enable or not. This parameter can be a value of
[`WWDG_EWI_Mode`](#)

49.1.2 WWDG_HandleTypeDefDef

WWDG_HandleTypeDefDef is defined in the `stm32f0xx_hal_wwdg.h`

Data Fields

- `WWDG_TypeDef * Instance`
- `WWDG_InitTypeDef Init`

Field Documentation

- `WWDG_TypeDef* WWDG_HandleTypeDefDef::Instance`

Register base address

- `WWDG_InitTypeDef WWDG_HandleTypeDefDef::Init`

WWDG required parameters

49.2 WWDG Firmware driver API description

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

49.2.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (T[6:0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDGRST flag in RCC_CSR register informs when a WWDG reset has occurred (check available with `__HAL_RCC_GET_FLAG(RCC_FLAG_WWDGRST)`).
- The WWDG downcounter input clock is derived from the APB clock divided by a programmable prescaler.

- WWDG downcounter clock (Hz) = PCLK / (4096 * Prescaler)
- WWDG timeout (ms) = (1000 * (T[5;0] + 1)) / (WWDG downcounter clock) where T[5;0] are the lowest 6 bits of downcounter.
- WWDG Counter refresh is allowed between the following limits :
 - min time (ms) = (1000 * (T[5;0] - Window)) / (WWDG downcounter clock)
 - max time (ms) = (1000 * (T[5;0] - 0x40)) / (WWDG downcounter clock)
- Min-max timeout value @48 MHz(PCLK): ~85,3us / ~5,46 ms
- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches the value 0x40, an EWI interrupt is generated and the corresponding interrupt service routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device. In some applications, the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case, the corresponding interrupt service routine (ISR) should reload the WWDG counter to avoid the WWDG reset, then trigger the required actions.
Note:When the EWI interrupt cannot be served, e.g. due to a system lock in a higher priority task, the WWDG reset will eventually be generated.
- Debug mode : When the microcontroller enters debug mode (core halted), the WWDG counter either continues to work normally or stops, depending on DBG_WWDG_STOP configuration bit in DBG module, accessible through `_HAL_DBGMCU_FREEZE_WWDG()` and `_HAL_DBGMCU_UNFREEZE_WWDG()` macros

49.2.2

How to use this driver

- Enable WWDG APB1 clock using `_HAL_RCC_WWDG_CLK_ENABLE()`.
- Set the WWDG prescaler, refresh window, counter value and Early Wakeup Interrupt mode using using `HAL_WWDG_Init()` function. This enables WWDG peripheral and the downcounter starts downcounting from given counter value. Init function can be called again to modify all watchdog parameters, however if EWI mode has been set once, it can't be clear until next reset.
- The application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset using `HAL_WWDG_Refresh()` function. This operation must occur only when the counter is lower than the window value already programmed.
- if Early Wakeup Interrupt mode is enable an interrupt is generated when the counter reaches 0x40. User can add his own code in weak function `HAL_WWDG_EarlyWakeUpCallback()`.

WWDG HAL driver macros list

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

- `_HAL_WWDG_GET_IT_SOURCE`: Check the selected WWDG's interrupt source.
- `_HAL_WWDG_GET_FLAG`: Get the selected WWDG's flag status.
- `_HAL_WWDG_CLEAR_FLAG`: Clear the WWDG's pending flags.

49.2.3

Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and start the WWDG according to the specified parameters in the `WWDG_InitTypeDef` of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:

- **`HAL_WWDG_Init`**
- **`HAL_WWDG_MspInit`**

49.2.4

IO operation functions

This section provides functions allowing to:

- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- [*HAL_WWDG_Refresh*](#)
- [*HAL_WWDG_IRQHandler*](#)
- [*HAL_WWDG_EarlyWakeupCallback*](#)

49.2.5 Detailed description of functions

HAL_WWDG_Init

Function name [**HAL_StatusTypeDef HAL_WWDG_Init \(WWDG_HandleTypeDef * hwdg\)**](#)

Function description Initialize the WWDG according to the specified.

Parameters • **hwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values • **HAL:** status

HAL_WWDG_MspInit

Function name [**void HAL_WWDG_MspInit \(WWDG_HandleTypeDef * hwdg\)**](#)

Function description Initialize the WWDG MSP.

Parameters • **hwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values • **None:**

Notes • When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_Refresh

Function name [**HAL_StatusTypeDef HAL_WWDG_Refresh \(WWDG_HandleTypeDef * hwdg\)**](#)

Function description Refresh the WWDG.

Parameters • **hwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values • **HAL:** status

HAL_WWDG_IRQHandler

Function name [**void HAL_WWDG_IRQHandler \(WWDG_HandleTypeDef * hwdg\)**](#)

Function description Handle WWDG interrupt request.

Parameters • **hwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values • **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name `void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)`

Function description WWDG Early Wakeup callback.

Parameters

- hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- None:**

49.3 WWDG Firmware driver defines

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

49.3.1 WWDG

WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE EWI Disable

WWDG_EWI_ENABLE EWI Enable

WWDG Exported Macros

_HAL_WWDG_ENABLE **Description:**

- Enable the WWDG peripheral.

Parameters:

- _HANDLE_**: WWDG handle

Return value:

- None

_HAL_WWDG_ENABLE_I **Description:**

T

- Enable the WWDG early wakeup interrupt.

Parameters:

- _HANDLE_**: WWDG handle
- _INTERRUPT_**: specifies the interrupt to enable. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

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

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

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

__HAL_WWDG_CLEAR_FL

AG

Description:

- Clear the WWDG's pending flags.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

__HAL_WWDG_GET_IT_S

OURCE

Description:

- Check whether the specified WWDG interrupt source is enabled or not.

Parameters:

- __HANDLE__: WWDG Handle.
- __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1 WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2 WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4 WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8 WWDG counter clock = (PCLK1/4096)/8

50 LL ADC Generic Driver

50.1 ADC Firmware driver registers structures

50.1.1 LL_ADC_InitTypeDef

`LL_ADC_InitTypeDef` is defined in the `stm32f0xx_ll_adc.h`

Data Fields

- `uint32_t Clock`
- `uint32_t Resolution`
- `uint32_t DataAlignment`
- `uint32_t LowPowerMode`

Field Documentation

- `uint32_t LL_ADC_InitTypeDef::Clock`

Set ADC instance clock source and prescaler. This parameter can be a value of `ADC_LL_EC_CLOCK_SOURCE`

Note:

- On this STM32 serie, this parameter has some clock ratio constraints: ADC clock synchronous (from PCLK) with prescaler 1 must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle).

This feature can be modified afterwards using unitary function `LL_ADC_SetClock()`. For more details, refer to description of this function.

- `uint32_t LL_ADC_InitTypeDef::Resolution`

Set ADC resolution. This parameter can be a value of `ADC_LL_EC_RESOLUTION` This feature can be modified afterwards using unitary function `LL_ADC_SetResolution()`.

- `uint32_t LL_ADC_InitTypeDef::DataAlignment`

Set ADC conversion data alignment. This parameter can be a value of `ADC_LL_EC_DATA_ALIGN` This feature can be modified afterwards using unitary function `LL_ADC_SetDataAlignment()`.

- `uint32_t LL_ADC_InitTypeDef::LowPowerMode`

Set ADC low power mode. This parameter can be a value of `ADC_LL_EC_LP_MODE` This feature can be modified afterwards using unitary function `LL_ADC_SetLowPowerMode()`.

50.1.2 LL_ADC_REG_InitTypeDef

`LL_ADC_REG_InitTypeDef` is defined in the `stm32f0xx_ll_adc.h`

Data Fields

- `uint32_t TriggerSource`
- `uint32_t SequencerDiscont`
- `uint32_t ContinuousMode`
- `uint32_t DMATransfer`
- `uint32_t Overrun`

Field Documentation

- `uint32_t LL_ADC_REG_InitTypeDef::TriggerSource`

Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of `ADC_LL_EC_REG_TRIGGER_SOURCE`

Note:

- On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function `LL_ADC_REG_SetTriggerEdge()`.

This feature can be modified afterwards using unitary function `LL_ADC_REG_SetTriggerSource()`.

- **`uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont`**
Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of **`ADC_LL_EC_REG_SEQ_DISCONT_MODE`**
Note:
 - This parameter has an effect only if group regular sequencer is enabled (several ADC channels enabled in group regular sequencer).
This feature can be modified afterwards using unitary function **`LL_ADC_REG_SetSequencerDiscont()`**.
- **`uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode`**
Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of **`ADC_LL_EC_REG_CONTINUOUS_MODE`** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode. This feature can be modified afterwards using unitary function **`LL_ADC_REG_SetContinuousMode()`**.
- **`uint32_t LL_ADC_REG_InitTypeDef::DMATransfer`**
Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode. This parameter can be a value of **`ADC_LL_EC_REG_DMA_TRANSFER`** This feature can be modified afterwards using unitary function **`LL_ADC_REG_SetDMATransfer()`**.
- **`uint32_t LL_ADC_REG_InitTypeDef::Overrun`**
Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of **`ADC_LL_EC_REG_OVR_DATA_BEHAVIOR`** This feature can be modified afterwards using unitary function **`LL_ADC_REG_SetOverrun()`**.

50.2 ADC Firmware driver API description

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

50.2.1 Detailed description of functions

`LL_ADC_DMA_GetRegAddr`

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (ADC_TypeDef * ADCx, uint32_t Register)</code>
Function description	Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• Register: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_ADC_DMA_REG.Regular_Data</code>
Return values	<ul style="list-style-type: none">• ADC: register address
Notes	<ul style="list-style-type: none">• These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.• This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: <code>LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG.Regular_Data), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);</code>• For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter.

Reference Manual to LL API cross reference:

- DR DATA LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonPathInternalCh

Function name	<code>__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)</code>
Function description	Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	<ul style="list-style-type: none">• ADCxy_COMMON: ADC common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_ADC_COMMON_INSTANCE()</code>)• PathInternal: This parameter can be a combination of the following values:<ul style="list-style-type: none">– <code>LL_ADC_PATH_INTERNAL_NONE</code>– <code>LL_ADC_PATH_INTERNAL_VREFINT</code>– <code>LL_ADC_PATH_INTERNAL_TEMPSENSOR</code>– <code>LL_ADC_PATH_INTERNAL_VBAT</code> (1) (1) On STM32F0, parameter not available on all devices: all devices except STM32F030x6, STM32F030x8, STM32F030xC, STM32F070x6, STM32F070xB.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• One or several values can be selected. Example: (<code>LL_ADC_PATH_INTERNAL_VREFINT LL_ADC_PATH_INTERNAL_TEMPSENSOR</code>)• Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal <code>LL_ADC_DELAY_VREFINT_STAB_US</code>. Refer to literal <code>LL_ADC_DELAY_TEMPSENSOR_STAB_US</code>.• ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.• On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function <code>LL_ADC_IsEnabled()</code> for each ADC instance or by using helper macro helper macro <code>__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE()</code>.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR VREFEN <code>LL_ADC_SetCommonPathInternalCh</code>• CCR TSEN <code>LL_ADC_SetCommonPathInternalCh</code>• CCR VBATEN <code>LL_ADC_SetCommonPathInternalCh</code>

LL_ADC_GetCommonPathInternalCh

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON)</code>
Function description	Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	<ul style="list-style-type: none">• ADCxy_COMMON: ADC common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_ADC_COMMON_INSTANCE()</code>)

- Return values**
- **Returned:** value can be a combination of the following values:
 - LL_ADC_PATH_INTERNAL_NONE
 - LL_ADC_PATH_INTERNAL_VREFINT
 - LL_ADC_PATH_INTERNAL_TEMPSENSOR
 - LL_ADC_PATH_INTERNAL_VBAT (1)
- (1) On STM32F0, parameter not available on all devices: all devices except STM32F030x6, STM32F030x8, STM32F030xC, STM32F070x6, STM32F070xB.
- Notes**
- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Reference Manual to LL API cross reference:**
- CCR VREFEN LL_ADC_GetCommonPathInternalCh
 - CCR TSEN LL_ADC_GetCommonPathInternalCh
 - CCR VBATEN LL_ADC_GetCommonPathInternalCh

LL_ADC_SetClock

- Function name**
- _STATIC_INLINE void LL_ADC_SetClock (ADC_TypeDef * ADCx, uint32_t ClockSource)**
- Function description**
- Set ADC instance clock source and prescaler.
- Parameters**
- **ADCx:** ADC instance
 - **ClockSource:** This parameter can be one of the following values:
 - LL_ADC_CLOCK_SYNC_PCLK_DIV4
 - LL_ADC_CLOCK_SYNC_PCLK_DIV2
 - LL_ADC_CLOCK_ASYNC (1)
- (1) On this STM32 serie, synchronous clock has no prescaler.
- Return values**
- **None:**
- Notes**
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled.
- Reference Manual to LL API cross reference:**
- CFGR2 CKMODE LL_ADC_SetClock

LL_ADC_GetClock

- Function name**
- _STATIC_INLINE uint32_t LL_ADC_GetClock (ADC_TypeDef * ADCx)**
- Function description**
- Get ADC instance clock source and prescaler.
- Parameters**
- **ADCx:** ADC instance
- Return values**
- **Returned:** value can be one of the following values:
 - LL_ADC_CLOCK_SYNC_PCLK_DIV4
 - LL_ADC_CLOCK_SYNC_PCLK_DIV2
 - LL_ADC_CLOCK_ASYNC (1)
- (1) On this STM32 serie, synchronous clock has no prescaler.

- [Reference Manual to LL API cross reference:](#)
- CFGR2 CKMODE LL_ADC_GetClock

LL_ADC_SetResolution

Function name `__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)`

Function description Set ADC resolution.

- Parameters**
- **ADCx:** ADC instance
 - **Resolution:** This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

- Return values**
- **None:**

- Notes**
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

- [Reference Manual to LL API cross reference:](#)
- CFGR1 RES LL_ADC_SetResolution

LL_ADC_GetResolution

Function name `__STATIC_INLINE uint32_t LL_ADC_GetResolution (ADC_TypeDef * ADCx)`

Function description Get ADC resolution.

- Parameters**
- **ADCx:** ADC instance

- Return values**
- **Returned:** value can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

- [Reference Manual to LL API cross reference:](#)
- CFGR1 RES LL_ADC_GetResolution

LL_ADC_SetDataAlignment

Function name `__STATIC_INLINE void LL_ADC_SetDataAlignment (ADC_TypeDef * ADCx, uint32_t DataAlignment)`

Function description Set ADC conversion data alignment.

Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• DataAlignment: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_DATA_ALIGN_RIGHT– LL_ADC_DATA_ALIGN_LEFT
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Refer to reference manual for alignments formats dependencies to ADC resolutions.• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 ALIGN LL_ADC_SetDataAlignment

LL_ADC_GetDataAlignment

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_GetDataAlignment (ADC_TypeDef * ADCx)</code>
Function description	Get ADC conversion data alignment.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_DATA_ALIGN_RIGHT– LL_ADC_DATA_ALIGN_LEFT
Notes	<ul style="list-style-type: none">• Refer to reference manual for alignments formats dependencies to ADC resolutions.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 ALIGN LL_ADC_SetDataAlignment

LL_ADC_SetLowPowerMode

Function name	<code>__STATIC_INLINE void LL_ADC_SetLowPowerMode (ADC_TypeDef * ADCx, uint32_t LowPowerMode)</code>
Function description	Set ADC low power mode.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• LowPowerMode: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_LP_MODE_NONE– LL_ADC_LP_AUTOWAIT– LL_ADC_LP_AUTOPOWEROFF– LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF
Return values	<ul style="list-style-type: none">• None:

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_MODE_AUTOOFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 WAIT LL_ADC_SetLowPowerMode
- CFGR1 AUTOFF LL_ADC_SetLowPowerMode

LL_ADC_GetLowPowerMode

Function name `__STATIC_INLINE uint32_t LL_ADC_GetLowPowerMode (ADC_TypeDef * ADCx)`

Function description Get ADC low power mode:

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
 - LL_ADC_LP_MODE_NONE
 - LL_ADC_LP_AUTOWAIT
 - LL_ADC_LP_AUTOPOWEROFF
 - LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_MODE_AUTOOFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to LL API cross reference:

- CFGR1 WAIT LL_ADC_GetLowPowerMode
- CFGR1 AUTOFF LL_ADC_GetLowPowerMode

LL_ADC_SetSamplingTimeCommonChannels**Function name**

**STATIC_INLINE void LL_ADC_SetSamplingTimeCommonChannels (ADC_TypeDef *
ADCx, uint32_t SamplingTime)**

Function description

Set sampling time common to a group of channels.

Parameters

- **ADCx:** ADC instance
- **SamplingTime:** This parameter can be one of the following values:
 - LL_ADC_SAMPLINGTIME_1CYCLE_5
 - LL_ADC_SAMPLINGTIME_7CYCLES_5
 - LL_ADC_SAMPLINGTIME_13CYCLES_5
 - LL_ADC_SAMPLINGTIME_28CYCLES_5
 - LL_ADC_SAMPLINGTIME_41CYCLES_5
 - LL_ADC_SAMPLINGTIME_55CYCLES_5
 - LL_ADC_SAMPLINGTIME_71CYCLES_5
 - LL_ADC_SAMPLINGTIME_239CYCLES_5

Return values

- **None:**

Notes

- Unit: ADC clock cycles.
- On this STM32 serie, sampling time scope is on ADC instance: Sampling time common to all channels. (on some other STM32 families, sampling time is channel wise)
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 serie, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SMPR SMP LL_ADC_SetSamplingTimeCommonChannels

LL_ADC_GetSamplingTimeCommonChannels

Function name `__STATIC_INLINE uint32_t LL_ADC_GetSamplingTimeCommonChannels (ADC_TypeDef * ADCx)`

Function description Get sampling time common to a group of channels.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
 - `LL_ADC_SAMPLINGTIME_1CYCLE_5`
 - `LL_ADC_SAMPLINGTIME_7CYCLES_5`
 - `LL_ADC_SAMPLINGTIME_13CYCLES_5`
 - `LL_ADC_SAMPLINGTIME_28CYCLES_5`
 - `LL_ADC_SAMPLINGTIME_41CYCLES_5`
 - `LL_ADC_SAMPLINGTIME_55CYCLES_5`
 - `LL_ADC_SAMPLINGTIME_71CYCLES_5`
 - `LL_ADC_SAMPLINGTIME_239CYCLES_5`

Notes

- Unit: ADC clock cycles.
- On this STM32 serie, sampling time scope is on ADC instance: Sampling time common to all channels. (on some other STM32 families, sampling time is channel wise)
- Conversion time is the addition of sampling time and processing time. Refer to reference manual for ADC processing time of this STM32 serie.

Reference Manual to LL API cross reference:

- SMPR SMP LL_ADC_SetSamplingTimeCommonChannels

LL_ADC_REG_SetTriggerSource

Function name `__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)`

Function description	Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• TriggerSource: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_TRIG_SOFTWARE– LL_ADC_REG_TRIG_EXT_TIM1_TRGO– LL_ADC_REG_TRIG_EXT_TIM1_CH4– LL_ADC_REG_TRIG_EXT_TIM2_TRGO (1)– LL_ADC_REG_TRIG_EXT_TIM3_TRGO– LL_ADC_REG_TRIG_EXT_TIM15_TRGO (1) <p>(1) On STM32F0, parameter not available on all devices</p>
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_REG_SetTriggerEdge().• Availability of parameters of trigger sources from timer depends on timers availability on the selected device.• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 EXTSEL LL_ADC_REG_SetTriggerSource• CFGR1 EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (ADC_TypeDef * ADCx)</code>
Function description	Get ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_TRIG_SOFTWARE– LL_ADC_REG_TRIG_EXT_TIM1_TRGO– LL_ADC_REG_TRIG_EXT_TIM1_CH4– LL_ADC_REG_TRIG_EXT_TIM2_TRGO (1)– LL_ADC_REG_TRIG_EXT_TIM3_TRGO– LL_ADC_REG_TRIG_EXT_TIM15_TRGO (1) <p>(1) On STM32F0, parameter not available on all devices</p>
Notes	<ul style="list-style-type: none">• To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart.• Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

- Reference Manual to LL API cross reference:**
- CFGR1 EXTSEL LL_ADC_REG_GetTriggerSource
 - CFGR1 EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name `__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)`

Function description Get ADC group regular conversion trigger source internal (SW start) or external.

Parameters

- ADCx:** ADC instance

Return values

- Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function `LL_ADC_REG_GetTriggerSource()`.

- Reference Manual to LL API cross reference:**
- CFGR1 EXTEN LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetTriggerEdge

Function name `__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)`

Function description Set ADC group regular conversion trigger polarity.

Parameters

- ADCx:** ADC instance
- ExternalTriggerEdge:** This parameter can be one of the following values:
 - `LL_ADC_REG_TRIG_EXT_RISING`
 - `LL_ADC_REG_TRIG_EXT_FALLING`
 - `LL_ADC_REG_TRIG_EXT_RISINGFALLING`

Return values

- None:**

Notes

- Applicable only for trigger source set to external trigger.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

- Reference Manual to LL API cross reference:**
- CFGR1 EXTEN LL_ADC_REG_SetTriggerEdge

LL_ADC_REG_GetTriggerEdge

Function name `__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge (ADC_TypeDef * ADCx)`

Function description Get ADC group regular conversion trigger polarity.

Parameters

- ADCx:** ADC instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_TRIG_EXT_RISING– LL_ADC_REG_TRIG_EXT_FALLING– LL_ADC_REG_TRIG_EXT_RISINGFALLING
Notes	<ul style="list-style-type: none">• Applicable only for trigger source set to external trigger.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 EXTEN LL_ADC_REG_GetTriggerEdge
LL_ADC_REG_SetSequencerScanDirection	
Function name	<code>__STATIC_INLINE void LL_ADC_REG_SetSequencerScanDirection (ADC_TypeDef * ADCx, uint32_t ScanDirection)</code>
Function description	Set ADC group regular sequencer scan direction.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• ScanDirection: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_SEQ_SCAN_DIR_FORWARD– LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• On some other STM32 families, this setting is not available and the default scan direction is forward.• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 SCANDIR LL_ADC_REG_SetSequencerScanDirection
LL_ADC_REG_GetSequencerScanDirection	
Function name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerScanDirection (ADC_TypeDef * ADCx)</code>
Function description	Get ADC group regular sequencer scan direction.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_SEQ_SCAN_DIR_FORWARD– LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD
Notes	<ul style="list-style-type: none">• On some other STM32 families, this setting is not available and the default scan direction is forward.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 SCANDIR LL_ADC_REG_GetSequencerScanDirection

LL_ADC_REG_SetSequencerDiscont

Function name	<code>__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)</code>
Function description	Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• SeqDiscont: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_SEQ_DISCONT_DISABLE– LL_ADC_REG_SEQ_DISCONT_1RANK
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 DISCEN LL_ADC_REG_SetSequencerDiscont•

LL_ADC_REG_GetSequencerDiscont

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (ADC_TypeDef * ADCx)</code>
Function description	Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_SEQ_DISCONT_DISABLE– LL_ADC_REG_SEQ_DISCONT_1RANK
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 DISCEN LL_ADC_REG_SetSequencerDiscont•

LL_ADC_REG_SetSequencerChannels

Function name	<code>__STATIC_INLINE void LL_ADC_REG_SetSequencerChannels (ADC_TypeDef * ADCx, uint32_t Channel)</code>
Function description	Set ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

(1) On STM32F0, parameter not available on all devices: all devices except STM32F030x6, STM32F030x8, STM32F030xC, STM32F070x6, STM32F070xB.

Return values

- **None:**

Notes

- This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Set channels selected by overwriting the current sequencer configuration.
- On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSEL0 LL_ADC_REG_SetSequencerChannels
- CHSEL1 LL_ADC_REG_SetSequencerChannels
- CHSEL2 LL_ADC_REG_SetSequencerChannels
- CHSEL3 LL_ADC_REG_SetSequencerChannels
- CHSEL4 LL_ADC_REG_SetSequencerChannels
- CHSEL5 LL_ADC_REG_SetSequencerChannels
- CHSEL6 LL_ADC_REG_SetSequencerChannels
- CHSEL7 LL_ADC_REG_SetSequencerChannels
- CHSEL8 LL_ADC_REG_SetSequencerChannels
- CHSEL9 LL_ADC_REG_SetSequencerChannels
- CHSEL10 LL_ADC_REG_SetSequencerChannels
- CHSEL11 LL_ADC_REG_SetSequencerChannels
- CHSEL12 LL_ADC_REG_SetSequencerChannels
- CHSEL13 LL_ADC_REG_SetSequencerChannels
- CHSEL14 LL_ADC_REG_SetSequencerChannels
- CHSEL15 LL_ADC_REG_SetSequencerChannels
- CHSEL16 LL_ADC_REG_SetSequencerChannels
- CHSEL17 LL_ADC_REG_SetSequencerChannels
- CHSEL18 LL_ADC_REG_SetSequencerChannels

LL_ADC_REG_SetSequencerChAdd

Function name `__STATIC_INLINE void LL_ADC_REG_SetSequencerChAdd (ADC_TypeDef * ADCx, uint32_t Channel)`

Function description Add channel to ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

(1) On STM32F0, parameter not available on all devices: all devices except STM32F030x6, STM32F030x8, STM32F030xC, STM32F070x6, STM32F070xB.

Return values

- **None:**

Notes

- This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Set channels selected by adding them to the current sequencer configuration.
- On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSEL0 LL_ADC_REG_SetSequencerChAdd
- CHSEL1 LL_ADC_REG_SetSequencerChAdd
- CHSEL2 LL_ADC_REG_SetSequencerChAdd
- CHSEL3 LL_ADC_REG_SetSequencerChAdd
- CHSEL4 LL_ADC_REG_SetSequencerChAdd
- CHSEL5 LL_ADC_REG_SetSequencerChAdd
- CHSEL6 LL_ADC_REG_SetSequencerChAdd
- CHSEL7 LL_ADC_REG_SetSequencerChAdd
- CHSEL8 LL_ADC_REG_SetSequencerChAdd
- CHSEL9 LL_ADC_REG_SetSequencerChAdd
- CHSEL10 LL_ADC_REG_SetSequencerChAdd
- CHSEL11 LL_ADC_REG_SetSequencerChAdd
- CHSEL12 LL_ADC_REG_SetSequencerChAdd
- CHSEL13 LL_ADC_REG_SetSequencerChAdd
- CHSEL14 LL_ADC_REG_SetSequencerChAdd
- CHSEL15 LL_ADC_REG_SetSequencerChAdd
- CHSEL16 LL_ADC_REG_SetSequencerChAdd
- CHSEL17 LL_ADC_REG_SetSequencerChAdd
- CHSEL18 LL_ADC_REG_SetSequencerChAdd

LL_ADC_REG_SetSequencerChRem**Function name**

```
__STATIC_INLINE void LL_ADC_REG_SetSequencerChRem (ADC_TypeDef * ADCx,  
                                                uint32_t Channel)
```

Function description

Remove channel to ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

(1) On STM32F0, parameter not available on all devices: all devices except STM32F030x6, STM32F030x8, STM32F030xC, STM32F070x6, STM32F070xB.

Return values

- **None:**

Notes

- This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Set channels selected by removing them to the current sequencer configuration.
- On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSEL0 LL_ADC_REG_SetSequencerChRem
- CHSEL1 LL_ADC_REG_SetSequencerChRem
- CHSEL2 LL_ADC_REG_SetSequencerChRem
- CHSEL3 LL_ADC_REG_SetSequencerChRem
- CHSEL4 LL_ADC_REG_SetSequencerChRem
- CHSEL5 LL_ADC_REG_SetSequencerChRem
- CHSEL6 LL_ADC_REG_SetSequencerChRem
- CHSEL7 LL_ADC_REG_SetSequencerChRem
- CHSEL8 LL_ADC_REG_SetSequencerChRem
- CHSEL9 LL_ADC_REG_SetSequencerChRem
- CHSEL10 LL_ADC_REG_SetSequencerChRem
- CHSEL11 LL_ADC_REG_SetSequencerChRem
- CHSEL12 LL_ADC_REG_SetSequencerChRem
- CHSEL13 LL_ADC_REG_SetSequencerChRem
- CHSEL14 LL_ADC_REG_SetSequencerChRem
- CHSEL15 LL_ADC_REG_SetSequencerChRem
- CHSEL16 LL_ADC_REG_SetSequencerChRem
- CHSEL17 LL_ADC_REG_SetSequencerChRem
- CHSEL18 LL_ADC_REG_SetSequencerChRem

LL_ADC_REG_GetSequencerChannels

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerChannels (ADC_TypeDef * ADCx)</code>
Function description	Get ADC group regular sequence: channel on rank corresponding to channel number.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance

Return values

- **Returned:** value can be a combination of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

(1) On STM32F0, parameter not available on all devices: all devices except STM32F030x6, STM32F030x8, STM32F030xC, STM32F070x6, STM32F070xB.

Notes

- This function performs: Channels order reading into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).
- On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (Vrefint, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be retrieved. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSELR CHSEL0 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL1 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL2 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL3 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL4 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL5 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL6 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL7 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL8 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL9 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL10 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL11 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL12 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL13 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL14 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL15 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL16 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL17 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL18 LL_ADC_REG_GetSequencerChannels

LL_ADC_REG_SetContinuousMode

Function name `__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)`

Function description Set ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance
- **Continuous:** This parameter can be one of the following values:
 - `LL_ADC_REG_CONV_SINGLE`
 - `LL_ADC_REG_CONV_CONTINUOUS`

Return values

- **None:**

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- `CFGR1 CONT LL_ADC_REG_SetContinuousMode`

LL_ADC_REG_GetContinuousMode

Function name `__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (ADC_TypeDef * ADCx)`

Function description Get ADC continuous conversion mode on ADC group regular.

Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_CONV_SINGLE– LL_ADC_REG_CONV_CONTINUOUS
Notes	<ul style="list-style-type: none">• Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 CONT LL_ADC_REG_GetContinuousMode
LL_ADC_REG_SetDMATransfer	
Function name	<code>__STATIC_INLINE void LL_ADC_REG_SetDMATransfer (ADC_TypeDef * ADCx, uint32_t DMATransfer)</code>
Function description	Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• DMATransfer: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_DMA_TRANSFER_NONE– LL_ADC_REG_DMA_TRANSFER_LIMITED– LL_ADC_REG_DMA_TRANSFER_UNLIMITED
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.• If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).• To configure DMA source address (peripheral address), use function <code>LL_ADC_DMA_GetRegAddr()</code>.• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 DMAEN LL_ADC_REG_SetDMATransfer• CFGR1 DMACFG LL_ADC_REG_SetDMATransfer
LL_ADC_REG_GetDMATransfer	
Function name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer (ADC_TypeDef * ADCx)</code>
Function description	Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_DMA_TRANSFER_NONE– LL_ADC_REG_DMA_TRANSFER_LIMITED– LL_ADC_REG_DMA_TRANSFER_UNLIMITED
Notes	<ul style="list-style-type: none">• If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.• If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).• To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 DMAEN LL_ADC_REG_GetDMATransfer• CFGR1 DMACFG LL_ADC_REG_GetDMATransfer

LL_ADC_REG_SetOverrun

Function name	STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)
Function description	Set ADC group regular behavior in case of overrun: data preserved or overwritten.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• Overrun: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_OVR_DATA_PRESERVED– LL_ADC_REG_OVR_DATA_OVERWRITTEN
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten.• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 OVRMOD LL_ADC_REG_SetOverrun

LL_ADC_REG_GetOverrun

Function name	STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun (ADC_TypeDef * ADCx)
Function description	Get ADC group regular behavior in case of overrun: data preserved or overwritten.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_ADC_REG_OVR_DATA_PRESERVED– LL_ADC_REG_OVR_DATA_OVERWRITTEN
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR1 OVRMOD LL_ADC_REG_GetOverrun
LL_ADC_SetAnalogWDMonitChannels	
Function name	_STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDChannelGroup)
Function description	Set ADC analog watchdog monitored channels: a single channel or all channels, on ADC group regular.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• AWDChannelGroup: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_AWD_DISABLE– LL_ADC_AWD_ALL_CHANNELS_REG– LL_ADC_AWD_CHANNEL_0_REG– LL_ADC_AWD_CHANNEL_1_REG– LL_ADC_AWD_CHANNEL_2_REG– LL_ADC_AWD_CHANNEL_3_REG– LL_ADC_AWD_CHANNEL_4_REG– LL_ADC_AWD_CHANNEL_5_REG– LL_ADC_AWD_CHANNEL_6_REG– LL_ADC_AWD_CHANNEL_7_REG– LL_ADC_AWD_CHANNEL_8_REG– LL_ADC_AWD_CHANNEL_9_REG– LL_ADC_AWD_CHANNEL_10_REG– LL_ADC_AWD_CHANNEL_11_REG– LL_ADC_AWD_CHANNEL_12_REG– LL_ADC_AWD_CHANNEL_13_REG– LL_ADC_AWD_CHANNEL_14_REG– LL_ADC_AWD_CHANNEL_15_REG– LL_ADC_AWD_CHANNEL_16_REG– LL_ADC_AWD_CHANNEL_17_REG– LL_ADC_AWD_CHANNEL_18_REG (1)– LL_ADC_AWD_CH_VREFINT_REG– LL_ADC_AWD_CH_TEMPSENSOR_REG– LL_ADC_AWD_CH_VBAT_REG (1)
	<p>(1) On STM32F0, parameter not available on all devices: all devices except STM32F030x6, STM32F030x8, STM32F030xC, STM32F070x6, STM32F070xB.</p>
Return values	<ul style="list-style-type: none">• None:

Notes

- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro `_LL_ADC_ANALOGWD_CHANNEL_GROUP()`.
- On this STM32 serie, there is only 1 kind of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- `CFGGR1 AWDCH LL_ADC_SetAnalogWDMonitChannels`
- `CFGGR1 AWDSGL LL_ADC_SetAnalogWDMonitChannels`
- `CFGGR1 AWDEN LL_ADC_SetAnalogWDMonitChannels`

LL_ADC_GetAnalogWDMonitChannels

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx)</code>
Function description	Get ADC analog watchdog monitored channel.
Parameters	<ul style="list-style-type: none">ADCx: ADC instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">- <code>LL_ADC_AWD_DISABLE</code>- <code>LL_ADC_AWD_ALL_CHANNELS_REG</code>- <code>LL_ADC_AWD_CHANNEL_0_REG</code>- <code>LL_ADC_AWD_CHANNEL_1_REG</code>- <code>LL_ADC_AWD_CHANNEL_2_REG</code>- <code>LL_ADC_AWD_CHANNEL_3_REG</code>- <code>LL_ADC_AWD_CHANNEL_4_REG</code>- <code>LL_ADC_AWD_CHANNEL_5_REG</code>- <code>LL_ADC_AWD_CHANNEL_6_REG</code>- <code>LL_ADC_AWD_CHANNEL_7_REG</code>- <code>LL_ADC_AWD_CHANNEL_8_REG</code>- <code>LL_ADC_AWD_CHANNEL_9_REG</code>- <code>LL_ADC_AWD_CHANNEL_10_REG</code>- <code>LL_ADC_AWD_CHANNEL_11_REG</code>- <code>LL_ADC_AWD_CHANNEL_12_REG</code>- <code>LL_ADC_AWD_CHANNEL_13_REG</code>- <code>LL_ADC_AWD_CHANNEL_14_REG</code>- <code>LL_ADC_AWD_CHANNEL_15_REG</code>- <code>LL_ADC_AWD_CHANNEL_16_REG</code>- <code>LL_ADC_AWD_CHANNEL_17_REG</code>- <code>LL_ADC_AWD_CHANNEL_18_REG</code>

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function. To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 serie, there is only 1 kind of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 AWDCH LL_ADC_GetAnalogWDMonitChannels
- CFGR1 AWDSGL LL_ADC_GetAnalogWDMonitChannels
- CFGR1 AWDEN LL_ADC_GetAnalogWDMonitChannels

LL_ADC_ConfigAnalogWDThresholds**Function name**

```
__STATIC_INLINE void LL_ADC_ConfigAnalogWDThresholds (ADC_TypeDef * ADCx,  
uint32_t AWDThresholdHighValue, uint32_t AWDThresholdLowValue)
```

Function description

Set ADC analog watchdog thresholds value of both thresholds high and low.

Parameters

- **ADCx:** ADC instance
- **AWDThresholdHighValue:** Value between Min_Data=0x000 and Max_Data=0xFFFF
- **AWDThresholdLowValue:** Value between Min_Data=0x000 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- If value of only one threshold high or low must be set, use function LL_ADC_SetAnalogWDThresholds().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 serie, there is only 1 kind of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- TR HT LL_ADC_ConfigAnalogWDThresholds
- TR LT LL_ADC_ConfigAnalogWDThresholds

LL_ADC_SetAnalogWDThresholds**Function name**

```
__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx,  
uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)
```

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• AWDThresholdsHighLow: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_AWD_THRESHOLD_HIGH– LL_ADC_AWD_THRESHOLD_LOW• AWDThresholdValue: Value between Min_Data=0x000 and Max_Data=0xFFFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• If values of both thresholds high or low must be set, use function LL_ADC_ConfigAnalogWDThresholds().• In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().• On this STM32 serie, there is only 1 kind of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TR HT LL_ADC_SetAnalogWDThresholds• TR LT LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow)</code>
Function description	Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• AWDThresholdsHighLow: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_ADC_AWD_THRESHOLD_HIGH– LL_ADC_AWD_THRESHOLD_LOW– LL_ADC_AWD_THRESHOLDS_HIGH_LOW
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x000 and Max_Data=0xFFFF
Notes	<ul style="list-style-type: none">• If raw data with ADC thresholds high and low is retrieved, the data of each threshold high or low can be isolated using helper macro __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW().• In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TR1 HT1 LL_ADC_GetAnalogWDThresholds• TR2 HT2 LL_ADC_GetAnalogWDThresholds• TR3 HT3 LL_ADC_GetAnalogWDThresholds• TR1 LT1 LL_ADC_GetAnalogWDThresholds• TR2 LT2 LL_ADC_GetAnalogWDThresholds• TR3 LT3 LL_ADC_GetAnalogWDThresholds

LL_ADC_Enable

Function name	<code>__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)</code>
Function description	Enable the selected ADC instance.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• On this STM32 serie, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB.• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name	<code>__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)</code>
Function description	Disable the selected ADC instance.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabled (ADC_TypeDef * ADCx)</code>
Function description	Get the selected ADC instance enable state.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• 0: ADC is disabled, 1: ADC is enabled.
Notes	<ul style="list-style-type: none">• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_IsEnabled

LL_ADC_IsDisableOngoing

Function name `__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (ADC_TypeDef * ADCx)`

Function description Get the selected ADC instance disable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_IsDisableOngoing

LL_ADC_StartCalibration

Function name `__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx)`

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

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal `LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES`.
- In case of usage of ADC with DMA transfer: On this STM32 serie, ADC DMA transfer request should be disabled during calibration: Calibration factor is available in data register and also transferred by DMA. To not insert ADC calibration factor among ADC conversion data in array variable, DMA transfer must be disabled during calibration. (DMA transfer setting backup and disable before calibration, DMA transfer setting restore after calibration. Refer to functions `LL_ADC_REG_GetDMATransfer()`, `LL_ADC_REG_SetDMATransfer()`).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name `__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing (ADC_TypeDef * ADCx)`

Function description Get ADC calibration state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** calibration complete, 1: calibration in progress.

[Reference Manual to LL API cross reference:](#)

- CR ADCAL LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversion

Function name

`__STATIC_INLINE void LL_ADC_REG_StartConversion (ADC_TypeDef * ADCx)`

Function description

Start ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately. If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

[Reference Manual to LL API cross reference:](#)

- CR ADSTART LL_ADC_REG_StartConversion

LL_ADC_REG_StopConversion

Function name

`__STATIC_INLINE void LL_ADC_REG_StopConversion (ADC_TypeDef * ADCx)`

Function description

Stop ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.

[Reference Manual to LL API cross reference:](#)

- CR ADSTP LL_ADC_REG_StopConversion

LL_ADC_REG_IsConversionOngoing

Function name

`__STATIC_INLINE uint32_t LL_ADC_REG_IsConversionOngoing (ADC_TypeDef * ADCx)`

Function description

Get ADC group regular conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_IsConversionOngoing

LL_ADC_REG_IsStopConversionOngoing

Function name `__STATIC_INLINE uint32_t LL_ADC_REG_IsStopConversionOngoing (ADC_TypeDef * ADCx)`

Function description Get ADC group regular command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_IsStopConversionOngoing

LL_ADC_REG_ReadConversionData32

Function name `__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (ADC_TypeDef * ADCx)`

Function description Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- DR DATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData12

Function name `__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (ADC_TypeDef * ADCx)`

Function description Get ADC group regular conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance

Return values

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

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR DATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadConversionData10

Function name `__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (ADC_TypeDef * ADCx)`

Function description	Get ADC group regular conversion data, range fit for ADC resolution 10 bits.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x000 and Max_Data=0x3FF
Notes	<ul style="list-style-type: none">• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR DATA LL_ADC_REG_ReadConversionData10
LL_ADC_REG_ReadConversionData8	
Function name	<u>__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (ADC_TypeDef * ADCx)</u>
Function description	Get ADC group regular conversion data, range fit for ADC resolution 8 bits.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0xFF
Notes	<ul style="list-style-type: none">• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR DATA LL_ADC_REG_ReadConversionData8
LL_ADC_REG_ReadConversionData6	
Function name	<u>__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (ADC_TypeDef * ADCx)</u>
Function description	Get ADC group regular conversion data, range fit for ADC resolution 6 bits.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0x3F
Notes	<ul style="list-style-type: none">• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR DATA LL_ADC_REG_ReadConversionData6
LL_ADC_IsActiveFlag_ADRDY	
Function name	<u>__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (ADC_TypeDef * ADCx)</u>
Function description	Get flag ADC ready.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance

- Return values**
- **State:** of bit (1 or 0).
- Notes**
- On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

- Reference Manual to LL API cross reference:**
- ISR ADRDY LL_ADC_IsActiveFlag_ADRDY

LL_ADC_IsActiveFlag_EOC

Function name `__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (ADC_TypeDef * ADCx)`

Function description Get flag ADC group regular end of unitary conversion.

- Parameters**
- **ADCx:** ADC instance

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

- Reference Manual to LL API cross reference:**
- ISR EOC LL_ADC_IsActiveFlag_EOC

LL_ADC_IsActiveFlag_EOS

Function name `__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (ADC_TypeDef * ADCx)`

Function description Get flag ADC group regular end of sequence conversions.

- Parameters**
- **ADCx:** ADC instance

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

- Reference Manual to LL API cross reference:**
- ISR EOSEQ LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_OVR

Function name `__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR (ADC_TypeDef * ADCx)`

Function description Get flag ADC group regular overrun.

- Parameters**
- **ADCx:** ADC instance

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

- Reference Manual to LL API cross reference:**
- ISR OVR LL_ADC_IsActiveFlag_OVR

LL_ADC_IsActiveFlag_EOSMP

Function name `__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP (ADC_TypeDef * ADCx)`

Function description Get flag ADC group regular end of sampling phase.

Parameters • **ADCx:** ADC instance

Return values • **State:** of bit (1 or 0).

Reference Manual to LL API cross reference: • ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

LL_ADC_IsActiveFlag_AWD1

Function name **_STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (ADC_TypeDef * ADCx)**

Function description Get flag ADC analog watchdog 1 flag.

Parameters • **ADCx:** ADC instance

Return values • **State:** of bit (1 or 0).

Reference Manual to LL API cross reference: • ISR AWD LL_ADC_IsActiveFlag_AWD1

LL_ADC_ClearFlag_ADRDY

Function name **_STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)**

Function description Clear flag ADC ready.

Parameters • **ADCx:** ADC instance

Return values • **None:**

Notes • On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference: • ISR ADRDY LL_ADC_ClearFlag_ADRDY

LL_ADC_ClearFlag_EOC

Function name **_STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)**

Function description Clear flag ADC group regular end of unitary conversion.

Parameters • **ADCx:** ADC instance

Return values • **None:**

Reference Manual to LL API cross reference: • ISR EOC LL_ADC_ClearFlag_EOC

LL_ADC_ClearFlag_EOS

Function name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)</code>
Function description	Clear flag ADC group regular end of sequence conversions.
Parameters	<ul style="list-style-type: none">• <code>ADCx</code>: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR EOSEQ LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)</code>
Function description	Clear flag ADC group regular overrun.
Parameters	<ul style="list-style-type: none">• <code>ADCx</code>: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)</code>
Function description	Clear flag ADC group regular end of sampling phase.
Parameters	<ul style="list-style-type: none">• <code>ADCx</code>: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_AWD1

Function name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)</code>
Function description	Clear flag ADC analog watchdog 1.
Parameters	<ul style="list-style-type: none">• <code>ADCx</code>: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR AWD LL_ADC_ClearFlag_AWD1

LL_ADC_EnableIT_ADRDY

Function name	<code>__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)</code>
Function description	Enable ADC ready.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name	<code>__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)</code>
Function description	Enable interruption ADC group regular end of unitary conversion.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name	<code>__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)</code>
Function description	Enable interruption ADC group regular end of sequence conversions.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOSEQIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name	<code>__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)</code>
Function description	Enable ADC group regular interruption overrun.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name	<code>__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)</code>
Function description	Enable interruption ADC group regular end of sampling.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_AWD1

Function name	<code>__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)</code>
Function description	Enable interruption ADC analog watchdog 1.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER AWDIE LL_ADC_EnableIT_AWD1

LL_ADC_DisableIT_ADRDY

Function name	<code>__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)</code>
Function description	Disable interruption ADC ready.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name	<code>__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)</code>
Function description	Disable interruption ADC group regular end of unitary conversion.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name	<code>__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)</code>
Function description	Disable interruption ADC group regular end of sequence conversions.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOSEQIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name	<code>__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)</code>
Function description	Disable interruption ADC group regular overrun.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name	<code>__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)</code>
Function description	Disable interruption ADC group regular end of sampling.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_AWD1

Function name	<code>__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)</code>
Function description	Disable interruption ADC analog watchdog 1.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER AWDIE LL_ADC_DisableIT_AWD1

LL_ADC_IsEnabledIT_ADRDY

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (ADC_TypeDef * ADCx)</code>
Function description	Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (ADC_TypeDef * ADCx)</code>
Function description	Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (ADC_TypeDef * ADCx)</code>
Function description	Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IER EOSEQIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (ADC_TypeDef * ADCx)</code>
Function description	Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).

[Reference Manual to LL API cross reference:](#)

- IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (ADC_TypeDef * ADCx)

[Function description](#)

Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).

[Parameters](#)

- **ADCx:** ADC instance

[Return values](#)

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

[Reference Manual to LL API cross reference:](#)

- IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP

LL_ADC_IsEnabledIT_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (ADC_TypeDef * ADCx)

[Function description](#)

Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).

[Parameters](#)

- **ADCx:** ADC instance

[Return values](#)

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

[Reference Manual to LL API cross reference:](#)

- IER AWDIE LL_ADC_IsEnabledIT_AWD1

LL_ADC_CommonDelinit

Function name

ErrorStatus LL_ADC_CommonDelinit (ADC_Common_TypeDef * ADCxy_COMMON)

[Function description](#)

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

[Parameters](#)

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

[Return values](#)

- **An:** ErrorStatus enumeration value:
 - SUCCESS: ADC common registers are de-initialized
 - ERROR: not applicable

[Notes](#)

- This function is performing a hard reset, using high level clock source RCC ADC reset.

LL_ADC_Delinit

Function name

ErrorStatus LL_ADC_Delinit (ADC_TypeDef * ADCx)

[Function description](#)

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

Parameters	<ul style="list-style-type: none">• ADCx: ADC instance
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: ADC registers are de-initialized– ERROR: ADC registers are not de-initialized
Notes	<ul style="list-style-type: none">• To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDelInit().• If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Refer to function LL_ADC_CommonDelInit().

LL_ADC_Init

Function name	ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_InitStruct)
Function description	Initialize some features of ADC instance.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• ADC_InitStruct: Pointer to a LL_ADC_REG_InitTypeDef structure
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: ADC registers are initialized– ERROR: ADC registers are not initialized
Notes	<ul style="list-style-type: none">• These parameters have an impact on ADC scope: ADC instance. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .• The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.• After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular sequencer: map channel on rank corresponding to channel number. Refer to function LL_ADC_REG_SetSequencerChannels(); Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name	void LL_ADC_StructInit (LL_ADC_InitTypeDef * ADC_InitStruct)
Function description	Set each LL_ADC_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none">• ADC_InitStruct: Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.
Return values	<ul style="list-style-type: none">• None:

LL_ADC_REG_Init

Function name	<code>ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)</code>
Function description	Initialize some features of ADC group regular.
Parameters	<ul style="list-style-type: none">• ADCx: ADC instance• ADC_REG_InitStruct: Pointer to a LL_ADC_REG_InitTypeDef structure
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: ADC registers are initialized– ERROR: ADC registers are not initialized
Notes	<ul style="list-style-type: none">• These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").• The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.• After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular sequencer: map channel on rank corresponding to channel number. Refer to function <code>LL_ADC_REG_SetSequencerChannels();</code> Set ADC channel sampling time Refer to function <code>LL_ADC_SetChannelSamplingTime();</code>

LL_ADC_REG_StructInit

Function name	<code>void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)</code>
Function description	Set each LL_ADC_REG_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none">• ADC_REG_InitStruct: Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.
Return values	<ul style="list-style-type: none">• None:

50.3 ADC Firmware driver defines

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

50.3.1 ADC

ADC

Analog watchdog - Monitored channels

`LL_ADC_AWD_DISABLE` ADC analog watchdog monitoring disabled

`LL_ADC_AWD_ALL_CHAN` ADC analog watchdog monitoring of all channels, converted by group regular only
`NELS_REG`

LL_ADC_AWD_CHANNEL_0_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN0, converted by group regular only)

LL_ADC_AWD_CHANNEL_1_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN1, converted by group regular only)

LL_ADC_AWD_CHANNEL_2_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN2, converted by group regular only)

LL_ADC_AWD_CHANNEL_3_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN3, converted by group regular only)

LL_ADC_AWD_CHANNEL_4_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN4, converted by group regular only)

LL_ADC_AWD_CHANNEL_5_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN5, converted by group regular only)

LL_ADC_AWD_CHANNEL_6_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN6, converted by group regular only)

LL_ADC_AWD_CHANNEL_7_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN7, converted by group regular only)

LL_ADC_AWD_CHANNEL_8_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN8, converted by group regular only)

LL_ADC_AWD_CHANNEL_9_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN9, converted by group regular only)

LL_ADC_AWD_CHANNEL_10_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN10, converted by group regular only)

LL_ADC_AWD_CHANNEL_11_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN11, converted by group regular only)

LL_ADC_AWD_CHANNEL_12_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN12, converted by group regular only)

LL_ADC_AWD_CHANNEL_13_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN13, converted by group regular only)

LL_ADC_AWD_CHANNEL_14_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN14, converted by group regular only)

LL_ADC_AWD_CHANNEL_15_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN15, converted by group regular only)

LL_ADC_AWD_CHANNEL_16_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN16, converted by group regular only)

LL_ADC_AWD_CHANNEL_17_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin ADCx_IN17, converted by group regular only)

LL_ADC_AWD_CH_VREFIN_T_REG ADC analog watchdog monitoring of ADC internal channel connected to Vrefint: Internal voltage reference, converted by group regular only

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

LL_ADC_AWD_CHANNEL_18_REG ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group regular only

LL_ADC_AWD_CH_VBAT_REG ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda, converted by group regular only

Analog watchdog - Analog watchdog number

LL_ADC_AWD1 ADC analog watchdog number 1

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOL_D_HIGH ADC analog watchdog threshold high

LL_ADC_AWD_THRESHOL_D_LOW ADC analog watchdog threshold low

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

ADC instance - Channel number

LL_ADC_CHANNEL_0 ADC external channel (channel connected to GPIO pin) ADCx_IN0

LL_ADC_CHANNEL_1 ADC external channel (channel connected to GPIO pin) ADCx_IN1

LL_ADC_CHANNEL_2 ADC external channel (channel connected to GPIO pin) ADCx_IN2

LL_ADC_CHANNEL_3 ADC external channel (channel connected to GPIO pin) ADCx_IN3

LL_ADC_CHANNEL_4 ADC external channel (channel connected to GPIO pin) ADCx_IN4

LL_ADC_CHANNEL_5 ADC external channel (channel connected to GPIO pin) ADCx_IN5

LL_ADC_CHANNEL_6 ADC external channel (channel connected to GPIO pin) ADCx_IN6

LL_ADC_CHANNEL_7 ADC external channel (channel connected to GPIO pin) ADCx_IN7

LL_ADC_CHANNEL_8 ADC external channel (channel connected to GPIO pin) ADCx_IN8

LL_ADC_CHANNEL_9 ADC external channel (channel connected to GPIO pin) ADCx_IN9

LL_ADC_CHANNEL_10 ADC external channel (channel connected to GPIO pin) ADCx_IN10

LL_ADC_CHANNEL_11 ADC external channel (channel connected to GPIO pin) ADCx_IN11

<code>LL_ADC_CHANNEL_12</code>	ADC external channel (channel connected to GPIO pin) ADCx_IN12
<code>LL_ADC_CHANNEL_13</code>	ADC external channel (channel connected to GPIO pin) ADCx_IN13
<code>LL_ADC_CHANNEL_14</code>	ADC external channel (channel connected to GPIO pin) ADCx_IN14
<code>LL_ADC_CHANNEL_15</code>	ADC external channel (channel connected to GPIO pin) ADCx_IN15
<code>LL_ADC_CHANNEL_16</code>	ADC external channel (channel connected to GPIO pin) ADCx_IN16
<code>LL_ADC_CHANNEL_17</code>	ADC external channel (channel connected to GPIO pin) ADCx_IN17
<code>LL_ADC_CHANNEL_VREFI</code>	ADC internal channel connected to VrefInt: Internal voltage reference. NT
<code>LL_ADC_CHANNEL_TEMP_SENSOR</code>	ADC internal channel connected to Temperature sensor.
<code>LL_ADC_CHANNEL_18</code>	ADC external channel (channel connected to GPIO pin) ADCx_IN18
<code>LL_ADC_CHANNEL_VBAT</code>	ADC internal channel connected to Vbat/2: Vbat voltage through a divider ladder of factor 1/2 to have Vbat always below Vdda.

Channel - Sampling time

`LL_ADC_SAMPLINGTIME_1CYCLE_5` Sampling time 1.5 ADC clock cycle

`LL_ADC_SAMPLINGTIME_7CYCLES_5` Sampling time 7.5 ADC clock cycles

`LL_ADC_SAMPLINGTIME_13CYCLES_5` Sampling time 13.5 ADC clock cycles

`LL_ADC_SAMPLINGTIME_28CYCLES_5` Sampling time 28.5 ADC clock cycles

`LL_ADC_SAMPLINGTIME_41CYCLES_5` Sampling time 41.5 ADC clock cycles

`LL_ADC_SAMPLINGTIME_55CYCLES_5` Sampling time 55.5 ADC clock cycles

`LL_ADC_SAMPLINGTIME_71CYCLES_5` Sampling time 71.5 ADC clock cycles

`LL_ADC_SAMPLINGTIME_239CYCLES_5` Sampling time 239.5 ADC clock cycles

ADC instance - Clock source

`LL_ADC_CLOCK_SYNC_PCLK_DIV4` ADC synchronous clock derived from AHB clock divided by 4

LL_ADC_CLOCK_SYNC_P ADC synchronous clock derived from AHB clock divided by 2
CLK_DIV2

LL_ADC_CLOCK_ASYNC ADC asynchronous clock. On this STM32 serie, asynchronous clock has no prescaler.

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL ADC measurement pathes all disabled
_NONE

LL_ADC_PATH_INTERNAL ADC measurement path to internal channel Vrefint
_VREFINT

LL_ADC_PATH_INTERNAL ADC measurement path to internal channel temperature sensor
_TEMPSENSOR

LL_ADC_PATH_INTERNAL ADC measurement path to internal channel Vbat
_VBAT

ADC instance - Data alignment

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

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

ADC flags

LL_ADC_FLAG_ADRDY ADC flag ADC instance ready

LL_ADC_FLAG_EOC ADC flag ADC group regular end of unitary conversion

LL_ADC_FLAG_EOS ADC flag ADC group regular end of sequence conversions

LL_ADC_FLAG_OVR ADC flag ADC group regular overrun

LL_ADC_FLAG_EOSMP ADC flag ADC group regular end of sampling phase

LL_ADC_FLAG_AWD1 ADC flag ADC analog watchdog 1

ADC instance - Groups

LL_ADC_GROUP_REGULA ADC group regular (available on all STM32 devices)
R

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_VREFINT Delay for internal voltage reference stabilization time
_STAB_US

LL_ADC_DELAY_TEMPSE Delay for temperature sensor stabilization time
NSOR_STAB_US

LL_ADC_DELAY_CALIB_E Delay required between ADC end of calibration and ADC enable
NABLE_ADC_CYCLES

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY	ADC interruption ADC instance ready
LL_ADC_IT_EOC	ADC interruption ADC group regular end of unitary conversion
LL_ADC_IT_EOS	ADC interruption ADC group regular end of sequence conversions
LL_ADC_IT_OVR	ADC interruption ADC group regular overrun
LL_ADC_IT_EOSMP	ADC interruption ADC group regular end of sampling phase
LL_ADC_IT_AWD1	ADC interruption ADC analog watchdog 1

ADC instance - Low power mode

LL_ADC_LP_MODE_NONE	No ADC low power mode activated
LL_ADC_LP_AUTOWAIT	ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function
LL_ADC_LP_AUTOPOWER OFF	ADC low power mode auto power-off: the ADC automatically powers-off after a ADC conversion and automatically wakes up when a new ADC conversion is triggered (with startup time between trigger and start of sampling). See description with function
LL_ADC_LP_AUTOWAIT_A UTOPOWEROFF	ADC low power modes auto wait and auto power-off combined. See description with function

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGU_LAR_DATA

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SING LE ADC conversions are performed in single mode: one conversion per trigger

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

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRAN SFER_NONE ADC conversions are not transferred by DMA

LL_ADC_REG_DMA_TRAN SFER_LIMITED ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

LL_ADC_REG_DMA_TRAN_SFER_UNLIMITED ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED ADC group regular behavior in case of overrun: data preserved

LL_ADC_REG_OVR_DATA_OVERWRITTEN ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCO_NT_DISABLE ADC group regular sequencer discontinuous mode disable

LL_ADC_REG_SEQ_DISCO_NT_1RANK ADC group regular sequencer discontinuous mode enable with sequence interruption every rank

ADC group regular - Sequencer scan direction

LL_ADC_REG_SEQ_SCAN_DIR_FORWARD ADC group regular sequencer scan direction forward: from lowest channel number to highest channel number (scan of all ranks, ADC conversion of ranks with channels enabled in sequencer). On some other STM32 families, this setting is not available and the default scan direction is forward.

LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD ADC group regular sequencer scan direction backward: from highest channel number to lowest channel number (scan of all ranks, ADC conversion of ranks with channels enabled in sequencer)

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING ADC group regular conversion trigger polarity set to rising edge

LL_ADC_REG_TRIG_EXT_FALLING ADC group regular conversion trigger polarity set to falling edge

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

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFT_WARE ADC group regular conversion trigger internal: SW start.

LL_ADC_REG_TRIG_EXT_TIM1_TRGO ADC group regular conversion trigger from external IP: TIM1 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH4 ADC group regular conversion trigger from external IP: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO ADC group regular conversion trigger from external IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).

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

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

ADC instance - Resolution

LL_ADC_RESOLUTION_12_B ADC resolution 12 bits

LL_ADC_RESOLUTION_10_B ADC resolution 10 bits

LL_ADC_RESOLUTION_8B ADC resolution 8 bits

LL_ADC_RESOLUTION_6B ADC resolution 6 bits

ADC helper macro

__LL_ADC_CHANNEL_TO_ **Description:****DECIMAL_NB**

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

Parameters:

- **_CHANNEL_**: This parameter can be one of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__LL_ADC_DECIMAL_NB_TO_CHANNEL **Description:**

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

Parameters:

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

Return value:

- Returned: value can be one of the following values:

- LL_ADC_CHANNEL_0
- LL_ADC_CHANNEL_1
- LL_ADC_CHANNEL_2
- LL_ADC_CHANNEL_3
- LL_ADC_CHANNEL_4
- LL_ADC_CHANNEL_5
- LL_ADC_CHANNEL_6
- LL_ADC_CHANNEL_7
- LL_ADC_CHANNEL_8
- LL_ADC_CHANNEL_9
- LL_ADC_CHANNEL_10
- LL_ADC_CHANNEL_11
- LL_ADC_CHANNEL_12
- LL_ADC_CHANNEL_13
- LL_ADC_CHANNEL_14
- LL_ADC_CHANNEL_15
- LL_ADC_CHANNEL_16
- LL_ADC_CHANNEL_17
- LL_ADC_CHANNEL_18 (1)
- LL_ADC_CHANNEL_VREFINT (2)
- LL_ADC_CHANNEL_TEMPSENSOR (2)
- LL_ADC_CHANNEL_VBAT (1)(2)

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

LL_ADC_IS_CHANNEL_I Description:

INTERNAL

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

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

Return value:

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

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

LL_ADC_CHANNEL_INT **Description:****INTERNAL_TO_EXTERNAL**

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- _CHANNEL_: This parameter can be one of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

Return value:

- Returned: value can be one of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17

- LL_ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_I **Description:****INTERNAL_AVAILABLE**

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

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
 - LL_ADC_CHANNEL_VREFINT
 - LL_ADC_CHANNEL_TEMPSENSOR
 - LL_ADC_CHANNEL_VBAT (1)

Return value:

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

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_ANALOGWD_C Description:**HANNEL_GROUP**

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- **__CHANNEL__**: This parameter can be one of the following values:
 - LL_ADC_CHANNEL_0
 - LL_ADC_CHANNEL_1
 - LL_ADC_CHANNEL_2
 - LL_ADC_CHANNEL_3
 - LL_ADC_CHANNEL_4
 - LL_ADC_CHANNEL_5
 - LL_ADC_CHANNEL_6
 - LL_ADC_CHANNEL_7
 - LL_ADC_CHANNEL_8
 - LL_ADC_CHANNEL_9
 - LL_ADC_CHANNEL_10
 - LL_ADC_CHANNEL_11
 - LL_ADC_CHANNEL_12
 - LL_ADC_CHANNEL_13
 - LL_ADC_CHANNEL_14
 - LL_ADC_CHANNEL_15
 - LL_ADC_CHANNEL_16
 - LL_ADC_CHANNEL_17
 - LL_ADC_CHANNEL_18 (1)
 - LL_ADC_CHANNEL_VREFINT (2)
 - LL_ADC_CHANNEL_TEMPSENSOR (2)
 - LL_ADC_CHANNEL_VBAT (1)(2)
- **__GROUP__**: This parameter can be one of the following values:
 - LL_ADC_GROUP_REGULAR

Return value:

- Returned: value can be one of the following values:
 - LL_ADC_AWD_DISABLE
 - LL_ADC_AWD_ALL_CHANNELS_REG
 - LL_ADC_AWD_CHANNEL_0_REG
 - LL_ADC_AWD_CHANNEL_1_REG
 - LL_ADC_AWD_CHANNEL_2_REG
 - LL_ADC_AWD_CHANNEL_3_REG
 - LL_ADC_AWD_CHANNEL_4_REG
 - LL_ADC_AWD_CHANNEL_5_REG
 - LL_ADC_AWD_CHANNEL_6_REG
 - LL_ADC_AWD_CHANNEL_7_REG
 - LL_ADC_AWD_CHANNEL_8_REG
 - LL_ADC_AWD_CHANNEL_9_REG
 - LL_ADC_AWD_CHANNEL_10_REG
 - LL_ADC_AWD_CHANNEL_11_REG
 - LL_ADC_AWD_CHANNEL_12_REG
 - LL_ADC_AWD_CHANNEL_13_REG
 - LL_ADC_AWD_CHANNEL_14_REG
 - LL_ADC_AWD_CHANNEL_15_REG

- LL_ADC_AWD_CHANNEL_16_REG
- LL_ADC_AWD_CHANNEL_17_REG
- LL_ADC_AWD_CHANNEL_18_REG (1)
- LL_ADC_AWD_CH_VREFINT_REG
- LL_ADC_AWD_CH_TEMPSENSOR_REG
- LL_ADC_AWD_CH_VBAT_REG (1)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels(). Example:
`LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1,
 __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4,
 LL_ADC_GROUP_REGULAR))`

__LL_ADC_ANALOGWD_S Description:

ET_THRESHOLD_RESOLUTION • Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

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

Return value:

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

Notes:

- To be used with function LL_ADC_ConfigAnalogWDThresholds() or LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (<ADCx param>,
`__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8
B, <threshold_value_8_bits>);`)

LL_ADC_ANALOGWD_G Description:

- ET_THRESHOLD_RESOLUTION**
- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- ADC_RESOLUTION: This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B
- AWD_THRESHOLD_12_BITS: Value between Min_Data=0x000 and Max_Data=0xFFFF

Return value:

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

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): <threshold_value_6_bits> =
LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

LL_ADC_ANALOGWD_T Description:

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

Parameters:

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

Return value:

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

Notes:

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

LL_ADC_COMMON_INST Description:

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

Parameters:

- ADCx: ADC instance

Return value:

- ADC: common register instance

Notes:

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

LL_ADC_IS_ENABLED_ **Description:****ALL_COMMON_INSTANCE**

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

Parameters:

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

Return value:

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

Notes:

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

LL_ADC_DIGITAL_SCAL **Description:****E**

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

Parameters:

- ADC_RESOLUTION_: This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

Return value:

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

Notes:

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

__LL_ADC_CONVERT_DAT **Description:****A_RESOLUTION**

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

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_T **Description:****O_VOLTAGE**

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

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

Return value:

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

Notes:

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

LL_ADC_CALC_VREFAN Description:**ALOG_VOLTAGE**

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

Parameters:

- VREFINT_ADC_DATA: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- ADC_RESOLUTION: This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

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

LL_ADC_CALC_TEMPER Description:

ATURE

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

Parameters:

- VREFANALOG_VOLTAGE: Analog reference voltage (unit: mV)
- TEMPSENSOR_ADC_DATA: ADC conversion data of internal temperature sensor (unit: digital value).
- ADC_RESOLUTION: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

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

LL_ADC_CALC_TEMPER Description:**ATURE_TYP_PARAMS**

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

Parameters:

- TEMPSENSOR_TYP_AVGSLOPE: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32F0, refer to device datasheet parameter "Avg_Slope".
- TEMPSENSOR_TYP_CALX_V: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32F0, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- TEMPSENSOR_CALX_TEMP: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- VREFANALOG_VOLTAGE: Analog voltage reference (Vref+) voltage (unit: mV)
- TEMPSENSOR_ADC_DATA: ADC conversion data of internal temperature sensor (unit: digital value).
- ADC_RESOLUTION: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
 - LL_ADC_RESOLUTION_12B
 - LL_ADC_RESOLUTION_10B
 - LL_ADC_RESOLUTION_8B
 - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

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

Common write and read registers Macros**LL_ADC_WriteReg****Description:**

- Write a value in ADC register.

Parameters:

- INSTANCE: ADC Instance
- REG: Register to be written
- VALUE: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg**Description:**

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

51 LL BUS Generic Driver

51.1 BUS Firmware driver API description

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

51.1.1 Detailed description of functions

LL_AHB1_GRP1_EnableClock

Function name `__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periph)`

Function description Enable AHB1 peripherals clock.

- Parameters**
- **Periph:** This parameter can be a combination of the following values:
 - `LL_AHB1_GRP1_PERIPH_DMA1`
 - `LL_AHB1_GRP1_PERIPH_DMA2 (*)`
 - `LL_AHB1_GRP1_PERIPH_SRAM`
 - `LL_AHB1_GRP1_PERIPH_FLASH`
 - `LL_AHB1_GRP1_PERIPH_CRC`
 - `LL_AHB1_GRP1_PERIPH_GPIOA`
 - `LL_AHB1_GRP1_PERIPH_GPIOB`
 - `LL_AHB1_GRP1_PERIPH_GPIOC`
 - `LL_AHB1_GRP1_PERIPH_GPIOD (*)`
 - `LL_AHB1_GRP1_PERIPH_GPIOE (*)`
 - `LL_AHB1_GRP1_PERIPH_GPIOF`
 - `LL_AHB1_GRP1_PERIPH_TSC (*)`

(*) value not defined in all devices.

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- `AHBENR DMA1EN LL_AHB1_GRP1_EnableClock`
 - `AHBENR DMA2EN LL_AHB1_GRP1_EnableClock`
 - `AHBENR SRAMEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR FLITFEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR CRCEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR GPIOAEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR GPIOBEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR GPIOCEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR GPIODEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR GPIOEEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR GPIOFEN LL_AHB1_GRP1_EnableClock`
 - `AHBENR TSCEN LL_AHB1_GRP1_EnableClock`

LL_AHB1_GRP1_IsEnabledClock

Function name `__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periph)`

Function description Check if AHB1 peripheral clock is enabled or not.

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_AHB1_GRP1_PERIPH_DMA1
 - LL_AHB1_GRP1_PERIPH_DMA2 (*)
 - LL_AHB1_GRP1_PERIPH_SRAM
 - LL_AHB1_GRP1_PERIPH_FLASH
 - LL_AHB1_GRP1_PERIPH_CRC
 - LL_AHB1_GRP1_PERIPH_GPIOA
 - LL_AHB1_GRP1_PERIPH_GPIOB
 - LL_AHB1_GRP1_PERIPH_GPIOC
 - LL_AHB1_GRP1_PERIPH_GPIOD (*)
 - LL_AHB1_GRP1_PERIPH_GPIOE (*)
 - LL_AHB1_GRP1_PERIPH_GPIOF
 - LL_AHB1_GRP1_PERIPH_TSC (*)
- (*) value not defined in all devices.

Return values

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

Reference Manual to LL API cross reference:

- AHBENR DMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHBENR DMA2EN LL_AHB1_GRP1_IsEnabledClock
- AHBENR SRAMEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR FLITFEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR CRCEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR GPIOAEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR GPIOBEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR GPIOCEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR GPIODEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR GPIOEEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR GPIOFEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR TSCEN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock**Function name**

__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periph)

Function description

Disable AHB1 peripherals clock.

- Parameters**
- **Periph:** This parameter can be a combination of the following values:
 - LL_AHB1_GRP1_PERIPH_DMA1
 - LL_AHB1_GRP1_PERIPH_DMA2 (*)
 - LL_AHB1_GRP1_PERIPH_SRAM
 - LL_AHB1_GRP1_PERIPH_FLASH
 - LL_AHB1_GRP1_PERIPH_CRC
 - LL_AHB1_GRP1_PERIPH_GPIOA
 - LL_AHB1_GRP1_PERIPH_GPIOB
 - LL_AHB1_GRP1_PERIPH_GPIOC
 - LL_AHB1_GRP1_PERIPH_GPIOD (*)
 - LL_AHB1_GRP1_PERIPH_GPIOE (*)
 - LL_AHB1_GRP1_PERIPH_GPIOF
 - LL_AHB1_GRP1_PERIPH_TSC (*)
- (*) value not defined in all devices.

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- AHBENR DMA1EN LL_AHB1_GRP1_DisableClock
 - AHBENR DMA2EN LL_AHB1_GRP1_DisableClock
 - AHBENR SRAMEN LL_AHB1_GRP1_DisableClock
 - AHBENR FLITFEN LL_AHB1_GRP1_DisableClock
 - AHBENR CRCEN LL_AHB1_GRP1_DisableClock
 - AHBENR GPIOAEN LL_AHB1_GRP1_DisableClock
 - AHBENR GPIOBEN LL_AHB1_GRP1_DisableClock
 - AHBENR GPIOCEN LL_AHB1_GRP1_DisableClock
 - AHBENR GPIODEN LL_AHB1_GRP1_DisableClock
 - AHBENR GPIOEEN LL_AHB1_GRP1_DisableClock
 - AHBENR GPIOFEN LL_AHB1_GRP1_DisableClock
 - AHBENR TSCEN LL_AHB1_GRP1_DisableClock

LL_AHB1_GRP1_ForceReset

Function name `__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periph)`

Function description Force AHB1 peripherals reset.

- Parameters**
- **Periph:** This parameter can be a combination of the following values:
 - LL_AHB1_GRP1_PERIPH_ALL
 - LL_AHB1_GRP1_PERIPH_GPIOA
 - LL_AHB1_GRP1_PERIPH_GPIOB
 - LL_AHB1_GRP1_PERIPH_GPIOC
 - LL_AHB1_GRP1_PERIPH_GPIOD (*)
 - LL_AHB1_GRP1_PERIPH_GPIOE (*)
 - LL_AHB1_GRP1_PERIPH_GPIOF
 - LL_AHB1_GRP1_PERIPH_TSC (*)
- (*) value not defined in all devices.

- Return values**
- **None:**

Reference Manual to LL API cross reference:

- AHBRSTR GPIOARST LL_AHB1_GRP1_ForceReset
- AHBRSTR GPIOBRST LL_AHB1_GRP1_ForceReset
- AHBRSTR GPIOCRST LL_AHB1_GRP1_ForceReset
- AHBRSTR GPIODRST LL_AHB1_GRP1_ForceReset
- AHBRSTR GPIOERST LL_AHB1_GRP1_ForceReset
- AHBRSTR GPIOFRST LL_AHB1_GRP1_ForceReset
- AHBRSTR TSCRST LL_AHB1_GRP1_ForceReset

LL_AHB1_GRP1_ReleaseReset**Function name** `__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periph)`**Function description** Release AHB1 peripherals reset.**Parameters**

- **Periph:** This parameter can be a combination of the following values:
 - LL_AHB1_GRP1_PERIPH_ALL
 - LL_AHB1_GRP1_PERIPH_GPIOA
 - LL_AHB1_GRP1_PERIPH_GPIOB
 - LL_AHB1_GRP1_PERIPH_GPIOC
 - LL_AHB1_GRP1_PERIPH_GPIOD (*)
 - LL_AHB1_GRP1_PERIPH_GPIOE (*)
 - LL_AHB1_GRP1_PERIPH_GPIOF
 - LL_AHB1_GRP1_PERIPH_TSC (*)
- (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHBRSTR GPIOARST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR GPIOBRST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR GPIOCRST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR GPIODRST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR GPIOERST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR GPIOFRST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR TSCRST LL_AHB1_GRP1_ReleaseReset

LL_APB1_GRP1_EnableClock**Function name** `__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periph)`**Function description** Enable APB1 peripherals clock (available in register 1).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP1_PERIPH_TIM2 (*)
 - LL_APB1_GRP1_PERIPH_TIM3
 - LL_APB1_GRP1_PERIPH_TIM6 (*)
 - LL_APB1_GRP1_PERIPH_TIM7 (*)
 - LL_APB1_GRP1_PERIPH_TIM14
 - LL_APB1_GRP1_PERIPH_WWDG
 - LL_APB1_GRP1_PERIPH_SPI2 (*)
 - LL_APB1_GRP1_PERIPH_USART2 (*)
 - LL_APB1_GRP1_PERIPH_USART3 (*)
 - LL_APB1_GRP1_PERIPH_USART4 (*)
 - LL_APB1_GRP1_PERIPH_USART5 (*)
 - LL_APB1_GRP1_PERIPH_I2C1
 - LL_APB1_GRP1_PERIPH_I2C2 (*)
 - LL_APB1_GRP1_PERIPH_USB (*)
 - LL_APB1_GRP1_PERIPH_CAN (*)
 - LL_APB1_GRP1_PERIPH_CRS (*)
 - LL_APB1_GRP1_PERIPH_PWR
 - LL_APB1_GRP1_PERIPH_DAC1 (*)
 - LL_APB1_GRP1_PERIPH_CEC (*)
- (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR TIM2EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM3EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM6EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM7EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM14EN LL_APB1_GRP1_EnableClock
- APB1ENR WWDGEN LL_APB1_GRP1_EnableClock
- APB1ENR SPI2EN LL_APB1_GRP1_EnableClock
- APB1ENR USART2EN LL_APB1_GRP1_EnableClock
- APB1ENR USART3EN LL_APB1_GRP1_EnableClock
- APB1ENR USART4EN LL_APB1_GRP1_EnableClock
- APB1ENR USART5EN LL_APB1_GRP1_EnableClock
- APB1ENR I2C1EN LL_APB1_GRP1_EnableClock
- APB1ENR I2C2EN LL_APB1_GRP1_EnableClock
- APB1ENR USBEN LL_APB1_GRP1_EnableClock
- APB1ENR CANEN LL_APB1_GRP1_EnableClock
- APB1ENR CRSEN LL_APB1_GRP1_EnableClock
- APB1ENR PWREN LL_APB1_GRP1_EnableClock
- APB1ENR DACEN LL_APB1_GRP1_EnableClock
- APB1ENR CECEN LL_APB1_GRP1_EnableClock

LL_APB1_GRP1_IsEnabledClock**Function name**

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periph)

Function description

Check if APB1 peripheral clock is enabled or not (available in register 1).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP1_PERIPH_TIM2 (*)
 - LL_APB1_GRP1_PERIPH_TIM3
 - LL_APB1_GRP1_PERIPH_TIM6 (*)
 - LL_APB1_GRP1_PERIPH_TIM7 (*)
 - LL_APB1_GRP1_PERIPH_TIM14
 - LL_APB1_GRP1_PERIPH_WWDG
 - LL_APB1_GRP1_PERIPH_SPI2 (*)
 - LL_APB1_GRP1_PERIPH_USART2 (*)
 - LL_APB1_GRP1_PERIPH_USART3 (*)
 - LL_APB1_GRP1_PERIPH_USART4 (*)
 - LL_APB1_GRP1_PERIPH_USART5 (*)
 - LL_APB1_GRP1_PERIPH_I2C1
 - LL_APB1_GRP1_PERIPH_I2C2 (*)
 - LL_APB1_GRP1_PERIPH_USB (*)
 - LL_APB1_GRP1_PERIPH_CAN (*)
 - LL_APB1_GRP1_PERIPH_CRS (*)
 - LL_APB1_GRP1_PERIPH_PWR
 - LL_APB1_GRP1_PERIPH_DAC1 (*)
 - LL_APB1_GRP1_PERIPH_CEC (*)
- (*) value not defined in all devices.

Return values

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

Reference Manual to LL API cross reference:

- APB1ENR TIM2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM6EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM7EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM14EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR WWDGEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR SPI2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USART2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USART3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USART4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USART5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR I2C1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR I2C2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USBEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR CANEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR CRSEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR PWREN LL_APB1_GRP1_IsEnabledClock
- APB1ENR DACEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR CECEN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP1_DisableClock**Function name**

`__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periph)`

Function description

Disable APB1 peripherals clock (available in register 1).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP1_PERIPH_TIM2 (*)
 - LL_APB1_GRP1_PERIPH_TIM3
 - LL_APB1_GRP1_PERIPH_TIM6 (*)
 - LL_APB1_GRP1_PERIPH_TIM7 (*)
 - LL_APB1_GRP1_PERIPH_TIM14
 - LL_APB1_GRP1_PERIPH_WWDG
 - LL_APB1_GRP1_PERIPH_SPI2 (*)
 - LL_APB1_GRP1_PERIPH_USART2 (*)
 - LL_APB1_GRP1_PERIPH_USART3 (*)
 - LL_APB1_GRP1_PERIPH_USART4 (*)
 - LL_APB1_GRP1_PERIPH_USART5 (*)
 - LL_APB1_GRP1_PERIPH_I2C1
 - LL_APB1_GRP1_PERIPH_I2C2 (*)
 - LL_APB1_GRP1_PERIPH_USB (*)
 - LL_APB1_GRP1_PERIPH_CAN (*)
 - LL_APB1_GRP1_PERIPH_CRS (*)
 - LL_APB1_GRP1_PERIPH_PWR
 - LL_APB1_GRP1_PERIPH_DAC1 (*)
 - LL_APB1_GRP1_PERIPH_CEC (*)
- (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR TIM2EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM3EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM6EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM7EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM14EN LL_APB1_GRP1_DisableClock
- APB1ENR WWDGEN LL_APB1_GRP1_DisableClock
- APB1ENR SPI2EN LL_APB1_GRP1_DisableClock
- APB1ENR USART2EN LL_APB1_GRP1_DisableClock
- APB1ENR USART3EN LL_APB1_GRP1_DisableClock
- APB1ENR USART4EN LL_APB1_GRP1_DisableClock
- APB1ENR USART5EN LL_APB1_GRP1_DisableClock
- APB1ENR I2C1EN LL_APB1_GRP1_DisableClock
- APB1ENR I2C2EN LL_APB1_GRP1_DisableClock
- APB1ENR USBEN LL_APB1_GRP1_DisableClock
- APB1ENR CANEN LL_APB1_GRP1_DisableClock
- APB1ENR CRSEN LL_APB1_GRP1_DisableClock
- APB1ENR PWREN LL_APB1_GRP1_DisableClock
- APB1ENR DACEN LL_APB1_GRP1_DisableClock
- APB1ENR CECEN LL_APB1_GRP1_DisableClock

LL_APB1_GRP1_ForceReset**Function name**

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periph)

Function description

Force APB1 peripherals reset (available in register 1).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP1_PERIPH_ALL
 - LL_APB1_GRP1_PERIPH_TIM2 (*)
 - LL_APB1_GRP1_PERIPH_TIM3
 - LL_APB1_GRP1_PERIPH_TIM6 (*)
 - LL_APB1_GRP1_PERIPH_TIM7 (*)
 - LL_APB1_GRP1_PERIPH_TIM14
 - LL_APB1_GRP1_PERIPH_WWDG
 - LL_APB1_GRP1_PERIPH_SPI2 (*)
 - LL_APB1_GRP1_PERIPH_USART2 (*)
 - LL_APB1_GRP1_PERIPH_USART3 (*)
 - LL_APB1_GRP1_PERIPH_USART4 (*)
 - LL_APB1_GRP1_PERIPH_USART5 (*)
 - LL_APB1_GRP1_PERIPH_I2C1
 - LL_APB1_GRP1_PERIPH_I2C2 (*)
 - LL_APB1_GRP1_PERIPH_USB (*)
 - LL_APB1_GRP1_PERIPH_CAN (*)
 - LL_APB1_GRP1_PERIPH_CRS (*)
 - LL_APB1_GRP1_PERIPH_PWR
 - LL_APB1_GRP1_PERIPH_DAC1 (*)
 - LL_APB1_GRP1_PERIPH_CEC (*)

(*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR TIM2RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM3RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM6RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM7RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM14RST LL_APB1_GRP1_ForceReset
- APB1RSTR WWDGRST LL_APB1_GRP1_ForceReset
- APB1RSTR SPI2RST LL_APB1_GRP1_ForceReset
- APB1RSTR USART2RST LL_APB1_GRP1_ForceReset
- APB1RSTR USART3RST LL_APB1_GRP1_ForceReset
- APB1RSTR USART4RST LL_APB1_GRP1_ForceReset
- APB1RSTR USART5RST LL_APB1_GRP1_ForceReset
- APB1RSTR I2C1RST LL_APB1_GRP1_ForceReset
- APB1RSTR I2C2RST LL_APB1_GRP1_ForceReset
- APB1RSTR USBRST LL_APB1_GRP1_ForceReset
- APB1RSTR CANRST LL_APB1_GRP1_ForceReset
- APB1RSTR CRSRST LL_APB1_GRP1_ForceReset
- APB1RSTR PWRRST LL_APB1_GRP1_ForceReset
- APB1RSTR DACRST LL_APB1_GRP1_ForceReset
- APB1RSTR CECRST LL_APB1_GRP1_ForceReset

LL_APB1_GRP1_ReleaseReset**Function name**

_STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periph)

Function description Release APB1 peripherals reset (available in register 1).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP1_PERIPH_ALL
 - LL_APB1_GRP1_PERIPH_TIM2 (*)
 - LL_APB1_GRP1_PERIPH_TIM3
 - LL_APB1_GRP1_PERIPH_TIM6 (*)
 - LL_APB1_GRP1_PERIPH_TIM7 (*)
 - LL_APB1_GRP1_PERIPH_TIM14
 - LL_APB1_GRP1_PERIPH_WWDG
 - LL_APB1_GRP1_PERIPH_SPI2 (*)
 - LL_APB1_GRP1_PERIPH_USART2 (*)
 - LL_APB1_GRP1_PERIPH_USART3 (*)
 - LL_APB1_GRP1_PERIPH_USART4 (*)
 - LL_APB1_GRP1_PERIPH_USART5 (*)
 - LL_APB1_GRP1_PERIPH_I2C1
 - LL_APB1_GRP1_PERIPH_I2C2 (*)
 - LL_APB1_GRP1_PERIPH_USB (*)
 - LL_APB1_GRP1_PERIPH_CAN (*)
 - LL_APB1_GRP1_PERIPH_CRS (*)
 - LL_APB1_GRP1_PERIPH_PWR
 - LL_APB1_GRP1_PERIPH_DAC1 (*)
 - LL_APB1_GRP1_PERIPH_CEC (*)
- (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR TIM2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM6RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM7RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM14RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR WWDGRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR SPI2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USART2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USART3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USART4RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USART5RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR I2C1RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR I2C2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USBRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR CANRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR CRSRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR PWRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR DACRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR CECRST LL_APB1_GRP1_ReleaseReset

LL_APB1_GRP2_EnableClock

Function name `__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periph)`

Function description Enable APB1 peripherals clock (available in register 2).

- Parameters**
- **Periph:** This parameter can be a combination of the following values:
 - `LL_APB1_GRP2_PERIPH_SYSCFG`
 - `LL_APB1_GRP2_PERIPH_ADC1`
 - `LL_APB1_GRP2_PERIPH_USART8 (*)`
 - `LL_APB1_GRP2_PERIPH_USART7 (*)`
 - `LL_APB1_GRP2_PERIPH_USART6 (*)`
 - `LL_APB1_GRP2_PERIPH_TIM1`
 - `LL_APB1_GRP2_PERIPH_SPI1`
 - `LL_APB1_GRP2_PERIPH_USART1`
 - `LL_APB1_GRP2_PERIPH_TIM15 (*)`
 - `LL_APB1_GRP2_PERIPH_TIM16`
 - `LL_APB1_GRP2_PERIPH_TIM17`
 - `LL_APB1_GRP2_PERIPH_DBGMCU`
- (*) value not defined in all devices.

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- `APB2ENR SYSCFGEN LL_APB1_GRP2_EnableClock`
 - `APB2ENR ADC1EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR USART8EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR USART7EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR USART6EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR TIM1EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR SPI1EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR USART1EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR TIM15EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR TIM16EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR TIM17EN LL_APB1_GRP2_EnableClock`
 - `APB2ENR DBGMCUEN LL_APB1_GRP2_EnableClock`

LL_APB1_GRP2_IsEnabledClock

Function name `__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periph)`

Function description Check if APB1 peripheral clock is enabled or not (available in register 2).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP2_PERIPH_SYSCFG
 - LL_APB1_GRP2_PERIPH_ADC1
 - LL_APB1_GRP2_PERIPH_USART8 (*)
 - LL_APB1_GRP2_PERIPH_USART7 (*)
 - LL_APB1_GRP2_PERIPH_USART6 (*)
 - LL_APB1_GRP2_PERIPH_TIM1
 - LL_APB1_GRP2_PERIPH_SPI1
 - LL_APB1_GRP2_PERIPH_USART1
 - LL_APB1_GRP2_PERIPH_TIM15 (*)
 - LL_APB1_GRP2_PERIPH_TIM16
 - LL_APB1_GRP2_PERIPH_TIM17
 - LL_APB1_GRP2_PERIPH_DBGMCU
- (*) value not defined in all devices.

Return values

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

Reference Manual to LL API cross reference:

- APB2ENR SYSCFGEN LL_APB1_GRP2_IsEnabledClock
- APB2ENR ADC1EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR USART8EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR USART7EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR USART6EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR TIM1EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR SPI1EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR USART1EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR TIM15EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR TIM16EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR TIM17EN LL_APB1_GRP2_IsEnabledClock
- APB2ENR DBGMCUEN LL_APB1_GRP2_IsEnabledClock

LL_APB1_GRP2_DisableClock**Function name**

_STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periph)

Function description

Disable APB1 peripherals clock (available in register 2).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP2_PERIPH_SYSCFG
 - LL_APB1_GRP2_PERIPH_ADC1
 - LL_APB1_GRP2_PERIPH_USART8 (*)
 - LL_APB1_GRP2_PERIPH_USART7 (*)
 - LL_APB1_GRP2_PERIPH_USART6 (*)
 - LL_APB1_GRP2_PERIPH_TIM1
 - LL_APB1_GRP2_PERIPH_SPI1
 - LL_APB1_GRP2_PERIPH_USART1
 - LL_APB1_GRP2_PERIPH_TIM15 (*)
 - LL_APB1_GRP2_PERIPH_TIM16
 - LL_APB1_GRP2_PERIPH_TIM17
 - LL_APB1_GRP2_PERIPH_DBGMCU
- (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR SYSCFGEN LL_APB1_GRP2_DisableClock
- APB2ENR ADC1EN LL_APB1_GRP2_DisableClock
- APB2ENR USART8EN LL_APB1_GRP2_DisableClock
- APB2ENR USART7EN LL_APB1_GRP2_DisableClock
- APB2ENR USART6EN LL_APB1_GRP2_DisableClock
- APB2ENR TIM1EN LL_APB1_GRP2_DisableClock
- APB2ENR SPI1EN LL_APB1_GRP2_DisableClock
- APB2ENR USART1EN LL_APB1_GRP2_DisableClock
- APB2ENR TIM15EN LL_APB1_GRP2_DisableClock
- APB2ENR TIM16EN LL_APB1_GRP2_DisableClock
- APB2ENR TIM17EN LL_APB1_GRP2_DisableClock
- APB2ENR DBGMCUEN LL_APB1_GRP2_DisableClock

LL_APB1_GRP2_ForceReset**Function name**

_STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periph)

Function description

Force APB1 peripherals reset (available in register 2).

Parameters

- **Periph:** This parameter can be a combination of the following values:
 - LL_APB1_GRP2_PERIPH_ALL
 - LL_APB1_GRP2_PERIPH_SYSCFG
 - LL_APB1_GRP2_PERIPH_ADC1
 - LL_APB1_GRP2_PERIPH_USART8 (*)
 - LL_APB1_GRP2_PERIPH_USART7 (*)
 - LL_APB1_GRP2_PERIPH_USART6 (*)
 - LL_APB1_GRP2_PERIPH_TIM1
 - LL_APB1_GRP2_PERIPH_SPI1
 - LL_APB1_GRP2_PERIPH_USART1
 - LL_APB1_GRP2_PERIPH_TIM15 (*)
 - LL_APB1_GRP2_PERIPH_TIM16
 - LL_APB1_GRP2_PERIPH_TIM17
 - LL_APB1_GRP2_PERIPH_DBGMCU
- (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR SYSCFGRST LL_APB1_GRP2_ForceReset
- APB2RSTR ADC1RST LL_APB1_GRP2_ForceReset
- APB2RSTR USART8RST LL_APB1_GRP2_ForceReset
- APB2RSTR USART7RST LL_APB1_GRP2_ForceReset
- APB2RSTR USART6RST LL_APB1_GRP2_ForceReset
- APB2RSTR TIM1RST LL_APB1_GRP2_ForceReset
- APB2RSTR SPI1RST LL_APB1_GRP2_ForceReset
- APB2RSTR USART1RST LL_APB1_GRP2_ForceReset
- APB2RSTR TIM15RST LL_APB1_GRP2_ForceReset
- APB2RSTR TIM16RST LL_APB1_GRP2_ForceReset
- APB2RSTR TIM17RST LL_APB1_GRP2_ForceReset
- APB2RSTR DBGMURST LL_APB1_GRP2_ForceReset

LL_APB1_GRP2_ReleaseReset**Function name**

`__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periph)`

Function description

Release APB1 peripherals reset (available in register 2).

Parameters	<ul style="list-style-type: none">• Periph: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_APB1_GRP2_PERIPH_ALL– LL_APB1_GRP2_PERIPH_SYSCFG– LL_APB1_GRP2_PERIPH_ADC1– LL_APB1_GRP2_PERIPH_USART8 (*)– LL_APB1_GRP2_PERIPH_USART7 (*)– LL_APB1_GRP2_PERIPH_USART6 (*)– LL_APB1_GRP2_PERIPH_TIM1– LL_APB1_GRP2_PERIPH_SPI1– LL_APB1_GRP2_PERIPH_USART1– LL_APB1_GRP2_PERIPH_TIM15 (*)– LL_APB1_GRP2_PERIPH_TIM16– LL_APB1_GRP2_PERIPH_TIM17– LL_APB1_GRP2_PERIPH_DBGMCU
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• APB2RSTR SYSCFGRST LL_APB1_GRP2_ReleaseReset• APB2RSTR ADC1RST LL_APB1_GRP2_ReleaseReset• APB2RSTR USART8RST LL_APB1_GRP2_ReleaseReset• APB2RSTR USART7RST LL_APB1_GRP2_ReleaseReset• APB2RSTR USART6RST LL_APB1_GRP2_ReleaseReset• APB2RSTR TIM1RST LL_APB1_GRP2_ReleaseReset• APB2RSTR SPI1RST LL_APB1_GRP2_ReleaseReset• APB2RSTR USART1RST LL_APB1_GRP2_ReleaseReset• APB2RSTR TIM15RST LL_APB1_GRP2_ReleaseReset• APB2RSTR TIM16RST LL_APB1_GRP2_ReleaseReset• APB2RSTR TIM17RST LL_APB1_GRP2_ReleaseReset• APB2RSTR DBGMURST LL_APB1_GRP2_ReleaseReset

51.2 BUS Firmware driver defines

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

51.2.1 BUS

BUS

AHB1_GRP1_PERIPH

`LL_AHB1_GRP1_PERIPH_
ALL`

`LL_AHB1_GRP1_PERIPH_
DMA1`

`LL_AHB1_GRP1_PERIPH_
DMA2`

`LL_AHB1_GRP1_PERIPH_
SRAM`

LL_AHB1_GRP1_PERIPH_
FLASH

LL_AHB1_GRP1_PERIPH_
CRC

LL_AHB1_GRP1_PERIPH_
GPIOA

LL_AHB1_GRP1_PERIPH_
GPIOB

LL_AHB1_GRP1_PERIPH_
GPIOC

LL_AHB1_GRP1_PERIPH_
GPIOD

LL_AHB1_GRP1_PERIPH_
GPIOE

LL_AHB1_GRP1_PERIPH_
GPIOF

LL_AHB1_GRP1_PERIPH_
TSC

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_
ALL

LL_APB1_GRP1_PERIPH_
TIM2

LL_APB1_GRP1_PERIPH_
TIM3

LL_APB1_GRP1_PERIPH_
TIM6

LL_APB1_GRP1_PERIPH_
TIM7

LL_APB1_GRP1_PERIPH_
TIM14

LL_APB1_GRP1_PERIPH_
WWDG

LL_APB1_GRP1_PERIPH_
SPI2

LL_APB1_GRP1_PERIPH_
USART2

LL_APB1_GRP1_PERIPH_
USART3

LL_APB1_GRP1_PERIPH_
USART4

LL_APB1_GRP1_PERIPH_
USART5

LL_APB1_GRP1_PERIPH_I
2C1

LL_APB1_GRP1_PERIPH_I
2C2

LL_APB1_GRP1_PERIPH_
CAN

LL_APB1_GRP1_PERIPH_
CRS

LL_APB1_GRP1_PERIPH_
PWR

LL_APB1_GRP1_PERIPH_
DAC1

LL_APB1_GRP1_PERIPH_
CEC

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_
ALL

LL_APB1_GRP2_PERIPH_
SYSCFG

LL_APB1_GRP2_PERIPH_
ADC1

LL_APB1_GRP2_PERIPH_
USART8

LL_APB1_GRP2_PERIPH_
USART7

LL_APB1_GRP2_PERIPH_
USART6

LL_APB1_GRP2_PERIPH_
TIM1

LL_APB1_GRP2_PERIPH_
SPI1

LL_APB1_GRP2_PERIPH_
USART1

LL_APB1_GRP2_PERIPH_
TIM15

LL_APB1_GRP2_PERIPH_
TIM16

LL_APB1_GRP2_PERIPH_
TIM17

LL_APB1_GRP2_PERIPH_
DBGMCU

52 LL COMP Generic Driver

52.1 COMP Firmware driver registers structures

52.1.1 LL_COMP_InitTypeDef

`LL_COMP_InitTypeDef` is defined in the `stm32f0xx_ll_comp.h`

Data Fields

- `uint32_t PowerMode`
- `uint32_t InputPlus`
- `uint32_t InputMinus`
- `uint32_t InputHysteresis`
- `uint32_t OutputSelection`
- `uint32_t OutputPolarity`

Field Documentation

- **`uint32_t LL_COMP_InitTypeDef::PowerMode`**

Set comparator operating mode to adjust power and speed. This parameter can be a value of `COMP_LL_EC_POWERMODE`This feature can be modified afterwards using unitary function `LL_COMP_SetPowerMode()`.

- **`uint32_t LL_COMP_InitTypeDef::InputPlus`**

Set comparator input plus (non-inverting input). This parameter can be a value of `COMP_LL_EC_INPUT_PLUS`This feature can be modified afterwards using unitary function `LL_COMP_SetInputPlus()`.

- **`uint32_t LL_COMP_InitTypeDef::InputMinus`**

Set comparator input minus (inverting input). This parameter can be a value of `COMP_LL_EC_INPUT_MINUS`This feature can be modified afterwards using unitary function `LL_COMP_SetInputMinus()`.

- **`uint32_t LL_COMP_InitTypeDef::InputHysteresis`**

Set comparator hysteresis mode of the input minus. This parameter can be a value of `COMP_LL_EC_INPUT_HYSTERESIS`This feature can be modified afterwards using unitary function `LL_COMP_SetInputHysteresis()`.

- **`uint32_t LL_COMP_InitTypeDef::OutputSelection`**

Set comparator output selection. This parameter can be a value of `COMP_LL_EC_OUTPUT_SELECTION`This feature can be modified afterwards using unitary function `LL_COMP_SetOutputSelection()`.

- **`uint32_t LL_COMP_InitTypeDef::OutputPolarity`**

Set comparator output polarity. This parameter can be a value of `COMP_LL_EC_OUTPUT_POLARITY`This feature can be modified afterwards using unitary function `LL_COMP_SetOutputPolarity()`.

52.2 COMP Firmware driver API description

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

52.2.1 Detailed description of functions

`LL_COMP_SetCommonWindowMode`

Function name `__STATIC_INLINE void LL_COMP_SetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON, uint32_t WindowMode)`

Function description Set window mode of a pair of comparators instances (2 consecutive COMP instances odd and even COMP<x> and COMP<x+1>).

Parameters	<ul style="list-style-type: none">COMPxy_COMMON: Comparator common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_COMP_COMMON_INSTANCE()</code>)WindowMode: This parameter can be one of the following values:<ul style="list-style-type: none">- <code>LL_COMP_WINDOWMODE_DISABLE</code>- <code>LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON</code>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR WNDWEN <code>LL_COMP_SetCommonWindowMode</code>
	LL_COMP_GetCommonWindowMode
Function name	<code>__STATIC_INLINE uint32_t LL_COMP_GetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON)</code>
Function description	Get window mode of a pair of comparators instances (2 consecutive COMP instances odd and even COMP<x> and COMP<x+1>).
Parameters	<ul style="list-style-type: none">COMPxy_COMMON: Comparator common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_COMP_COMMON_INSTANCE()</code>)
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">- <code>LL_COMP_WINDOWMODE_DISABLE</code>- <code>LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR WNDWEN <code>LL_COMP_GetCommonWindowMode</code>
	LL_COMP_SetPowerMode
Function name	<code>__STATIC_INLINE void LL_COMP_SetPowerMode (COMP_TypeDef * COMPx, uint32_t PowerMode)</code>
Function description	Set comparator instance operating mode to adjust power and speed.
Parameters	<ul style="list-style-type: none">COMPx: Comparator instancePowerMode: This parameter can be one of the following values:<ul style="list-style-type: none">- <code>LL_COMP_POWERMODE_HIGHSPEED</code>- <code>LL_COMP_POWERMODE_MEDIUMSPEED</code>- <code>LL_COMP_POWERMODE_LOWPOWER</code>- <code>LL_COMP_POWERMODE_ULTRALOWPOWER</code>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1MODE <code>LL_COMP_SetPowerMode</code>COMP2MODE <code>LL_COMP_SetPowerMode</code>
	LL_COMP_GetPowerMode
Function name	<code>__STATIC_INLINE uint32_t LL_COMP_GetPowerMode (COMP_TypeDef * COMPx)</code>

Function description Get comparator instance operating mode to adjust power and speed.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
 - LL_COMP_POWERMODE_HIGHSPEED
 - LL_COMP_POWERMODE_MEDIUMSPEED
 - LL_COMP_POWERMODE_LOWPOWER
 - LL_COMP_POWERMODE_ULTRALOWPOWER

Reference Manual to LL API cross reference:

- CSR COMP1MODE LL_COMP_GetPowerMode
- COMP2MODE LL_COMP_GetPowerMode

LL_COMP_ConfigInputs

Function name `__STATIC_INLINE void LL_COMP_ConfigInputs (COMP_TypeDef * COMPx, uint32_t InputMinus, uint32_t InputPlus)`

Function description Set comparator inputs minus (inverting) and plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
 - LL_COMP_INPUT_MINUS_1_4VREFINT
 - LL_COMP_INPUT_MINUS_1_2VREFINT
 - LL_COMP_INPUT_MINUS_3_4VREFINT
 - LL_COMP_INPUT_MINUS_VREFINT
 - LL_COMP_INPUT_MINUS_DAC1_CH1
 - LL_COMP_INPUT_MINUS_DAC1_CH2
 - LL_COMP_INPUT_MINUS_IO1
- **InputPlus:** This parameter can be one of the following values:
 - LL_COMP_INPUT_PLUS_IO1
 - LL_COMP_INPUT_PLUS_DAC1_CH1 (1)

(1) Parameter available only on COMP instance: COMP1.

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR COMP1INSEL LL_COMP_ConfigInputs
- CSR COMP2INSEL LL_COMP_ConfigInputs
- CSR COMP1SW1 LL_COMP_ConfigInputs

LL_COMP_SetInputPlus

Function name `__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)`

Function description Set comparator input plus (non-inverting).

Parameters	<ul style="list-style-type: none">• COMPx: Comparator instance• InputPlus: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_COMP_INPUT_PLUS_IO1– LL_COMP_INPUT_PLUS_DAC1_CH1 (1)(1) Parameter available only on COMP instance: COMP1.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CSR COMP1INSEL LL_COMP_SetInputPlus• CSR COMP2INSEL LL_COMP_SetInputPlus

LL_COMP_GetInputPlus

Function name	<code>__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (COMP_TypeDef * COMPx)</code>
Function description	Get comparator input plus (non-inverting).
Parameters	<ul style="list-style-type: none">• COMPx: Comparator instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_COMP_INPUT_PLUS_IO1– LL_COMP_INPUT_PLUS_DAC1_CH1 (1)(1) Parameter available only on COMP instance: COMP1.
Notes	<ul style="list-style-type: none">• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CSR COMP1INSEL LL_COMP_GetInputPlus• CSR COMP2INSEL LL_COMP_GetInputPlus

LL_COMP_SetInputMinus

Function name	<code>__STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)</code>
Function description	Set comparator input minus (inverting).
Parameters	<ul style="list-style-type: none">• COMPx: Comparator instance• InputMinus: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_COMP_INPUT_MINUS_1_4VREFINT– LL_COMP_INPUT_MINUS_1_2VREFINT– LL_COMP_INPUT_MINUS_3_4VREFINT– LL_COMP_INPUT_MINUS_VREFINT– LL_COMP_INPUT_MINUS_DAC1_CH1– LL_COMP_INPUT_MINUS_DAC1_CH2– LL_COMP_INPUT_MINUS_IO1

- Return values**
- **None:**
- Notes**
- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- Reference Manual to LL API cross reference:**
- CSR COMP1SW1 LL_COMP_SetInputMinus

LL_COMP_GetInputMinus

- Function name**
- _STATIC_INLINE uint32_t LL_COMP_GetInputMinus (COMP_TypeDef * COMPx)**
- Function description**
- Get comparator input minus (inverting).
- Parameters**
- **COMPx:** Comparator instance
- Return values**
- **Returned:** value can be one of the following values:
 - LL_COMP_INPUT_MINUS_1_4VREFINT
 - LL_COMP_INPUT_MINUS_1_2VREFINT
 - LL_COMP_INPUT_MINUS_3_4VREFINT
 - LL_COMP_INPUT_MINUS_VREFINT
 - LL_COMP_INPUT_MINUS_DAC1_CH1
 - LL_COMP_INPUT_MINUS_DAC1_CH2
 - LL_COMP_INPUT_MINUS_IO1
- Notes**
- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- Reference Manual to LL API cross reference:**
- CSR COMP1SW1 LL_COMP_GetInputMinus

LL_COMP_SetInputHysteresis

- Function name**
- _STATIC_INLINE void LL_COMP_SetInputHysteresis (COMP_TypeDef * COMPx, uint32_t InputHysteresis)**
- Function description**
- Set comparator instance hysteresis mode of the input minus (inverting input).
- Parameters**
- **COMPx:** Comparator instance
 - **InputHysteresis:** This parameter can be one of the following values:
 - LL_COMP_HYSTERESIS_NONE
 - LL_COMP_HYSTERESIS_LOW
 - LL_COMP_HYSTERESIS_MEDIUM
 - LL_COMP_HYSTERESIS_HIGH
- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CSR COMP1HYST LL_COMP_SetInputHysteresis
 - COMP2HYST LL_COMP_SetInputHysteresis

LL_COMP_GetInputHysteresis

Function name	<code>__STATIC_INLINE uint32_t LL_COMP_GetInputHysteresis (COMP_TypeDef * COMPx)</code>
Function description	Get comparator instance hysteresis mode of the minus (inverting) input.
Parameters	<ul style="list-style-type: none">COMPx: Comparator instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_COMP_HYSTERESIS_NONELL_COMP_HYSTERESIS_LOWLL_COMP_HYSTERESIS_MEDIUMLL_COMP_HYSTERESIS_HIGH
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1HYST LL_COMP_GetInputHysteresisCOMP2HYST LL_COMP_GetInputHysteresis

LL_COMP_SetOutputSelection

Function name	<code>__STATIC_INLINE void LL_COMP_SetOutputSelection (COMP_TypeDef * COMPx, uint32_t OutputSelection)</code>
Function description	Set comparator output selection.
Parameters	<ul style="list-style-type: none">COMPx: Comparator instanceOutputSelection: This parameter can be one of the following values:<ul style="list-style-type: none">LL_COMP_OUTPUT_NONELL_COMP_OUTPUT_TIM1_BKIN (1)LL_COMP_OUTPUT_TIM1_IC1 (1)LL_COMP_OUTPUT_TIM1_OCCLR (1)LL_COMP_OUTPUT_TIM2_IC4 (1)LL_COMP_OUTPUT_TIM2_OCCLR (1)LL_COMP_OUTPUT_TIM3_IC1 (1)LL_COMP_OUTPUT_TIM3_OCCLR (1)
	(1) Parameter availability depending on timer availability on the selected device.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Availability of parameters of output selection to timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1OUTSEL LL_COMP_SetOutputSelectionCOMP2OUTSEL LL_COMP_SetOutputSelection

LL_COMP_GetOutputSelection

Function name	<code>__STATIC_INLINE uint32_t LL_COMP_GetOutputSelection (COMP_TypeDef * COMPx)</code>
Function description	Get comparator output selection.

Parameters	<ul style="list-style-type: none">• COMPx: Comparator instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_COMP_OUTPUT_NONE– LL_COMP_OUTPUT_TIM1_BKIN (1)– LL_COMP_OUTPUT_TIM1_IC1 (1)– LL_COMP_OUTPUT_TIM1_OCCCLR (1)– LL_COMP_OUTPUT_TIM2_IC4 (1)– LL_COMP_OUTPUT_TIM2_OCCCLR (1)– LL_COMP_OUTPUT_TIM3_IC1 (1)– LL_COMP_OUTPUT_TIM3_OCCCLR (1)
	<p>(1) Parameter availability depending on timer availability on the selected device.</p>
Notes	<ul style="list-style-type: none">• Availability of parameters of output selection to timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CSR COMP1OUTSEL LL_COMP_GetOutputSelection• COMP2OUTSEL LL_COMP_GetOutputSelection

LL_COMP_SetOutputPolarity

Function name	<code>__STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)</code>
Function description	Set comparator instance output polarity.
Parameters	<ul style="list-style-type: none">• COMPx: Comparator instance• OutputPolarity: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_COMP_OUTPUTPOL_NONINVERTED– LL_COMP_OUTPUTPOL_INVERTED
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CSR COMP1POL LL_COMP_SetOutputPolarity• COMP2POL LL_COMP_SetOutputPolarity

LL_COMP_GetOutputPolarity

Function name	<code>__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (COMP_TypeDef * COMPx)</code>
Function description	Get comparator instance output polarity.
Parameters	<ul style="list-style-type: none">• COMPx: Comparator instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_COMP_OUTPUTPOL_NONINVERTED– LL_COMP_OUTPUTPOL_INVERTED
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CSR COMP1POL LL_COMP_GetOutputPolarity• COMP2POL LL_COMP_GetOutputPolarity

LL_COMP_Enable

Function name	<code>__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)</code>
Function description	Enable comparator instance.
Parameters	<ul style="list-style-type: none">COMPx: Comparator instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">After enable from off state, comparator requires a delay to reach propagation delay specification. Refer to device datasheet, parameter "tSTART".
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1EN LL_COMP_EnableCSR COMP2EN LL_COMP_Enable

LL_COMP_Disable

Function name	<code>__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)</code>
Function description	Disable comparator instance.
Parameters	<ul style="list-style-type: none">COMPx: Comparator instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1EN LL_COMP_DisableCSR COMP2EN LL_COMP_Disable

LL_COMP_IsEnabled

Function name	<code>__STATIC_INLINE uint32_t LL_COMP_IsEnabled (COMP_TypeDef * COMPx)</code>
Function description	Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)
Parameters	<ul style="list-style-type: none">COMPx: Comparator instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1EN LL_COMP_IsEnabledCSR COMP2EN LL_COMP_IsEnabled

LL_COMP_Lock

Function name	<code>__STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)</code>
Function description	Lock comparator instance.
Parameters	<ul style="list-style-type: none">COMPx: Comparator instance

Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Once locked, comparator configuration can be accessed in read-only.The only way to unlock the comparator is a device hardware reset.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1LOCK LL_COMP_LockCSR COMP2LOCK LL_COMP_Lock

LL_COMP_IsLocked

Function name	<code>__STATIC_INLINE uint32_t LL_COMP_IsLocked (COMP_TypeDef * COMPx)</code>
Function description	Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).
Parameters	<ul style="list-style-type: none">COMPx: Comparator instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">Once locked, comparator configuration can be accessed in read-only.The only way to unlock the comparator is a device hardware reset.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1LOCK LL_COMP_IsLockedCSR COMP2LOCK LL_COMP_IsLocked

LL_COMP_ReadOutputLevel

Function name	<code>__STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (COMP_TypeDef * COMPx)</code>
Function description	Read comparator instance output level.
Parameters	<ul style="list-style-type: none">COMPx: Comparator instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_COMP_OUTPUT_LEVEL_LOWLL_COMP_OUTPUT_LEVEL_HIGH
Notes	<ul style="list-style-type: none">The comparator output level depends on the selected polarity (Refer to function <code>LL_COMP_SetOutputPolarity()</code>). If the comparator polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the comparator polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR COMP1OUT LL_COMP_ReadOutputLevelCSR COMP2OUT LL_COMP_ReadOutputLevel

LL_COMP_DelInit

Function name	<code>ErrorStatus LL_COMP_DelInit (COMP_TypeDef * COMPx)</code>
Function description	De-initialize registers of the selected COMP instance to their default reset values.

Parameters	<ul style="list-style-type: none">• COMPx: COMP instance
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: COMP registers are de-initialized– ERROR: COMP registers are not de-initialized
Notes	<ul style="list-style-type: none">• If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_COMP_Init

Function name	ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, LL_COMP_InitTypeDef * COMP_InitStruct)
Function description	Initialize some features of COMP instance.
Parameters	<ul style="list-style-type: none">• COMPx: COMP instance• COMP_InitStruct: Pointer to a LL_COMP_InitTypeDef structure
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: COMP registers are initialized– ERROR: COMP registers are not initialized
Notes	<ul style="list-style-type: none">• This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.

LL_COMP_StructInit

Function name	void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)
Function description	Set each LL_COMP_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none">• COMP_InitStruct: pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.
Return values	<ul style="list-style-type: none">• None:

52.3 COMP Firmware driver defines

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

52.3.1 COMP

COMP

Comparator common modes - Window mode

LL_COMP_WINDOWMODE_DISABLE Window mode disable: Comparators 1 and 2 are independent

LL_COMP_WINDOWMODE Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus _COMP1_INPUT_PLUS_CO connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

Definitions of COMP hardware constraints delays

LL_COMP_DELAY_START_UP_US Delay for COMP startup time

LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US Delay for COMP voltage scaler stabilization time

Comparator input - Hysteresis

LL_COMP_HYSTERESIS_N No hysteresis

ONE

LL_COMP_HYSTERESIS_L Hysteresis level low

LOW

LL_COMP_HYSTERESIS_M Hysteresis level medium

MEDIUM

LL_COMP_HYSTERESIS_H Hysteresis level high

HIGH

Comparator inputs - Input minus (input inverting) selection

LL_COMP_INPUT_MINUS_1_4VREFINT Comparator input minus connected to 1/4 Vrefint

LL_COMP_INPUT_MINUS_1_2VREFINT Comparator input minus connected to 1/2 Vrefint

LL_COMP_INPUT_MINUS_3_4VREFINT Comparator input minus connected to 3/4 Vrefint

LL_COMP_INPUT_MINUS_VREFINT Comparator input minus connected to Vrefint

LL_COMP_INPUT_MINUS_DAC1_CH1 Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)

LL_COMP_INPUT_MINUS_DAC1_CH2 Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)

LL_COMP_INPUT_MINUS_I01 Comparator input minus connected to IO1 (pin PA0 for COMP1, pin PA2 for COMP2)

Comparator inputs - Input plus (input non-inverting) selection

LL_COMP_INPUT_PLUS_I01 Comparator input plus connected to IO1 (pin PA1 for COMP1, pin PA3 for COMP2)

LL_COMP_INPUT_PLUS_DAC1_CH1 Comparator input plus connected to DAC1 channel 1 (DAC_OUT1), through dedicated switch
(Note: this switch is solely intended to redirect signals onto high impedance input, such as
COMP1 input plus (highly resistive switch)) (specific to COMP instance: COMP1)

Comparator output - Output level

LL_COMP_OUTPUT_LEVEL_L Comparator output level low (if the polarity is not inverted, otherwise to be complemented)

LL_COMP_OUTPUT_LEVEL_H Comparator output level high (if the polarity is not inverted, otherwise to be complemented)

Comparator output - Output polarity

LL_COMP_OUTPUTPOL_NONINVERTED COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input

LL_COMP_OUTPUTPOL_INVERTED COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

Comparator output - Output selection

LL_COMP_OUTPUT_NONE COMP output is not connected to other peripherals (except GPIO and EXTI that are always connected to COMP output)

LL_COMP_OUTPUT_TIM1_BKIN COMP output connected to TIM1 break input (BKIN)

LL_COMP_OUTPUT_TIM1_C1 COMP output connected to TIM1 input capture 1

LL_COMP_OUTPUT_TIM1_OCCLR COMP output connected to TIM1 OCREF clear

LL_COMP_OUTPUT_TIM2_C4 COMP output connected to TIM2 input capture 4

LL_COMP_OUTPUT_TIM2_OCCLR COMP output connected to TIM2 OCREF clear

LL_COMP_OUTPUT_TIM3_C1 COMP output connected to TIM3 input capture 1

LL_COMP_OUTPUT_TIM3_OCCLR COMP output connected to TIM3 OCREF clear

Comparator modes - Power mode

LL_COMP_POWERMODE_HIGHSPEED COMP power mode to high speed

LL_COMP_POWERMODE_MEDIUMSPEED COMP power mode to medium speed

LL_COMP_POWERMODE_LOWPOWER COMP power mode to low power

LL_COMP_POWERMODE_ULTRALOWPOWER COMP power mode to ultra-low power

COMP helper macro

LL_COMP_COMMON_INSTANCE **Description:**

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

Parameters:

- COMPx: COMP instance

Return value:

- COMP: common instance or value "0" if there is no COMP common instance.

Notes:

- COMP common register instance can be used to set parameters common to several COMP instances. Refer to functions having argument "COMPxy_COMMON" as parameter.

Common write and read registers macro**LL_COMP_WriteReg****Description:**

- Write a value in COMP register.

Parameters:

- INSTANCE: comparator instance
- REG: Register to be written
- VALUE: Value to be written in the register

Return value:

- None

LL_COMP_ReadReg**Description:**

- Read a value in COMP register.

Parameters:

- INSTANCE: comparator instance
- REG: Register to be read

Return value:

- Register: value

53 LL CORTEX Generic Driver

53.1 CORTEX Firmware driver API description

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

53.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name `__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)`

Function description This function checks if the Systick counter flag is active or not.

Return values

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

Notes

- It can be used in timeout function on application side.

Reference Manual to LL API cross reference:

LL_SYSTICK_SetClkSource

Function name `__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)`

Function description Configures the SysTick clock source.

Parameters

- **Source:** This parameter can be one of the following values:
 - `LL_SYSTICK_CLKSOURCE_HCLK_DIV8`
 - `LL_SYSTICK_CLKSOURCE_HCLK`

Return values

- **None:**

Reference Manual to LL API cross reference:

LL_SYSTICK_GetClkSource

Function name `__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)`

Function description Get the SysTick clock source.

Return values

- **Returned:** value can be one of the following values:
 - `LL_SYSTICK_CLKSOURCE_HCLK_DIV8`
 - `LL_SYSTICK_CLKSOURCE_HCLK`

Reference Manual to LL API cross reference:

LL_SYSTICK_EnableIT

Function name `__STATIC_INLINE void LL_SYSTICK_EnableIT (void)`

Function description Enable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name `__STATIC_INLINE void LL_SYSTICK_DisableIT (void)`

Function description Disable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name `__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)`

Function description Checks if the SYSTICK interrupt is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name `__STATIC_INLINE void LL_LPM_EnableSleep (void)`

Function description Processor uses sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name `__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)`

Function description Processor uses deep sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name `__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)`

Function description Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name `__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)`

Function description Do not sleep when returning to Thread mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name `__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)`

Function description Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name `__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)`

Function description Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_CPUID_GetImplementer

Function name `__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)`

Function description Get Implementer code.

Return values

- **Value:** should be equal to 0x41 for ARM

Reference Manual to LL API cross reference:

- SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name `__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)`

Function description Get Variant number (The r value in the rnpn product revision identifier)

Return values

- **Value:** between 0 and 255 (0x0: revision 0)

Reference Manual to LL API cross reference:

- SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetArchitecture

Function name `__STATIC_INLINE uint32_t LL_CPUID_GetArchitecture (void)`

Function description Get Architecture number.

Return values

- **Value:** should be equal to 0xC for Cortex-M0 devices

Reference Manual to LL API cross reference:

- SCB_CPUID ARCHITECTURE LL_CPUID_GetArchitecture

LL_CPUID_GetParNo

Function name `__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)`

Function description Get Part number.

Return values

- **Value:** should be equal to 0xC20 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

53.2 CORTEX Firmware driver defines

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

53.2.1 CORTEX

CORTEX

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE AHB clock divided by 8 selected as SysTick clock source.
_HCLK_DIV8

LL_SYSTICK_CLKSOURCE AHB clock selected as SysTick clock source.
_HCLK

54 LL CRC Generic Driver

54.1 CRC Firmware driver API description

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

54.1.1 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:**

Notes

- This function is available only on devices supporting Programmable Polynomial feature.

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_SetPolynomialSize

[LL_CRC_GetPolynomialSize](#)

Function name `__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (CRC_TypeDef * CRCx)`

Function description Return size of the polynomial.

Parameters	<ul style="list-style-type: none">• CRCx: CRC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_CRC_POLYLENGTH_32B– LL_CRC_POLYLENGTH_16B– LL_CRC_POLYLENGTH_8B– LL_CRC_POLYLENGTH_7B
Notes	<ul style="list-style-type: none">• This function is available only on devices supporting Programmable Polynomial feature.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseMode

Function name	<code>__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)</code>
Function description	Configure the reversal of the bit order of the input data.
Parameters	<ul style="list-style-type: none">• CRCx: CRC Instance• ReverseMode: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_CRC_INDATA_REVERSE_NONE– LL_CRC_INDATA_REVERSE_BYTE– LL_CRC_INDATA_REVERSE_HALFWORD– LL_CRC_INDATA_REVERSE_WORD
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name	<code>__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (CRC_TypeDef * CRCx)</code>
Function description	Return type of reversal for input data bit order.
Parameters	<ul style="list-style-type: none">• CRCx: CRC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name	<code>__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)</code>
Function description	Configure the reversal of the bit order of the Output data.
Parameters	<ul style="list-style-type: none">CRCx: CRC InstanceReverseMode: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_CRC_OUTDATA_REVERSE_NONE– LL_CRC_OUTDATA_REVERSE_BIT
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name	<code>__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (CRC_TypeDef * CRCx)</code>
Function description	Configure the reversal of the bit order of the Output data.
Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_CRC_OUTDATA_REVERSE_NONE– LL_CRC_OUTDATA_REVERSE_BIT
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name	<code>__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)</code>
Function description	Initialize the Programmable initial CRC value.
Parameters	<ul style="list-style-type: none">CRCx: CRC InstanceInitCrc: Value to be programmed in Programmable initial CRC value register
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">If the CRC size is less than 32 bits, the least significant bits are used to write the correct valueLL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name	<code>__STATIC_INLINE uint32_t LL_CRC_GetInitialData (CRC_TypeDef * CRCx)</code>
Function description	Return current Initial CRC value.
Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
Return values	<ul style="list-style-type: none">Value: programmed in Programmable initial CRC value register
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">INIT INIT LL_CRC_GetInitialData

LL_CRC_SetPolynomialCoef

Function name	<code>__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)</code>
Function description	Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).
Parameters	<ul style="list-style-type: none">CRCx: CRC InstancePolynomCoef: Value to be programmed in Programmable Polynomial value register
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">This function is available only on devices supporting Programmable Polynomial feature.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:	<ul style="list-style-type: none">POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name	<code>__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (CRC_TypeDef * CRCx)</code>
Function description	Return current Programmable polynomial value.
Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
Return values	<ul style="list-style-type: none">Value: programmed in Programmable Polynomial value register

Notes

- This function is available only on devices supporting Programmable Polynomial feature.
- 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 Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name `__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)`

Function description Write given 16-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 16 bit value to be provided to CRC calculator between 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 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	<code>__STATIC_INLINE uint32_t LL_CRC_ReadData32 (CRC_TypeDef * CRCx)</code>
Function description	Return current CRC calculation result.
Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
Return values	<ul style="list-style-type: none">Current: CRC calculation result as stored in CRC_DR register (32 bits).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name	<code>__STATIC_INLINE uint16_t LL_CRC_ReadData16 (CRC_TypeDef * CRCx)</code>
Function description	Return current CRC calculation result.
Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
Return values	<ul style="list-style-type: none">Current: CRC calculation result as stored in CRC_DR register (16 bits).
Notes	<ul style="list-style-type: none">This function is expected to be used in a 16 bits CRC polynomial size context.This function is available only on devices supporting Programmable Polynomial feature.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name	<code>__STATIC_INLINE uint8_t LL_CRC_ReadData8 (CRC_TypeDef * CRCx)</code>
Function description	Return current CRC calculation result.
Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
Return values	<ul style="list-style-type: none">Current: CRC calculation result as stored in CRC_DR register (8 bits).
Notes	<ul style="list-style-type: none">This function is expected to be used in a 8 bits CRC polynomial size context.This function is available only on devices supporting Programmable Polynomial feature.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name	<code>__STATIC_INLINE uint8_t LL_CRC_ReadData7 (CRC_TypeDef * CRCx)</code>
Function description	Return current CRC calculation result.

Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
Return values	<ul style="list-style-type: none">Current: CRC calculation result as stored in CRC_DR register (7 bits).
Notes	<ul style="list-style-type: none">This function is expected to be used in a 7 bits CRC polynomial size context.This function is available only on devices supporting Programmable Polynomial feature.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name	<code>__STATIC_INLINE uint32_t LL_CRC_Read_IDR (CRC_TypeDef * CRCx)</code>
---------------	---

Function description	Return data stored in the Independent Data(IDR) register.
----------------------	---

Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
------------	---

Return values	<ul style="list-style-type: none">Value: stored in CRC_IDR register (General-purpose 8-bit data register).
---------------	---

Notes	<ul style="list-style-type: none">This register can be used as a temporary storage location for one byte.
-------	---

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">IDR IDR LL_CRC_Read_IDR
---	---

LL_CRC_Write_IDR

Function name	<code>__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)</code>
---------------	---

Function description	Store data in the Independent Data(IDR) register.
----------------------	---

Parameters	<ul style="list-style-type: none">CRCx: CRC InstanceInData: value to be stored in CRC_IDR register (8-bit) between Min_Data=0 and Max_Data=0xFF
------------	--

Return values	<ul style="list-style-type: none">None:
---------------	--

Notes	<ul style="list-style-type: none">This register can be used as a temporary storage location for one byte.
-------	---

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">IDR IDR LL_CRC_Write_IDR
---	--

LL_CRC_DeInit

Function name	<code>ErrorStatus LL_CRC_DeInit (CRC_TypeDef * CRCx)</code>
---------------	--

Function description	De-initialize CRC registers (Registers restored to their default values).
----------------------	---

Parameters	<ul style="list-style-type: none">CRCx: CRC Instance
------------	---

- Return values**
- **An:** ErrorStatus enumeration value:
 - SUCCESS: CRC registers are de-initialized
 - ERROR: CRC registers are not de-initialized

54.2 CRC Firmware driver defines

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

54.2.1 CRC

CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_I Default CRC computation initialization value
NITVALUE

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32 Default CRC generating polynomial value
_POLY

Input Data Reverse

LL_CRC_INDATA_REVERS Input Data bit order not affected
E_NONE

LL_CRC_INDATA_REVERS Input Data bit reversal done by byte
E_BYTE

LL_CRC_INDATA_REVERS Input Data bit reversal done by half-word
E_HALFWORD

LL_CRC_INDATA_REVERS Input Data bit reversal done by word
E_WORD

Output Data Reverse

LL_CRC_OUTDATA_REV Output Data bit order not affected
RSE_NONE

LL_CRC_OUTDATA_REV Output Data bit reversal done by bit
RSE_BIT

Polynomial length

LL_CRC_POLYLENGTH_32 32 bits Polynomial size
B

LL_CRC_POLYLENGTH_16 16 bits Polynomial size
B

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

55 LL CRS Generic Driver

55.1 CRS Firmware driver API description

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

55.1.1 Detailed description of functions

`LL_CRS_EnableFreqErrorCounter`

Function name `__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter (void)`

Function description Enable Frequency error counter.

Return values

- **None:**

Notes

- When this bit is set, the CRS_CFGR register is write-protected and cannot be modified

Reference Manual to LL API cross reference:

`LL_CRS_DisableFreqErrorCounter`

Function name `__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter (void)`

Function description Disable Frequency error counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

`LL_CRS_IsEnabledFreqErrorCounter`

Function name `__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter (void)`

Function description Check if Frequency error counter is enabled or not.

Return values

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

Reference Manual to LL API cross reference:

`LL_CRS_EnableAutoTrimming`

Function name `__STATIC_INLINE void LL_CRS_EnableAutoTrimming (void)`

Function description Enable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_EnableAutoTrimming

LL_CRS_DisableAutoTrimming

Function name `__STATIC_INLINE void LL_CRS_DisableAutoTrimming (void)`

Function description Disable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_DisableAutoTrimming

LL_CRS_IsEnabledAutoTrimming

Function name `__STATIC_INLINE uint32_t LL_CRS_IsEnabledAutoTrimming (void)`

Function description Check if Automatic trimming is enabled or not.

Return values

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

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_IsEnabledAutoTrimming

LL_CRS_SetHSI48SmoothTrimming

Function name `__STATIC_INLINE void LL_CRS_SetHSI48SmoothTrimming (uint32_t Value)`

Function description Set HSI48 oscillator smooth trimming.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 63

Return values

- **None:**

Notes

- When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only
- Default value can be set thanks to LL_CRS_HSI48CALIBRATION_DEFAULT

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_SetHSI48SmoothTrimming

LL_CRS_GetHSI48SmoothTrimming

Function name `__STATIC_INLINE uint32_t LL_CRS_GetHSI48SmoothTrimming (void)`

Function description Get HSI48 oscillator smooth trimming.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 63

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_GetHSI48SmoothTrimming

LL_CRS_SetReloadCounter

Function name	<code>__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)</code>
Function description	Set counter reload value.
Parameters	<ul style="list-style-type: none">• Value: a number between Min_Data = 0 and Max_Data = 0xFFFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro __LL_CRS_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR RELOAD LL_CRS_SetReloadCounter

LL_CRS_GetReloadCounter

Function name	<code>__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void)</code>
Function description	Get counter reload value.
Return values	<ul style="list-style-type: none">• a: number between Min_Data = 0 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR RELOAD LL_CRS_GetReloadCounter

LL_CRS_SetFreqErrorLimit

Function name	<code>__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)</code>
Function description	Set frequency error limit.
Parameters	<ul style="list-style-type: none">• Value: a number between Min_Data = 0 and Max_Data = 255
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR FELIM LL_CRS_SetFreqErrorLimit

LL_CRS_GetFreqErrorLimit

Function name	<code>__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void)</code>
Function description	Get frequency error limit.
Return values	<ul style="list-style-type: none">• A: number between Min_Data = 0 and Max_Data = 255

- Reference Manual to LL API cross reference:**
- CFGR FELIM LL_CRS_GetFreqErrorLimit

LL_CRS_SetSyncDivider

Function name `__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)`

Function description Set division factor for SYNC signal.

- Parameters**
- **Divider:** This parameter can be one of the following values:
 - LL_CRS_SYNC_DIV_1
 - LL_CRS_SYNC_DIV_2
 - LL_CRS_SYNC_DIV_4
 - LL_CRS_SYNC_DIV_8
 - LL_CRS_SYNC_DIV_16
 - LL_CRS_SYNC_DIV_32
 - LL_CRS_SYNC_DIV_64
 - LL_CRS_SYNC_DIV_128

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- CFGR SYNCDIV LL_CRS_SetSyncDivider

LL_CRS_GetSyncDivider

Function name `__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void)`

Function description Get division factor for SYNC signal.

- Return values**
- **Returned:** value can be one of the following values:
 - LL_CRS_SYNC_DIV_1
 - LL_CRS_SYNC_DIV_2
 - LL_CRS_SYNC_DIV_4
 - LL_CRS_SYNC_DIV_8
 - LL_CRS_SYNC_DIV_16
 - LL_CRS_SYNC_DIV_32
 - LL_CRS_SYNC_DIV_64
 - LL_CRS_SYNC_DIV_128

- Reference Manual to LL API cross reference:**
- CFGR SYNCDIV LL_CRS_GetSyncDivider

LL_CRS_SetSyncSignalSource

Function name `__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)`

Function description Set SYNC signal source.

- Parameters**
- **Source:** This parameter can be one of the following values:
 - LL_CRS_SYNC_SOURCE_GPIO
 - LL_CRS_SYNC_SOURCE_LSE
 - LL_CRS_SYNC_SOURCE_USB

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- CFGR SYNC SRC LL_CRS_SetSyncSignalSource

LL_CRS_GetSyncSignalSource

Function name `__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource (void)`

Function description Get SYNC signal source.

- Return values**
- **Returned:** value can be one of the following values:
 - LL_CRS_SYNC_SOURCE_GPIO
 - LL_CRS_SYNC_SOURCE_LSE
 - LL_CRS_SYNC_SOURCE_USB

- Reference Manual to LL API cross reference:**
- CFGR SYNC SRC LL_CRS_GetSyncSignalSource

LL_CRS_SetSyncPolarity

Function name `__STATIC_INLINE void LL_CRS_SetSyncPolarity (uint32_t Polarity)`

Function description Set input polarity for the SYNC signal source.

- Parameters**
- **Polarity:** This parameter can be one of the following values:
 - LL_CRS_SYNC_POLARITY_RISING
 - LL_CRS_SYNC_POLARITY_FALLING

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- CFGR SYNC POL LL_CRS_SetSyncPolarity

LL_CRS_GetSyncPolarity

Function name `__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity (void)`

Function description Get input polarity for the SYNC signal source.

- Return values**
- **Returned:** value can be one of the following values:
 - LL_CRS_SYNC_POLARITY_RISING
 - LL_CRS_SYNC_POLARITY_FALLING

- Reference Manual to LL API cross reference:**
- CFGR SYNC POL LL_CRS_GetSyncPolarity

LL_CRS_ConfigSynchronization

Function name	<code>__STATIC_INLINE void LL_CRS_ConfigSynchronization (uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)</code>
Function description	Configure CRS for the synchronization.
Parameters	<ul style="list-style-type: none">HSI48CalibrationValue: a number between Min_Data = 0 and Max_Data = 63ErrorLimitValue: a number between Min_Data = 0 and Max_Data = 0xFFFFReloadValue: a number between Min_Data = 0 and Max_Data = 255Settings: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128– LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB– LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR TRIM LL_CRS_ConfigSynchronizationCFGR RELOAD LL_CRS_ConfigSynchronizationCFGR FELIM LL_CRS_ConfigSynchronizationCFGR SYNCDIV LL_CRS_ConfigSynchronizationCFGR SYNCSRC LL_CRS_ConfigSynchronizationCFGR SYNCPOL LL_CRS_ConfigSynchronization

LL_CRS_GenerateEvent_SWSYNC

Function name	<code>__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC (void)</code>
Function description	Generate software SYNC event.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR SWSYNC LL_CRS_GenerateEvent_SWSYNC

LL_CRS_GetFreqErrorDirection

Function name	<code>__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection (void)</code>
Function description	Get the frequency error direction latched in the time of the last SYNC event.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_CRS_FREQ_ERROR_DIR_UP– LL_CRS_FREQ_ERROR_DIR_DOWN
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR FEDIR LL_CRS_GetFreqErrorDirection

LL_CRS_GetFreqErrorCapture

Function name `__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void)`

Function description Get the frequency error counter value latched in the time of the last SYNC event.

Return values

- **A:** number between Min_Data = 0x0000 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- ISR FECAP LL_CRS_GetFreqErrorCapture

LL_CRS_IsActiveFlag_SYNCOK

Function name `__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void)`

Function description Check if SYNC event OK signal occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK

LL_CRS_IsActiveFlag_SYNCWARN

Function name `__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void)`

Function description Check if SYNC warning signal occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN

LL_CRS_IsActiveFlag_ERR

Function name `__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void)`

Function description Check if Synchronization or trimming error signal occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- ISR ERRF LL_CRS_IsActiveFlag_ERR

LL_CRS_IsActiveFlag_ESYNC

Function name `__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void)`

Function description Check if Expected SYNC signal occurred or not.

Return values

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

[Reference Manual to LL](#)
[API cross reference:](#)

- ISR ESYNCF LL_CRS_IsActiveFlag_ESYNC

LL_CRS_IsActiveFlag_SYNCERR

Function name `__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR (void)`

Function description Check if SYNC error signal occurred or not.

Return values

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

[Reference Manual to LL](#)
[API cross reference:](#)

- ISR SYNCERR LL_CRS_IsActiveFlag_SYNCERR

LL_CRS_IsActiveFlag_SYNCMISS

Function name `__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS (void)`

Function description Check if SYNC missed error signal occurred or not.

Return values

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

[Reference Manual to LL](#)
[API cross reference:](#)

- ISR SYNCMISS LL_CRS_IsActiveFlag_SYNCMISS

LL_CRS_IsActiveFlag_TRIMOVF

Function name `__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF (void)`

Function description Check if Trimming overflow or underflow occurred or not.

Return values

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

[Reference Manual to LL](#)
[API cross reference:](#)

- ISR TRIMOVF LL_CRS_IsActiveFlag_TRIMOVF

LL_CRS_ClearFlag_SYNCOK

Function name `__STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void)`

Function description Clear the SYNC event OK flag.

Return values

- **None:**

[Reference Manual to LL](#)
[API cross reference:](#)

- ICR SYNCOKC LL_CRS_ClearFlag_SYNCOK

LL_CRS_ClearFlag_SYNCWARN

Function name `__STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void)`

Function description Clear the SYNC warning flag.

Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ICR SYNCWARN NC LL_CRS_ClearFlag_SYNCWARN

LL_CRS_ClearFlag_ERR

Function name	<u>__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void)</u>
Function description	Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ICR ERRC LL_CRS_ClearFlag_ERR

LL_CRS_ClearFlag_ESYNC

Function name	<u>__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void)</u>
Function description	Clear Expected SYNC flag.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ICR ESYNCC LL_CRS_ClearFlag_ESYNC

LL_CRS_EnableIT_SYNCOK

Function name	<u>__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void)</u>
Function description	Enable SYNC event OK interrupt.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR SYNCOKIE LL_CRS_EnableIT_SYNCOK

LL_CRS_DisableIT_SYNCOK

Function name	<u>__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void)</u>
Function description	Disable SYNC event OK interrupt.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR SYNCOKIE LL_CRS_DisableIT_SYNCOK

LL_CRS_IsEnabledIT_SYNCOK

Function name	<u>__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK (void)</u>
----------------------	--

Function description Check if SYNC event OK interrupt is enabled or not.

Return values

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

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

LL_CRS_EnableIT_SYNCWARN

Function name `__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void)`

Function description Enable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

LL_CRS_DisableIT_SYNCWARN

Function name `__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void)`

Function description Disable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

LL_CRS_IsEnabledIT_SYNCWARN

Function name `__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void)`

Function description Check if SYNC warning interrupt is enabled or not.

Return values

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

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

LL_CRS_EnableIT_ERR

Function name `__STATIC_INLINE void LL_CRS_EnableIT_ERR (void)`

Function description Enable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_EnableIT_ERR

LL_CRS_DisableIT_ERR

Function name `__STATIC_INLINE void LL_CRS_DisableIT_ERR (void)`

Function description Disable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRERRIE LL_CRS_DisableIT_ERR

LL_CRS_IsEnabledIT_ERR

Function name `__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void)`

Function description Check if Synchronization or trimming error interrupt is enabled or not.

Return values

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

Reference Manual to LL API cross reference:

- CRERRIE LL_CRS_IsEnabledIT_ERR

LL_CRS_EnableIT_ESYNC

Function name `__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void)`

Function description Enable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRESYNCIE LL_CRS_EnableIT_ESYNC

LL_CRS_DisableIT_ESYNC

Function name `__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void)`

Function description Disable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRESYNCIE LL_CRS_DisableIT_ESYNC

LL_CRS_IsEnabledIT_ESYNC

Function name `__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void)`

Function description Check if Expected SYNC interrupt is enabled or not.

Return values

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

- Reference Manual to LL API cross reference:**
- CR ESYNCIE LL_CRS_IsEnabledIT_ESYNC

LL_CRS_DeInit

Function name **ErrorStatus LL_CRS_DeInit (void)**

Function description De-Initializes CRS peripheral registers to their default reset values.

- Return values**
- **An:** ErrorStatus enumeration value:
 - SUCCESS: CRS registers are de-initialized
 - ERROR: not applicable

55.2 CRS Firmware driver defines

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

55.2.1 CRS

CRS

Default Values

LL_CRS_RELOADVALUE_DEFAULT Notes:

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL_CRS_ERRORLIMIT_DEFAULT Notes:

- The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

Frequency Error Direction

LL_CRS_FREQ_ERROR_R_UP Upcounting direction, the actual frequency is above the target

LL_CRS_FREQ_ERROR_R_DOWN Downcounting direction, the actual frequency is below the target

Get Flags Defines

LL_CRS_ISR_SYNCOKF

LL_CRS_ISR_SYNCWARNF

LL_CRS_ISR_ERRF

LL_CRS_ISR_ESYNCF

LL_CRS_ISR_SYNCERR

`LL_CRS_ISR_SYNCMISS`

`LL_CRS_ISR_TRIMOVF`

IT Defines

`LL_CRS_CR_SYNCOKIE`

`LL_CRS_CR_SYNCWARNI`
E

`LL_CRS_CR_ERRIE`

`LL_CRS_CR_ESYNCIE`

Synchronization Signal Divider

`LL_CRS_SYNC_DIV_1` Synchro Signal not divided (default)

`LL_CRS_SYNC_DIV_2` Synchro Signal divided by 2

`LL_CRS_SYNC_DIV_4` Synchro Signal divided by 4

`LL_CRS_SYNC_DIV_8` Synchro Signal divided by 8

`LL_CRS_SYNC_DIV_16` Synchro Signal divided by 16

`LL_CRS_SYNC_DIV_32` Synchro Signal divided by 32

`LL_CRS_SYNC_DIV_64` Synchro Signal divided by 64

`LL_CRS_SYNC_DIV_128` Synchro Signal divided by 128

Synchronization Signal Polarity

`LL_CRS_SYNC_POLARITY_RISING` Synchro Active on rising edge (default)

`LL_CRS_SYNC_POLARITY_FALLING` Synchro Active on falling edge

Synchronization Signal Source

`LL_CRS_SYNC_SOURCE_GPIO` Synchro Signal soucre GPIO

`LL_CRS_SYNC_SOURCE_LSE` Synchro Signal source LSE

`LL_CRS_SYNC_SOURCE_USB` Synchro Signal source USB SOF (default)

Exported Macros Calculate Reload

LL_CRS_CALC_CALCUL Description:**ATE_RELOADVALUE**

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- `_FTARGET_`: Target frequency (value in Hz)
- `_FSYNC_`: Synchronization signal frequency (value in Hz)

Return value:

- Reload: value (in Hz)

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: $\text{RELOAD} = (\text{FTARGET} / \text{fSYNC}) - 1$

Common Write and read registers Macros**LL_CRS_WriteReg****Description:**

- Write a value in CRS register.

Parameters:

- `_INSTANCE_`: CRS Instance
- `_REG_`: Register to be written
- `_VALUE_`: Value to be written in the register

Return value:

- None

LL_CRS_ReadReg**Description:**

- Read a value in CRS register.

Parameters:

- `_INSTANCE_`: CRS Instance
- `_REG_`: Register to be read

Return value:

- Register: value

56 LL DAC Generic Driver

56.1 DAC Firmware driver registers structures

56.1.1 LL_DAC_InitTypeDef

`LL_DAC_InitTypeDef` is defined in the `stm32f0xx_ll_dac.h`

Data Fields

- `uint32_t TriggerSource`
- `uint32_t WaveAutoGeneration`
- `uint32_t WaveAutoGenerationConfig`
- `uint32_t OutputBuffer`

Field Documentation

• `uint32_t LL_DAC_InitTypeDef::TriggerSource`

Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of `DAC_LL_EC_TRIGGER_SOURCE`This feature can be modified afterwards using unitary function `LL_DAC_SetTriggerSource()`.

• `uint32_t LL_DAC_InitTypeDef::WaveAutoGeneration`

Set the waveform automatic generation mode for the selected DAC channel. This parameter can be a value of `DAC_LL_EC_WAVE_AUTO_GENERATION_MODE`This feature can be modified afterwards using unitary function `LL_DAC_SetWaveAutoGeneration()`.

• `uint32_t LL_DAC_InitTypeDef::WaveAutoGenerationConfig`

Set the waveform automatic generation mode for the selected DAC channel. If waveform automatic generation mode is set to noise, this parameter can be a value of `DAC_LL_EC_WAVE_NOISE_LFSR_UNMASK_BITS` If waveform automatic generation mode is set to triangle, this parameter can be a value of `DAC_LL_EC_WAVE_TRIANGLE_AMPLITUDE`

Note:

- If waveform automatic generation mode is disabled, this parameter is discarded.

This feature can be modified afterwards using unitary function `LL_DAC_SetWaveNoiseLFSR()` or `LL_DAC_SetWaveTriangleAmplitude()`, depending on the wave automatic generation selected.

• `uint32_t LL_DAC_InitTypeDef::OutputBuffer`

Set the output buffer for the selected DAC channel. This parameter can be a value of `DAC_LL_EC_OUTPUT_BUFFER`This feature can be modified afterwards using unitary function `LL_DAC_SetOutputBuffer()`.

56.2 DAC Firmware driver API description

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

56.2.1 Detailed description of functions

`LL_DAC_SetTriggerSource`

Function name `__STATIC_INLINE void LL_DAC_SetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)`

Function description Set the conversion trigger source for the selected DAC channel.

- | | |
|--|--|
| Parameters | <ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.• TriggerSource: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_TRIG_SOFTWARE– LL_DAC_TRIG_EXT_TIM2_TRGO– LL_DAC_TRIG_EXT_TIM3_TRGO– LL_DAC_TRIG_EXT_TIM4_TRGO– LL_DAC_TRIG_EXT_TIM6_TRGO– LL_DAC_TRIG_EXT_TIM7_TRGO– LL_DAC_TRIG_EXT_TIM15_TRGO– LL_DAC_TRIG_EXT EXTI_LINE9 |
| Return values | <ul style="list-style-type: none">• None: |
| Notes | <ul style="list-style-type: none">• For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().• To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.• Availability of parameters of trigger sources from timer depends on timers availability on the selected device. |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none">• CR TSEL1 LL_DAC_SetTriggerSource• CR TSEL2 LL_DAC_SetTriggerSource |

LL_DAC_GetTriggerSource

- | | |
|-----------------------------|---|
| Function name | <u>_STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)</u> |
| Function description | Get the conversion trigger source for the selected DAC channel. |
| Parameters | <ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. |

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_DAC_TRIG_SOFTWARE– LL_DAC_TRIG_EXT_TIM2_TRGO– LL_DAC_TRIG_EXT_TIM3_TRGO– LL_DAC_TRIG_EXT_TIM4_TRGO– LL_DAC_TRIG_EXT_TIM6_TRGO– LL_DAC_TRIG_EXT_TIM7_TRGO– LL_DAC_TRIG_EXT_TIM15_TRGO– LL_DAC_TRIG_EXT EXTI_LINE9
Notes	<ul style="list-style-type: none">• For conversion trigger source to be effective, DAC trigger must be enabled using function <code>LL_DAC_EnableTrigger()</code>.• Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR TSEL1 <code>LL_DAC_GetTriggerSource</code>• CR TSEL2 <code>LL_DAC_GetTriggerSource</code>

`LL_DAC_SetWaveAutoGeneration`

Function name	<code>__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)</code>
Function description	Set the waveform automatic generation mode for the selected DAC channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.• WaveAutoGeneration: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_WAVE_AUTO_GENERATION_NONE– LL_DAC_WAVE_AUTO_GENERATION_NOISE– LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR WAVE1 <code>LL_DAC_SetWaveAutoGeneration</code>• CR WAVE2 <code>LL_DAC_SetWaveAutoGeneration</code>

`LL_DAC_GetWaveAutoGeneration`

Function name	<code>__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Get the waveform automatic generation mode for the selected DAC channel.

- Parameters**
- **DACx:** DAC instance
 - **DAC_Channel:** This parameter can be one of the following values:
 - LL_DAC_CHANNEL_1
 - LL_DAC_CHANNEL_2 (1)
- (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.

- Return values**
- **Returned:** value can be one of the following values:
 - LL_DAC_WAVE_AUTO_GENERATION_NONE
 - LL_DAC_WAVE_AUTO_GENERATION_NOISE
 - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

- Reference Manual to LL API cross reference:**
- CR WAVE1 LL_DAC_SetWaveAutoGeneration
 - CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name

```
__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)
```

Function description

Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

- Parameters**
- **DACx:** DAC instance
 - **DAC_Channel:** This parameter can be one of the following values:
 - LL_DAC_CHANNEL_1
 - LL_DAC_CHANNEL_2 (1)
- (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
- **NoiseLFSRMask:** This parameter can be one of the following values:
 - LL_DAC_NOISE_LFSR_UNMASK_BIT0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
 - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

- Return values**
- **None:**

- Notes**
- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
 - This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

- Reference Manual to LL API cross reference:**
- CR MAMP1 LL_DAC_SetWaveNoiseLFSR
 - CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name	<code>__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_DAC_NOISE_LFSR_UNMASK_BIT0– LL_DAC_NOISE_LFSR_UNMASK_BITS1_0– LL_DAC_NOISE_LFSR_UNMASK_BITS2_0– LL_DAC_NOISE_LFSR_UNMASK_BITS3_0– LL_DAC_NOISE_LFSR_UNMASK_BITS4_0– LL_DAC_NOISE_LFSR_UNMASK_BITS5_0– LL_DAC_NOISE_LFSR_UNMASK_BITS6_0– LL_DAC_NOISE_LFSR_UNMASK_BITS7_0– LL_DAC_NOISE_LFSR_UNMASK_BITS8_0– LL_DAC_NOISE_LFSR_UNMASK_BITS9_0– LL_DAC_NOISE_LFSR_UNMASK_BITS10_0– LL_DAC_NOISE_LFSR_UNMASK_BITS11_0
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR MAMP1 LL_DAC_GetWaveNoiseLFSR• CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name	<code>__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)</code>
Function description	Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.• TriangleAmplitude: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_TRIANGLE_AMPLITUDE_1– LL_DAC_TRIANGLE_AMPLITUDE_3– LL_DAC_TRIANGLE_AMPLITUDE_7– LL_DAC_TRIANGLE_AMPLITUDE_15– LL_DAC_TRIANGLE_AMPLITUDE_31– LL_DAC_TRIANGLE_AMPLITUDE_63– LL_DAC_TRIANGLE_AMPLITUDE_127– LL_DAC_TRIANGLE_AMPLITUDE_255– LL_DAC_TRIANGLE_AMPLITUDE_511– LL_DAC_TRIANGLE_AMPLITUDE_1023– LL_DAC_TRIANGLE_AMPLITUDE_2047– LL_DAC_TRIANGLE_AMPLITUDE_4095
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• For wave generation to be effective, DAC channel wave generation mode must be enabled using function <code>LL_DAC_SetWaveAutoGeneration()</code>.• This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR MAMP1 <code>LL_DAC_SetWaveTriangleAmplitude</code>• CR MAMP2 <code>LL_DAC_SetWaveTriangleAmplitude</code>

`LL_DAC_GetWaveTriangleAmplitude`

Function name	<code>__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.

- | | |
|----------------------|---|
| Return values | <ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">- LL_DAC_TRIANGLE_AMPLITUDE_1- LL_DAC_TRIANGLE_AMPLITUDE_3- LL_DAC_TRIANGLE_AMPLITUDE_7- LL_DAC_TRIANGLE_AMPLITUDE_15- LL_DAC_TRIANGLE_AMPLITUDE_31- LL_DAC_TRIANGLE_AMPLITUDE_63- LL_DAC_TRIANGLE_AMPLITUDE_127- LL_DAC_TRIANGLE_AMPLITUDE_255- LL_DAC_TRIANGLE_AMPLITUDE_511- LL_DAC_TRIANGLE_AMPLITUDE_1023- LL_DAC_TRIANGLE_AMPLITUDE_2047- LL_DAC_TRIANGLE_AMPLITUDE_4095 |
|----------------------|---|

- | | |
|--|---|
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none">• CR MAMP1 LL_DAC_SetWaveTriangleAmplitude• CR MAMP2 LL_DAC_SetWaveTriangleAmplitude |
|--|---|

LL_DAC_SetOutputBuffer

- | | |
|--|---|
| Function name | <code>__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)</code> |
| Function description | Set the output buffer for the selected DAC channel. |
| Parameters | <ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_DAC_CHANNEL_1- LL_DAC_CHANNEL_2 (1)• OutputBuffer: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_DAC_OUTPUT_BUFFER_ENABLE- LL_DAC_OUTPUT_BUFFER_DISABLE |
| Return values | <ul style="list-style-type: none">• None: |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none">• CR BOFF1 LL_DAC_SetOutputBuffer• CR BOFF2 LL_DAC_SetOutputBuffer |

LL_DAC_GetOutputBuffer

- | | |
|-----------------------------|--|
| Function name | <code>__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code> |
| Function description | Get the output buffer state for the selected DAC channel. |

Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_DAC_OUTPUT_BUFFER_ENABLE– LL_DAC_OUTPUT_BUFFER_DISABLE
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR BOFF1 LL_DAC_GetOutputBuffer• CR BOFF2 LL_DAC_GetOutputBuffer

LL_DAC_EnableDMAReq

Function name	<code>__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Enable DAC DMA transfer request of the selected channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• To configure DMA source address (peripheral address), use function <code>LL_DAC_DMA_GetRegAddr()</code>.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR DMAEN1 LL_DAC_EnableDMAReq• CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name	<code>__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Disable DAC DMA transfer request of the selected channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• None:

Notes	<ul style="list-style-type: none">To configure DMA source address (peripheral address), use function <code>LL_DAC_DMA_GetRegAddr()</code>.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR DMAEN1 <code>LL_DAC_DisableDMAReq</code>CR DMAEN2 <code>LL_DAC_DisableDMAReq</code>
<code>LL_DAC_IsDMAReqEnabled</code>	
Function name	<code>__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Get DAC DMA transfer request state of the selected channel.
Parameters	<ul style="list-style-type: none">DACx: DAC instanceDAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_DAC_CHANNEL_1</code>– <code>LL_DAC_CHANNEL_2</code> (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR DMAEN1 <code>LL_DAC_IsDMAReqEnabled</code>CR DMAEN2 <code>LL_DAC_IsDMAReqEnabled</code>
<code>LL_DAC_DMA_GetRegAddr</code>	
Function name	<code>__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)</code>
Function description	Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.
Parameters	<ul style="list-style-type: none">DACx: DAC instanceDAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_DAC_CHANNEL_1</code>– <code>LL_DAC_CHANNEL_2</code> (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.Register: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED</code>– <code>LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED</code>– <code>LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED</code>
Return values	<ul style="list-style-type: none">DAC: register address

Notes

- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name	<code>__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Enable DAC selected channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)
	(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR EN1 LL_DAC_Enable• CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name	<code>__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Disable DAC selected channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)
	(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• None:

- Reference Manual to LL**
- API cross reference:**
- CR EN1 LL_DAC_Disable
 - CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name `__STATIC_INLINE uint32_t LL_DAC_IsEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)`

Function description Get DAC enable state of the selected channel.

- Parameters**
- **DACx:** DAC instance
 - **DAC_Channel:** This parameter can be one of the following values:
 - LL_DAC_CHANNEL_1
 - LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.

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

- Reference Manual to LL**
- API cross reference:**
- CR EN1 LL_DAC_IsEnabled
 - CR EN2 LL_DAC_IsEnabled

LL_DAC_EnableTrigger

Function name `__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)`

Function description Enable DAC trigger of the selected channel.

- Parameters**
- **DACx:** DAC instance
 - **DAC_Channel:** This parameter can be one of the following values:
 - LL_DAC_CHANNEL_1
 - LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.

- Return values**
- **None:**

- Notes**
- - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware or software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

- Reference Manual to LL**
- API cross reference:**
- CR TEN1 LL_DAC_EnableTrigger
 - CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name `__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)`

Function description	Disable DAC trigger of the selected channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR TEN1 LL_DAC_DisableTrigger• CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name	<code>__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Get DAC trigger state of the selected channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR TEN1 LL_DAC_IsTriggerEnabled• CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name	<code>__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Trig DAC conversion by software for the selected DAC channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can a combination of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• None:

Notes

- Preliminarily, DAC trigger must be set to software trigger using function `LL_DAC_SetTriggerSource()` with parameter "`LL_DAC_TRIGGER_SOFTWARE`". and DAC trigger must be enabled using function `LL_DAC_EnableTrigger()`.
- For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (`LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2`)

Reference Manual to LL API cross reference:

- `SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion`
- `SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion`

`LL_DAC_ConvertData12RightAligned`**Function name**

`__STATIC_INLINE void LL_DAC_ConvertData12RightAligned (DAC_TypeDef * DACx,
uint32_t DAC_Channel, uint32_t Data)`

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
 - `LL_DAC_CHANNEL_1`
 - `LL_DAC_CHANNEL_2` (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
- **Data:** Value between `Min_Data=0x000` and `Max_Data=0xFFFF`

Return values

- **None:**

Reference Manual to LL API cross reference:

- `DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned`
- `DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned`

`LL_DAC_ConvertData12LeftAligned`**Function name**

`__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx,
uint32_t DAC_Channel, uint32_t Data)`

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
 - `LL_DAC_CHANNEL_1`
 - `LL_DAC_CHANNEL_2` (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
- **Data:** Value between `Min_Data=0x000` and `Max_Data=0xFFFF`

Return values

- **None:**

Reference Manual to LL API cross reference:

- `DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned`
- `DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned`

LL_DAC_ConvertData8RightAligned

Function name `__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)`

Function description Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

- Parameters**
- **DACx:** DAC instance
 - **DAC_Channel:** This parameter can be one of the following values:
 - LL_DAC_CHANNEL_1
 - LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
 - **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

- Return values**
- **None:**

Reference Manual to LL API cross reference:

- DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned
- DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name `__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)`

Function description Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.

- Parameters**
- **DACx:** DAC instance
 - **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFFF
 - **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFFF

- Return values**
- **None:**

Reference Manual to LL API cross reference:

- DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned
- DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name `__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)`

Function description Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.

- Parameters**
- **DACx:** DAC instance
 - **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFFF
 - **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFFF

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned
 - DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name	<code>__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)</code>
Function description	Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DataChannel1: Value between Min_Data=0x00 and Max_Data=0xFF• DataChannel2: Value between Min_Data=0x00 and Max_Data=0xFF
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned• DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name	<code>__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function description	Retrieve output data currently generated for the selected DAC channel.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance• DAC_Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DAC_CHANNEL_1– LL_DAC_CHANNEL_2 (1)(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x000 and Max_Data=0xFFFF
Notes	<ul style="list-style-type: none">• Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DOR1 DACC1DOR LL_DAC_RetrieveOutputData• DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_IsActiveFlag_DMAUDR1

Function name	<code>__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1 (DAC_TypeDef * DACx)</code>
Function description	Get DAC underrun flag for DAC channel 1.
Parameters	<ul style="list-style-type: none">• DACx: DAC instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1

LL_DAC_IsActiveFlag_DMAUDR2

Function name `__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2 (DAC_TypeDef * DACx)`

Function description Get DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

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

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_IsActiveFlag_DMAUDR2

LL_DAC_ClearFlag_DMAUDR1

Function name `__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR1 (DAC_TypeDef * DACx)`

Function description Clear DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_ClearFlag_DMAUDR1

LL_DAC_ClearFlag_DMAUDR2

Function name `__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR2 (DAC_TypeDef * DACx)`

Function description Clear DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_ClearFlag_DMAUDR2

LL_DAC_EnableIT_DMAUDR1

Function name `__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR1 (DAC_TypeDef * DACx)`

Function description Enable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_EnableIT_DMAUDR1

LL_DAC_EnableIT_DMAUDR2

Function name `__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR2 (DAC_TypeDef * DACx)`

Function description Enable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_EnableIT_DMAUDR2

LL_DAC_DisableIT_DMAUDR1

Function name `__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR1 (DAC_TypeDef * DACx)`

Function description Disable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_DisableIT_DMAUDR1

LL_DAC_DisableIT_DMAUDR2

Function name `__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR2 (DAC_TypeDef * DACx)`

Function description Disable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_DisableIT_DMAUDR2

LL_DAC_IsEnabledIT_DMAUDR1

Function name `__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR1 (DAC_TypeDef * DACx)`

Function description Get DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

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

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_IsEnabledIT_DMAUDR1

LL_DAC_IsEnabledIT_DMAUDR2

Function name **`__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR2 (DAC_TypeDef * DACx)`**

Function description Get DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

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

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_IsEnabledIT_DMAUDR2

LL_DAC_DeInit

Function name **`ErrorStatus LL_DAC_DeInit (DAC_TypeDef * DACx)`**

Function description De-initialize registers of the selected DAC instance to their default reset values.

Parameters

- **DACx:** DAC instance

Return values

- **An:** ErrorStatus enumeration value:
 - **SUCCESS:** DAC registers are de-initialized
 - **ERROR:** not applicable

LL_DAC_Init

Function name **`ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, LL_DAC_InitTypeDef * DAC_InitStruct)`**

Function description Initialize some features of DAC instance.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
 - **LL_DAC_CHANNEL_1**
 - **LL_DAC_CHANNEL_2** (1)
- **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure

(1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.

Return values

- **An:** ErrorStatus enumeration value:
 - **SUCCESS:** DAC registers are initialized
 - **ERROR:** DAC registers are not initialized

Notes

- The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC instance must be disabled.

LL_DAC_StructInit

Function name `void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)`

Function description Set each LL_DAC_InitTypeDef field to default value.

Parameters • **DAC_InitStruct**: pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.

Return values • **None**:

56.3 DAC Firmware driver defines

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

56.3.1 DAC

DAC

DAC channels

`LL_DAC_CHANNEL_1` DAC channel 1

`LL_DAC_CHANNEL_2` DAC channel 2

DAC flags

`LL_DAC_FLAG_DMAUDR1` DAC channel 1 flag DMA underrun

`LL_DAC_FLAG_DMAUDR2` DAC channel 2 flag DMA underrun

Definitions of DAC hardware constraints delays

`LL_DAC_DELAY_STARTUP` Delay for DAC channel voltage settling time from DAC channel startup (transition from disable _VOLTAGE_SETTLING_US to enable)

`LL_DAC_DELAY_VOLTAGE` Delay for DAC channel voltage settling time
_SETTLING_US

DAC interruptions

`LL_DAC_IT_DMAUDRIE1` DAC channel 1 interruption DMA underrun

`LL_DAC_IT_DMAUDRIE2` DAC channel 2 interruption DMA underrun

DAC channel output buffer

`LL_DAC_OUTPUT_BUFFE_R_ENABLE` The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption

`LL_DAC_OUTPUT_BUFFE_R_DISABLE` The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA DAC channel data holding register 12 bits right aligned
_12BITS_RIGHT_ALIGNED

LL_DAC_DMA_REG_DATA DAC channel data holding register 12 bits left aligned
_12BITS_LEFT_ALIGNED

LL_DAC_DMA_REG_DATA DAC channel data holding register 8 bits right aligned
_8BITS_RIGHT_ALIGNED

DAC channel output resolution

LL_DAC_RESOLUTION_12 DAC channel resolution 12 bits
B

LL_DAC_RESOLUTION_8B DAC channel resolution 8 bits

DAC trigger source

LL_DAC_TRIG_SOFTWARE DAC channel conversion trigger internal (SW start)

LL_DAC_TRIG_EXT_TIM2 DAC channel conversion trigger from external IP: TIM2 TRGO.
TRGO

LL_DAC_TRIG_EXT_TIM3 DAC channel conversion trigger from external IP: TIM3 TRGO.
TRGO

LL_DAC_TRIG_EXT_TIM4 DAC channel conversion trigger from external IP: TIM4 TRGO.
TRGO

LL_DAC_TRIG_EXT_TIM6 DAC channel conversion trigger from external IP: TIM6 TRGO.
TRGO

LL_DAC_TRIG_EXT_TIM7 DAC channel conversion trigger from external IP: TIM7 TRGO.
TRGO

LL_DAC_TRIG_EXT_TIM15 DAC channel conversion trigger from external IP: TIM15 TRGO.
_TRGO

LL_DAC_TRIG_EXT_EXTI DAC channel conversion trigger from external IP: external interrupt line 9.
LINE9

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GE DAC channel wave auto generation mode disabled.
NERATION_NONE

LL_DAC_WAVE_AUTO_GE DAC channel wave auto generation mode enabled, set generated noise waveform.
NERATION_NOISE

LL_DAC_WAVE_AUTO_GE DAC channel wave auto generation mode enabled, set generated triangle waveform.
NERATION_TRIANGLE

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bit0, for the selected DAC channel
MASK_BIT0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel
MASK_BITS1_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel
MASK_BITS2_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[3:0], for the selected DAC channel
MASK_BITS3_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel
MASK_BITS4_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[5:0], for the selected DAC channel
MASK_BITS5_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel
MASK_BITS6_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel
MASK_BITS7_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel
MASK_BITS8_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel
MASK_BITS9_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel
MASK_BITS10_0

LL_DAC_NOISE_LFSR_UN Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel
MASK_BITS11_0

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPL Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC
ITUDE_1 channel

LL_DAC_TRIANGLE_AMPL Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC
ITUDE_3 channel

LL_DAC_TRIANGLE_AMPL Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC
ITUDE_7 channel

LL_DAC_TRIANGLE_AMPL Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC
ITUDE_15 channel

LL_DAC_TRIANGLE_AMPL Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC
ITUDE_31 channel

LL_DAC_TRIANGLE_AMPLITUDE_63 Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_127 Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_255 Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_511 Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_1023 Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_2047 Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_4095 Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

LL_DAC_CHANNEL_TO_DECIMAL_NB Description:

- Helper macro to get DAC channel number in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- CHANNEL: This parameter can be one of the following values:
 - LL_DAC_CHANNEL_1
 - LL_DAC_CHANNEL_2 (1)

Return value:

- 1...2: (value "2" depending on DAC channel 2 availability)

Notes:

- The input can be a value from functions where a channel number is returned.

LL_DAC_DECIMAL_NB_TO_CHANNEL Description:

- Helper macro to get DAC channel in literal format LL_DAC_CHANNEL_x from number in decimal format.

Parameters:

- DECIMAL_NB: 1...2 (value "2" depending on DAC channel 2 availability)

Return value:

- Returned: value can be one of the following values:
 - LL_DAC_CHANNEL_1
 - LL_DAC_CHANNEL_2 (1)

Notes:

- If the input parameter does not correspond to a DAC channel, this macro returns value '0'.

LL_DAC_DIGITAL_SCAL Description:

E

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

Parameters:

- DAC_RESOLUTION: This parameter can be one of the following values:
 - LL_DAC_RESOLUTION_12B
 - LL_DAC_RESOLUTION_8B

Return value:

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

Notes:

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

LL_DAC_CALC_VOLTAG Description:

E_TO_DATA

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

Parameters:

- VREFANALOG_VOLTAGE: Analog reference voltage (unit: mV)
- DAC_VOLTAGE: Voltage to be generated by DAC channel (unit: mVolt).
- DAC_RESOLUTION: This parameter can be one of the following values:
 - LL_DAC_RESOLUTION_12B
 - LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro LL_ADC_CALC_VREFANALOG_VOLTAGE().

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- INSTANCE: DAC Instance
- REG: Register to be written
- VALUE: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- INSTANCE: DAC Instance
- REG: Register to be read

Return value:

- Register: value

57 LL DMA Generic Driver

57.1 DMA Firmware driver registers structures

57.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the `stm32f0xx_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 `DMA_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()`.

57.2 DMA Firmware driver API description

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

57.2.1 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`

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	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR EN LL_DMA_DisableChannel
	LL_DMA_IsEnabledChannel
Function name	<u>__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (DMA_TypeDef * DMAx, uint32_t Channel)</u>
Function description	Check if DMA channel is enabled or disabled.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR EN LL_DMA_IsEnabledChannel
	LL_DMA_ConfigTransfer
Function name	<u>__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)</u>
Function description	Configure all parameters link to DMA transfer.

Parameters

- **DMAX:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
 - LL_DMA_CHANNEL_1
 - LL_DMA_CHANNEL_2
 - LL_DMA_CHANNEL_3
 - LL_DMA_CHANNEL_4
 - LL_DMA_CHANNEL_5
 - LL_DMA_CHANNEL_6
 - LL_DMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
 - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or
LL_DMA_DIRECTION_MEMORY_TO_PERIPH or
LL_DMA_DIRECTION_MEMORY_TO_MEMORY
 - LL_DMA_MODE_NORMAL or LL_DMA_MODE_CIRCULAR
 - LL_DMA_PERIPH_INCREMENT or LL_DMA_PERIPH_NOINCREMENT
 - LL_DMA_MEMORY_INCREMENT or LL_DMA_MEMORY_NOINCREMENT
 - LL_DMA_PDATAALIGN_BYTE or LL_DMA_PDATAALIGN_HALFWORD or
LL_DMA_PDATAALIGN_WORD
 - LL_DMA_MDATAALIGN_BYTE or LL_DMA_MDATAALIGN_HALFWORD or
LL_DMA_MDATAALIGN_WORD
 - LL_DMA_PRIORITY_LOW or LL_DMA_PRIORITY_MEDIUM or
LL_DMA_PRIORITY_HIGH or LL_DMA_PRIORITY_VERYHIGH

Return values

- **None:**

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	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• Direction: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_DIRECTION_PERIPH_TO_MEMORY– LL_DMA_DIRECTION_MEMORY_TO_PERIPH– LL_DMA_DIRECTION_MEMORY_TO_MEMORY
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR DIR LL_DMA_SetDataTransferDirection• CCR MEM2MEM LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (DMA_TypeDef * DMAX, uint32_t Channel)</code>
Function description	Get Data transfer direction (read from peripheral or from memory).
Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• CCR DIR LL_DMA_GetDataTransferDirection• CCR MEM2MEM LL_DMA_GetDataTransferDirection

LL_DMA_SetMode

Function name	<code>__STATIC_INLINE void LL_DMA_SetMode (DMA_TypeDef * DMAX, uint32_t Channel, uint32_t Mode)</code>
Function description	Set DMA mode circular or normal.

Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• Mode: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_MODE_NORMAL– LL_DMA_MODE_CIRCULAR
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CCR CIRC LL_DMA_SetMode

LL_DMA_GetMode

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Get DMA mode circular or normal.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_DMA_MODE_NORMAL– LL_DMA_MODE_CIRCULAR
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR CIRC LL_DMA_SetMode

LL_DMA_SetPeriphIncMode

Function name	<code>__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)</code>
Function description	Set Peripheral increment mode.

- Parameters**
- **DMAx:** DMAx Instance
 - **Channel:** This parameter can be one of the following values:
 - LL_DMA_CHANNEL_1
 - LL_DMA_CHANNEL_2
 - LL_DMA_CHANNEL_3
 - LL_DMA_CHANNEL_4
 - LL_DMA_CHANNEL_5
 - LL_DMA_CHANNEL_6
 - LL_DMA_CHANNEL_7
 - **PeriphOrM2MSrcIncMode:** This parameter can be one of the following values:
 - LL_DMA_PERIPH_INCREMENT
 - LL_DMA_PERIPH_NOINCREMENT

- Return values**
- **None:**

- 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

- 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_SetPeriphIncMode

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
 - **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

- 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_SetMemoryIncMode

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	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• PeriphOrM2MSrcDataSize: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_PDATAALIGN_BYTE– LL_DMA_PDATAALIGN_HALFWORD– LL_DMA_PDATAALIGN_WORD
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name	<u>__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (DMA_TypeDef * DMAX, uint32_t Channel)</u>
Function description	Get Peripheral size.
Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_DMA_PDATAALIGN_BYTE– LL_DMA_PDATAALIGN_HALFWORD– LL_DMA_PDATAALIGN_WORD
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name	<u>__STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAX, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)</u>
Function description	Set Memory size.

Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• MemoryOrM2MDstDataSize: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_MDATAALIGN_BYTE– LL_DMA_MDATAALIGN_HALFWORD– LL_DMA_MDATAALIGN_WORD
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAX, uint32_t Channel)</code>
Function description	Get Memory size.
Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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	<code>__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (DMA_TypeDef * DMAX, uint32_t Channel, uint32_t Priority)</code>
Function description	Set Channel priority level.

Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• Priority: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_PRIORITY_LOW– LL_DMA_PRIORITY_MEDIUM– LL_DMA_PRIORITY_HIGH– LL_DMA_PRIORITY VERYHIGH
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR PL LL_DMA_SetChannelPriorityLevel
LL_DMA_GetChannelPriorityLevel	
Function name	<code>__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Get Channel priority level.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_DMA_PRIORITY_LOW– LL_DMA_PRIORITY_MEDIUM– LL_DMA_PRIORITY_HIGH– LL_DMA_PRIORITY VERYHIGH
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR PL LL_DMA_GetChannelPriorityLevel
LL_DMA_SetDataLength	
Function name	<code>__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t NbData)</code>

Function description	Set Number of data to transfer.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• NbData: Between Min_Data = 0 and Max_Data = 0x0000FFFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This action has no effect if channel is enabled.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CNDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name	<code>_STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Get Number of data to transfer.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Between: Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	<ul style="list-style-type: none">• Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CNDTR NDT LL_DMA_GetDataLength

LL_DMA_ConfigAddresses

Function name	<code>_STATIC_INLINE void LL_DMA_ConfigAddresses (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)</code>
Function description	Configure the Source and Destination addresses.

Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• SrcAddress: Between Min_Data = 0 and Max_Data = 0xFFFFFFFF• DstAddress: Between Min_Data = 0 and Max_Data = 0xFFFFFFFF• Direction: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_DIRECTION_PERIPH_TO_MEMORY– LL_DMA_DIRECTION_MEMORY_TO_PERIPH– LL_DMA_DIRECTION_MEMORY_TO_MEMORY
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CPAR PA LL_DMA_ConfigAddresses• CMAR MA LL_DMA_ConfigAddresses

LL_DMA_SetMemoryAddress

Function name	<code>__STATIC_INLINE void LL_DMA_SetMemoryAddress (DMA_TypeDef * DMAX, uint32_t Channel, uint32_t MemoryAddress)</code>
Function description	Set the Memory address.
Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• MemoryAddress: Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CMAR MA LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name	<code>__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphAddress)</code>
Function description	Set the Peripheral address.
Parameters	<ul style="list-style-type: none">DMAx: DMAx InstanceChannel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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_7PeriphAddress: Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CPAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Get Memory address.
Parameters	<ul style="list-style-type: none">DMAx: DMAx InstanceChannel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">Between: Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	<ul style="list-style-type: none">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	<code>__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Get Peripheral address.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Between: Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CPAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name	<code>__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)</code>
Function description	Set the Memory to Memory Source address.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• MemoryAddress: Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)</code>
Function description	Set the Memory to Memory Destination address.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7• MemoryAddress: Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CMAR MA LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Get the Memory to Memory Source address.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Between: Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	<ul style="list-style-type: none">• 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

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 PeriphRequest)`

Function description Set DMA request for DMA instance on Channel x.

Parameters	<ul style="list-style-type: none">• DMAX: DMAX Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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• PeriphRequest: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_REQUEST_0– LL_DMA_REQUEST_1– LL_DMA_REQUEST_2– LL_DMA_REQUEST_3– LL_DMA_REQUEST_4– LL_DMA_REQUEST_5– LL_DMA_REQUEST_6– LL_DMA_REQUEST_7– LL_DMA_REQUEST_8– LL_DMA_REQUEST_9– LL_DMA_REQUEST_10– LL_DMA_REQUEST_11– LL_DMA_REQUEST_12– LL_DMA_REQUEST_13– LL_DMA_REQUEST_14– LL_DMA_REQUEST_15
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Please refer to Reference Manual to get the available mapping of Request value link to Channel Selection.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CSELR C1S LL_DMA_SetPeriphRequest• CSELR C2S LL_DMA_SetPeriphRequest• CSELR C3S LL_DMA_SetPeriphRequest• CSELR C4S LL_DMA_SetPeriphRequest• CSELR C5S LL_DMA_SetPeriphRequest• CSELR C6S LL_DMA_SetPeriphRequest• CSELR C7S LL_DMA_SetPeriphRequest
LL_DMA_GetPeriphRequest	
Function name	<code>__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (DMA_TypeDef * DMAX, uint32_t Channel)</code>
Function description	Get DMA request for DMA instance on Channel x.

Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_DMA_REQUEST_0– LL_DMA_REQUEST_1– LL_DMA_REQUEST_2– LL_DMA_REQUEST_3– LL_DMA_REQUEST_4– LL_DMA_REQUEST_5– LL_DMA_REQUEST_6– LL_DMA_REQUEST_7– LL_DMA_REQUEST_8– LL_DMA_REQUEST_9– LL_DMA_REQUEST_10– LL_DMA_REQUEST_11– LL_DMA_REQUEST_12– LL_DMA_REQUEST_13– LL_DMA_REQUEST_14– LL_DMA_REQUEST_15
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CSELR C1S LL_DMA_GetPeriphRequest• CSELR C2S LL_DMA_GetPeriphRequest• CSELR C3S LL_DMA_GetPeriphRequest• CSELR C4S LL_DMA_GetPeriphRequest• CSELR C5S LL_DMA_GetPeriphRequest• CSELR C6S LL_DMA_GetPeriphRequest• CSELR C7S LL_DMA_GetPeriphRequest

LL_DMA_IsActiveFlag_GI1

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI1 (DMA_TypeDef * DMAX)</code>
Function description	Get Channel 1 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAX: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR GIF1 LL_DMA_IsActiveFlag_GI1

LL_DMA_IsActiveFlag_GI2

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI2 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 2 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR GIF2 LL_DMA_IsActiveFlag_GI2

LL_DMA_IsActiveFlag_GI3

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI3 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 3 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR GIF3 LL_DMA_IsActiveFlag_GI3

LL_DMA_IsActiveFlag_GI4

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI4 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 4 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR GIF4 LL_DMA_IsActiveFlag_GI4

LL_DMA_IsActiveFlag_GI5

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI5 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 5 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR GIF5 LL_DMA_IsActiveFlag_GI5

LL_DMA_IsActiveFlag_GI6

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI6 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 6 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR GIF6 LL_DMA_IsActiveFlag_GI6

LL_DMA_IsActiveFlag_GI7

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI7 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 7 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR GIF7 LL_DMA_IsActiveFlag_GI7

LL_DMA_IsActiveFlag_TC1

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 1 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TCIF1 LL_DMA_IsActiveFlag_TC1

LL_DMA_IsActiveFlag_TC2

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC2 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 2 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TCIF2 LL_DMA_IsActiveFlag_TC2

LL_DMA_IsActiveFlag_TC3

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC3 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 3 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TCIF3 LL_DMA_IsActiveFlag_TC3

LL_DMA_IsActiveFlag_TC4

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC4 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 4 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TCIF4 LL_DMA_IsActiveFlag_TC4

LL_DMA_IsActiveFlag_TC5

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 5 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TCIF5 LL_DMA_IsActiveFlag_TC5

LL_DMA_IsActiveFlag_TC6

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 6 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TCIF6 LL_DMA_IsActiveFlag_TC6

LL_DMA_IsActiveFlag_TC7

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 7 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TCIF7 LL_DMA_IsActiveFlag_TC7

LL_DMA_IsActiveFlag_HT1

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 1 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR HTIF1 LL_DMA_IsActiveFlag_HT1

LL_DMA_IsActiveFlag_HT2

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT2 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 2 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR HTIF2 LL_DMA_IsActiveFlag_HT2

LL_DMA_IsActiveFlag_HT3

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT3 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 3 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR HTIF3 LL_DMA_IsActiveFlag_HT3

LL_DMA_IsActiveFlag_HT4

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT4 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 4 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR HTIF4 LL_DMA_IsActiveFlag_HT4

LL_DMA_IsActiveFlag_HT5

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT5 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 5 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR HTIF5 LL_DMA_IsActiveFlag_HT5

LL_DMA_IsActiveFlag_HT6

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 6 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR HTIF6 LL_DMA_IsActiveFlag_HT6

LL_DMA_IsActiveFlag_HT7

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 7 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR HTIF7 LL_DMA_IsActiveFlag_HT7

LL_DMA_IsActiveFlag_TE1

Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 1 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 2 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE3 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 3 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE4 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 4 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE5 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 5 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 6 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7 (DMA_TypeDef * DMAx)</code>
Function description	Get Channel 7 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).

Reference Manual to LL
API cross reference:

- ISR TEIF7 LL_DMA_IsActiveFlag_TE7

LL_DMA_ClearFlag_GI1

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI1 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 1 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL
API cross reference:

- IFCR CGIF1 LL_DMA_ClearFlag_GI1

LL_DMA_ClearFlag_GI2

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI2 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 2 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CGIF2 LL_DMA_ClearFlag_GI2

LL_DMA_ClearFlag_GI3

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI3 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 3 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CGIF3 LL_DMA_ClearFlag_GI3

LL_DMA_ClearFlag_GI4

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI4 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 4 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CGIF4 LL_DMA_ClearFlag_GI4

LL_DMA_ClearFlag_GI5

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI5 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 5 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CGIF5 LL_DMA_ClearFlag_GI5

LL_DMA_ClearFlag_GI6

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI6 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 6 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CGIF6 LL_DMA_ClearFlag_GI6

LL_DMA_ClearFlag_GI7

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI7 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 7 global interrupt flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CGIF7 LL_DMA_ClearFlag_GI7

LL_DMA_ClearFlag_TC1

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC1 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 1 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTCIF1 LL_DMA_ClearFlag_TC1

LL_DMA_ClearFlag_TC2

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 2 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTCIF2 LL_DMA_ClearFlag_TC2

LL_DMA_ClearFlag_TC3

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 3 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTCIF3 LL_DMA_ClearFlag_TC3

LL_DMA_ClearFlag_TC4

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 4 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTCIF4 LL_DMA_ClearFlag_TC4

LL_DMA_ClearFlag_TC5

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 5 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTCIF5 LL_DMA_ClearFlag_TC5

LL_DMA_ClearFlag_TC6

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 6 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTCIF6 LL_DMA_ClearFlag_TC6

LL_DMA_ClearFlag_TC7

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 7 transfer complete flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CTCIF7 LL_DMA_ClearFlag_TC7

LL_DMA_ClearFlag_HT1

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 1 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CHTIF1 LL_DMA_ClearFlag_HT1

LL_DMA_ClearFlag_HT2

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 2 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CHTIF2 LL_DMA_ClearFlag_HT2

LL_DMA_ClearFlag_HT3

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 3 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CHTIF3 LL_DMA_ClearFlag_HT3

LL_DMA_ClearFlag_HT4

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 4 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CHTIF4 LL_DMA_ClearFlag_HT4

LL_DMA_ClearFlag_HT5

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 5 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CHTIF5 LL_DMA_ClearFlag_HT5

LL_DMA_ClearFlag_HT6

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 6 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CHTIF6 LL_DMA_ClearFlag_HT6

LL_DMA_ClearFlag_HT7

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT7 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 7 half transfer flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- IFCR CHTIF7 LL_DMA_ClearFlag_HT7

LL_DMA_ClearFlag_TE1

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE1 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 1 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTEIF1 LL_DMA_ClearFlag_TE1

LL_DMA_ClearFlag_TE2

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE2 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 2 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTEIF2 LL_DMA_ClearFlag_TE2

LL_DMA_ClearFlag_TE3

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE3 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 3 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTEIF3 LL_DMA_ClearFlag_TE3

LL_DMA_ClearFlag_TE4

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 4 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTEIF4 LL_DMA_ClearFlag_TE4

LL_DMA_ClearFlag_TE5

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 5 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTEIF5 LL_DMA_ClearFlag_TE5

LL_DMA_ClearFlag_TE6

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 6 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTEIF6 LL_DMA_ClearFlag_TE6

LL_DMA_ClearFlag_TE7

Function name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)</code>
Function description	Clear Channel 7 transfer error flag.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• IFCR CTEIF7 LL_DMA_ClearFlag_TE7

LL_DMA_EnableIT_TC

Function name	<code>__STATIC_INLINE void LL_DMA_EnableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Enable Transfer complete interrupt.

Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR TCIE LL_DMA_EnableIT_TC
	LL_DMA_EnableIT_HT
Function name	<u>__STATIC_INLINE void LL_DMA_EnableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)</u>
Function description	Enable Half transfer interrupt.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR HTIE LL_DMA_EnableIT_HT
	LL_DMA_EnableIT_TE
Function name	<u>__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)</u>
Function description	Enable Transfer error interrupt.

Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR HTIE LL_DMA_DisableIT_HT
	LL_DMA_DisableIT_TE
Function name	<u>__STATIC_INLINE void LL_DMA_DisableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)</u>
Function description	Disable Transfer error interrupt.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR TEIE LL_DMA_DisableIT_TE
	LL_DMA_IsEnabledIT_TC
Function name	<u>__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)</u>
Function description	Check if Transfer complete Interrupt is enabled.

Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR TCIE LL_DMA_IsEnabledIT_TC
LL_DMA_IsEnabledIT_HT	
Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Check if Half transfer Interrupt is enabled.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR HTIE LL_DMA_IsEnabledIT_HT
LL_DMA_IsEnabledIT_TE	
Function name	<code>__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	Check if Transfer error Interrupt is enabled.

Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6– LL_DMA_CHANNEL_7
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCR TEIE LL_DMA_IsEnabledIT_TE
LL_DMA_Init	
Function name	<code>uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)</code>
Function description	Initialize the DMA registers according to the specified parameters in DMA_InitStruct.
Parameters	<ul style="list-style-type: none">• DMAx: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_DMA_CHANNEL_1– LL_DMA_CHANNEL_2– LL_DMA_CHANNEL_3– LL_DMA_CHANNEL_4– LL_DMA_CHANNEL_5– LL_DMA_CHANNEL_6 (*)– LL_DMA_CHANNEL_7 (*)• DMA_InitStruct: pointer to a LL_DMA_InitTypeDef structure.
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: DMA registers are initialized– ERROR: Not applicable
Notes	<ul style="list-style-type: none">• To convert DMAx_Channelny Instance to DMAx Instance and Channely, use helper macros : __LL_DMA_GET_INSTANCE __LL_DMA_GET_CHANNEL

LL_DMA_DeInit

Function name	<code>uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function description	De-initialize the DMA registers to their default reset values.

- | | |
|----------------------|---|
| Parameters | <ul style="list-style-type: none">• DMAX: DMAx Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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 (*) |
| | (*) value not defined in all devices |
| Return values | <ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• DMA_InitStruct: Pointer to a LL_DMA_InitTypeDef structure.
Return values	<ul style="list-style-type: none">• None:

57.3 DMA Firmware driver defines

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

57.3.1 DMA

DMA

CHANNEL

LL_DMA_CHANNEL_1	DMA Channel 1
LL_DMA_CHANNEL_2	DMA Channel 2
LL_DMA_CHANNEL_3	DMA Channel 3
LL_DMA_CHANNEL_4	DMA Channel 4
LL_DMA_CHANNEL_5	DMA Channel 5
LL_DMA_CHANNEL_6	DMA Channel 6
LL_DMA_CHANNEL_7	DMA Channel 7
LL_DMA_CHANNEL_ALL	DMA Channel all (used only for function

Clear Flags Defines

LL_DMA_IFCR_CGIF1	Channel 1 global flag
--------------------------	-----------------------

LL_DMA_IFCR_CTCIF1	Channel 1 transfer complete flag
LL_DMA_IFCR_CHTIF1	Channel 1 half transfer flag
LL_DMA_IFCR_CTEIF1	Channel 1 transfer error flag
LL_DMA_IFCR_CGIF2	Channel 2 global flag
LL_DMA_IFCR_CTCIF2	Channel 2 transfer complete flag
LL_DMA_IFCR_CHTIF2	Channel 2 half transfer flag
LL_DMA_IFCR_CTEIF2	Channel 2 transfer error flag
LL_DMA_IFCR_CGIF3	Channel 3 global flag
LL_DMA_IFCR_CTCIF3	Channel 3 transfer complete flag
LL_DMA_IFCR_CHTIF3	Channel 3 half transfer flag
LL_DMA_IFCR_CTEIF3	Channel 3 transfer error flag
LL_DMA_IFCR_CGIF4	Channel 4 global flag
LL_DMA_IFCR_CTCIF4	Channel 4 transfer complete flag
LL_DMA_IFCR_CHTIF4	Channel 4 half transfer flag
LL_DMA_IFCR_CTEIF4	Channel 4 transfer error flag
LL_DMA_IFCR_CGIF5	Channel 5 global flag
LL_DMA_IFCR_CTCIF5	Channel 5 transfer complete flag
LL_DMA_IFCR_CHTIF5	Channel 5 half transfer flag
LL_DMA_IFCR_CTEIF5	Channel 5 transfer error flag
LL_DMA_IFCR_CGIF6	Channel 6 global flag
LL_DMA_IFCR_CTCIF6	Channel 6 transfer complete flag
LL_DMA_IFCR_CHTIF6	Channel 6 half transfer flag
LL_DMA_IFCR_CTEIF6	Channel 6 transfer error flag
LL_DMA_IFCR_CGIF7	Channel 7 global flag
LL_DMA_IFCR_CTCIF7	Channel 7 transfer complete flag

`LL_DMA_IFCR_CHTIF7` Channel 7 half transfer flag

`LL_DMA_IFCR_CTEIF7` Channel 7 transfer error flag

Transfer Direction

`LL_DMA_DIRECTION_PERI` Peripheral to memory direction
`PH_TO_MEMORY`

`LL_DMA_DIRECTION_MEM` Memory to peripheral direction
`ORY_TO_PERIPH`

`LL_DMA_DIRECTION_MEM` Memory to memory direction
`ORY_TO_MEMORY`

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

<code>LL_DMA_ISR_HTIF5</code>	Channel 5 half transfer flag
<code>LL_DMA_ISR_TEIF5</code>	Channel 5 transfer error flag
<code>LL_DMA_ISR_GIF6</code>	Channel 6 global flag
<code>LL_DMA_ISR_TCIF6</code>	Channel 6 transfer complete flag
<code>LL_DMA_ISR_HTIF6</code>	Channel 6 half transfer flag
<code>LL_DMA_ISR_TEIF6</code>	Channel 6 transfer error flag
<code>LL_DMA_ISR_GIF7</code>	Channel 7 global flag
<code>LL_DMA_ISR_TCIF7</code>	Channel 7 transfer complete flag
<code>LL_DMA_ISR_HTIF7</code>	Channel 7 half transfer flag
<code>LL_DMA_ISR_TEIF7</code>	Channel 7 transfer error flag

IT Defines

<code>LL_DMA_CCR_TCIE</code>	Transfer complete interrupt
<code>LL_DMA_CCR_HTIE</code>	Half Transfer interrupt
<code>LL_DMA_CCR_TEIE</code>	Transfer error interrupt

Memory data alignment

`LL_DMA_MDATAALIGN_BY` Memory data alignment : Byte
TE

`LL_DMA_MDATAALIGN_H` Memory data alignment : HalfWord
ALFWORD

`LL_DMA_MDATAALIGN_W` Memory data alignment : Word
ORD

Memory increment mode

`LL_DMA_MEMORY_INCRE` Memory increment mode Enable
MENT

`LL_DMA_MEMORY_NOINC` Memory increment mode Disable
REMENT

Transfer mode

`LL_DMA_MODE_NORMAL` Normal Mode

`LL_DMA_MODE_CIRCULAR` Circular Mode
R

Peripheral data alignment

`LL_DMA_PDATAALIGN_BY` Peripheral data alignment : Byte
TE

`LL_DMA_PDATAALIGN_HA` Peripheral data alignment : HalfWord
LFWORD

`LL_DMA_PDATAALIGN_W` Peripheral data alignment : Word
ORD

Peripheral increment mode

`LL_DMA_PERIPH_INCREM` Peripheral increment mode Enable
ENT

`LL_DMA_PERIPH_NOINCR` Peripheral increment mode Disable
EMENT

Transfer Priority level

`LL_DMA_PRIORITY_LOW` Priority level : Low

`LL_DMA_PRIORITY_MEDIU` Priority level : Medium
M

`LL_DMA_PRIORITY_HIGH` Priority level : High

`LL_DMA_PRIORITY_VERYHIGH` Priority level : Very_High

Transfer peripheral request

`LL_DMA_REQUEST_0` DMA peripheral request 0

`LL_DMA_REQUEST_1` DMA peripheral request 1

`LL_DMA_REQUEST_2` DMA peripheral request 2

`LL_DMA_REQUEST_3` DMA peripheral request 3

`LL_DMA_REQUEST_4` DMA peripheral request 4

`LL_DMA_REQUEST_5` DMA peripheral request 5

`LL_DMA_REQUEST_6` DMA peripheral request 6

`LL_DMA_REQUEST_7` DMA peripheral request 7

`LL_DMA_REQUEST_8` DMA peripheral request 8

`LL_DMA_REQUEST_9` DMA peripheral request 9

LL_DMA_REQUEST_10 DMA peripheral request 10

LL_DMA_REQUEST_11 DMA peripheral request 11

LL_DMA_REQUEST_12 DMA peripheral request 12

LL_DMA_REQUEST_13 DMA peripheral request 13

LL_DMA_REQUEST_14 DMA peripheral request 14

LL_DMA_REQUEST_15 DMA peripheral request 15

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

58 LL EXTI Generic Driver

58.1 EXTI Firmware driver registers structures

58.1.1 LL_EXTI_InitTypeDef

LL_EXTI_InitTypeDef is defined in the `stm32f0xx_ll_exti.h`

Data Fields

- *uint32_t Line_0_31*
- *FunctionalState LineCommand*
- *uint8_t Mode*
- *uint8_t Trigger*

Field Documentation

- *uint32_t LL_EXTI_InitTypeDef::Line_0_31*

Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31. This parameter can be any combination of `EXTI_LL_EC_LINE`

- *FunctionalState LL_EXTI_InitTypeDef::LineCommand*

Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE

- *uint8_t LL_EXTI_InitTypeDef::Mode*

Specifies the mode for the EXTI lines. This parameter can be a value of `EXTI_LL_EC_MODE`.

- *uint8_t LL_EXTI_InitTypeDef::Trigger*

Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of `EXTI_LL_EC_TRIGGER`.

58.2 EXTI Firmware driver API description

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

58.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name `__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)`

Function description Enable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
 - LL_EXTI_LINE_0
 - LL_EXTI_LINE_1
 - LL_EXTI_LINE_2
 - LL_EXTI_LINE_3
 - LL_EXTI_LINE_4
 - LL_EXTI_LINE_5
 - LL_EXTI_LINE_6
 - LL_EXTI_LINE_7
 - LL_EXTI_LINE_8
 - LL_EXTI_LINE_9
 - LL_EXTI_LINE_10
 - LL_EXTI_LINE_11
 - LL_EXTI_LINE_12
 - LL_EXTI_LINE_13
 - LL_EXTI_LINE_14
 - LL_EXTI_LINE_15
 - LL_EXTI_LINE_16
 - LL_EXTI_LINE_17
 - LL_EXTI_LINE_18
 - LL_EXTI_LINE_19
 - LL_EXTI_LINE_20
 - LL_EXTI_LINE_21
 - LL_EXTI_LINE_22
 - LL_EXTI_LINE_23
 - LL_EXTI_LINE_24
 - LL_EXTI_LINE_25
 - LL_EXTI_LINE_26
 - LL_EXTI_LINE_27
 - LL_EXTI_LINE_28
 - LL_EXTI_LINE_29
 - LL_EXTI_LINE_30
 - LL_EXTI_LINE_31
 - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR IMx LL_EXTI_EnableIT_0_31

LL_EXTI_DisableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
 - LL_EXTI_LINE_0
 - LL_EXTI_LINE_1
 - LL_EXTI_LINE_2
 - LL_EXTI_LINE_3
 - LL_EXTI_LINE_4
 - LL_EXTI_LINE_5
 - LL_EXTI_LINE_6
 - LL_EXTI_LINE_7
 - LL_EXTI_LINE_8
 - LL_EXTI_LINE_9
 - LL_EXTI_LINE_10
 - LL_EXTI_LINE_11
 - LL_EXTI_LINE_12
 - LL_EXTI_LINE_13
 - LL_EXTI_LINE_14
 - LL_EXTI_LINE_15
 - LL_EXTI_LINE_16
 - LL_EXTI_LINE_17
 - LL_EXTI_LINE_18
 - LL_EXTI_LINE_19
 - LL_EXTI_LINE_20
 - LL_EXTI_LINE_21
 - LL_EXTI_LINE_22
 - LL_EXTI_LINE_23
 - LL_EXTI_LINE_24
 - LL_EXTI_LINE_25
 - LL_EXTI_LINE_26
 - LL_EXTI_LINE_27
 - LL_EXTI_LINE_28
 - LL_EXTI_LINE_29
 - LL_EXTI_LINE_30
 - LL_EXTI_LINE_31
 - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR IMx LL_EXTI_DisableIT_0_31

LL_EXTI_IsEnabledIT_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
 - LL_EXTI_LINE_0
 - LL_EXTI_LINE_1
 - LL_EXTI_LINE_2
 - LL_EXTI_LINE_3
 - LL_EXTI_LINE_4
 - LL_EXTI_LINE_5
 - LL_EXTI_LINE_6
 - LL_EXTI_LINE_7
 - LL_EXTI_LINE_8
 - LL_EXTI_LINE_9
 - LL_EXTI_LINE_10
 - LL_EXTI_LINE_11
 - LL_EXTI_LINE_12
 - LL_EXTI_LINE_13
 - LL_EXTI_LINE_14
 - LL_EXTI_LINE_15
 - LL_EXTI_LINE_16
 - LL_EXTI_LINE_17
 - LL_EXTI_LINE_18
 - LL_EXTI_LINE_19
 - LL_EXTI_LINE_20
 - LL_EXTI_LINE_21
 - LL_EXTI_LINE_22
 - LL_EXTI_LINE_23
 - LL_EXTI_LINE_24
 - LL_EXTI_LINE_25
 - LL_EXTI_LINE_26
 - LL_EXTI_LINE_27
 - LL_EXTI_LINE_28
 - LL_EXTI_LINE_29
 - LL_EXTI_LINE_30
 - LL_EXTI_LINE_31
 - LL_EXTI_LINE_ALL_0_31

Return values

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

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_EnableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_17– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_23– LL_EXTI_LINE_24– LL_EXTI_LINE_25– LL_EXTI_LINE_26– LL_EXTI_LINE_27– LL_EXTI_LINE_28– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31– LL_EXTI_LINE_ALL_0_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• EMR EMx LL_EXTI_EnableEvent_0_31
LL_EXTI_DisableEvent_0_31	
Function name	_STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)
Function description	Disable ExtiLine Event request for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_17– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_23– LL_EXTI_LINE_24– LL_EXTI_LINE_25– LL_EXTI_LINE_26– LL_EXTI_LINE_27– LL_EXTI_LINE_28– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31– LL_EXTI_LINE_ALL_0_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• EMR EMx LL_EXTI_DisableEvent_0_31
LL_EXTI_IsEnabledEvent_0_31	
Function name	_STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)
Function description	Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
 - LL_EXTI_LINE_0
 - LL_EXTI_LINE_1
 - LL_EXTI_LINE_2
 - LL_EXTI_LINE_3
 - LL_EXTI_LINE_4
 - LL_EXTI_LINE_5
 - LL_EXTI_LINE_6
 - LL_EXTI_LINE_7
 - LL_EXTI_LINE_8
 - LL_EXTI_LINE_9
 - LL_EXTI_LINE_10
 - LL_EXTI_LINE_11
 - LL_EXTI_LINE_12
 - LL_EXTI_LINE_13
 - LL_EXTI_LINE_14
 - LL_EXTI_LINE_15
 - LL_EXTI_LINE_16
 - LL_EXTI_LINE_17
 - LL_EXTI_LINE_18
 - LL_EXTI_LINE_19
 - LL_EXTI_LINE_20
 - LL_EXTI_LINE_21
 - LL_EXTI_LINE_22
 - LL_EXTI_LINE_23
 - LL_EXTI_LINE_24
 - LL_EXTI_LINE_25
 - LL_EXTI_LINE_26
 - LL_EXTI_LINE_27
 - LL_EXTI_LINE_28
 - LL_EXTI_LINE_29
 - LL_EXTI_LINE_30
 - LL_EXTI_LINE_31
 - LL_EXTI_LINE_ALL_0_31

Return values

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

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_EnableRisingTrig_0_31**Function name**

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RTSR RTx LL_EXTI_EnableRisingTrig_0_31
Function name	<u>LL_EXTI_DisableRisingTrig_0_31</u>
Function description	Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RTSR RTx LL_EXTI_DisableRisingTrig_0_31
Function name	<code>LL_EXTI_IsEnabledRisingTrig_0_31</code>
Function description	Check if rising edge trigger is enabled for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RTSR RTx LL_EXTI_IsEnabledRisingTrig_0_31
	<u>LL_EXTI_EnableFallingTrig_0_31</u>
Function name	<u>__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)</u>
Function description	Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• FTSR FTx LL_EXTI_EnableFallingTrig_0_31
Function name	<u>LL_EXTI_DisableFallingTrig_0_31</u>
Function description	Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• FTSR FTx LL_EXTI_DisableFallingTrig_0_31
Function name	<code>LL_EXTI_IsEnabledFallingTrig_0_31</code>
Function description	Check if falling edge trigger is enabled for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• FTSR FTx LL_EXTI_IsEnabledFallingTrig_0_31
	LL_EXTI_GenerateSWI_0_31
Function name	_STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)
Function description	Generate a software Interrupt Event for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• If the interrupt is enabled on this line in the EXTI_IMR, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR register (by writing a 1 into the bit)• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SWIER SWIx LL_EXTI_GenerateSWI_0_31
Function name	<code>LL_EXTI_IsActiveFlag_0_31</code>
Function description	Check if the ExtLine Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
 - LL_EXTI_LINE_0
 - LL_EXTI_LINE_1
 - LL_EXTI_LINE_2
 - LL_EXTI_LINE_3
 - LL_EXTI_LINE_4
 - LL_EXTI_LINE_5
 - LL_EXTI_LINE_6
 - LL_EXTI_LINE_7
 - LL_EXTI_LINE_8
 - LL_EXTI_LINE_9
 - LL_EXTI_LINE_10
 - LL_EXTI_LINE_11
 - LL_EXTI_LINE_12
 - LL_EXTI_LINE_13
 - LL_EXTI_LINE_14
 - LL_EXTI_LINE_15
 - LL_EXTI_LINE_16
 - LL_EXTI_LINE_18
 - LL_EXTI_LINE_19
 - LL_EXTI_LINE_20
 - LL_EXTI_LINE_21
 - LL_EXTI_LINE_22
 - LL_EXTI_LINE_29
 - LL_EXTI_LINE_30
 - LL_EXTI_LINE_31

Return values

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

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PR PIFx LL_EXTI_IsActiveFlag_0_31

LL_EXTI_ReadFlag_0_31**Function name**

`__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_0_31 (uint32_t ExtiLine)`

Function description

Read ExtLine Combination Flag for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_EXTI_LINE_0– LL_EXTI_LINE_1– LL_EXTI_LINE_2– LL_EXTI_LINE_3– LL_EXTI_LINE_4– LL_EXTI_LINE_5– LL_EXTI_LINE_6– LL_EXTI_LINE_7– LL_EXTI_LINE_8– LL_EXTI_LINE_9– LL_EXTI_LINE_10– LL_EXTI_LINE_11– LL_EXTI_LINE_12– LL_EXTI_LINE_13– LL_EXTI_LINE_14– LL_EXTI_LINE_15– LL_EXTI_LINE_16– LL_EXTI_LINE_18– LL_EXTI_LINE_19– LL_EXTI_LINE_20– LL_EXTI_LINE_21– LL_EXTI_LINE_22– LL_EXTI_LINE_29– LL_EXTI_LINE_30– LL_EXTI_LINE_31
Return values	<ul style="list-style-type: none">• @note: This bit is set when the selected edge event arrives on the interrupt
Notes	<ul style="list-style-type: none">• This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• PR PIFx LL_EXTI_ReadFlag_0_31
	LL_EXTI_ClearFlag_0_31
Function name	_STATIC_INLINE void LL_EXTI_ClearFlag_0_31 (uint32_t ExtiLine)
Function description	Clear ExtLine Flags for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none">• ExtiLine: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL EXTI_LINE_0– LL EXTI_LINE_1– LL EXTI_LINE_2– LL EXTI_LINE_3– LL EXTI_LINE_4– LL EXTI_LINE_5– LL EXTI_LINE_6– LL EXTI_LINE_7– LL EXTI_LINE_8– LL EXTI_LINE_9– LL EXTI_LINE_10– LL EXTI_LINE_11– LL EXTI_LINE_12– LL EXTI_LINE_13– LL EXTI_LINE_14– LL EXTI_LINE_15– LL EXTI_LINE_16– LL EXTI_LINE_18– LL EXTI_LINE_19– LL EXTI_LINE_20– LL EXTI_LINE_21– LL EXTI_LINE_22– LL EXTI_LINE_29– LL EXTI_LINE_30– LL EXTI_LINE_31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• PR PIFx LL_EXTI_ClearFlag_0_31
LL_EXTI_Init	
Function name	<code>uint32_t LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)</code>
Function description	Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.
Parameters	<ul style="list-style-type: none">• EXTI_InitStruct: pointer to a LL_EXTI_InitTypeDef structure.
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: EXTI registers are initialized– ERROR: not applicable

LL_EXTI_DeInit

Function name	<code>uint32_t LL_EXTI_DeInit (void)</code>
Function description	De-initialize the EXTI registers to their default reset values.
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: EXTI registers are de-initialized– ERROR: not applicable

LL_EXTI_StructInit

Function name	<code>void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)</code>
Function description	Set each LL_EXTI_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none">• EXTI_InitStruct: Pointer to a LL_EXTI_InitTypeDef structure.
Return values	<ul style="list-style-type: none">• None:

58.3 EXTI Firmware driver defines

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

58.3.1 EXTI

EXTI
LINE

<code>LL_EXTI_LINE_0</code>	Extended line 0
<code>LL_EXTI_LINE_1</code>	Extended line 1
<code>LL_EXTI_LINE_2</code>	Extended line 2
<code>LL_EXTI_LINE_3</code>	Extended line 3
<code>LL_EXTI_LINE_4</code>	Extended line 4
<code>LL_EXTI_LINE_5</code>	Extended line 5
<code>LL_EXTI_LINE_6</code>	Extended line 6
<code>LL_EXTI_LINE_7</code>	Extended line 7
<code>LL_EXTI_LINE_8</code>	Extended line 8
<code>LL_EXTI_LINE_9</code>	Extended line 9
<code>LL_EXTI_LINE_10</code>	Extended line 10
<code>LL_EXTI_LINE_11</code>	Extended line 11

<code>LL_EXTI_LINE_12</code>	Extended line 12
<code>LL_EXTI_LINE_13</code>	Extended line 13
<code>LL_EXTI_LINE_14</code>	Extended line 14
<code>LL_EXTI_LINE_15</code>	Extended line 15
<code>LL_EXTI_LINE_16</code>	Extended line 16
<code>LL_EXTI_LINE_17</code>	Extended line 17
<code>LL_EXTI_LINE_18</code>	Extended line 18
<code>LL_EXTI_LINE_19</code>	Extended line 19
<code>LL_EXTI_LINE_20</code>	Extended line 20
<code>LL_EXTI_LINE_21</code>	Extended line 21
<code>LL_EXTI_LINE_22</code>	Extended line 22
<code>LL_EXTI_LINE_23</code>	Extended line 23
<code>LL_EXTI_LINE_25</code>	Extended line 25
<code>LL_EXTI_LINE_26</code>	Extended line 26
<code>LL_EXTI_LINE_27</code>	Extended line 27
<code>LL_EXTI_LINE_28</code>	Extended line 28
<code>LL_EXTI_LINE_31</code>	Extended line 31
<code>LL_EXTI_LINE_ALL_0_31</code>	All Extended line not reserved
<code>LL_EXTI_LINE_ALL</code>	All Extended line
<code>LL_EXTI_LINE_NONE</code>	None Extended line

Mode

<code>LL_EXTI_MODE_IT</code>	Interrupt Mode
<code>LL_EXTI_MODE_EVENT</code>	Event Mode
<code>LL_EXTI_MODE_IT_EVENT</code>	Interrupt & Event Mode

Edge Trigger

<code>LL_EXTI_TRIGGER_NONE</code>	No Trigger Mode
-----------------------------------	-----------------

LL_EXTI_TRIGGER_RISING Trigger Rising Mode

LL_EXTI_TRIGGER_FALLING Trigger Falling Mode

LL_EXTI_TRIGGER_RISING_FALLING Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

59 LL GPIO Generic Driver

59.1 GPIO Firmware driver registers structures

59.1.1 LL_GPIO_InitTypeDef

`LL_GPIO_InitTypeDef` is defined in the `stm32f0xx_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()`.

59.2 GPIO Firmware driver API description

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

59.2.1 Detailed description of functions

`LL_GPIO_SetPinMode`

Function name	<code>__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)</code>
----------------------	--

Function description	Configure gpio mode for a dedicated pin on dedicated port.
-----------------------------	--

Parameters	<ul style="list-style-type: none">• GPIOx: GPIO Port• Pin: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– LL_GPIO_MODE_INPUT– LL_GPIO_MODE_OUTPUT– LL_GPIO_MODE_ALTERNATE– LL_GPIO_MODE_ANALOG
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• MODER MODEy LL_GPIO_SetPinMode
	LL_GPIO_GetPinMode
Function name	<u>__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin)</u>
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_SetPinMode

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_SetPinOutputType

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	<ul style="list-style-type: none">• GPIOx: GPIO Port• Pin: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– LL_GPIO_SPEED_FREQ_LOW– LL_GPIO_SPEED_FREQ_MEDIUM– LL_GPIO_SPEED_FREQ_HIGH
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• 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

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_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	<ul style="list-style-type: none">• GPIOx: GPIO Port• Pin: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– LL_GPIO_PULL_NO– LL_GPIO_PULL_UP– LL_GPIO_PULL_DOWN
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• PUPDR PUPDy LL_GPIO_SetPinPull
LL_GPIO_GetPinPull	
Function name	<code>__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin)</code>
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_SetPinPull

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	<ul style="list-style-type: none">• GPIOx: GPIO Port• Pin: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• AFRL AFSELy LL_GPIO_SetAFPin_0_7
LL_GPIO_GetAFPin_0_7	
Function name	<code>__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin)</code>
Function description	Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.
Parameters	<ul style="list-style-type: none">• GPIOx: GPIO Port• Pin: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• AFRL AFSELy LL_GPIO_SetAFPin_0_7
	LL_GPIO_SetAFPin_8_15
Function name	<pre>__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)</pre>
Function description	Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	<ul style="list-style-type: none">• GPIOx: GPIO Port• Pin: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name	<code>__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin)</code>
Function description	Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	<ul style="list-style-type: none">• GPIOx: GPIO Port• Pin: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• Possible values are from AF0 to AF7 depending on target.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name	<code>__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)</code>
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	<code>__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)</code>
Function description	Return if input data level for several pins of dedicated port is high or low.
Parameters	<ul style="list-style-type: none">GPIOx: GPIO PortPinMask: This parameter can be a combination of the following values:<ul style="list-style-type: none">– <code>LL_GPIO_PIN_0</code>– <code>LL_GPIO_PIN_1</code>– <code>LL_GPIO_PIN_2</code>– <code>LL_GPIO_PIN_3</code>– <code>LL_GPIO_PIN_4</code>– <code>LL_GPIO_PIN_5</code>– <code>LL_GPIO_PIN_6</code>– <code>LL_GPIO_PIN_7</code>– <code>LL_GPIO_PIN_8</code>– <code>LL_GPIO_PIN_9</code>– <code>LL_GPIO_PIN_10</code>– <code>LL_GPIO_PIN_11</code>– <code>LL_GPIO_PIN_12</code>– <code>LL_GPIO_PIN_13</code>– <code>LL_GPIO_PIN_14</code>– <code>LL_GPIO_PIN_15</code>– <code>LL_GPIO_PIN_ALL</code>
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name	<code>__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)</code>
Function description	Write output data register for the port.
Parameters	<ul style="list-style-type: none">GPIOx: GPIO PortPortValue: Level value for each pin of the port
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name	<code>__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (GPIO_TypeDef * GPIOx)</code>
---------------	---

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:

[**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	<ul style="list-style-type: none">• GPIOx: GPIO Port• PinMask: This parameter can be a combination of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• BRR BRy LL_GPIO_ResetOutputPin
	LL_GPIO_TogglePin
Function name	<u>__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)</u>
Function description	Toggle data value for several pin of dedicated port.
Parameters	<ul style="list-style-type: none">• GPIOx: GPIO Port• PinMask: This parameter can be a combination of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ODR ODy LL_GPIO_TogglePin
LL_GPIO_DelInit	
Function name	ErrorStatus LL_GPIO_DelInit (GPIO_TypeDef * GPIOx)
Function description	De-initialize GPIO registers (Registers restored to their default values).
Parameters	<ul style="list-style-type: none">• GPIOx: GPIO Port
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• GPIOx: GPIO Port• GPIO_InitStruct: pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content– ERROR: Not applicable
LL_GPIO_StructInit	
Function name	void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)
Function description	Set each LL_GPIO_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none">• GPIO_InitStruct: pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.
Return values	<ul style="list-style-type: none">• None:

59.3 GPIO Firmware driver defines

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

59.3.1 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_ALTERNA
TE` Select alternate function mode

`LL_GPIO_MODE_ANALOG` Select analog mode

Output Type

`LL_GPIO_OUTPUT_PUSH
ULL` Select push-pull as output type

`LL_GPIO_OUTPUT_OPEND
RAIN` 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_L` Select I/O low output speed
OW

`LL_GPIO_SPEED_FREQ_M` Select I/O medium output speed
EDIUM

`LL_GPIO_SPEED_FREQ_HI` Select I/O high output speed
GH

Common Write and read registers Macros

`LL_GPIO_WriteReg`

Description:

- Write a value in GPIO register.

Parameters:

- `_INSTANCE_`: GPIO Instance
- `_REG_`: Register to be written
- `_VALUE_`: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg**Description:**

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants**LL_GPIO_SPEED_LOW****LL_GPIO_SPEED_MEDIUM****LL_GPIO_SPEED_HIGH**

60 LL I2C Generic Driver

60.1 I2C Firmware driver registers structures

60.1.1 LL_I2C_InitTypeDef

`LL_I2C_InitTypeDef` is defined in the `stm32f0xx_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 = 0x0FThis 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 = 0x3FFThis 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()`.

60.2 I2C Firmware driver API description

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

60.2.1 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 (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	<ul style="list-style-type: none">• I2Cx: I2C Instance.• AnalogFilter: This parameter can be one of the following values:<ul style="list-style-type: none">– 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*t_{i2cclk}). 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]*t_{i2cclk}.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR1 ANFOFF LL_I2C_ConfigFilters• CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name	<u>__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)</u>
Function description	Configure Digital Noise Filter.
Parameters	<ul style="list-style-type: none">• 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*t_{i2cclk}). 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]*t_{i2cclk}.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name	<u>__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (I2C_TypeDef * I2Cx)</u>
Function description	Get the current Digital Noise Filter configuration.
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

LL_I2C_EnableAnalogFilter

Function name	<u>__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)</u>
----------------------	---

Function description	Enable Analog Noise Filter.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name	<u>__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)</u>
Function description	Disable Analog Noise Filter.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name	<u>__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (I2C_TypeDef * I2Cx)</u>
Function description	Check if Analog Noise Filter is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name	<u>__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)</u>
Function description	Enable DMA transmission requests.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">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 (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	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (I2C_TypeDef * I2Cx)</code>
Function description	Check if DMA reception requests are enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (I2C_TypeDef * I2Cx, uint32_t Direction)</code>
Function description	Get the data register address used for DMA transfer.
Parameters	<ul style="list-style-type: none">I2Cx: I2C InstanceDirection: This parameter can be one of the following values:<ul style="list-style-type: none">LL_I2C_DMA_REG_DATA_TRANSMITLL_I2C_DMA_REG_DATA_RECEIVE
Return values	<ul style="list-style-type: none">Address: of data register
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TXDR TXDATA LL_I2C_DMA_GetRegAddrRXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name	<code>__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)</code>
Function description	Enable Clock stretching.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name	<code>__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)</code>
Function description	Disable Clock stretching.

Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 NOSTRETCH LL_I2C_DisableClockStretching
LL_I2C_IsEnabledClockStretching	
Function name	<u>__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (I2C_TypeDef * I2Cx)</u>
Function description	Check if Clock stretching is enabled or disabled.
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching
LL_I2C_EnableSlaveByteControl	
Function name	<u>__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)</u>
Function description	Enable hardware byte control in slave mode.
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 SBC LL_I2C_EnableSlaveByteControl
LL_I2C_DisableSlaveByteControl	
Function name	<u>__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)</u>
Function description	Disable hardware byte control in slave mode.
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 SBC LL_I2C_DisableSlaveByteControl
LL_I2C_IsEnabledSlaveByteControl	
Function name	<u>__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (I2C_TypeDef * I2Cx)</u>

Function description	Check if hardware byte control in slave mode is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 SBC LL_I2C_IsEnabledSlaveByteControl
LL_I2C_EnableWakeUpFromStop	
Function name	<u>__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)</u>
Function description	Enable Wakeup from STOP.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.This bit can only be programmed when Digital Filter is disabled.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 WUPEN LL_I2C_EnableWakeUpFromStop
LL_I2C_DisableWakeUpFromStop	
Function name	<u>__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)</u>
Function description	Disable Wakeup from STOP.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 WUPEN LL_I2C_DisableWakeUpFromStop
LL_I2C_IsEnabledWakeUpFromStop	
Function name	<u>__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (I2C_TypeDef * I2Cx)</u>
Function description	Check if Wakeup from STOP is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).

Notes

- Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop

LL_I2C_EnableGeneralCall

Function name `__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)`

Function description Enable General Call.

Parameters

- I2Cx: I2C Instance.

Return values

- None:**

Notes

- When enabled the Address 0x00 is ACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name `__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)`

Function description Disable General Call.

Parameters

- I2Cx: I2C Instance.

Return values

- None:**

Notes

- When disabled the Address 0x00 is NACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name `__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (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	<code>__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)</code>
Function description	Configure the Master to operate in 7-bit or 10-bit addressing mode.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.AddressingMode: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_I2C_ADDRESSING_MODE_7BIT</code>– <code>LL_I2C_ADDRESSING_MODE_10BIT</code>
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Changing this bit is not allowed, when the START bit is set.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (I2C_TypeDef * I2Cx)</code>
Function description	Get the Master addressing mode.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_I2C_ADDRESSING_MODE_7BIT</code>– <code>LL_I2C_ADDRESSING_MODE_10BIT</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name	<code>__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)</code>
Function description	Set the Own Address1.
Parameters	<ul style="list-style-type: none">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:<ul style="list-style-type: none">– <code>LL_I2C_OWNADDRESS1_7BIT</code>– <code>LL_I2C_OWNADDRESS1_10BIT</code>
Return values	<ul style="list-style-type: none">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 (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	<ul style="list-style-type: none">• I2Cx: I2C Instance.• OwnAddress2: Value between Min_Data=0 and Max_Data=0x7F.• OwnAddrMask: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This action has no effect if own address2 is enabled.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• OAR2 OA2 LL_I2C_SetOwnAddress2• OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name	<code>__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)</code>
Function description	Enable acknowledge on Own Address2 match address.
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name	<code>__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)</code>
Function description	Disable acknowledge on Own Address2 match address.
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (I2C_TypeDef * I2Cx)</code>
Function description	Check if Own Address1 acknowledge is enabled or disabled.

Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name	<code>__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)</code>
Function description	Configure the SDA setup, hold time and the SCL high, low period.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.Timing: This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (I2C_TypeDef * I2Cx)</code>
Function description	Get the Timing Prescaler setting.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TIMINGR PRESC LL_I2C_GetTimingPrescaler
---	---

LL_I2C_GetClockLowPeriod

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (I2C_TypeDef * I2Cx)</code>
Function description	Get the SCL low period setting.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TIMINGR SCLL LL_I2C_GetClockLowPeriod
---	---

LL_I2C_GetClockHighPeriod

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (I2C_TypeDef * I2Cx)</code>
Function description	Get the SCL high period setting.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">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	<code>__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (I2C_TypeDef * I2Cx)</code>
Function description	Get the SDA hold time.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">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	<code>__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (I2C_TypeDef * I2Cx)</code>
Function description	Get the SDA setup time.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">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	<code>__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)</code>
Function description	Configure peripheral mode.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.PeripheralMode: This parameter can be one of the following values:<ul style="list-style-type: none">LL_I2C_MODE_I2CLL_I2C_MODE_SMBUS_HOSTLL_I2C_MODE_SMBUS_DEVICELL_I2C_MODE_SMBUS_DEVICE_ARP

Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR1 SMBHEN LL_I2C_SetModeCR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetMode (I2C_TypeDef * I2Cx)</code>
Function description	Get peripheral mode.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_I2C_MODE_I2CLL_I2C_MODE_SMBUS_HOSTLL_I2C_MODE_SMBUS_DEVICELL_I2C_MODE_SMBUS_DEVICE_ARP
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR1 SMBHEN LL_I2C_GetModeCR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name	<code>__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)</code>
Function description	Enable SMBus alert (Host or Device mode)
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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 driven 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:	<ul style="list-style-type: none">CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name	<code>__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)</code>
Function description	Disable SMBus alert (Host or Device mode)

Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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 driven (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:	<ul style="list-style-type: none">CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name	__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (I2C_TypeDef * I2Cx)
---------------	---

Function description Check if SMBus alert (Host or Device mode) is enabled or disabled.

Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
------------	---

Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
---------------	---

Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">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	<ul style="list-style-type: none">I2Cx: I2C Instance.
------------	---

Return values	<ul style="list-style-type: none">None:
---------------	---

Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">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	<ul style="list-style-type: none">I2Cx: I2C Instance.
------------	---

Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR1 PECEN LL_I2C_DisableSMBusPEC
LL_I2C_IsEnabledSMBusPEC	
Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (I2C_TypeDef * I2Cx)</code>
Function description	Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR1 PECEN LL_I2C_IsEnabledSMBusPEC
LL_I2C_ConfigSMBusTimeout	
Function name	<code>__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)</code>
Function description	Configure the SMBus Clock Timeout.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.TimeoutA: This parameter must be a value between Min_Data=0 and Max_Data=0FFF.TimeoutAMode: This parameter can be one of the following values:<ul style="list-style-type: none">LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOWLL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGHTimeoutB:
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeoutTIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeoutTIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name	<code>__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)</code>
Function description	Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.TimeoutA: This parameter must be a value between Min_Data=0 and Max_Data=0xFFFF.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (I2C_TypeDef * I2Cx)</code>
Function description	Get the SMBus Clock TimeoutA setting.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">Value: between Min_Data=0 and Max_Data=0xFFFF
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name	<code>__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)</code>
Function description	Set the SMBus Clock TimeoutA mode.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.TimeoutAMode: This parameter can be one of the following values:<ul style="list-style-type: none">LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOWLL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
Return values	<ul style="list-style-type: none">None:

Notes

- 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 (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

- 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=0xFFFF.

Return values

- None:**

Notes

- 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 (I2C_TypeDef * I2Cx)`

Function description Get the SMBus Extended Cumulative Clock TimeoutB setting.

Parameters

- I2Cx:** I2C Instance.

Return values	<ul style="list-style-type: none">• Value: between Min_Data=0 and Max_Data=0xFFFF
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• I2Cx: I2C Instance.• ClockTimeout: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_I2C_SMBUS_TIMEOUTA– LL_I2C_SMBUS_TIMEOUTB– LL_I2C_SMBUS_ALL_TIMEOUT
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• I2Cx: I2C Instance.• ClockTimeout: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_I2C_SMBUS_TIMEOUTA– LL_I2C_SMBUS_TIMEOUTB– LL_I2C_SMBUS_ALL_TIMEOUT
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout• TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name `__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (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

- 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 (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 (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 (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 (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 (I2C_TypeDef * I2Cx)**

Function description	Check if STOP detection interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">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	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">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	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 TCIE LL_I2C_DisableIT_TC
LL_I2C_IsEnabledIT_TC	
Function name	__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (I2C_TypeDef * I2Cx)
Function description	Check if Transfer Complete interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">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

- 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

- 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 (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	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Transmit data register empty flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Transmit interrupt flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Receive data register not empty flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (I2C_TypeDef * I2Cx)</code>
----------------------	---

Function description	Indicate the status of Address matched flag (slave mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">ISR ADDR LL_I2C_IsActiveFlag_ADDR
LL_I2C_IsActiveFlag_NACK	
Function name	<u>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (I2C_TypeDef * I2Cx)</u>
Function description	Indicate the status of Not Acknowledge received flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When a NACK is received after a byte transmission.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR NACKF LL_I2C_IsActiveFlag_NACK
LL_I2C_IsActiveFlag_STOP	
Function name	<u>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (I2C_TypeDef * I2Cx)</u>
Function description	Indicate the status of Stop detection flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When a Stop condition is detected.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR STOPF LL_I2C_IsActiveFlag_STOP
LL_I2C_IsActiveFlag_TC	
Function name	<u>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (I2C_TypeDef * I2Cx)</u>
Function description	Indicate the status of Transfer complete flag (master mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).

Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR TC LL_I2C_IsActiveFlag_TC
LL_I2C_IsActiveFlag_TCR	
Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Transfer complete flag (master mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR TCR LL_I2C_IsActiveFlag_TCR
LL_I2C_IsActiveFlag_BERR	
Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Bus error flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR BERR LL_I2C_IsActiveFlag_BERR
LL_I2C_IsActiveFlag_ARLO	
Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Arbitration lost flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When arbitration lost.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Overrun/Underrun flag (slave mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of SMBus PEC error flag in reception.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of SMBus Timeout detection flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of SMBus alert flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the status of Bus Busy flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">RESET: Clear default value. SET: When a Start condition is detected.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name	<code>__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)</code>
Function description	Clear Address Matched flag.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name	<code>__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)</code>
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

- 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

- 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

- 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 transferred, stretching SCL low.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (I2C_TypeDef * I2Cx)</code>
Function description	Check if automatic STOP condition is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name	<code>__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)</code>
Function description	Enable reload mode (master mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name	<code>__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)</code>
Function description	Disable reload mode (master mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (I2C_TypeDef * I2Cx)</code>
Function description	Check if reload mode is enabled or disabled.
Parameters	<ul style="list-style-type: none">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 (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	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">The master only sends the first 7 bits of 10bit address in Read direction.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (I2C_TypeDef * I2Cx)</code>
----------------------	---

Function description	Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name	<code>__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)</code>
----------------------	---

Function description	Configure the transfer direction (master mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.TransferRequest: This parameter can be one of the following values:<ul style="list-style-type: none">LL_I2C_REQUEST_WRITELL_I2C_REQUEST_READ
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (I2C_TypeDef * I2Cx)</code>
----------------------	---

Function description	Get the transfer direction requested (master mode).
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_I2C_REQUEST_WRITE– LL_I2C_REQUEST_READ
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• I2Cx: I2C Instance.• SlaveAddr: This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 SADD LL_I2C_SetSlaveAddr
LL_I2C_GetSlaveAddr	
Function name	__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (I2C_TypeDef * I2Cx)
Function description	Get the slave address programmed for transfer.
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x0 and Max_Data=0x3F
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• I2Cx: I2C Instance.• SlaveAddr: Specifies the slave address to be programmed.• SlaveAddrSize: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_I2C_ADDRSRSLAVE_7BIT– LL_I2C_ADDRSRSLAVE_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:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (I2C_TypeDef * I2Cx)</code>
Function description	Indicate the value of transfer direction (slave mode).
Parameters	<ul style="list-style-type: none">• I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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 (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_EnableSMBusPECCCompare

Function name `__STATIC_INLINE void LL_I2C_EnableSMBusPECCCompare (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

- 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 PECPBYTE LL_I2C_EnableSMBusPECCCompare

LL_I2C_IsEnabledSMBusPECCCompare

Function name `__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCCompare (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

- 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 PECPBYTE LL_I2C_IsEnabledSMBusPECCCompare

LL_I2C_GetSMBusPEC

Function name	<code>__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (I2C_TypeDef * I2Cx)</code>
Function description	Get the SMBus Packet Error byte calculated.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0xFF
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name	<code>__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (I2C_TypeDef * I2Cx)</code>
Function description	Read Receive Data register.
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name	<code>__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)</code>
Function description	Write in Transmit Data Register .
Parameters	<ul style="list-style-type: none">I2Cx: I2C Instance.Data: Value between Min_Data=0x00 and Max_Data=0xFF
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Init

Function name	<code>uint32_t LL_I2C_Init (I2C_TypeDef * I2Cx, LL_I2C_InitTypeDef * I2C_InitStruct)</code>
Function description	Initialize the I2C registers according to the specified parameters in I2C_InitStruct.
Parameters	<ul style="list-style-type: none">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_DelInit

Function name `uint32_t LL_I2C_DelInit (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:**

60.3 I2C Firmware driver defines

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

60.3.1 I2C

I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MO Master operates in 7-bit addressing mode.
DE_7BIT

LL_I2C_ADDRESSING_MO Master operates in 10-bit addressing mode.
DE_10BIT

Slave Address Length

LL_I2C_ADDRSOLVE_7BIT Slave Address in 7-bit.

LL_I2C_ADDRSOLVE_10BIT Slave Address in 10-bit.
T

Analog Filter Selection

LL_I2C_ANALOGFILTER_E Analog filter is enabled.
ENABLE

LL_I2C_ANALOGFILTER_D Analog filter is disabled.
ISABLE

Clear Flags Defines

<code>LL_I2C_ICR_ADDRCF</code>	Address Matched flag
<code>LL_I2C_ICR_NACKCF</code>	Not Acknowledge flag
<code>LL_I2C_ICR_STOPCF</code>	Stop detection flag
<code>LL_I2C_ICR_BERRCF</code>	Bus error flag
<code>LL_I2C_ICR_ARLOCF</code>	Arbitration Lost flag
<code>LL_I2C_ICR_OVRCF</code>	Overrun/Underrun flag
<code>LL_I2C_ICR_PECCF</code>	PEC error flag
<code>LL_I2C_ICR_TIMOUTCF</code>	Timeout detection flag
<code>LL_I2C_ICR_ALERTCF</code>	Alert flag

Read Write Direction

`LL_I2C_DIRECTION_WRIT` Write transfer request by master, slave enters receiver mode.
`E`

`LL_I2C_DIRECTION_READ` Read transfer request by master, slave enters transmitter mode.

DMA Register Data

`LL_I2C_DMA_REG_DATA_` Get address of data register used for transmission
`TRANSMIT`

`LL_I2C_DMA_REG_DATA_` Get address of data register used for reception
`RECEIVE`

Start And Stop Generation

`LL_I2C_GENERATE_NOST` Don't Generate Stop and Start condition.
`ARTSTOP`

`LL_I2C_GENERATE_STOP` Generate Stop condition (Size should be set to 0).

`LL_I2C_GENERATE_STAR` Generate Start for read request.
`T_READ`

`LL_I2C_GENERATE_STAR` Generate Start for write request.
`T_WRITE`

`LL_I2C_GENERATE_REST` Generate Restart for read request, slave 7Bit address.
`ART_7BIT_READ`

`LL_I2C_GENERATE_REST` Generate Restart for write request, slave 7Bit address.
`ART_7BIT_WRITE`

LL_I2C_GENERATE_RST Generate Restart for read request, slave 10Bit address.
ART_10BIT_READ

LL_I2C_GENERATE_RST Generate Restart for write request, slave 10Bit address.
ART_10BIT_WRITE

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_AU Enable SMBUS Automatic end mode with HW PEC comparison.
TOEND_NO_PEC

LL_I2C_MODE_SMBUS_SO Enable SMBUS Software end mode with HW PEC comparison.
FTEND_NO_PEC

LL_I2C_MODE_SMBUS_AU Enable SMBUS Automatic end mode with HW PEC comparison.
TOEND_WITH_PEC

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_ Only Address2 bits[7:5] are compared.
MASK04

LL_I2C_OWNADDRESS2_ Only Address2 bits[7:6] are compared.
MASK05

LL_I2C_OWNADDRESS2_ Only Address2 bits[7] are compared.
MASK06

LL_I2C_OWNADDRESS2_ No comparison is done. All Address2 are acknowledged.
MASK07

Peripheral Mode

LL_I2C_MODE_I2C I2C Master or Slave mode

LL_I2C_MODE_SMBUS_HO SMBus Host address acknowledge
ST

LL_I2C_MODE_SMBUS_DE SMBus Device default mode (Default address not acknowledge)
VICE

LL_I2C_MODE_SMBUS_DE SMBus Device Default address acknowledge
VICE_ARP

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_TIMEOUT TimeoutA is used to detect SCL low level timeout.
A_MODE_SCL_LOW

LL_I2C_SMBUS_TIMEOUT TimeoutA is used to detect both SCL and SDA high level timeout.
A_MODE_SDA_SCL_HIGH

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUT TimeoutA enable bit
A

LL_I2C_SMBUS_TIMEOUT TimeoutB (extended clock) enable bit
B

LL_I2C_SMBUS_ALL_TIME TimeoutA and TimeoutB (extended clock) enable bits
OUT

Convert SDA SCL timings

LL_I2C_CONVERT_TIMING **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=0xFF.
- _DATA_SETUP_TIME_**: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. ($tscldel = (SCLDEL+1)xtpresc$)
- _DATA_HOLD_TIME_**: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. ($tsdadel = SDADELxtpresc$)
- _CLOCK_HIGH_PERIOD_**: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. ($tschl = (SCLH+1)xtpresc$)
- _CLOCK_LOW_PERIOD_**: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. ($tschl = (SCLL+1)xtpresc$)

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

61 LL I2S Generic Driver

61.1 I2S Firmware driver registers structures

61.1.1 LL_I2S_InitTypeDef

`LL_I2S_InitTypeDef` is defined in the `stm32f0xx_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()`.

61.2 I2S Firmware driver API description

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

61.2.1 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	<code>__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (SPI_TypeDef * SPIx)</code>
Function description	Get I2S data frame length.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_I2S_DATAFORMAT_16BLL_I2S_DATAFORMAT_16B_EXTENDEDLL_I2S_DATAFORMAT_24BLL_I2S_DATAFORMAT_32B
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SCFGR DATLEN LL_I2S_GetDataFormatI2SCFGR CHLEN LL_I2S_GetDataFormat

LL_I2S_SetClockPolarity

Function name	<code>__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)</code>
Function description	Set I2S clock polarity.
Parameters	<ul style="list-style-type: none">SPIx: SPI InstanceClockPolarity: This parameter can be one of the following values:<ul style="list-style-type: none">LL_I2S_POLARITY_LOWLL_I2S_POLARITY_HIGH
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name	<code>__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (SPI_TypeDef * SPIx)</code>
Function description	Get I2S clock polarity.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_I2S_POLARITY_LOWLL_I2S_POLARITY_HIGH
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name	<code>__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)</code>
Function description	Set I2S standard protocol.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• Standard: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_I2S_STANDARD_PHILIPS– LL_I2S_STANDARD_MSB– LL_I2S_STANDARD_LSB– LL_I2S_STANDARD_PCM_SHORT– LL_I2S_STANDARD_PCM_LONG
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• I2SCFGR I2SSTD LL_I2S_SetStandard• I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name	<code>__STATIC_INLINE uint32_t LL_I2S_GetStandard (SPI_TypeDef * SPIx)</code>
Function description	Get I2S standard protocol.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• I2SCFGR I2SSTD LL_I2S_SetStandard• I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_SetTransferMode

Function name	<code>__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Mode)</code>
Function description	Set I2S transfer mode.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• Mode: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_I2S_MODE_SLAVE_TX– LL_I2S_MODE_SLAVE_RX– LL_I2S_MODE_MASTER_TX– LL_I2S_MODE_MASTER_RX

Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SCFGR I2SCFG LL_I2S_SetTransferMode
LL_I2S_GetTransferMode	
Function name	<code>__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (SPI_TypeDef * SPIx)</code>
Function description	Get I2S transfer mode.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_I2S_MODE_SLAVE_TXLL_I2S_MODE_SLAVE_RXLL_I2S_MODE_MASTER_TXLL_I2S_MODE_MASTER_RX
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SCFGR I2SCFG LL_I2S_SetTransferMode
LL_I2S_SetPrescalerLinear	
Function name	<code>__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint8_t PrescalerLinear)</code>
Function description	Set I2S linear prescaler.
Parameters	<ul style="list-style-type: none">SPIx: SPI InstancePrescalerLinear: Value between Min_Data=0x02 and Max_Data=0xFF
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SPR I2SDIV LL_I2S_SetPrescalerLinear
LL_I2S_GetPrescalerLinear	
Function name	<code>__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (SPI_TypeDef * SPIx)</code>
Function description	Get I2S linear prescaler.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">PrescalerLinear: Value between Min_Data=0x02 and Max_Data=0xFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SPR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name	<code>__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)</code>
Function description	Set I2S parity prescaler.
Parameters	<ul style="list-style-type: none">SPIx: SPI InstancePrescalerParity: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_I2S_PRESCALER_PARITY EVEN</code>– <code>LL_I2S_PRESCALER_PARITY ODD</code>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SPR ODD <code>LL_I2S_SetPrescalerParity</code>

LL_I2S_GetPrescalerParity

Function name	<code>__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (SPI_TypeDef * SPIx)</code>
Function description	Get I2S parity prescaler.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_I2S_PRESCALER_PARITY EVEN</code>– <code>LL_I2S_PRESCALER_PARITY ODD</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SPR ODD <code>LL_I2S_GetPrescalerParity</code>

LL_I2S_EnableMasterClock

Function name	<code>__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)</code>
Function description	Enable the master clock output (Pin MCK)
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">I2SPR MCKOE <code>LL_I2S_EnableMasterClock</code>

LL_I2S_DisableMasterClock

Function name	<code>__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)</code>
Function description	Disable the master clock output (Pin MCK)

Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• I2SPR MCKOE LL_I2S_DisableMasterClock
LL_I2S_IsEnabledMasterClock	
Function name	<u>__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (SPI_TypeDef * SPIx)</u>
Function description	Check if the master clock output (Pin MCK) is enabled.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• I2SPR MCKOE LL_I2S_IsEnabledMasterClock
LL_I2S_IsActiveFlag_RXNE	
Function name	<u>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)</u>
Function description	Check if Rx buffer is not empty.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR RXNE LL_I2S_IsActiveFlag_RXNE
LL_I2S_IsActiveFlag_TXE	
Function name	<u>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXE (SPI_TypeDef * SPIx)</u>
Function description	Check if Tx buffer is empty.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR TXE LL_I2S_IsActiveFlag_TXE
LL_I2S_IsActiveFlag_BSY	
Function name	<u>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_BSY (SPI_TypeDef * SPIx)</u>
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	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• SR CHSIDE LL_I2S_IsActiveFlag_CHSIDE

LL_I2S_ClearFlag_OVR

Function name	<code>__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)</code>
Function description	Clear overrun error flag.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR OVR LL_I2S_ClearFlag_OVR
---	---

LL_I2S_ClearFlag_UDR

Function name	<code>__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)</code>
Function description	Clear underrun error flag.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR UDR LL_I2S_ClearFlag_UDR
---	---

LL_I2S_ClearFlag_FRE

Function name	<code>__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)</code>
Function description	Clear frame format error flag.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR FRE LL_I2S_ClearFlag_FRE
---	---

LL_I2S_EnableIT_ERR

Function name	<code>__STATIC_INLINE void LL_I2S_EnableIT_ERR (SPI_TypeDef * SPIx)</code>
---------------	--

Function description	Enable error IT.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR2 ERRIE LL_I2S_EnableIT_ERR
LL_I2S_EnableIT_RXNE	
Function name	<u>__STATIC_INLINE void LL_I2S_EnableIT_RXNE (SPI_TypeDef * SPIx)</u>
Function description	Enable Rx buffer not empty IT.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 RXNEIE LL_I2S_EnableIT_RXNE
LL_I2S_EnableIT_TXE	
Function name	<u>__STATIC_INLINE void LL_I2S_EnableIT_TXE (SPI_TypeDef * SPIx)</u>
Function description	Enable Tx buffer empty IT.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 TXEIE LL_I2S_EnableIT_TXE
LL_I2S_DisableIT_ERR	
Function name	<u>__STATIC_INLINE void LL_I2S_DisableIT_ERR (SPI_TypeDef * SPIx)</u>
Function description	Disable error IT.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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 IP 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).

61.3 I2S Firmware driver defines

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

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_DEFA Audio Freq not specified. Register I2SDIV = 2
ULT

Data format

LL_I2S_DATAFORMAT_16B Data length 16 bits, Channel lenght 16bit

LL_I2S_DATAFORMAT_16B_EXTENDED Data length 16 bits, Channel lenght 32bit

LL_I2S_DATAFORMAT_24B Data length 24 bits, Channel lenght 32bit

LL_I2S_DATAFORMAT_32B Data length 16 bits, Channel lenght 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

MCLK Output

LL_I2S_MCLK_OUTPUT_DI_SABLE Master clock output is disabled

LL_I2S_MCLK_OUTPUT_E_NABLE 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_T_X Master Tx configuration

LL_I2S_MODE_MASTER_R_X 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

Prescaler Factor

LL_I2S_PRESCALER_PARY_EVEN Odd factor: Real divider value is = I2SDIV * 2

LL_I2S_PRESCALER_PARRY_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

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

62 LL IWDG Generic Driver

62.1 IWDG Firmware driver API description

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

62.1.1 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	<code>__STATIC_INLINE void LL_IWDG_DisableWriteAccess (IWDG_TypeDef * IWDGx)</code>
Function description	Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
Parameters	<ul style="list-style-type: none">IWDGx: IWDG Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name	<code>__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)</code>
Function description	Select the prescaler of the IWDG.
Parameters	<ul style="list-style-type: none">IWDGx: IWDG InstancePrescaler: This parameter can be one of the following values:<ul style="list-style-type: none">LL_IWDG_PRESCALER_4LL_IWDG_PRESCALER_8LL_IWDG_PRESCALER_16LL_IWDG_PRESCALER_32LL_IWDG_PRESCALER_64LL_IWDG_PRESCALER_128LL_IWDG_PRESCALER_256
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name	<code>__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (IWDG_TypeDef * IWDGx)</code>
Function description	Get the selected prescaler of the IWDG.
Parameters	<ul style="list-style-type: none">IWDGx: IWDG Instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• PR PR LL_IWDG_GetPrescaler
	LL_IWDG_SetReloadCounter
Function name	<u>__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)</u>
Function description	Specify the IWDG down-counter reload value.
Parameters	<ul style="list-style-type: none">• IWDGx: IWDG Instance• Counter: Value between Min_Data=0 and Max_Data=0xFFFF
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RLR RL LL_IWDG_SetReloadCounter
	LL_IWDG_GetReloadCounter
Function name	<u>__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (IWDG_TypeDef * IWDGx)</u>
Function description	Get the specified IWDG down-counter reload value.
Parameters	<ul style="list-style-type: none">• IWDGx: IWDG Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RLR RL LL_IWDG_GetReloadCounter
	LL_IWDG_SetWindow
Function name	<u>__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)</u>
Function description	Specify high limit of the window value to be compared to the down-counter.
Parameters	<ul style="list-style-type: none">• IWDGx: IWDG Instance• Window: Value between Min_Data=0 and Max_Data=0xFFFF
Return values	<ul style="list-style-type: none">• 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=0xFFFF

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_SetWindow

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 WVU LL_IWDG_IsReady
- SR RVU LL_IWDG_IsReady

62.2 IWDG Firmware driver defines

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

62.2.1 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
8

LL_IWDG_PRESCALER_256 Divider by 256
6

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

63 LL PWR Generic Driver

63.1 PWR Firmware driver API description

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

63.1.1 Detailed description of functions

`LL_PWR_EnableBkUpAccess`

Function name `__STATIC_INLINE void LL_PWR_EnableBkUpAccess (void)`

Function description Enable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

`LL_PWR_DisableBkUpAccess`

Function name `__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)`

Function description Disable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

`LL_PWR_IsEnabledBkUpAccess`

Function name `__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess (void)`

Function description Check if the backup domain is enabled.

Return values

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

Reference Manual to LL API cross reference:

`LL_PWR_SetRegulModeDS`

Function name `__STATIC_INLINE void LL_PWR_SetRegulModeDS (uint32_t RegulMode)`

Function description Set voltage Regulator mode during deep sleep mode.

Parameters

- **RegulMode:** This parameter can be one of the following values:
 - `LL_PWR_REGU_DSMODE_MAIN`
 - `LL_PWR_REGU_DSMODE_LOW_POWER`

Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR LPDS LL_PWR_SetRegulModeDS
LL_PWR_GetRegulModeDS	
Function name	<code>__STATIC_INLINE uint32_t LL_PWR_GetRegulModeDS (void)</code>
Function description	Get voltage Regulator mode during deep sleep mode.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_PWR_REGU_DSMODE_MAINLL_PWR_REGU_DSMODE_LOW_POWER
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR LPDS LL_PWR_GetRegulModeDS
LL_PWR_SetPowerMode	
Function name	<code>__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t PDMode)</code>
Function description	Set Power Down mode when CPU enters deepsleep.
Parameters	<ul style="list-style-type: none">PDMode: This parameter can be one of the following values:<ul style="list-style-type: none">LL_PWR_MODE_STOP_MAINREGULL_PWR_MODE_STOP_LPREGULL_PWR_MODE_STANDBY
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR PDDS LL_PWR_SetPowerModeCR LPDS LL_PWR_SetPowerMode
LL_PWR_GetPowerMode	
Function name	<code>__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void)</code>
Function description	Get Power Down mode when CPU enters deepsleep.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_PWR_MODE_STOP_MAINREGULL_PWR_MODE_STOP_LPREGULL_PWR_MODE_STANDBY
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR PDDS LL_PWR_GetPowerModeCR LPDS LL_PWR_GetPowerMode

LL_PWR_SetPVDLevel

Function name	<code>__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)</code>
Function description	Configure the voltage threshold detected by the Power Voltage Detector.
Parameters	<ul style="list-style-type: none">PVDLevel: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_PWR_PVDLEVEL_0</code>– <code>LL_PWR_PVDLEVEL_1</code>– <code>LL_PWR_PVDLEVEL_2</code>– <code>LL_PWR_PVDLEVEL_3</code>– <code>LL_PWR_PVDLEVEL_4</code>– <code>LL_PWR_PVDLEVEL_5</code>– <code>LL_PWR_PVDLEVEL_6</code>– <code>LL_PWR_PVDLEVEL_7</code>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR PLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name	<code>__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)</code>
Function description	Get the voltage threshold detection.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_PWR_PVDLEVEL_0</code>– <code>LL_PWR_PVDLEVEL_1</code>– <code>LL_PWR_PVDLEVEL_2</code>– <code>LL_PWR_PVDLEVEL_3</code>– <code>LL_PWR_PVDLEVEL_4</code>– <code>LL_PWR_PVDLEVEL_5</code>– <code>LL_PWR_PVDLEVEL_6</code>– <code>LL_PWR_PVDLEVEL_7</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR PLS LL_PWR_GetPVDLevel

LL_PWR_EnablePVD

Function name	<code>__STATIC_INLINE void LL_PWR_EnablePVD (void)</code>
Function description	Enable Power Voltage Detector.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR PVDE LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name **__STATIC_INLINE void LL_PWR_DisablePVD (void)**

Function description Disable Power Voltage Detector.

Return values • **None:**

**Reference Manual to LL
API cross reference:** • CR 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.

Return values • **State:** of bit (1 or 0).

**Reference Manual to LL
API cross reference:** • CR PVDE LL_PWR_IsEnabledPVD

LL_PWR_EnableWakeUpPin

Function name **__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)**

Function description Enable the WakeUp PINx functionality.

Parameters • **WakeUpPin:** This parameter can be one of the following values:

- LL_PWR_WAKEUP_PIN1
- LL_PWR_WAKEUP_PIN2
- LL_PWR_WAKEUP_PIN3 (*)
- LL_PWR_WAKEUP_PIN4 (*)
- LL_PWR_WAKEUP_PIN5 (*)
- LL_PWR_WAKEUP_PIN6 (*)
- LL_PWR_WAKEUP_PIN7 (*)
- LL_PWR_WAKEUP_PIN8 (*)

(*) not available on all devices

Return values • **None:**

Reference Manual to LL API cross reference:

- CSR EWUP1 LL_PWR_EnableWakeUpPin
- CSR EWUP2 LL_PWR_EnableWakeUpPin
- CSR EWUP3 LL_PWR_EnableWakeUpPin
- CSR EWUP4 LL_PWR_EnableWakeUpPin
- CSR EWUP5 LL_PWR_EnableWakeUpPin
- CSR EWUP6 LL_PWR_EnableWakeUpPin
- CSR EWUP7 LL_PWR_EnableWakeUpPin
- CSR EWUP8 LL_PWR_EnableWakeUpPin

LL_PWR_DisableWakeUpPin

Function name	<code>__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)</code>
Function description	Disable the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none">• WakeUpPin: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_PWR_WAKEUP_PIN1</code>– <code>LL_PWR_WAKEUP_PIN2</code>– <code>LL_PWR_WAKEUP_PIN3 (*)</code>– <code>LL_PWR_WAKEUP_PIN4 (*)</code>– <code>LL_PWR_WAKEUP_PIN5 (*)</code>– <code>LL_PWR_WAKEUP_PIN6 (*)</code>– <code>LL_PWR_WAKEUP_PIN7 (*)</code>– <code>LL_PWR_WAKEUP_PIN8 (*)</code> <p>(*) not available on all devices</p>
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- CSR EWUP1 LL_PWR_DisableWakeUpPin
- CSR EWUP2 LL_PWR_DisableWakeUpPin
- CSR EWUP3 LL_PWR_DisableWakeUpPin
- CSR EWUP4 LL_PWR_DisableWakeUpPin
- CSR EWUP5 LL_PWR_DisableWakeUpPin
- CSR EWUP6 LL_PWR_DisableWakeUpPin
- CSR EWUP7 LL_PWR_DisableWakeUpPin
- CSR EWUP8 LL_PWR_DisableWakeUpPin

LL_PWR_IsEnabledWakeUpPin

Function name	<code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)</code>
Function description	Check if the WakeUp PINx functionality is enabled.
Parameters	<ul style="list-style-type: none">• WakeUpPin: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_PWR_WAKEUP_PIN1– LL_PWR_WAKEUP_PIN2– LL_PWR_WAKEUP_PIN3 (*)– LL_PWR_WAKEUP_PIN4 (*)– LL_PWR_WAKEUP_PIN5 (*)– LL_PWR_WAKEUP_PIN6 (*)– LL_PWR_WAKEUP_PIN7 (*)– LL_PWR_WAKEUP_PIN8 (*) <p>(*) not available on all devices</p>
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).

**Reference Manual to LL
API cross reference:**

- CSR EWUP1 LL_PWR_IsEnabledWakeUpPin
- CSR EWUP2 LL_PWR_IsEnabledWakeUpPin
- CSR EWUP3 LL_PWR_IsEnabledWakeUpPin
- CSR EWUP4 LL_PWR_IsEnabledWakeUpPin
- CSR EWUP5 LL_PWR_IsEnabledWakeUpPin
- CSR EWUP6 LL_PWR_IsEnabledWakeUpPin
- CSR EWUP7 LL_PWR_IsEnabledWakeUpPin
- CSR EWUP8 LL_PWR_IsEnabledWakeUpPin

LL_PWR_IsActiveFlag_WU

Function name `__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU (void)`

Function description Get Wake-up Flag.

Return values • **State:** of bit (1 or 0).

**Reference Manual to LL
API cross reference:** • CSR WUF LL_PWR_IsActiveFlag_WU

LL_PWR_IsActiveFlag_SB

Function name `__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void)`

Function description Get Standby Flag.

Return values • **State:** of bit (1 or 0).

**Reference Manual to LL
API cross reference:** • CSR SBF LL_PWR_IsActiveFlag_SB

LL_PWR_IsActiveFlag_PVDO

Function name `__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)`

Function description Indicate whether VDD voltage is below the selected PVD threshold.

Return values • **State:** of bit (1 or 0).

**Reference Manual to LL
API cross reference:** • CSR PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_IsActiveFlag_VREFINTRDY

Function name `__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VREFINTRDY (void)`

Function description	Get Internal Reference Vreflnt Flag.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CSR VREFINTRDYF LL_PWR_IsActiveFlag_VREFINTRDY

LL_PWR_ClearFlag_SB

Function name	__STATIC_INLINE void LL_PWR_ClearFlag_SB (void)
----------------------	---

Function description	Clear Standby Flag.
-----------------------------	---------------------

Return values	<ul style="list-style-type: none">None:
----------------------	--

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR CSBF LL_PWR_ClearFlag_SB
--	---

LL_PWR_ClearFlag_WU

Function name	__STATIC_INLINE void LL_PWR_ClearFlag_WU (void)
----------------------	---

Function description	Clear Wake-up Flags.
-----------------------------	----------------------

Return values	<ul style="list-style-type: none">None:
----------------------	--

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR CWUF LL_PWR_ClearFlag_WU
--	---

LL_PWR_DeInit

Function name	ErrorStatus LL_PWR_DeInit (void)
----------------------	--

Function description	De-initialize the PWR registers to their default reset values.
-----------------------------	--

Return values	<ul style="list-style-type: none">An: ErrorStatus enumeration value:<ul style="list-style-type: none">SUCCESS: PWR registers are de-initializedERROR: not applicable
----------------------	--

63.2 PWR Firmware driver defines

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

63.2.1 PWR

PWR

Clear Flags Defines

LL_PWR_CR_CSBF	Clear standby flag
-----------------------	--------------------

LL_PWR_CR_CWUF	Clear wakeup flag
-----------------------	-------------------

Get Flags Defines

LL_PWR_CSR_WUF	Wakeup flag
LL_PWR_CSR_SBF	Standby flag
LL_PWR_CSR_PVDO	Power voltage detector output flag
LL_PWR_CSR_VREFINTRD_YF	VREFINT ready flag
LL_PWR_CSR_EWUP1	Enable WKUP pin 1
LL_PWR_CSR_EWUP2	Enable WKUP pin 2
LL_PWR_CSR_EWUP3	Enable WKUP pin 3
LL_PWR_CSR_EWUP4	Enable WKUP pin 4
LL_PWR_CSR_EWUP5	Enable WKUP pin 5
LL_PWR_CSR_EWUP6	Enable WKUP pin 6
LL_PWR_CSR_EWUP7	Enable WKUP pin 7
LL_PWR_CSR_EWUP8	Enable WKUP pin 8

Mode Power

[LL_PWR_MODE_STOP_MA](#) Enter Stop mode when the CPU enters deepsleep
INREGU

[LL_PWR_MODE_STOP_LP](#) Enter Stop mode (with low power Regulator ON) when the CPU enters deepsleep
REGU

[LL_PWR_MODE_STANDBY](#) Enter Standby mode when the CPU enters deepsleep

Power Voltage Detector Level

LL_PWR_PVDLEVEL_0	Voltage threshold 0
LL_PWR_PVDLEVEL_1	Voltage threshold 1
LL_PWR_PVDLEVEL_2	Voltage threshold 2
LL_PWR_PVDLEVEL_3	Voltage threshold 3
LL_PWR_PVDLEVEL_4	Voltage threshold 4
LL_PWR_PVDLEVEL_5	Voltage threshold 5
LL_PWR_PVDLEVEL_6	Voltage threshold 6
LL_PWR_PVDLEVEL_7	Voltage threshold 7

Regulator Mode In Deep Sleep Mode

LL_PWR_REGU_DSMODE_MAIN Voltage Regulator in main mode during deepsleep mode

LL_PWR_REGU_DSMODE_LOW_POWER Voltage Regulator in low-power mode during deepsleep mode

Wakeup Pins

LL_PWR_WAKEUP_PIN1 WKUP pin 1 : PA0

LL_PWR_WAKEUP_PIN2 WKUP pin 2 : PC13

LL_PWR_WAKEUP_PIN3 WKUP pin 3 : PE6 or PA2 according to device

LL_PWR_WAKEUP_PIN4 WKUP pin 4 : LLG TBD

LL_PWR_WAKEUP_PIN5 WKUP pin 5 : LLG TBD

LL_PWR_WAKEUP_PIN6 WKUP pin 6 : LLG TBD

LL_PWR_WAKEUP_PIN7 WKUP pin 7 : LLG TBD

LL_PWR_WAKEUP_PIN8 WKUP pin 8 : LLG TBD

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

64 LL RCC Generic Driver

64.1 RCC Firmware driver registers structures

64.1.1 LL_RCC_ClocksTypeDef

`LL_RCC_ClocksTypeDef` is defined in the `stm32f0xx_ll_rcc.h`

Data Fields

- `uint32_t SYSCLK_Frequency`
- `uint32_t HCLK_Frequency`
- `uint32_t PCLK1_Frequency`

Field Documentation

- `uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency`
SYSCLK clock frequency
- `uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency`
HCLK clock frequency
- `uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency`
PCLK1 clock frequency

64.2 RCC Firmware driver API description

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

64.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name `__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)`

Function description Enable the Clock Security System.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSSON `LL_RCC_HSE_EnableCSS`

LL_RCC_HSE_DisableCSS

Function name `__STATIC_INLINE void LL_RCC_HSE_DisableCSS (void)`

Function description Disable the Clock Security System.

Return values

- **None:**

Notes

- Cannot be disabled in HSE is ready (only by hardware)

Reference Manual to LL API cross reference:

- CR CSSON `LL_RCC_HSE_DisableCSS`

LL_RCC_HSE_EnableBypass

Function name **__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)**

Function description Enable HSE external oscillator (HSE Bypass)

Return values • **None:**

Reference Manual to LL API cross reference: • CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name **__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)**

Function description Disable HSE external oscillator (HSE Bypass)

Return values • **None:**

Reference Manual to LL API cross reference: • CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name **__STATIC_INLINE void LL_RCC_HSE_Enable (void)**

Function description Enable HSE crystal oscillator (HSE ON)

Return values • **None:**

Reference Manual to LL API cross reference: • CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name **__STATIC_INLINE void LL_RCC_HSE_Disable (void)**

Function description Disable HSE crystal oscillator (HSE ON)

Return values • **None:**

Reference Manual to LL API cross reference: • CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name **__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)**

Function description Check if HSE oscillator Ready.

Return values • **State:** of bit (1 or 0).

[Reference Manual to LL API cross reference:](#)

- CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSI_Enable

Function name `__STATIC_INLINE void LL_RCC_HSI_Enable (void)`

Function description Enable HSI oscillator.

Return values

- **None:**

[Reference Manual to LL API cross reference:](#)

- CR HSION LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name `__STATIC_INLINE void LL_RCC_HSI_Disable (void)`

Function description Disable HSI oscillator.

Return values

- **None:**

[Reference Manual to LL API cross reference:](#)

- CR HSION LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name `__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)`

Function description Check if HSI clock is ready.

Return values

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

[Reference Manual to LL API cross reference:](#)

- CR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_GetCalibration

Function name `__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)`

Function description Get HSI Calibration value.

Return values

- **Between:** Min_Data = 0x00 and Max_Data = 0xFF

Notes

- When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value

[Reference Manual to LL API cross reference:](#)

- CR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name `__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)`

Function description	Set HSI Calibration trimming.
Parameters	<ul style="list-style-type: none">Value: between Min_Data = 0x00 and Max_Data = 0x1F
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">user-programmable trimming value that is added to the HSICALDefault value is 16, which, when added to the HSICAL value, should trim the HSI to 16 MHz +/- 1 %
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR HSITRIM LL_RCC_HSI_SetCalibTrimming

[LL_RCC_HSI_GetCalibTrimming](#)

Function name	<code>__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)</code>
Function description	Get HSI Calibration trimming.
Return values	<ul style="list-style-type: none">Between: Min_Data = 0x00 and Max_Data = 0x1F
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR HSITRIM LL_RCC_HSI_GetCalibTrimming

[LL_RCC_HSI48_Enable](#)

Function name	<code>__STATIC_INLINE void LL_RCC_HSI48_Enable (void)</code>
Function description	Enable HSI48.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 HSI48ON LL_RCC_HSI48_Enable

[LL_RCC_HSI48_Disable](#)

Function name	<code>__STATIC_INLINE void LL_RCC_HSI48_Disable (void)</code>
Function description	Disable HSI48.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 HSI48ON LL_RCC_HSI48_Disable

[LL_RCC_HSI48_IsReady](#)

Function name	<code>__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void)</code>
Function description	Check if HSI48 oscillator Ready.

Return values

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

Reference Manual to LL API cross reference:

- CR2 HSI48RDY LL_RCC_HSI48_IsReady

LL_RCC_HSI48_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_GetCalibration (void)

Function description

Get HSI48 Calibration value.

Return values

- **Between:** Min_Data = 0x00 and Max_Data = 0xFF

Reference Manual to LL API cross reference:

- CR2 HSI48CAL LL_RCC_HSI48_GetCalibration

LL_RCC_HSI14_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI14_Enable (void)

Function description

Enable HSI14.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 HSI14ON LL_RCC_HSI14_Enable

LL_RCC_HSI14_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI14_Disable (void)

Function description

Disable HSI14.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 HSI14ON LL_RCC_HSI14_Disable

LL_RCC_HSI14_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI14_IsReady (void)

Function description

Check if HSI14 oscillator Ready.

Return values

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

Reference Manual to LL API cross reference:

- CR2 HSI14RDY LL_RCC_HSI14_IsReady

LL_RCC_HSI14_EnableADCControl

Function name

__STATIC_INLINE void LL_RCC_HSI14_EnableADCControl (void)

Function description ADC interface can turn on the HSI14 oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

LL_RCC_HSI14_DisableADCControl

Function name `__STATIC_INLINE void LL_RCC_HSI14_DisableADCControl (void)`

Function description ADC interface can not turn on the HSI14 oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

LL_RCC_HSI14_SetCalibTrimming

Function name `__STATIC_INLINE void LL_RCC_HSI14_SetCalibTrimming (uint32_t Value)`

Function description Set HSI14 Calibration trimming.

Parameters

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

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the HSI14CAL
- Default value is 16, which, when added to the HSI14CAL value, should trim the HSI14 to 14 MHz +/- 1 %

Reference Manual to LL API cross reference:

LL_RCC_HSI14_GetCalibTrimming

Function name `__STATIC_INLINE uint32_t LL_RCC_HSI14_GetCalibTrimming (void)`

Function description Get HSI14 Calibration value.

Return values

- **Between:** Min_Data = 0x00 and Max_Data = 0x1F

Notes

- When HSI14TRIM is written, HSI14CAL is updated with the sum of HSI14TRIM and the factory trim value

Reference Manual to LL API cross reference:

LL_RCC_HSI14_GetCalibration

Function name `__STATIC_INLINE uint32_t LL_RCC_HSI14_GetCalibration (void)`

Function description	Get HSI14 Calibration trimming.
Return values	<ul style="list-style-type: none">Between: Min_Data = 0x00 and Max_Data = 0x1F
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 HSI14CAL LL_RCC_HSI14_GetCalibration

LL_RCC_LSE_Enable

Function name	<u>__STATIC_INLINE void LL_RCC_LSE_Enable (void)</u>
Function description	Enable Low Speed External (LSE) crystal.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name	<u>__STATIC_INLINE void LL_RCC_LSE_Disable (void)</u>
Function description	Disable Low Speed External (LSE) crystal.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_EnableBypass

Function name	<u>__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)</u>
Function description	Enable external clock source (LSE bypass).
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name	<u>__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)</u>
Function description	Disable external clock source (LSE bypass).
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SetDriveCapability

Function name	<code>__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)</code>
Function description	Set LSE oscillator drive capability.
Parameters	<ul style="list-style-type: none">LSEDrive: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_RCC_LSEDRIVE_LOW</code>– <code>LL_RCC_LSEDRIVE_MEDIUMLOW</code>– <code>LL_RCC_LSEDRIVE_MEDIUMHIGH</code>– <code>LL_RCC_LSEDRIVE_HIGH</code>
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">The oscillator is in Xtal mode when it is not in bypass mode.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name	<code>__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void)</code>
Function description	Get LSE oscillator drive capability.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_RCC_LSEDRIVE_LOW</code>– <code>LL_RCC_LSEDRIVE_MEDIUMLOW</code>– <code>LL_RCC_LSEDRIVE_MEDIUMHIGH</code>– <code>LL_RCC_LSEDRIVE_HIGH</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_IsReady

Function name	<code>__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)</code>
Function description	Check if LSE oscillator Ready.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSI_Enable

Function name	<code>__STATIC_INLINE void LL_RCC_LSI_Enable (void)</code>
Function description	Enable LSI Oscillator.

- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CSR 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:**
- CSR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name `__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)`

Function description Check if LSI is Ready.

- Return values**
- **State:** of bit (1 or 0).
- Reference Manual to LL API cross reference:**
- CSR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_SetSysClkSource

Function name `__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)`

Function description Configure the system clock source.

- Parameters**
- **Source:** This parameter can be one of the following values:
 - `LL_RCC_SYS_CLKSOURCE_HSI`
 - `LL_RCC_SYS_CLKSOURCE_HSE`
 - `LL_RCC_SYS_CLKSOURCE_PLL`
 - `LL_RCC_SYS_CLKSOURCE_HSI48 (*)`
- (*) value not defined in all devices

- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CFGR SW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name `__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)`

Function description Get the system clock source.

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RCC_SYS_CLKSOURCE_STATUS_HSI– LL_RCC_SYS_CLKSOURCE_STATUS_HSE– LL_RCC_SYS_CLKSOURCE_STATUS_PLL– LL_RCC_SYS_CLKSOURCE_STATUS_HSI48 (*)
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR SWS LL_RCC_GetSysClkSource
	LL_RCC_SetAHBPrescaler
Function name	<u>__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)</u>
Function description	Set AHB prescaler.
Parameters	<ul style="list-style-type: none">• Prescaler: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RCC_SYSCLK_DIV_1– LL_RCC_SYSCLK_DIV_2– LL_RCC_SYSCLK_DIV_4– LL_RCC_SYSCLK_DIV_8– LL_RCC_SYSCLK_DIV_16– LL_RCC_SYSCLK_DIV_64– LL_RCC_SYSCLK_DIV_128– LL_RCC_SYSCLK_DIV_256– LL_RCC_SYSCLK_DIV_512
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR HPRE LL_RCC_SetAHBPrescaler
	LL_RCC_SetAPB1Prescaler
Function name	<u>__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)</u>
Function description	Set APB1 prescaler.
Parameters	<ul style="list-style-type: none">• Prescaler: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RCC_APB1_DIV_1– LL_RCC_APB1_DIV_2– LL_RCC_APB1_DIV_4– LL_RCC_APB1_DIV_8– LL_RCC_APB1_DIV_16
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR PPRE LL_RCC_SetAPB1Prescaler

LL_RCC_GetAHBPrescaler

Function name	<code>__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)</code>
Function description	Get AHB prescaler.
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_RCC_SYSCLK_DIV_1</code>– <code>LL_RCC_SYSCLK_DIV_2</code>– <code>LL_RCC_SYSCLK_DIV_4</code>– <code>LL_RCC_SYSCLK_DIV_8</code>– <code>LL_RCC_SYSCLK_DIV_16</code>– <code>LL_RCC_SYSCLK_DIV_64</code>– <code>LL_RCC_SYSCLK_DIV_128</code>– <code>LL_RCC_SYSCLK_DIV_256</code>– <code>LL_RCC_SYSCLK_DIV_512</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• <code>CFGR HPRE</code> <code>LL_RCC_GetAHBPrescaler</code>

LL_RCC_GetAPB1Prescaler

Function name	<code>__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)</code>
Function description	Get APB1 prescaler.
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_RCC_APB1_DIV_1</code>– <code>LL_RCC_APB1_DIV_2</code>– <code>LL_RCC_APB1_DIV_4</code>– <code>LL_RCC_APB1_DIV_8</code>– <code>LL_RCC_APB1_DIV_16</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• <code>CFGR PPRE</code> <code>LL_RCC_GetAPB1Prescaler</code>

LL_RCC_ConfigMCO

Function name	<code>__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)</code>
Function description	Configure MCOx.

- | | |
|-------------------|--|
| Parameters | <ul style="list-style-type: none">• MCOxSource: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_RCC_MCO1SOURCE_NOCLOCK- LL_RCC_MCO1SOURCE_HSI14- LL_RCC_MCO1SOURCE_SYSCLK- LL_RCC_MCO1SOURCE_HSI- LL_RCC_MCO1SOURCE_HSE- LL_RCC_MCO1SOURCE_LSI- LL_RCC_MCO1SOURCE_LSE- LL_RCC_MCO1SOURCE_HSI48 (*)- LL_RCC_MCO1SOURCE_PLLCLK (*)- LL_RCC_MCO1SOURCE_PLLCLK_DIV_2(*) value not defined in all devices• MCOxPrescaler: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_RCC_MCO1_DIV_1- LL_RCC_MCO1_DIV_2 (*)- LL_RCC_MCO1_DIV_4 (*)- LL_RCC_MCO1_DIV_8 (*)- LL_RCC_MCO1_DIV_16 (*)- LL_RCC_MCO1_DIV_32 (*)- LL_RCC_MCO1_DIV_64 (*)- LL_RCC_MCO1_DIV_128 (*)(*) value not defined in all devices |
|-------------------|--|

- | | |
|----------------------|--|
| Return values | <ul style="list-style-type: none">• None: |
|----------------------|--|

- | | |
|--|---|
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none">• CFGR MCO LL_RCC_ConfigMCO• CFGR MCOPRE LL_RCC_ConfigMCO• CFGR PLLNODIV LL_RCC_ConfigMCO |
|--|---|

LL_RCC_SetUSARTClockSource

Function name	<u>__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t USARTTxSource)</u>
----------------------	--

Function description	Configure USARTTx clock source.
-----------------------------	---------------------------------

- | | |
|-------------------|--|
| Parameters | <ul style="list-style-type: none">• USARTTxSource: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_RCC_USART1_CLKSOURCE_PCLK1- LL_RCC_USART1_CLKSOURCE_SYSCLK- LL_RCC_USART1_CLKSOURCE_LSE- LL_RCC_USART1_CLKSOURCE_HSI- LL_RCC_USART2_CLKSOURCE_PCLK1 (*)- LL_RCC_USART2_CLKSOURCE_SYSCLK (*)- LL_RCC_USART2_CLKSOURCE_LSE (*)- LL_RCC_USART2_CLKSOURCE_HSI (*)- LL_RCC_USART3_CLKSOURCE_PCLK1 (*)- LL_RCC_USART3_CLKSOURCE_SYSCLK (*)- LL_RCC_USART3_CLKSOURCE_LSE (*)- LL_RCC_USART3_CLKSOURCE_HSI (*)(*) value not defined in all devices. |
|-------------------|--|

Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR3 USART1SW LL_RCC_SetUSARTClockSource• CFGR3 USART2SW LL_RCC_SetUSARTClockSource• CFGR3 USART3SW LL_RCC_SetUSARTClockSource
LL_RCC_SetI2CClockSource	
Function name	<code>__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t I2CxSource)</code>
Function description	Configure I2Cx clock source.
Parameters	<ul style="list-style-type: none">• I2CxSource: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RCC_I2C1_CLKSOURCE_HSI– LL_RCC_I2C1_CLKSOURCE_SYSCLK
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR3 I2C1SW LL_RCC_SetI2CClockSource
LL_RCC_SetCECClockSource	
Function name	<code>__STATIC_INLINE void LL_RCC_SetCECClockSource (uint32_t CECxSource)</code>
Function description	Configure CEC clock source.
Parameters	<ul style="list-style-type: none">• CECxSource: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RCC_CEC_CLKSOURCE_HSI_DIV244– LL_RCC_CEC_CLKSOURCE_LSE
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFGR3 CECSW LL_RCC_SetCECClockSource
LL_RCC_GetUSARTClockSource	
Function name	<code>__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t USARTTx)</code>
Function description	Get USARTx clock source.
Parameters	<ul style="list-style-type: none">• USARTTx: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RCC_USART1_CLKSOURCE– LL_RCC_USART2_CLKSOURCE (*)– LL_RCC_USART3_CLKSOURCE (*)

(*) value not defined in all devices.

Return values

- **Returned:** value can be one of the following values:
 - LL_RCC_USART1_CLKSOURCE_PCLK1
 - LL_RCC_USART1_CLKSOURCE_SYSCLK
 - LL_RCC_USART1_CLKSOURCE_LSE
 - LL_RCC_USART1_CLKSOURCE_HSI
 - LL_RCC_USART2_CLKSOURCE_PCLK1 (*)
 - LL_RCC_USART2_CLKSOURCE_SYSCLK (*)
 - LL_RCC_USART2_CLKSOURCE_LSE (*)
 - LL_RCC_USART2_CLKSOURCE_HSI (*)
 - LL_RCC_USART3_CLKSOURCE_PCLK1 (*)
 - LL_RCC_USART3_CLKSOURCE_SYSCLK (*)
 - LL_RCC_USART3_CLKSOURCE_LSE (*)
 - LL_RCC_USART3_CLKSOURCE_HSI (*)
- (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- CFGR3 USART1SW LL_RCC_GetUSARTClockSource
- CFGR3 USART2SW LL_RCC_GetUSARTClockSource
- CFGR3 USART3SW LL_RCC_GetUSARTClockSource

LL_RCC_GetI2CClockSource**Function name**

__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t I2Cx)

Function description

Get I2Cx clock source.

Parameters

- **I2Cx:** This parameter can be one of the following values:
 - LL_RCC_I2C1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
 - LL_RCC_I2C1_CLKSOURCE_HSI
 - LL_RCC_I2C1_CLKSOURCE_SYSCLK

Reference Manual to LL API cross reference:

- CFGR3 I2C1SW LL_RCC_GetI2CClockSource

LL_RCC_GetCECClockSource**Function name**

__STATIC_INLINE uint32_t LL_RCC_GetCECClockSource (uint32_t CECx)

Function description

Get CEC clock source.

Parameters

- **CECx:** This parameter can be one of the following values:
 - LL_RCC_CEC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
 - LL_RCC_CEC_CLKSOURCE_HSI_DIV244
 - LL_RCC_CEC_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CFGR3 CECSW LL_RCC_GetCECClockSource

LL_RCC_SetRTCClockSource

Function name	<code>__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)</code>
Function description	Set RTC Clock Source.
Parameters	<ul style="list-style-type: none">Source: This parameter can be one of the following values:<ul style="list-style-type: none">- <code>LL_RCC_RTC_CLKSOURCE_NONE</code>- <code>LL_RCC_RTC_CLKSOURCE_LSE</code>- <code>LL_RCC_RTC_CLKSOURCE_LSI</code>- <code>LL_RCC_RTC_CLKSOURCE_HSE_DIV32</code>
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Once the RTC clock source has been selected, it cannot be changed any more unless the Backup domain is reset. The BDRST bit can be used to reset them.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name	<code>__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)</code>
Function description	Get RTC Clock Source.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">- <code>LL_RCC_RTC_CLKSOURCE_NONE</code>- <code>LL_RCC_RTC_CLKSOURCE_LSE</code>- <code>LL_RCC_RTC_CLKSOURCE_LSI</code>- <code>LL_RCC_RTC_CLKSOURCE_HSE_DIV32</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name	<code>__STATIC_INLINE void LL_RCC_EnableRTC (void)</code>
Function description	Enable RTC.
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">BDCR RTCEN LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name	<code>__STATIC_INLINE void LL_RCC_DisableRTC (void)</code>
Function description	Disable RTC.

Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• BDCR RTCEN LL_RCC_DisableRTC
LL_RCC_IsEnabledRTC	
Function name	<code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)</code>
Function description	Check if RTC has been enabled or not.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• BDCR RTCEN LL_RCC_IsEnabledRTC
LL_RCC_ForceBackupDomainReset	
Function name	<code>__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)</code>
Function description	Force the Backup domain reset.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• BDCR BDRST LL_RCC_ForceBackupDomainReset
LL_RCC_ReleaseBackupDomainReset	
Function name	<code>__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)</code>
Function description	Release the Backup domain reset.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• BDCR BDRST LL_RCC_ReleaseBackupDomainReset
LL_RCC_PLL_Enable	
Function name	<code>__STATIC_INLINE void LL_RCC_PLL_Enable (void)</code>
Function description	Enable PLL.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR PLLON LL_RCC_PLL_Enable
LL_RCC_PLL_Disable	
Function name	<code>__STATIC_INLINE void LL_RCC_PLL_Disable (void)</code>

Function description	Disable PLL.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Cannot be disabled if the PLL clock is used as the system clock
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR PLLON LL_RCC_PLL_Disable
	LL_RCC_PLL_IsReady
Function name	<code>__STATIC_INLINE uint32_t LL_RCC_PLL_IsReady (void)</code>
Function description	Check if PLL Ready.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR PLLRDY LL_RCC_PLL_IsReady
	LL_RCC_PLL_ConfigDomain_SYS
Function name	<code>__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SYS (uint32_t Source, uint32_t PLLMul, uint32_t PLLDiv)</code>
Function description	Configure PLL used for SYSCLK Domain.

- | | |
|--|---|
| Parameters | <ul style="list-style-type: none">• Source: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_RCC_PLLSOURCE_HSI- LL_RCC_PLLSOURCE_HSE- LL_RCC_PLLSOURCE_HSI48 (*)<p>(*) value not defined in all devices</p>• PLLMul: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_RCC_PLL_MUL_2- LL_RCC_PLL_MUL_3- LL_RCC_PLL_MUL_4- LL_RCC_PLL_MUL_5- LL_RCC_PLL_MUL_6- LL_RCC_PLL_MUL_7- LL_RCC_PLL_MUL_8- LL_RCC_PLL_MUL_9- LL_RCC_PLL_MUL_10- LL_RCC_PLL_MUL_11- LL_RCC_PLL_MUL_12- LL_RCC_PLL_MUL_13- LL_RCC_PLL_MUL_14- LL_RCC_PLL_MUL_15- LL_RCC_PLL_MUL_16• PLLDiv: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_RCC_PREDIV_DIV_1- LL_RCC_PREDIV_DIV_2- LL_RCC_PREDIV_DIV_3- LL_RCC_PREDIV_DIV_4- LL_RCC_PREDIV_DIV_5- LL_RCC_PREDIV_DIV_6- LL_RCC_PREDIV_DIV_7- LL_RCC_PREDIV_DIV_8- LL_RCC_PREDIV_DIV_9- LL_RCC_PREDIV_DIV_10- LL_RCC_PREDIV_DIV_11- LL_RCC_PREDIV_DIV_12- LL_RCC_PREDIV_DIV_13- LL_RCC_PREDIV_DIV_14- LL_RCC_PREDIV_DIV_15- LL_RCC_PREDIV_DIV_16 |
| Return values | <ul style="list-style-type: none">• None: |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none">• CFGR PLLSRC LL_RCC_PLL_ConfigDomain_SYS• CFGR PLLMUL LL_RCC_PLL_ConfigDomain_SYS• CFGR2 PREDIV LL_RCC_PLL_ConfigDomain_SYS |

LL_RCC_PLL_GetMainSource

Function name **__STATIC_INLINE uint32_t LL_RCC_PLL_GetMainSource (void)**

Function description Get the oscillator used as PLL clock source.

- Return values**
- **Returned:** value can be one of the following values:
 - LL_RCC_PLLSOURCE_HSI (*)
 - LL_RCC_PLLSOURCE_HSI_DIV_2 (*)
 - LL_RCC_PLLSOURCE_HSE
 - LL_RCC_PLLSOURCE_HSI48 (*)
- (*) value not defined in all devices

Reference Manual to LL API cross reference:

- CFGR PLLSRC LL_RCC_PLL_GetMainSource

LL_RCC_PLL_GetMultiplicator

Function name `__STATIC_INLINE uint32_t LL_RCC_PLL_GetMultiplicator (void)`

Function description Get PLL multiplication Factor.

- Return values**
- **Returned:** value can be one of the following values:
 - LL_RCC_PLL_MUL_2
 - LL_RCC_PLL_MUL_3
 - LL_RCC_PLL_MUL_4
 - LL_RCC_PLL_MUL_5
 - LL_RCC_PLL_MUL_6
 - LL_RCC_PLL_MUL_7
 - LL_RCC_PLL_MUL_8
 - LL_RCC_PLL_MUL_9
 - LL_RCC_PLL_MUL_10
 - LL_RCC_PLL_MUL_11
 - LL_RCC_PLL_MUL_12
 - LL_RCC_PLL_MUL_13
 - LL_RCC_PLL_MUL_14
 - LL_RCC_PLL_MUL_15
 - LL_RCC_PLL_MUL_16

Reference Manual to LL API cross reference:

- CFGR PLLMUL LL_RCC_PLL_GetMultiplicator

LL_RCC_PLL_GetPrediv

Function name `__STATIC_INLINE uint32_t LL_RCC_PLL_GetPrediv (void)`

Function description Get PREDIV division factor for the main PLL.

- Return values**
- **Returned:** value can be one of the following values:
 - LL_RCC_PREDIV_DIV_1
 - LL_RCC_PREDIV_DIV_2
 - LL_RCC_PREDIV_DIV_3
 - LL_RCC_PREDIV_DIV_4
 - LL_RCC_PREDIV_DIV_5
 - LL_RCC_PREDIV_DIV_6
 - LL_RCC_PREDIV_DIV_7
 - LL_RCC_PREDIV_DIV_8
 - LL_RCC_PREDIV_DIV_9
 - LL_RCC_PREDIV_DIV_10
 - LL_RCC_PREDIV_DIV_11
 - LL_RCC_PREDIV_DIV_12
 - LL_RCC_PREDIV_DIV_13
 - LL_RCC_PREDIV_DIV_14
 - LL_RCC_PREDIV_DIV_15
 - LL_RCC_PREDIV_DIV_16

- Notes**
- They can be written only when the PLL is disabled

- Reference Manual to LL API cross reference:**
- CFGR2 PREDIV LL_RCC_PLL_GetPrediv

`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:**
- CIR LSIRDYC LL_RCC_ClearFlag_LSIRDY

`LL_RCC_ClearFlag_LSERDY`

Function name `__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)`

Function description Clear LSE ready interrupt flag.

- Return values**
- **None:**

- Reference Manual to LL API cross reference:**
- CIR LSERDYC LL_RCC_ClearFlag_LSERDY

`LL_RCC_ClearFlag_HSIRDY`

Function name `__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)`

Function description Clear HSI ready interrupt flag.

- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CIR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSIRDY

Function name `__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)`

Function description Clear HSE ready interrupt flag.

- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CIR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_PLLRDY

Function name `__STATIC_INLINE void LL_RCC_ClearFlag_PLLRDY (void)`

Function description Clear PLL ready interrupt flag.

- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CIR PLLRDYC LL_RCC_ClearFlag_PLLRDY

LL_RCC_ClearFlag_HSI14RDY

Function name `__STATIC_INLINE void LL_RCC_ClearFlag_HSI14RDY (void)`

Function description Clear HSI14 ready interrupt flag.

- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CIR HSI14RDYC LL_RCC_ClearFlag_HSI14RDY

LL_RCC_ClearFlag_HSI48RDY

Function name `__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void)`

Function description Clear HSI48 ready interrupt flag.

- Return values**
- **None:**
- Reference Manual to LL API cross reference:**
- CIR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY

LL_RCC_ClearFlag_HSECSS

Function name `__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)`

Function description Clear Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIR CSSC LL_RCC_ClearFlag_HSECSS

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:

- CIR 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:

- CIR 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:

- CIR 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:

- CIR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_PLLRDY

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLRDY (void)`

Function description Check if PLL ready interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIR PLLRDYF LL_RCC_IsActiveFlag_PLLRDY

LL_RCC_IsActiveFlag_HSI14RDY

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI14RDY (void)`

Function description Check if HSI14 ready interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIR HSI14RDYF LL_RCC_IsActiveFlag_HSI14RDY

LL_RCC_IsActiveFlag_HSI48RDY

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY (void)`

Function description Check if HSI48 ready interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIR HSI48RDYF LL_RCC_IsActiveFlag_HSI48RDY

LL_RCC_IsActiveFlag_HSECSS

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)`

Function description Check if Clock security system interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIR CSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_IWDGRST

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)`

Function description Check if RCC flag Independent Watchdog reset is set or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- CSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_LPWRST

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRST (void)`

Function description Check if RCC flag Low Power reset is set or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRST

LL_RCC_IsActiveFlag_OBLRST

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_OBLRST (void)`

Function description Check if RCC flag is set or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- CSR OBLRSTF LL_RCC_IsActiveFlag_OBLRST

LL_RCC_IsActiveFlag_PINRST

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)`

Function description Check if RCC flag Pin reset is set or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- CSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_PORRST

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PORRST (void)`

Function description Check if RCC flag POR/PDR 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_SFTRST

Function name `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)`

Function description Check if RCC flag Software reset is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_WWDGRST

Function name

`__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST (void)`

Function description

Check if RCC flag Window Watchdog reset is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_IsActiveFlag_V18PWRRST

Function name

`__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_V18PWRRST (void)`

Function description

Check if RCC Reset flag of the 1.8 V domain is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR V18PWRRSTF LL_RCC_IsActiveFlag_V18PWRRST

LL_RCC_ClearResetFlags

Function name

`__STATIC_INLINE void LL_RCC_ClearResetFlags (void)`

Function description

Set RMVF bit to clear the reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RMVF LL_RCC_ClearResetFlags

LL_RCC_EnableIT_LSIRDY

Function name

`__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)`

Function description

Enable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIR 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:

- CIR 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:

- CIR 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:

- CIR HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_PLLRDY

Function name `__STATIC_INLINE void LL_RCC_EnableIT_PLLRDY (void)`

Function description Enable PLL ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIR PLLRDYIE LL_RCC_EnableIT_PLLRDY

LL_RCC_EnableIT_HSI14RDY

Function name `__STATIC_INLINE void LL_RCC_EnableIT_HSI14RDY (void)`

Function description Enable HSI14 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIR HSI14RDYIE LL_RCC_EnableIT_HSI14RDY

LL_RCC_EnableIT_HSI48RDY

Function name `__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void)`

Function description Enable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIR HSI48RDYIE LL_RCC_EnableIT_HSI48RDY

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:

- CIR 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:

- CIR 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:

- CIR 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:](#)

- CIR HSERDYIE LL_RCC_DisableIT_HSERDY

`LL_RCC_DisableIT_PLLRDY`

Function name `__STATIC_INLINE void LL_RCC_DisableIT_PLLRDY (void)`

Function description Disable PLL ready interrupt.

Return values

- **None:**

[Reference Manual to LL API cross reference:](#)

- CIR PLLRDYIE LL_RCC_DisableIT_PLLRDY

`LL_RCC_DisableIT_HSI14RDY`

Function name `__STATIC_INLINE void LL_RCC_DisableIT_HSI14RDY (void)`

Function description Disable HSI14 ready interrupt.

Return values

- **None:**

[Reference Manual to LL API cross reference:](#)

- CIR HSI14RDYIE LL_RCC_DisableIT_HSI14RDY

`LL_RCC_DisableIT_HSI48RDY`

Function name `__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void)`

Function description Disable HSI48 ready interrupt.

Return values

- **None:**

[Reference Manual to LL API cross reference:](#)

- CIR HSI48RDYIE LL_RCC_DisableIT_HSI48RDY

`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:](#)

- CIR 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:

- CIR 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:

- CIR 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:

- CIR HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_PLLRDY

Function name

`__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLRDY (void)`

Function description

Checks if PLL ready interrupt source is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CIR PLLRDYIE LL_RCC_IsEnabledIT_PLLRDY

LL_RCC_IsEnabledIT_HSI14RDY

Function name

`__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI14RDY (void)`

Function description

Checks if HSI14 ready interrupt source is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CIR HSI14RDYIE LL_RCC_IsEnabledIT_HSI14RDY

LL_RCC_IsEnabledIT_HSI48RDY

Function name

`__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI48RDY (void)`

Function description Checks if HSI48 ready interrupt source is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CIR HSI48RDYIE LL_RCC_IsEnabledIT_HSI48RDY

LL_RCC_DelInit

Function name **ErrorStatus LL_RCC_DelInit (void)**

Function description Reset the RCC clock configuration to the default reset state.

Return values

- **An:** ErrorStatus enumeration value:
 - SUCCESS: RCC registers are de-initialized
 - ERROR: not applicable

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE and PLL OFFAHB and APB1 prescaler set to 1.CSS, MCO OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name **void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)**

Function description Return the frequencies of different on chip clocks; System, AHB and APB1 buses clocks.

Parameters

- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

Notes

- Each time SYSCLK, HCLK and/or PCLK1 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetUSARTClockFreq

Function name **uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)**

Function description Return USARTx clock frequency.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
 - LL_RCC_USART1_CLKSOURCE
 - LL_RCC_USART2_CLKSOURCE (*)
 - LL_RCC_USART3_CLKSOURCE (*)

(*) value not defined in all devices.

- Return values**
- **USART:** clock frequency (in Hz)
 - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

`LL_RCC_GetI2CClockFreq`

Function name `uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)`

Function description Return I2Cx clock frequency.

- Parameters**
- **I2CxSource:** This parameter can be one of the following values:
 - LL_RCC_I2C1_CLKSOURCE

- Return values**
- **I2C:** clock frequency (in Hz)
 - LL_RCC_PERIPH_FREQUENCY_NO indicates that HSI oscillator is not ready

`LL_RCC_GetCECClockFreq`

Function name `uint32_t LL_RCC_GetCECClockFreq (uint32_t CECxSource)`

Function description Return CECx clock frequency.

- Parameters**
- **CECxSource:** This parameter can be one of the following values:
 - LL_RCC_CEC_CLKSOURCE

- Return values**
- **CEC:** clock frequency (in Hz)
 - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillators (HSI or LSE) are not ready

64.3 RCC Firmware driver defines

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

64.3.1 RCC

RCC

APB low-speed prescaler (APB1)

`LL_RCC_APB1_DIV_1` HCLK not divided

`LL_RCC_APB1_DIV_2` HCLK divided by 2

`LL_RCC_APB1_DIV_4` HCLK divided by 4

`LL_RCC_APB1_DIV_8` HCLK divided by 8

`LL_RCC_APB1_DIV_16` HCLK divided by 16

Peripheral CEC get clock source

`LL_RCC_CEC_CLKSOURC_E` CEC Clock source selection

Peripheral CEC clock source selection

LL_RCC_CEC_CLKSOURC HSI clock divided by 244 selected as HDMI CEC entry clock source
E_HSI_DIV244

LL_RCC_CEC_CLKSOURC LSE clock selected as HDMI CEC entry clock source
E_LSE

Clear Flags Defines

LL_RCC_CIR_LSIRDYC	LSI Ready Interrupt Clear
LL_RCC_CIR_LSERDYC	LSE Ready Interrupt Clear
LL_RCC_CIR_HSIRDYC	HSI Ready Interrupt Clear
LL_RCC_CIR_HSERDYC	HSE Ready Interrupt Clear
LL_RCC_CIR_PLLRDYC	PLL Ready Interrupt Clear
LL_RCC_CIR_HSI14RDYC	HSI14 Ready Interrupt Clear
LL_RCC_CIR_HSI48RDYC	HSI48 Ready Interrupt Clear
LL_RCC_CIR_CSSC	Clock Security System Interrupt Clear

Get Flags Defines

LL_RCC_CIR_LSIRDYF	LSI Ready Interrupt flag
LL_RCC_CIR_LSERDYF	LSE Ready Interrupt flag
LL_RCC_CIR_HSIRDYF	HSI Ready Interrupt flag
LL_RCC_CIR_HSERDYF	HSE Ready Interrupt flag
LL_RCC_CIR_PLLRDYF	PLL Ready Interrupt flag
LL_RCC_CIR_HSI14RDYF	HSI14 Ready Interrupt flag
LL_RCC_CIR_HSI48RDYF	HSI48 Ready Interrupt flag
LL_RCC_CIR_CSSF	Clock Security System Interrupt flag
LL_RCC_CSR_OBLRSTF	OBL reset flag
LL_RCC_CSR_PINRSTF	PIN reset flag
LL_RCC_CSR_PORRSTF	POR/PDR reset flag
LL_RCC_CSR_SFTRSTF	Software Reset flag
LL_RCC_CSR_IWDGRSTF	Independent Watchdog reset flag

LL_RCC_CSR_WWDGRST Window watchdog reset flag
F

LL_RCC_CSR_LPWRSTF Low-Power reset flag

LL_RCC_CSR_V18PWRRS Reset flag of the 1.8 V domain.
TF

Peripheral I2C get clock source

LL_RCC_I2C1_CLKSOURC I2C1 Clock source selection
E

Peripheral I2C clock source selection

LL_RCC_I2C1_CLKSOURC HSI oscillator clock used as I2C1 clock source
E_HSI

LL_RCC_I2C1_CLKSOURC System clock selected as I2C1 clock source
E_SYSCLK

IT Defines

LL_RCC_CIR_LSIRDYIE LSI Ready Interrupt Enable

LL_RCC_CIR_LSERDYIE LSE Ready Interrupt Enable

LL_RCC_CIR_HSIRDYIE HSI Ready Interrupt Enable

LL_RCC_CIR_HSERDYIE HSE Ready Interrupt Enable

LL_RCC_CIR_PLLRDYIE PLL Ready Interrupt Enable

LL_RCC_CIR_HSI14RDYIE HSI14 Ready Interrupt Enable

LL_RCC_CIR_HSI48RDYIE HSI48 Ready Interrupt Enable

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW Xtal mode lower driving capability

LL_RCC_LSEDRIVE_MEDI Xtal mode medium low driving capability
UMLOW

LL_RCC_LSEDRIVE_MEDI Xtal mode medium high driving capability
UMHIGH

LL_RCC_LSEDRIVE_HIGH Xtal mode higher driving capability

MCO1 SOURCE selection

LL_RCC_MCO1SOURCE_N MCO output disabled, no clock on MCO
OCLOCK

LL_RCC_MCO1SOURCE_H HSI14 oscillator clock selected
SI14

LL_RCC_MCO1SOURCE_S SYSCLK selection as MCO source
YSCLK

LL_RCC_MCO1SOURCE_H HSI selection as MCO source
SI

LL_RCC_MCO1SOURCE_H HSE selection as MCO source
SE

LL_RCC_MCO1SOURCE_L LSI selection as MCO source
SI

LL_RCC_MCO1SOURCE_L LSE selection as MCO source
SE

LL_RCC_MCO1SOURCE_H HSI48 selection as MCO source
SI48

LL_RCC_MCO1SOURCE_P PLL clock divided by 2
LLCLK_DIV_2

LL_RCC_MCO1SOURCE_P PLL clock selected
LLCLK

MCO1 prescaler

LL_RCC_MCO1_DIV_1 MCO Clock divided by 1

LL_RCC_MCO1_DIV_2 MCO Clock divided by 2

LL_RCC_MCO1_DIV_4 MCO Clock divided by 4

LL_RCC_MCO1_DIV_8 MCO Clock divided by 8

LL_RCC_MCO1_DIV_16 MCO Clock divided by 16

LL_RCC_MCO1_DIV_32 MCO Clock divided by 32

LL_RCC_MCO1_DIV_64 MCO Clock divided by 64

LL_RCC_MCO1_DIV_128 MCO Clock divided by 128

Oscillator Values adaptation

HSE_VALUE Value of the HSE oscillator in Hz

HSI_VALUE Value of the HSI oscillator in Hz

LSE_VALUE Value of the LSE oscillator in Hz

LSI_VALUE Value of the LSI oscillator in Hz

HSI48_VALUE Value of the HSI48 oscillator in Hz

Peripheral clock frequency

LL_RCC_PERIPH_FREQUE No clock enabled for the peripheral
NCY_NO

LL_RCC_PERIPH_FREQUE Frequency cannot be provided as external clock
NCY_NA

PLL SOURCE

LL_RCC_PLLSOURCE_HS_E HSE/PREDIV clock selected as PLL entry clock source

LL_RCC_PLLSOURCE_HSI HSI/PREDIV clock selected as PLL entry clock source

LL_RCC_PLLSOURCE_HSI48_48 HSI48/PREDIV clock selected as PLL entry clock source

PLL Multiplicator factor

LL_RCC_PLL_MUL_2 PLL input clock*2

LL_RCC_PLL_MUL_3 PLL input clock*3

LL_RCC_PLL_MUL_4 PLL input clock*4

LL_RCC_PLL_MUL_5 PLL input clock*5

LL_RCC_PLL_MUL_6 PLL input clock*6

LL_RCC_PLL_MUL_7 PLL input clock*7

LL_RCC_PLL_MUL_8 PLL input clock*8

LL_RCC_PLL_MUL_9 PLL input clock*9

LL_RCC_PLL_MUL_10 PLL input clock*10

LL_RCC_PLL_MUL_11 PLL input clock*11

LL_RCC_PLL_MUL_12 PLL input clock*12

LL_RCC_PLL_MUL_13 PLL input clock*13

LL_RCC_PLL_MUL_14 PLL input clock*14

LL_RCC_PLL_MUL_15 PLL input clock*15

LL_RCC_PLL_MUL_16 PLL input clock*16

PREDIV Division factor

LL_RCC_PREDIV_DIV_1 PREDIV input clock not divided

LL_RCC_PREDIV_DIV_2 PREDIV input clock divided by 2

LL_RCC_PREDIV_DIV_3 PREDIV input clock divided by 3

LL_RCC_PREDIV_DIV_4 PREDIV input clock divided by 4

LL_RCC_PREDIV_DIV_5 PREDIV input clock divided by 5

LL_RCC_PREDIV_DIV_6 PREDIV input clock divided by 6

LL_RCC_PREDIV_DIV_7 PREDIV input clock divided by 7

LL_RCC_PREDIV_DIV_8 PREDIV input clock divided by 8

LL_RCC_PREDIV_DIV_9 PREDIV input clock divided by 9

LL_RCC_PREDIV_DIV_10 PREDIV input clock divided by 10

LL_RCC_PREDIV_DIV_11 PREDIV input clock divided by 11

LL_RCC_PREDIV_DIV_12 PREDIV input clock divided by 12

LL_RCC_PREDIV_DIV_13 PREDIV input clock divided by 13

LL_RCC_PREDIV_DIV_14 PREDIV input clock divided by 14

LL_RCC_PREDIV_DIV_15 PREDIV input clock divided by 15

LL_RCC_PREDIV_DIV_16 PREDIV input clock divided by 16

RTC clock source selection

LL_RCC_RTC_CLKSOURC_E_NONE No clock used as RTC clock

LL_RCC_RTC_CLKSOURC_E_LSE LSE oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURC_E_LSI LSI oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURC_E_HSE_DIV32 HSE oscillator clock divided by 32 used as RTC clock

AHB prescaler

- `LL_RCC_SYSCLK_DIV_1` SYSCLK not divided
- `LL_RCC_SYSCLK_DIV_2` SYSCLK divided by 2
- `LL_RCC_SYSCLK_DIV_4` SYSCLK divided by 4
- `LL_RCC_SYSCLK_DIV_8` SYSCLK divided by 8
- `LL_RCC_SYSCLK_DIV_16` SYSCLK divided by 16
- `LL_RCC_SYSCLK_DIV_64` SYSCLK divided by 64
- `LL_RCC_SYSCLK_DIV_128` SYSCLK divided by 128
- `LL_RCC_SYSCLK_DIV_256` SYSCLK divided by 256
- `LL_RCC_SYSCLK_DIV_512` SYSCLK divided by 512

System clock switch

`LL_RCC_SYS_CLKSOURC` HSI selection as system clock
`E_HSI`

`LL_RCC_SYS_CLKSOURC` HSE selection as system clock
`E_HSE`

`LL_RCC_SYS_CLKSOURC` PLL selection as system clock
`E_PLL`

`LL_RCC_SYS_CLKSOURC` HSI48 selection as system clock
`E_HSI48`

System clock switch status

`LL_RCC_SYS_CLKSOURC` HSI used as system clock
`E_STATUS_HSI`

`LL_RCC_SYS_CLKSOURC` HSE used as system clock
`E_STATUS_HSE`

`LL_RCC_SYS_CLKSOURC` PLL used as system clock
`E_STATUS_PLL`

`LL_RCC_SYS_CLKSOURC` HSI48 used as system clock
`E_STATUS_HSI48`

Peripheral USART get clock source

`LL_RCC_USART1_CLKSO` USART1 Clock source selection
`URCE`

`LL_RCC_USART2_CLKSO` USART2 Clock source selection
`URCE`

LL_RCC_USART3_CLKSO USART3 Clock source selection
URCE

Peripheral USART clock source selection

LL_RCC_USART1_CLKSO PCLK1 clock used as USART1 clock source
URCE_PCLK1

LL_RCC_USART1_CLKSO System clock selected as USART1 clock source
URCE_SYSCLK

LL_RCC_USART1_CLKSO LSE oscillator clock used as USART1 clock source
URCE_LSE

LL_RCC_USART1_CLKSO HSI oscillator clock used as USART1 clock source
URCE_HSI

LL_RCC_USART2_CLKSO PCLK1 clock used as USART2 clock source
URCE_PCLK1

LL_RCC_USART2_CLKSO System clock selected as USART2 clock source
URCE_SYSCLK

LL_RCC_USART2_CLKSO LSE oscillator clock used as USART2 clock source
URCE_LSE

LL_RCC_USART2_CLKSO HSI oscillator clock used as USART2 clock source
URCE_HSI

LL_RCC_USART3_CLKSO PCLK1 clock used as USART3 clock source
URCE_PCLK1

LL_RCC_USART3_CLKSO System clock selected as USART3 clock source
URCE_SYSCLK

LL_RCC_USART3_CLKSO LSE oscillator clock used as USART3 clock source
URCE_LSE

LL_RCC_USART3_CLKSO HSI oscillator clock used as USART3 clock source
URCE_HSI

Calculate frequencies

LL_RCC_CALC_PLLCLK **Description:****_FREQ**

- Helper macro to calculate the PLLCLK frequency.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSE/HSI/HSI48)
- __PLLMUL__: This parameter can be one of the following values:
 - LL_RCC_PLL_MUL_2
 - LL_RCC_PLL_MUL_3
 - LL_RCC_PLL_MUL_4
 - LL_RCC_PLL_MUL_5
 - LL_RCC_PLL_MUL_6
 - LL_RCC_PLL_MUL_7
 - LL_RCC_PLL_MUL_8
 - LL_RCC_PLL_MUL_9
 - LL_RCC_PLL_MUL_10
 - LL_RCC_PLL_MUL_11
 - LL_RCC_PLL_MUL_12
 - LL_RCC_PLL_MUL_13
 - LL_RCC_PLL_MUL_14
 - LL_RCC_PLL_MUL_15
 - LL_RCC_PLL_MUL_16
- __PLLPREDIV__: This parameter can be one of the following values:
 - LL_RCC_PREDIV_DIV_1
 - LL_RCC_PREDIV_DIV_2
 - LL_RCC_PREDIV_DIV_3
 - LL_RCC_PREDIV_DIV_4
 - LL_RCC_PREDIV_DIV_5
 - LL_RCC_PREDIV_DIV_6
 - LL_RCC_PREDIV_DIV_7
 - LL_RCC_PREDIV_DIV_8
 - LL_RCC_PREDIV_DIV_9
 - LL_RCC_PREDIV_DIV_10
 - LL_RCC_PREDIV_DIV_11
 - LL_RCC_PREDIV_DIV_12
 - LL_RCC_PREDIV_DIV_13
 - LL_RCC_PREDIV_DIV_14
 - LL_RCC_PREDIV_DIV_15
 - LL_RCC_PREDIV_DIV_16

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: LL_RCC_CALC_PLLCLK_FREQ (HSE_VALUE, LL_RCC_PLL_GetMultiplicator(), LL_RCC_PLL_GetPrediv());

LL_RCC_CALC_HCLK_F **Description:**

REQ

- Helper macro to calculate the HCLK frequency.

Parameters:

- SYSCLKFREQ: SYSCLK frequency (based on HSE/HSI/PLLCLK)
- AHBPRESCALER: This parameter can be one of the following values:
 - LL_RCC_SYSCLK_DIV_1
 - LL_RCC_SYSCLK_DIV_2
 - LL_RCC_SYSCLK_DIV_4
 - LL_RCC_SYSCLK_DIV_8
 - LL_RCC_SYSCLK_DIV_16
 - LL_RCC_SYSCLK_DIV_64
 - LL_RCC_SYSCLK_DIV_128
 - LL_RCC_SYSCLK_DIV_256
 - LL_RCC_SYSCLK_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

Notes:

- : AHBPRESCALER be retrieved by LL_RCC_GetAHBPrescaler ex:
LL_RCC_CALC_HCLK_FREQ(LL_RCC_GetAHBPrescaler())

LL_RCC_CALC_PCLK1 **Description:**

FREQ

- Helper macro to calculate the PCLK1 frequency (APB1)

Parameters:

- HCLKFREQ: HCLK frequency
- APB1PRESCALER: This parameter can be one of the following values:
 - LL_RCC_APB1_DIV_1
 - LL_RCC_APB1_DIV_2
 - LL_RCC_APB1_DIV_4
 - LL_RCC_APB1_DIV_8
 - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

Notes:

- : APB1PRESCALER be retrieved by LL_RCC_GetAPB1Prescaler ex:
LL_RCC_CALC_PCLK1_FREQ(LL_RCC_GetAPB1Prescaler())

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

65 LL RTC Generic Driver

65.1 RTC Firmware driver registers structures

65.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the `stm32f0xx_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()`.

65.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the `stm32f0xx_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()`.

65.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the `stm32f0xx_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()`.

65.1.4 LL_RTC_AlarmTypeDef

`LL_RTC_AlarmTypeDef` is defined in the `stm32f0xx_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` This feature can be modified afterwards using unitary function `LL_RTC_ALMA_SetMask()` for ALARM A.

- `uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel`

Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of `RTC_LL_EC_ALMA_WEEKDAY_SELECTION`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()`

65.2 RTC Firmware driver API description

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

65.2.1 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	<ul style="list-style-type: none">• RTCx: RTC Instance• HourFormat: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_HOURFORMAT_24HOUR– LL_RTC_HOURFORMAT_AMPM
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (RTC_TypeDef * RTCx)</code>
---------------	--

Function description	Get Hours format (24 hour/day or AM/PM hour format)
----------------------	---

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_HOURFORMAT_24HOUR– LL_RTC_HOURFORMAT_AMPM
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name	<code>__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)</code>
---------------	---

Function description	Select the flag to be routed to RTC_ALARM output.
----------------------	---

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• AlarmOutput: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_ALARMOUT_DISABLE– LL_RTC_ALARMOUT_ALMA– LL_RTC_ALARMOUT_ALMB– LL_RTC_ALARMOUT_WAKEUP
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (RTC_TypeDef * RTCx)</code>
Function description	Get the flag to be routed to RTC_ALARM output.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_ALARMOUT_DISABLE– LL_RTC_ALARMOUT_ALMA– LL_RTC_ALARMOUT_ALMB– LL_RTC_ALARMOUT_WAKEUP
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name	<code>__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)</code>
Function description	Set RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Output: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN– LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Used only when RTC_ALARM is mapped on PC13• If all RTC alternate functions are disabled and PC13MODE = 1, PC13VALUE configures the PC13 output data
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR ALARMOUTTYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (RTC_TypeDef * RTCx)</code>
Function description	Get RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN– LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Notes

- used only when RTC_ALARM is mapped on PC13
- If all RTC alternate functions are disabled and PC13MODE = 1, PC13VALUE configures the PC13 output data

Reference Manual to LL API cross reference:

- TAFCR ALARMOUTTYPE LL_RTC_GetAlarmOutputType

LL_RTC_EnablePushPullMode**Function name**

```
__STATIC_INLINE void LL_RTC_EnablePushPullMode (RTC_TypeDef * RTCx, uint32_t PinMask)
```

Function description

Enable push-pull output on PC13, PC14 and/or PC15.

Parameters

- **RTCx:** RTC Instance
- **PinMask:** This parameter can be a combination of the following values:
 - LL_RTC_PIN_PC13
 - LL_RTC_PIN_PC14
 - LL_RTC_PIN_PC15

Return values

- **None:**

Notes

- PC13 forced to push-pull output if all RTC alternate functions are disabled
- PC14 and PC15 forced to push-pull output if LSE is disabled

Reference Manual to LL API cross reference:

- TAFCR PC13MODE LL_RTC_EnablePushPullMode
- TAFCR PC14MODE LL_RTC_EnablePushPullMode
- TAFCR PC15MODE LL_RTC_EnablePushPullMode

LL_RTC_DisablePushPullMode**Function name**

```
__STATIC_INLINE void LL_RTC_DisablePushPullMode (RTC_TypeDef * RTCx, uint32_t PinMask)
```

Function description

Disable push-pull output on PC13, PC14 and/or PC15.

Parameters

- **RTCx:** RTC Instance
- **PinMask:** This parameter can be a combination of the following values:
 - LL_RTC_PIN_PC13
 - LL_RTC_PIN_PC14
 - LL_RTC_PIN_PC15

Return values

- **None:**

Notes

- PC13, PC14 and/or PC15 are controlled by the GPIO configuration registers. Consequently PC13, PC14 and/or PC15 are floating in Standby mode.

- Reference Manual to LL API cross reference:**
- TAFCR PC13MODE LL_RTC_DisablePushPullMode
 - TAFCR PC14MODE LL_RTC_DisablePushPullMode
 - TAFCR PC15MODE LL_RTC_DisablePushPullMode

LL_RTC_SetOutputPin

Function name	<code>__STATIC_INLINE void LL_RTC_SetOutputPin (RTC_TypeDef * RTCx, uint32_t PinMask)</code>
Function description	Set PC14 and/or PC15 to high level.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• PinMask: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_RTC_PIN_PC14– LL_RTC_PIN_PC15
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Output data configuration is possible if the LSE is disabled and PushPull output is enabled (through LL_RTC_EnablePushPullMode)
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR PC14VALUE LL_RTC_SetOutputPin• TAFCR PC15VALUE LL_RTC_SetOutputPin

LL_RTC_ResetOutputPin

Function name	<code>__STATIC_INLINE void LL_RTC_ResetOutputPin (RTC_TypeDef * RTCx, uint32_t PinMask)</code>
Function description	Set PC14 and/or PC15 to low level.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• PinMask: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_RTC_PIN_PC14– LL_RTC_PIN_PC15
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Output data configuration is possible if the LSE is disabled and PushPull output is enabled (through LL_RTC_EnablePushPullMode)
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR PC14VALUE LL_RTC_ResetOutputPin• TAFCR PC15VALUE LL_RTC_ResetOutputPin

LL_RTC_EnableInitMode

Function name	<code>__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)</code>
Function description	Enable initialization mode.
Parameters	<ul style="list-style-type: none">• 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_EnableRefClock

Function name `__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)`

Function description Enable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- `RTCx`: RTC Instance

Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name	<code>__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)</code>
Function description	Disable RTC_REFIN reference clock detection (50 or 60 Hz)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name	<code>__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)</code>
Function description	Set Asynchronous prescaler factor.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• AsynchPrescaler: Value between Min_Data = 0 and Max_Data = 0x7F
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name	<code>__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)</code>
Function description	Set Synchronous prescaler factor.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• SynchPrescaler: Value between Min_Data = 0 and Max_Data = 0xFFFF
Return values	<ul style="list-style-type: none">• 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_SetSynchPrescaler

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 = 0xFFFF

Reference Manual to LL API cross reference:

- PRER PREDIV_S LL_RTC_SetSynchPrescaler

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 shadow mode is disabled (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	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">TR HT LL_RTC_TIME_SetHourTR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (RTC_TypeDef * RTCx)</code>
Function description	Get Hours in BCD format.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	<ul style="list-style-type: none">if shadow mode is disabled (BYPASHAD=0), need to check if RSF flag is set before reading this bitRead 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:	<ul style="list-style-type: none">TR HT LL_RTC_TIME_GetHourTR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name	<code>__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)</code>
Function description	Set Minutes in BCD format.
Parameters	<ul style="list-style-type: none">RTCx: RTC InstanceMinutes: Value between Min_Data=0x00 and Max_Data=0x59
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">TR MNT LL_RTC_TIME_SetMinuteTR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (RTC_TypeDef * RTCx)</code>
Function description	Get Minutes in BCD format.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0x59
Notes	<ul style="list-style-type: none">if shadow mode is disabled (BYPASHD=0), need to check if RSF flag is set before reading this bitRead 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 <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert minute from BCD to Binary format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TR MNT LL_RTC_TIME_GetMinuteTR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name	<code>__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)</code>
Function description	Set Seconds in BCD format.
Parameters	<ul style="list-style-type: none">RTCx: RTC InstanceSeconds: Value between Min_Data=0x00 and Max_Data=0x59
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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 <code>__LL_RTC_CONVERT_BIN2BCD</code> is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TR ST LL_RTC_TIME_SetSecondTR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (RTC_TypeDef * RTCx)</code>
Function description	Get Seconds in BCD format.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0x59

Notes

- if shadow mode is disabled (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 shadow mode is disabled (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	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR BKP LL_RTC_TIME_IsDayLightStoreEnabled
LL_RTC_TIME_DecHour	
Function name	<code>__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)</code>
Function description	Subtract 1 hour (winter time change)
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR SUB1H LL_RTC_TIME_DecHour
LL_RTC_TIME_IncHour	
Function name	<code>__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)</code>
Function description	Add 1 hour (summer time change)
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR ADD1H LL_RTC_TIME_IncHour
LL_RTC_TIME_GetSubSecond	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (RTC_TypeDef * RTCx)</code>
Function description	Get Sub second value in the synchronous prescaler counter.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Sub: second value (number between 0 and 65535)
Notes	<ul style="list-style-type: none">You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSyncPrescaler 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	<code>__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)</code>
Function description	Synchronize to a remote clock with a high degree of precision.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• ShiftSecond: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_SHIFT_SECOND_DELAY– LL_RTC_SHIFT_SECOND_ADVANCE• Fraction: Number of Seconds Fractions (any value from 0 to 0x7FFF)
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• SHIFTR ADD1S LL_RTC_TIME_Synchronize• SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name	<code>__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)</code>
Function description	Set Year in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Year: Value between Min_Data=0x00 and Max_Data=0x99
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR YT LL_RTC_DATE_SetYear• DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (RTC_TypeDef * RTCx)</code>
Function description	Get Year in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance

Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0x99
Notes	<ul style="list-style-type: none">• if shadow mode is disabled (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:	<ul style="list-style-type: none">• DR YT LL_RTC_DATE_GetYear• DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name	<code>__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)</code>
----------------------	---

Function description	Set Week day.
-----------------------------	---------------

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• WeekDay: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
----------------------	--

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR WDU LL_RTC_DATE_SetWeekDay
--	---

LL_RTC_DATE_GetWeekDay

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (RTC_TypeDef * RTCx)</code>
----------------------	---

Function description	Get Week day.
-----------------------------	---------------

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
-------------------	---

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• if shadow mode is disabled (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_APRIIL
 - 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	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_MONTH_JANUARY– LL_RTC_MONTH_FEBRUARY– LL_RTC_MONTH_MARCH– LL_RTC_MONTH_APRIIL– LL_RTC_MONTH_MAY– LL_RTC_MONTH_JUNE– LL_RTC_MONTH_JULY– LL_RTC_MONTH_AUGUST– LL_RTC_MONTH_SEPTMBER– LL_RTC_MONTH_OCTOBER– LL_RTC_MONTH_NOVEMBER– LL_RTC_MONTH_DECEMBER
Notes	<ul style="list-style-type: none">• if shadow mode is disabled (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:	<ul style="list-style-type: none">• DR MT LL_RTC_DATE_GetMonth• DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name	<u><code>__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)</code></u>
Function description	Set Day in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Day: Value between Min_Data=0x01 and Max_Data=0x31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR DT LL_RTC_DATE_SetDay• DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name	<u><code>__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (RTC_TypeDef * RTCx)</code></u>
Function description	Get Day in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x01 and Max_Data=0x31

Notes

- if shadow mode is disabled (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_APRIIL
 - 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	<code>__STATIC_INLINE uint32_t LL_RTC_DATE_Get (RTC_TypeDef * RTCx)</code>
Function description	Get date (WeekDay, Day, Month and Year) in BCD format.
Parameters	<ul style="list-style-type: none">• <code>RTCx</code>: RTC Instance
Return values	<ul style="list-style-type: none">• Combination: of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).
Notes	<ul style="list-style-type: none">• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit• helper macros <code>__LL_RTC_GET_WEEKDAY</code>, <code>__LL_RTC_GET_YEAR</code>, <code>__LL_RTC_GET_MONTH</code>, and <code>__LL_RTC_GET_DAY</code> are available to get independently each parameter.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)</code>
Function description	Enable Alarm A.
Parameters	<ul style="list-style-type: none">• <code>RTCx</code>: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name	<code>__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)</code>
Function description	Disable Alarm A.
Parameters	<ul style="list-style-type: none">• <code>RTCx</code>: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday
	LL_RTC_ALMA_DisableWeekday
Function name	<u>__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)</u>
Function description	Disable AlarmA Week day selection (DU[3:0] represents the date)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday
	LL_RTC_ALMA_SetDay
Function name	<u>__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)</u>
Function description	Set ALARM A Day in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Day: Value between Min_Data=0x01 and Max_Data=0x31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• helper macro <u>__LL_RTC_CONVERT_BIN2BCD</u> is available to convert Day from binary to BCD format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR DT LL_RTC_ALMA_SetDay• ALRMAR DU LL_RTC_ALMA_SetDay
	LL_RTC_ALMA_GetDay
Function name	<u>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (RTC_TypeDef * RTCx)</u>
Function description	Get ALARM A Day in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x01 and Max_Data=0x31
Notes	<ul style="list-style-type: none">• helper macro <u>__LL_RTC_CONVERT_BCD2BIN</u> is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR DT LL_RTC_ALMA_GetDay• ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name	<code>__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)</code>
Function description	Set ALARM A Weekday.
Parameters	<ul style="list-style-type: none">RTCx: RTC InstanceWeekDay: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (RTC_TypeDef * RTCx)</code>
Function description	Get ALARM A Weekday.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name	<code>__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)</code>
Function description	Set Alarm A time format (AM/24-hour or PM notation)

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• TimeFormat: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_ALMA_TIME_FORMAT_AM– LL_RTC_ALMA_TIME_FORMAT_PM
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR PM LL_RTC_ALMA_SetTimeFormat
	LL_RTC_ALMA_GetTimeFormat
Function name	<u>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (RTC_TypeDef * RTCx)</u>
Function description	Get Alarm A time format (AM or PM notation)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_ALMA_TIME_FORMAT_AM– LL_RTC_ALMA_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR PM LL_RTC_ALMA_GetTimeFormat
	LL_RTC_ALMA_SetHour
Function name	<u>__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)</u>
Function description	Set ALARM A Hours in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Hours: Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• helper macro <u>__LL_RTC_CONVERT_BIN2BCD</u> is available to convert Hours from binary to BCD format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMAR HT LL_RTC_ALMA_SetHour• ALRMAR HU LL_RTC_ALMA_SetHour
	LL_RTC_ALMA_GetHour
Function name	<u>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (RTC_TypeDef * RTCx)</u>
Function description	Get ALARM A Hours in BCD format.
Parameters	<ul style="list-style-type: none">• 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_SetHour
 - ALRMAR HU LL_RTC_ALMA_SetHour

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	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">helper macro <code>__LL_RTC_CONVERT_BIN2BCD</code> is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ALRMAR ST LL_RTC_ALMA_SetSecondALRMAR SU LL_RTC_ALMA_SetSecond
LL_RTC_ALMA_GetSecond	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (RTC_TypeDef * RTCx)</code>
Function description	Get ALARM A Seconds in BCD format.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0x59
Notes	<ul style="list-style-type: none">helper macro <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ALRMAR ST LL_RTC_ALMA_GetSecondALRMAR SU LL_RTC_ALMA_GetSecond
LL_RTC_ALMA_ConfigTime	
Function name	<code>__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)</code>
Function description	Set Alarm A Time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none">RTCx: RTC InstanceFormat12_24: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_RTC_ALMA_TIME_FORMAT_AM</code>– <code>LL_RTC_ALMA_TIME_FORMAT_PM</code>Hours: Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23Minutes: Value between Min_Data=0x00 and Max_Data=0x59Seconds: Value between Min_Data=0x00 and Max_Data=0x59
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ALRMAR PM LL_RTC_ALMA_ConfigTimeALRMAR HT LL_RTC_ALMA_ConfigTimeALRMAR HU LL_RTC_ALMA_ConfigTimeALRMAR MNT LL_RTC_ALMA_ConfigTimeALRMAR MNU LL_RTC_ALMA_ConfigTimeALRMAR ST LL_RTC_ALMA_ConfigTimeALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (RTC_TypeDef * RTCx)</code>
Function description	Get Alarm B Time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none">• <code>RTCx</code>: RTC Instance
Return values	<ul style="list-style-type: none">• Combination: of hours, minutes and seconds.
Notes	<ul style="list-style-type: none">• helper macros <code>__LL_RTC_GET_HOUR</code>, <code>__LL_RTC_GET_MINUTE</code> and <code>__LL_RTC_GET_SECOND</code> are available to get independently each parameter.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)</code>
Function description	Set Alarm A Mask the most-significant bits starting at this bit.
Parameters	<ul style="list-style-type: none">• <code>RTCx</code>: RTC Instance• <code>Mask</code>: Value between Min_Data=0x00 and Max_Data=0xF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (RTC_TypeDef * RTCx)</code>
Function description	Get Alarm A Mask the most-significant bits starting at this bit.
Parameters	<ul style="list-style-type: none">• <code>RTCx</code>: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0xF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetSubSecond

Function name	<code>__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)</code>
Function description	Set Alarm A Sub seconds value.
Parameters	<ul style="list-style-type: none">RTCx: RTC InstanceSubsecond: Value between Min_Data=0x00 and Max_Data=0x7FFF
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (RTC_TypeDef * RTCx)</code>
Function description	Get Alarm A Sub seconds value.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0x7FFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_TS_Enable

Function name	<code>__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)</code>
Function description	Enable Timestamp.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR TSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name	<code>__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)</code>
Function description	Disable Timestamp.
Parameters	<ul style="list-style-type: none">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 TSE LL_RTC_TS_Disable

`LL_RTC_TS_SetActiveEdge`

- Function name**
- `__STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)`
- Function description**
- Set Time-stamp event active edge.
- Parameters**
- **RTCx:** RTC Instance
 - **Edge:** This parameter can be one of the following values:
 - LL_RTC_TIMESTAMP_EDGE_RISING
 - LL_RTC_TIMESTAMP_EDGE_FALLING
- Return values**
- **None:**
- Notes**
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
 - TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting
- Reference Manual to LL API cross reference:**
- CR TSEDGE LL_RTC_TS_SetActiveEdge

`LL_RTC_TS_GetActiveEdge`

- Function name**
- `__STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (RTC_TypeDef * RTCx)`
- Function description**
- Get Time-stamp event active edge.
- Parameters**
- **RTCx:** RTC Instance
- Return values**
- **Returned:** value can be one of the following values:
 - LL_RTC_TIMESTAMP_EDGE_RISING
 - LL_RTC_TIMESTAMP_EDGE_FALLING
- Notes**
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Reference Manual to LL API cross reference:**
- CR TSEDGE LL_RTC_TS_SetActiveEdge

`LL_RTC_TS_GetTimeFormat`

- Function name**
- `__STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (RTC_TypeDef * RTCx)`
- Function description**
- Get Timestamp AM/PM notation (AM or 24-hour format)
- Parameters**
- **RTCx:** RTC Instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_TS_TIME_FORMAT_AM– LL_RTC_TS_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TSTR PM LL_RTC_TS_GetTimeFormat
LL_RTC_TS_GetHour	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (RTC_TypeDef * RTCx)</code>
Function description	Get Timestamp Hours in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	<ul style="list-style-type: none">• helper macro <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert Hours from BCD to Binary format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TSTR HT LL_RTC_TS_GetHour• TSTR HU LL_RTC_TS_GetHour
LL_RTC_TS_GetMinute	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (RTC_TypeDef * RTCx)</code>
Function description	Get Timestamp Minutes in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0x59
Notes	<ul style="list-style-type: none">• helper macro <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert Minutes from BCD to Binary format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TSTR MNT LL_RTC_TS_GetMinute• TSTR MNU LL_RTC_TS_GetMinute
LL_RTC_TS_GetSecond	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (RTC_TypeDef * RTCx)</code>
Function description	Get Timestamp Seconds in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0x59
Notes	<ul style="list-style-type: none">• helper macro <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert Seconds from BCD to Binary format

- Reference Manual to LL API cross reference:**
- TSTR ST LL_RTC_TS_GetSecond
 - TSTR SU LL_RTC_TS_GetSecond

LL_RTC_TS_GetTime

Function name `__STATIC_INLINE uint32_t LL_RTC_TS_GetTime (RTC_TypeDef * RTCx)`

Function description Get Timestamp time (hour, minute and second) in BCD format.

- Parameters**
- **RTCx:** RTC Instance

- Return values**
- **Combination:** of hours, minutes and seconds.

- Notes**
- helper macros `__LL_RTC_GET_HOUR`, `__LL_RTC_GET_MINUTE` and `__LL_RTC_GET_SECOND` are available to get independently each parameter.

- Reference Manual to LL API cross reference:**
- TSTR HT LL_RTC_TS_GetTime
 - TSTR HU LL_RTC_TS_GetTime
 - TSTR MNT LL_RTC_TS_GetTime
 - TSTR MNU LL_RTC_TS_GetTime
 - TSTR ST LL_RTC_TS_GetTime
 - TSTR SU LL_RTC_TS_GetTime

LL_RTC_TS_GetWeekDay

Function name `__STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay (RTC_TypeDef * RTCx)`

Function description Get Timestamp Week day.

- Parameters**
- **RTCx:** RTC Instance

- Return values**
- **Returned:** value can be one of the following values:
 - `LL_RTC_WEEKDAY_MONDAY`
 - `LL_RTC_WEEKDAY_TUESDAY`
 - `LL_RTC_WEEKDAY_WEDNESDAY`
 - `LL_RTC_WEEKDAY_THURSDAY`
 - `LL_RTC_WEEKDAY_FRIDAY`
 - `LL_RTC_WEEKDAY_SATURDAY`
 - `LL_RTC_WEEKDAY_SUNDAY`

- Reference Manual to LL API cross reference:**
- TSDR WDU LL_RTC_TS_GetWeekDay

LL_RTC_TS_GetMonth

Function name `__STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (RTC_TypeDef * RTCx)`

Function description Get Timestamp Month in BCD format.

- Parameters**
- **RTCx:** RTC Instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_MONTH_JANUARY– LL_RTC_MONTH_FEBRUARY– LL_RTC_MONTH_MARCH– LL_RTC_MONTH_APRIIL– 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	<ul style="list-style-type: none">• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TSDR MT LL_RTC_TS_GetMonth• TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name	<u>__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (RTC_TypeDef * RTCx)</u>
Function description	Get Timestamp Day in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x01 and Max_Data=0x31
Notes	<ul style="list-style-type: none">• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TSDR DT LL_RTC_TS_GetDay• TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name	<u>__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (RTC_TypeDef * RTCx)</u>
Function description	Get Timestamp date (WeekDay, Day and Month) in BCD format.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Combination: of Weekday, Day and Month
Notes	<ul style="list-style-type: none">• helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

- Reference Manual to LL API cross reference:**
- TSDR WDU LL_RTC_TS_GetDate
 - TSDR MT LL_RTC_TS_GetDate
 - TSDR MU LL_RTC_TS_GetDate
 - TSDR DT LL_RTC_TS_GetDate
 - TSDR DU LL_RTC_TS_GetDate

LL_RTC_TS_GetSubSecond

Function name `__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond (RTC_TypeDef * RTCx)`

Function description Get time-stamp sub second value.

Parameters • **RTCx:** RTC Instance

Return values • **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

- Reference Manual to LL API cross reference:**
- TSSSR SS LL_RTC_TS_GetSubSecond

LL_RTC_TS_EnableOnTamper

Function name `__STATIC_INLINE void LL_RTC_TS_EnableOnTamper (RTC_TypeDef * RTCx)`

Function description Activate timestamp on tamper detection event.

Parameters • **RTCx:** RTC Instance

Return values • **None:**

- Reference Manual to LL API cross reference:**
- TAFCR TAMPTS LL_RTC_TS_EnableOnTamper

LL_RTC_TS_DisableOnTamper

Function name `__STATIC_INLINE void LL_RTC_TS_DisableOnTamper (RTC_TypeDef * RTCx)`

Function description Disable timestamp on tamper detection event.

Parameters • **RTCx:** RTC Instance

Return values • **None:**

- Reference Manual to LL API cross reference:**
- TAFCR TAMPTS LL_RTC_TS_DisableOnTamper

LL_RTC_TAMPER_Enable

Function name `__STATIC_INLINE void LL_RTC_TAMPER_Enable (RTC_TypeDef * RTCx, uint32_t Tamper)`

Function description Enable RTC_TAMPx input detection.

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Tamper: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_RTC_TAMPER_1– LL_RTC_TAMPER_2– LL_RTC_TAMPER_3 (*)(*) value not defined in all devices.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMP1E LL_RTC_TAMPER_Enable• TAFCR TAMP2E LL_RTC_TAMPER_Enable• TAFCR TAMP3E LL_RTC_TAMPER_Enable

LL_RTC_TAMPER_Disable

Function name	<u><code>__STATIC_INLINE void LL_RTC_TAMPER_Disable (RTC_TypeDef * RTCx, uint32_t Tamper)</code></u>
Function description	Clear RTC_TAMPx input detection.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Tamper: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_RTC_TAMPER_1– LL_RTC_TAMPER_2– LL_RTC_TAMPER_3 (*)(*) value not defined in all devices.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMP1E LL_RTC_TAMPER_Disable• TAFCR TAMP2E LL_RTC_TAMPER_Disable• TAFCR TAMP3E LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_DisablePullUp

Function name	<u><code>__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (RTC_TypeDef * RTCx)</code></u>
Function description	Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name	<u><code>__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (RTC_TypeDef * RTCx)</code></u>
Function description	Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp
LL_RTC_TAMPER_SetPrecharge	
Function name	<code>__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (RTC_TypeDef * RTCx, uint32_t Duration)</code>
Function description	Set RTC_TAMPx precharge duration.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Duration: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_TAMPER_DURATION_1RTCCLK– LL_RTC_TAMPER_DURATION_2RTCCLK– LL_RTC_TAMPER_DURATION_4RTCCLK– LL_RTC_TAMPER_DURATION_8RTCCLK
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge
LL_RTC_TAMPER_GetPrecharge	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (RTC_TypeDef * RTCx)</code>
Function description	Get RTC_TAMPx precharge duration.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_TAMPER_DURATION_1RTCCLK– LL_RTC_TAMPER_DURATION_2RTCCLK– LL_RTC_TAMPER_DURATION_4RTCCLK– LL_RTC_TAMPER_DURATION_8RTCCLK
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge
LL_RTC_TAMPER_SetFilterCount	
Function name	<code>__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (RTC_TypeDef * RTCx, uint32_t FilterCount)</code>
Function description	Set RTC_TAMPx filter count.

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• FilterCount: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_TAMPER_FILTER_DISABLE– LL_RTC_TAMPER_FILTER_2SAMPLE– LL_RTC_TAMPER_FILTER_4SAMPLE– LL_RTC_TAMPER_FILTER_8SAMPLE
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPFLT LL_RTC_TAMPER_SetFilterCount
	LL_RTC_TAMPER_GetFilterCount
Function name	<u>__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (RTC_TypeDef * RTCx)</u>
Function description	Get RTC_TAMPx filter count.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_TAMPER_FILTER_DISABLE– LL_RTC_TAMPER_FILTER_2SAMPLE– LL_RTC_TAMPER_FILTER_4SAMPLE– LL_RTC_TAMPER_FILTER_8SAMPLE
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPFLT LL_RTC_TAMPER_GetFilterCount
	LL_RTC_TAMPER_SetSamplingFreq
Function name	<u>__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (RTC_TypeDef * RTCx, uint32_t SamplingFreq)</u>
Function description	Set Tamper sampling frequency.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• SamplingFreq: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_TAMPER_SAMPLFREQDIV_32768– LL_RTC_TAMPER_SAMPLFREQDIV_16384– LL_RTC_TAMPER_SAMPLFREQDIV_8192– LL_RTC_TAMPER_SAMPLFREQDIV_4096– LL_RTC_TAMPER_SAMPLFREQDIV_2048– LL_RTC_TAMPER_SAMPLFREQDIV_1024– LL_RTC_TAMPER_SAMPLFREQDIV_512– LL_RTC_TAMPER_SAMPLFREQDIV_256
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (RTC_TypeDef * RTCx)</code>
Function description	Get Tamper sampling frequency.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_RTC_TAMPER_SAMPLFREQDIV_32768</code>– <code>LL_RTC_TAMPER_SAMPLFREQDIV_16384</code>– <code>LL_RTC_TAMPER_SAMPLFREQDIV_8192</code>– <code>LL_RTC_TAMPER_SAMPLFREQDIV_4096</code>– <code>LL_RTC_TAMPER_SAMPLFREQDIV_2048</code>– <code>LL_RTC_TAMPER_SAMPLFREQDIV_1024</code>– <code>LL_RTC_TAMPER_SAMPLFREQDIV_512</code>– <code>LL_RTC_TAMPER_SAMPLFREQDIV_256</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name	<code>__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)</code>
Function description	Enable Active level for Tamper input.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Tamper: This parameter can be a combination of the following values:<ul style="list-style-type: none">– <code>LL_RTC_TAMPER_ACTIVELEVEL_TAMP1</code>– <code>LL_RTC_TAMPER_ACTIVELEVEL_TAMP2</code>– <code>LL_RTC_TAMPER_ACTIVELEVEL_TAMP3 (*)</code>
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel• TAFCR TAMP2TRG LL_RTC_TAMPER_EnableActiveLevel• TAFCR TAMP3TRG LL_RTC_TAMPER_EnableActiveLevel

LL_RTC_TAMPER_DisableActiveLevel

Function name	<code>__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)</code>
Function description	Disable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
 - **Tamper:** This parameter can be a combination of the following values:
 - LL_RTC_TAMPER_ACTIVELEVEL_TAMP1
 - LL_RTC_TAMPER_ACTIVELEVEL_TAMP2
 - LL_RTC_TAMPER_ACTIVELEVEL_TAMP3 (*)
- (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAFCR TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel
- TAFCR TAMP2TRG LL_RTC_TAMPER_DisableActiveLevel
- TAFCR TAMP3TRG LL_RTC_TAMPER_DisableActiveLevel

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	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR WUTE LL_RTC_WAKEUP_IsEnabled
LL_RTC_WAKEUP_SetClock	
Function name	<u>__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)</u>
Function description	Select Wakeup clock.
Parameters	<ul style="list-style-type: none">RTCx: RTC InstanceWakeupClock: This parameter can be one of the following values:<ul style="list-style-type: none">LL_RTC_WAKEUPCLOCK_DIV_16LL_RTC_WAKEUPCLOCK_DIV_8LL_RTC_WAKEUPCLOCK_DIV_4LL_RTC_WAKEUPCLOCK_DIV_2LL_RTC_WAKEUPCLOCK_CKSPRELL_RTC_WAKEUPCLOCK_CKSPRE_WUT
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR WUCKSEL LL_RTC_WAKEUP_SetClock
LL_RTC_WAKEUP_GetClock	
Function name	<u>__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (RTC_TypeDef * RTCx)</u>
Function description	Get Wakeup clock.
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_RTC_WAKEUPCLOCK_DIV_16LL_RTC_WAKEUPCLOCK_DIV_8LL_RTC_WAKEUPCLOCK_DIV_4LL_RTC_WAKEUPCLOCK_DIV_2LL_RTC_WAKEUPCLOCK_CKSPRELL_RTC_WAKEUPCLOCK_CKSPRE_WUT
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR WUCKSEL LL_RTC_WAKEUP_GetClock
LL_RTC_WAKEUP_SetAutoReload	
Function name	<u>__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)</u>

Function description	Set Wakeup auto-reload value.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Value: Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Bit can be written only when WUTWF is set to 1 in RTC_ISR
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• WUTR WUT LL_RTC_WAKEUP_SetAutoReload
LL_RTC_WAKEUP_SetAutoReload	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx)</code>
Function description	Get Wakeup auto-reload value.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• WUTR WUT LL_RTC_WAKEUP_GetAutoReload
LL_RTC_BAK_SetRegister	
Function name	<code>__STATIC_INLINE void LL_RTC_BAK_SetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)</code>
Function description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• BackupRegister: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• BKPxR BKP LL_RTC_BAK_SetRegister
LL_RTC_BAK_GetRegister	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_BAK_GetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister)</code>
Function description	Reads data from the specified RTC Backup data Register.

Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• BackupRegister: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_BKP_DR0– LL_RTC_BKP_DR1– LL_RTC_BKP_DR2– LL_RTC_BKP_DR3– LL_RTC_BKP_DR4
Return values	<ul style="list-style-type: none">• Value: between Min_Data=0x00 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• BKPxR BKP LL_RTC_BAK_GetRegister
LL_RTC_CAL_SetOutputFreq	
Function name	<code>__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)</code>
Function description	Set Calibration output frequency (1 Hz or 512 Hz)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• Frequency: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_RTC_CALIB_OUTPUT_NONE– LL_RTC_CALIB_OUTPUT_1HZ– LL_RTC_CALIB_OUTPUT_512HZ
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR COE LL_RTC_CAL_SetOutputFreq• CR COSEL LL_RTC_CAL_SetOutputFreq
LL_RTC_CAL_GetOutputFreq	
Function name	<code>__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (RTC_TypeDef * RTCx)</code>
Function description	Get Calibration output frequency (1 Hz or 512 Hz)
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_RTC_CALIB_OUTPUT_NONE– LL_RTC_CALIB_OUTPUT_1HZ– LL_RTC_CALIB_OUTPUT_512HZ
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR COE LL_RTC_CAL_GetOutputFreq• CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name	<code>__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)</code>
Function description	Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)
Parameters	<ul style="list-style-type: none">RTCx: RTC InstancePulse: This parameter can be one of the following values:<ul style="list-style-type: none">- <code>LL_RTC_CALIB_INSERTPULSE_NONE</code>- <code>LL_RTC_CALIB_INSERTPULSE_SET</code>
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Bit is write-protected. <code>LL_RTC_DisableWriteProtection</code> function should be called before.Bit can be written only when <code>RECALPF</code> is set to 0 in <code>RTC_ISR</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CALR CALP <code>LL_RTC_CAL_SetPulse</code>

LL_RTC_CAL_IsPulseInserted

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (RTC_TypeDef * RTCx)</code>
Function description	Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)
Parameters	<ul style="list-style-type: none">RTCx: RTC Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CALR CALP <code>LL_RTC_CAL_IsPulseInserted</code>

LL_RTC_CAL_SetPeriod

Function name	<code>__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)</code>
Function description	Set the calibration cycle period.
Parameters	<ul style="list-style-type: none">RTCx: RTC InstancePeriod: This parameter can be one of the following values:<ul style="list-style-type: none">- <code>LL_RTC_CALIB_PERIOD_32SEC</code>- <code>LL_RTC_CALIB_PERIOD_16SEC</code>- <code>LL_RTC_CALIB_PERIOD_8SEC</code>
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Bit is write-protected. <code>LL_RTC_DisableWriteProtection</code> function should be called before.Bit can be written only when <code>RECALPF</code> is set to 0 in <code>RTC_ISR</code>

- 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 the calibration cycle period.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
 - LL_RTC_CALIB_PERIOD_32SEC
 - LL_RTC_CALIB_PERIOD_16SEC
 - LL_RTC_CALIB_PERIOD_8SEC

- Reference Manual to LL API cross reference:**
- CALR CALW8 LL_RTC_CAL_SetPeriod
 - CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_SetMinus

Function name `__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)`

Function description Set Calibration minus.

Parameters

- **RTCx:** RTC Instance
- **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_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 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_SetMinus

LL_RTC_IsActiveFlag_RECALP

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (RTC_TypeDef * RTCx)</code>
Function description	Get Recalibration pending Flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TAMP3

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (RTC_TypeDef * RTCx)</code>
Function description	Get RTC_TAMP3 detection flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_TAMP2

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (RTC_TypeDef * RTCx)</code>
Function description	Get RTC_TAMP2 detection flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP1

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (RTC_TypeDef * RTCx)</code>
Function description	Get RTC_TAMP1 detection flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TSOV

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (RTC_TypeDef * RTCx)</code>
Function description	Get Time-stamp overflow flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (RTC_TypeDef * RTCx)</code>
Function description	Get Time-stamp flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (RTC_TypeDef * RTCx)</code>
Function description	Get Wakeup timer flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRA

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (RTC_TypeDef * RTCx)</code>
Function description	Get Alarm A flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_ClearFlag_TAMP3

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (RTC_TypeDef * RTCx)</code>
Function description	Clear RTC_TAMP3 detection flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_TAMP2

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (RTC_TypeDef * RTCx)</code>
Function description	Clear RTC_TAMP2 detection flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP1

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (RTC_TypeDef * RTCx)</code>
Function description	Clear RTC_TAMP1 detection flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TAMP1F LL_RTC_ClearFlag_TAMP1

LL_RTC_ClearFlag_TSOV

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)</code>
Function description	Clear Time-stamp overflow flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)</code>
Function description	Clear Time-stamp flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)</code>
Function description	Clear Wakeup timer flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR WUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRA

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)</code>
Function description	Clear Alarm A flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR ALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_IsActiveFlag_INIT

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (RTC_TypeDef * RTCx)</code>
Function description	Get Initialization flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (RTC_TypeDef * RTCx)</code>
Function description	Get Registers synchronization flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)</code>
Function description	Clear Registers synchronization flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (RTC_TypeDef * RTCx)</code>
Function description	Get Initialization status flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (RTC_TypeDef * RTCx)</code>
Function description	Get Shift operation pending flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (RTC_TypeDef * RTCx)</code>
Function description	Get Wakeup timer write flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW (RTC_TypeDef * RTCx)</code>
Function description	Get Alarm A write flag.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ALRAWF LL_RTC_IsActiveFlag_ALRAW

LL_RTC_EnableIT_TS

Function name	<code>__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)</code>
Function description	Enable Time-stamp interrupt.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR TSIE LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name	<code>__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)</code>
Function description	Disable Time-stamp interrupt.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:

- Notes**
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Reference Manual to LL API cross reference:**
- CR TSIE LL_RTC_DisableIT_TS

LL_RTC_EnableIT_WUT

- Function name**
- `__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)`
- Function description**
- Enable Wakeup timer interrupt.
- Parameters**
- **RTCx:** RTC Instance
- Return values**
- **None:**
- Notes**
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Reference Manual to LL API cross reference:**
- 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	<code>__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)</code>
Function description	Disable Alarm A interrupt.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR ALRAIE LL_RTC_DisableIT_ALRA

[LL_RTC_EnableIT_TAMP](#)

Function name	<code>__STATIC_INLINE void LL_RTC_EnableIT_TAMP (RTC_TypeDef * RTCx)</code>
Function description	Enable all Tamper Interrupt.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPIE LL_RTC_EnableIT_TAMP

[LL_RTC_DisableIT_TAMP](#)

Function name	<code>__STATIC_INLINE void LL_RTC_DisableIT_TAMP (RTC_TypeDef * RTCx)</code>
Function description	Disable all Tamper Interrupt.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TAFCR TAMPIE LL_RTC_DisableIT_TAMP

[LL_RTC_IsEnabledIT_TS](#)

Function name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (RTC_TypeDef * RTCx)</code>
Function description	Check if Time-stamp interrupt is enabled or not.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TSIE LL_RTC_IsEnabledIT_TS

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_IsEnabledIT_TAMP

Function name `__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP (RTC_TypeDef * RTCx)`

Function description Check if all the TAMPER interrupts are enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

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

Reference Manual to LL API cross reference:

- TAFCR TAMPIE LL_RTC_IsEnabledIT_TAMP

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	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: RTC registers are de-initialized– ERROR: RTC registers are not de-initialized
Notes	<ul style="list-style-type: none">• This function doesn't 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	<ul style="list-style-type: none">• RTCx: RTC Instance• RTC_InitStruct: pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: RTC registers are initialized– ERROR: RTC registers are not initialized
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• RTC_InitStruct: pointer to a LL_RTC_InitTypeDef structure which will be initialized.
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• RTCx: RTC Instance• RTC_Format: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: RTC Time register is configured– ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name	<code>void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)</code>
Function description	Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).
Parameters	<ul style="list-style-type: none">• RTC_TimeStruct: pointer to a LL_RTC_TimeTypeDef structure which will be initialized.
Return values	<ul style="list-style-type: none">• None:

LL_RTC_DATE_Init

Function name	<code>ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)</code>
Function description	Set the RTC current date.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• RTC_Format: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: RTC Day register is configured– ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name	<code>void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)</code>
Function description	Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)
Parameters	<ul style="list-style-type: none">• RTC_DateStruct: pointer to a LL_RTC_DateTypeDef structure which will be initialized.
Return values	<ul style="list-style-type: none">• None:

LL_RTC_ALMA_Init

Function name	<code>ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)</code>
Function description	Set the RTC Alarm A.
Parameters	<ul style="list-style-type: none">• RTCx: RTC Instance• RTC_Format: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• RTCx: RTC Instance
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: RTC registers are synchronised– ERROR: RTC registers are not synchronised
Notes	<ul style="list-style-type: none">• The RTC Resynchronization mode is write protected, use the <code>LL_RTC_DisableWriteProtection</code> 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.

65.3 RTC Firmware driver defines

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

65.3.1 RTC

RTC

ALARM OUTPUT

LL_RTC_ALARMOUT_DISA Output disabled
BLE

LL_RTC_ALARMOUT_ALM Alarm A output enabled
A

LL_RTC_ALARMOUT_ALM Alarm B output enabled
B

LL_RTC_ALARMOUT_WAK Wakeup output enabled
EUP

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUT RTC_ALARM, when mapped on PC13, is open-drain output
TYPE_OPENDRAIN

LL_RTC_ALARM_OUTPUT RTC_ALARM, when mapped on PC13, is push-pull output
TYPE_PUSHULL

ALARMA MASK

LL_RTC_ALMA_MASK_NO No masks applied on Alarm A
NE

LL_RTC_ALMA_MASK_DA Date/day do not care in Alarm A comparison
TEWEEKDAY

LL_RTC_ALMA_MASK_HO Hours do not care in Alarm A comparison
URS

LL_RTC_ALMA_MASK_MI Minutes do not care in Alarm A comparison
NUTES

LL_RTC_ALMA_MASK_SE Seconds do not care in Alarm A comparison
CONDSE

LL_RTC_ALMA_MASK_AL Masks all
L

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FOR AM or 24-hour format
MAT_AM

LL_RTC_ALMA_TIME_FOR PM
MAT_PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEE Alarm A Date is selected
KDAYSEL_DATE

LL_RTC_ALMA_DATEWEE Alarm A WeekDay is selected
KDAYSEL_WEEKDAY

BACKUP

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

Calibration pulse insertion

LL_RTC_CALIB_INSERTPU No RTCCLK pulses are added
LSE_NONE

LL_RTC_CALIB_INSERTPU One RTCCLK pulse is effectively inserted every $2^{exp(11)}$ pulses (frequency increased by 488.5 ppm)
LSE_SET

Calibration output

LL_RTC_CALIB_OUTPUT_ Calibration output disabled
NONE

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_3SEC Use a 32-second calibration cycle period

LL_RTC_CALIB_PERIOD_6SEC 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_TAMP3F

LL_RTC_ISR_TAMP2F

LL_RTC_ISR_TAMP1F

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_2 24 hour/day format
4HOUR

LL_RTC_HOURFORMAT_A AM/PM hour format
MPM

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRAIE

LL_RTC_TAFCR_TAMPIE

MONTH

LL_RTC_MONTH_JANUAR January
Y

LL_RTC_MONTH_FEBRUA February
RY

LL_RTC_MONTH_MARCH March

LL_RTC_MONTH_APRL April

LL_RTC_MONTH_MAY May

LL_RTC_MONTH_JUNE June

LL_RTC_MONTH_JULY July

LL_RTC_MONTH_AUGUST August

LL_RTC_MONTH_SEPTEM September
BER

LL_RTC_MONTH_OCTOBE October
R

LL_RTC_MONTH_NOVEMB November
ER

LL_RTC_MONTH_DECEMB December
ER

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPolarit Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)
Y_PIN_HIGH

LL_RTC_OUTPUTPolarit Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)
Y_PIN_LOW

PIN

LL_RTC_PIN_PC13 PC13 is forced to push-pull output if all RTC alternate functions are disabled

LL_RTC_PIN_PC14 PC14 is forced to push-pull output if LSE is disabled

LL_RTC_PIN_PC15 PC15 is forced to push-pull output if LSE is disabled

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

LL_RTC_SHIFT_SECOND_ADVANCE

TAMPER

LL_RTC_TAMPER_1 RTC_TAMP1 input detection

LL_RTC_TAMPER_2 RTC_TAMP2 input detection

LL_RTC_TAMPER_3 RTC_TAMP3 input detection

TAMPER ACTIVE LEVEL

LL_RTC_TAMPER_ACTIVE_LEVEL_TAMP1 RTC_TAMP1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVE_LEVEL_TAMP2 RTC_TAMP2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVE_LEVEL_TAMP3 RTC_TAMP3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

TAMPER DURATION

LL_RTC_TAMPER_DURATION_ON_1RTCCLK Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

LL_RTC_TAMPER_DURATION_ON_2RTCCLK Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

LL_RTC_TAMPER_DURATION_ON_4RTCCLK Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

LL_RTC_TAMPER_DURATION_ON_8RTCCLK Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE Tamper filter is disabled

LL_RTC_TAMPER_FILTER_2SAMPLE Tamper is activated after 2 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_4SAMPLE Tamper is activated after 4 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_8SAMPLE Tamper is activated after 8 consecutive samples at the active level.

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1 Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware. The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER2 Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

LL_RTC_TAMPER_MASK_TAMPER3 Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased

TAMPER NO ERASE

LL_RTC_TAMPER_NOERA_SE_TAMPER1 Tamper 1 event does not erase the backup registers.

LL_RTC_TAMPER_NOERA_SE_TAMPER2 Tamper 2 event does not erase the backup registers.

LL_RTC_TAMPER_NOERA_SE_TAMPER3 Tamper 3 event does not erase the backup registers.

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPL_FREQDIV_32768 Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

LL_RTC_TAMPER_SAMPL_FREQDIV_16384 Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

LL_RTC_TAMPER_SAMPL_FREQDIV_8192 Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

LL_RTC_TAMPER_SAMPL_FREQDIV_4096 Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

LL_RTC_TAMPER_SAMPL_FREQDIV_2048 Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

LL_RTC_TAMPER_SAMPL Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
FREQDIV_1024

LL_RTC_TAMPER_SAMPL Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
FREQDIV_512

LL_RTC_TAMPER_SAMPL Each of the tamper inputs are sampled with a frequency = RTCCLK / 256
FREQDIV_256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDG RTC_TS input rising edge generates a time-stamp event
E_RISING

LL_RTC_TIMESTAMP_EDG RTC_TS input falling edge generates a time-stamp even
E_FALLING

TIME FORMAT

LL_RTC_TIME_FORMAT_A AM or 24-hour format
M_OR_24

LL_RTC_TIME_FORMAT_P PM
M

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMA AM or 24-hour format
T_AM

LL_RTC_TS_TIME_FORMA PM
T_PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_ RTC/16 clock is selected
DIV_16

LL_RTC_WAKEUPCLOCK_ RTC/8 clock is selected
DIV_8

LL_RTC_WAKEUPCLOCK_ RTC/4 clock is selected
DIV_4

LL_RTC_WAKEUPCLOCK_ RTC/2 clock is selected
DIV_2

LL_RTC_WAKEUPCLOCK_ ck_spre (usually 1 Hz) clock is selected
CKSPRE

LL_RTC_WAKEUPCLOCK_ ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value
CKSPRE_WUT

WEEK DAY

`LL_RTC_WEEKDAY_MOND` Monday
AY

`LL_RTC_WEEKDAY_TUES` Tuesday
DAY

`LL_RTC_WEEKDAY_WEDN` Wednesday
ESDAY

`LL_RTC_WEEKDAY_THUR` Thursday
SDAY

`LL_RTC_WEEKDAY_FRIDA` Friday
Y

`LL_RTC_WEEKDAY_SATU` Saturday
RDAY

`LL_RTC_WEEKDAY_SUND` Sunday
AY

Convert helper Macros

`_LL_RTC_CONVERT_BIN` **Description:**
`2BCD`

- 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_BCD` **Description:**
`2BIN`

- 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_APRIIL
 - 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

66 LL SPI Generic Driver

66.1 SPI Firmware driver registers structures

66.1.1 LL_SPI_InitTypeDef

`LL_SPI_InitTypeDef` is defined in the `stm32f0xx_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()**.

66.2 SPI Firmware driver API description

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

66.2.1 Detailed description of functions

LL_SPI_Enable

Function name	<code>__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)</code>
Function description	Enable SPI peripheral.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name	<code>__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)</code>
Function description	Disable SPI peripheral.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• When disabling the SPI, follow the procedure described in the Reference Manual.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabled (SPI_TypeDef * SPIx)</code>
Function description	Check if SPI peripheral is enabled.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• SPIx: SPI Instance• BaudRate: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• These bits should not be changed when communication is ongoing. SPI BaudRate = fPCLK/Prescaler.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 BR LL_SPI_SetBaudRatePrescaler
Function name	<code>LL_SPI_GetBaudRatePrescaler</code>
Function description	Get baud rate prescaler.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• CR1 BR LL_SPI_GetBaudRatePrescaler
Function name	<code>LL_SPI_SetTransferBitOrder</code>
Function description	Set transfer bit order.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• BitOrder: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_SPI_LSB_FIRST– LL_SPI_MSB_FIRST

Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">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_SIMPLE_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	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• CR1 RXONLY LL_SPI_GetTransferDirection• CR1 BIDIMODE LL_SPI_GetTransferDirection• CR1 BIDIOE LL_SPI_GetTransferDirection
LL_SPI_SetDataWidth	
Function name	<u>__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)</u>
Function description	Set frame data width.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• DataWidth: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 DS LL_SPI_SetDataWidth
LL_SPI.GetDataWidth	
Function name	<u>__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (SPI_TypeDef * SPIx)</u>
Function description	Get frame data width.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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:	<ul style="list-style-type: none">• CR2 DS LL_SPI_GetDataWidth
	LL_SPI_SetRxFIFOThreshold
Function name	<u>__STATIC_INLINE void LL_SPI_SetRxFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)</u>
Function description	Set threshold of RXFIFO that triggers an RXNE event.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• Threshold: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_SPI_RX_FIFO_TH_HALF– LL_SPI_RX_FIFO_TH_QUARTER
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 FRXTH LL_SPI_SetRxFIFOThreshold
	LL_SPI_GetRxFIFOThreshold
Function name	<u>__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOThreshold (SPI_TypeDef * SPIx)</u>
Function description	Get threshold of RXFIFO that triggers an RXNE event.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_SPI_RX_FIFO_TH_HALF– LL_SPI_RX_FIFO_TH_QUARTER
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 FRXTH LL_SPI_GetRxFIFOThreshold

LL_SPI_EnableCRC

Function name	<u>__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)</u>
Function description	Enable CRC.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name	<u>__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)</u>
Function description	Disable CRC.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name	<u>__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (SPI_TypeDef * SPIx)</u>
Function description	Check if CRC is enabled.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCWidth

Function name	<u>__STATIC_INLINE void LL_SPI_SetCRCWidth (SPI_TypeDef * SPIx, uint32_t CRCLength)</u>
Function description	Set CRC Length.

Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• CRCLength: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_SPI_CRC_8BIT– LL_SPI_CRC_16BIT
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 CRCL LL_SPI_SetCRCWidth
LL_SPI_GetCRCWidth	
Function name	<code>__STATIC_INLINE uint32_t LL_SPI_GetCRCWidth (SPI_TypeDef * SPIx)</code>
Function description	Get CRC Length.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_SPI_CRC_8BIT– LL_SPI_CRC_16BIT
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 CRCL LL_SPI_GetCRCWidth
LL_SPI_SetCRCNext	
Function name	<code>__STATIC_INLINE void LL_SPI_SetCRCNext (SPI_TypeDef * SPIx)</code>
Function description	Set CRCNext to transfer CRC on the line.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR1 CRCNEXT LL_SPI_SetCRCNext
LL_SPI_SetCRCPolynomial	
Function name	<code>__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)</code>
Function description	Set polynomial for CRC calculation.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• CRCPoly: This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CRCPR CRCPOLY LL_SPI_SetCRCPolynomial
LL_SPI_GetCRCPolynomial	
Function name	<code>__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (SPI_TypeDef * SPIx)</code>
Function description	Get polynomial for CRC calculation.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CRCPR CRCPOLY LL_SPI_GetCRCPolynomial
LL_SPI_GetRxCRC	
Function name	<code>__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (SPI_TypeDef * SPIx)</code>
Function description	Get Rx CRC.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RXCRCR RXCRC LL_SPI_GetRxCRC
LL_SPI_GetTxCRC	
Function name	<code>__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)</code>
Function description	Get Tx CRC.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• TXCRCR TXCRC LL_SPI_GetTxCRC
LL_SPI_SetNSSMode	
Function name	<code>__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)</code>
Function description	Set NSS mode.

Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance• NSS: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_SPI_NSS_SOFT– LL_SPI_NSS_HARD_INPUT– LL_SPI_NSS_HARD_OUTPUT
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 SSM LL_SPI_SetNSSMode• CR2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name	<code>__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (SPI_TypeDef * SPIx)</code>
Function description	Get NSS mode.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_SPI_NSS_SOFT– LL_SPI_NSS_HARD_INPUT– LL_SPI_NSS_HARD_OUTPUT
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 SSM LL_SPI_GetNSSMode• CR2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name	<code>__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)</code>
Function description	Enable NSS pulse management.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR2 NSSP LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name	<code>__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)</code>
---------------	--

Function description	Disable NSS pulse management.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 NSSP LL_SPI_DisableNSSPulseMgt

LL_SPI_IsEnabledNSSPulse

Function name	<u>__STATIC_INLINE uint32_t LL_SPI_IsEnabledNSSPulse (SPI_TypeDef * SPIx)</u>
Function description	Check if NSS pulse is enabled.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 NSSP LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXNE

Function name	<u>__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)</u>
Function description	Check if Rx buffer is not empty.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR RXNE LL_SPI_IsActiveFlag_RXNE

LL_SPI_IsActiveFlag_TXE

Function name	<u>__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXE (SPI_TypeDef * SPIx)</u>
Function description	Check if Tx buffer is empty.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Clearing this flag is done by reading SPIx_SR register
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR FRE LL_SPI_ClearFlag_FRE

LL_SPI_EnableIT_ERR

Function name	<u>__STATIC_INLINE void LL_SPI_EnableIT_ERR (SPI_TypeDef * SPIx)</u>
Function description	Enable error interrupt.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 ERRIE LL_SPI_EnableIT_ERR

LL_SPI_EnableIT_RXNE

Function name	<u>__STATIC_INLINE void LL_SPI_EnableIT_RXNE (SPI_TypeDef * SPIx)</u>
Function description	Enable Rx buffer not empty interrupt.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 RXNEIE LL_SPI_EnableIT_RXNE

LL_SPI_EnableIT_TXE

Function name	<u>__STATIC_INLINE void LL_SPI_EnableIT_TXE (SPI_TypeDef * SPIx)</u>
Function description	Enable Tx buffer empty interrupt.
Parameters	<ul style="list-style-type: none">SPIx: SPI Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 TXEIE LL_SPI_EnableIT_TXE

LL_SPI_DisableIT_ERR

Function name	<code>__STATIC_INLINE void LL_SPI_DisableIT_ERR (SPI_TypeDef * SPIx)</code>
Function description	Disable error interrupt.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR2 ERRIE LL_SPI_DisableIT_ERR

LL_SPI_DisableIT_RXNE

Function name	<code>__STATIC_INLINE void LL_SPI_DisableIT_RXNE (SPI_TypeDef * SPIx)</code>
Function description	Disable Rx buffer not empty interrupt.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 RXNEIE LL_SPI_DisableIT_RXNE

LL_SPI_DisableIT_TXE

Function name	<code>__STATIC_INLINE void LL_SPI_DisableIT_TXE (SPI_TypeDef * SPIx)</code>
Function description	Disable Tx buffer empty interrupt.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 TXEIE LL_SPI_DisableIT_TXE

LL_SPI_IsEnabledIT_ERR

Function name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_ERR (SPI_TypeDef * SPIx)</code>
Function description	Check if error interrupt is enabled.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• 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:](#)

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:](#)

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:](#)

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:](#)

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:](#)

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:](#)

LL_SPI_GetDMAParity_TX

Function name	<u>__STATIC_INLINE uint32_t LL_SPI_GetDMAParity_TX (SPI_TypeDef * SPIx)</u>
Function description	Get parity configuration for Last DMA transmission.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_SPI_DMA_PARITY_ODD– LL_SPI_DMA_PARITY EVEN
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 LDMATX LL_SPI_GetDMAParity_TX

LL_SPI_DMA_GetRegAddr

Function name	<u>__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr (SPI_TypeDef * SPIx)</u>
Function description	Get the data register address used for DMA transfer.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• Address: of data register
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR DR LL_SPI_DMA_GetRegAddr

LL_SPI_ReceiveData8

Function name	<u>__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)</u>
Function description	Read 8-Bits in the data register.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• RxData: Value between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DR DR LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name	<u>__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)</u>
Function description	Read 16-Bits in the data register.
Parameters	<ul style="list-style-type: none">• SPIx: SPI Instance
Return values	<ul style="list-style-type: none">• 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_DelInit

Function name `ErrorStatus LL_SPI_DelInit (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 | <ul style="list-style-type: none">• An: ErrorStatus enumeration value. (Return always SUCCESS) |
| Notes | <ul style="list-style-type: none">• As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned. |

LL_SPI_StructInit

Function name **void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)**

Function description Set each LL_SPI_InitTypeDef field to default value.

Parameters

- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

66.3 SPI Firmware driver defines

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

66.3.1 SPI

SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRES2 BaudRate control equal to fPCLK/2
ALER_DIV2

LL_SPI_BAUDRATEPRES4 BaudRate control equal to fPCLK/4
ALER_DIV4

LL_SPI_BAUDRATEPRES8 BaudRate control equal to fPCLK/8
ALER_DIV8

LL_SPI_BAUDRATEPRES16 BaudRate control equal to fPCLK/16
ALER_DIV16

LL_SPI_BAUDRATEPRES32 BaudRate control equal to fPCLK/32
ALER_DIV32

LL_SPI_BAUDRATEPRES64 BaudRate control equal to fPCLK/64
ALER_DIV64

LL_SPI_BAUDRATEPRES128 BaudRate control equal to fPCLK/128
ALER_DIV128

LL_SPI_BAUDRATEPRES256 BaudRate control equal to fPCLK/256
ALER_DIV256

Transmission Bit Order

LL_SPI_LSB_FIRST Data is transmitted/received with the LSB first

LL_SPI_MSB_FIRST Data is transmitted/received with the MSB first

CRC Calculation

`LL_SPI_CRCCALCULATION_DISABLE` CRC calculation disabled

`LL_SPI_CRCCALCULATION_ENABLE` CRC calculation enabled

CRC Length

`LL_SPI_CRC_8BIT` 8-bit CRC length

`LL_SPI_CRC_16BIT` 16-bit CRC length

Datawidth

`LL_SPI_DATAWIDTH_4BIT` Data length for SPI transfer: 4 bits

`LL_SPI_DATAWIDTH_5BIT` Data length for SPI transfer: 5 bits

`LL_SPI_DATAWIDTH_6BIT` Data length for SPI transfer: 6 bits

`LL_SPI_DATAWIDTH_7BIT` Data length for SPI transfer: 7 bits

`LL_SPI_DATAWIDTH_8BIT` Data length for SPI transfer: 8 bits

`LL_SPI_DATAWIDTH_9BIT` Data length for SPI transfer: 9 bits

`LL_SPI_DATAWIDTH_10BIT` Data length for SPI transfer: 10 bits

`LL_SPI_DATAWIDTH_11BIT` Data length for SPI transfer: 11 bits

`LL_SPI_DATAWIDTH_12BIT` Data length for SPI transfer: 12 bits

`LL_SPI_DATAWIDTH_13BIT` Data length for SPI transfer: 13 bits

`LL_SPI_DATAWIDTH_14BIT` Data length for SPI transfer: 14 bits

`LL_SPI_DATAWIDTH_15BIT` Data length for SPI transfer: 15 bits

`LL_SPI_DATAWIDTH_16BIT` Data length for SPI transfer: 16 bits

DMA Parity

`LL_SPI_DMA_PARITY_EVEN` Select DMA parity Even

`LL_SPI_DMA_PARITY_ODD` Select DMA parity Odd

Get Flags Defines

`LL_SPI_SR_RXNE` Rx buffer not empty flag

<code>LL_SPI_SR_TXE</code>	Tx buffer empty flag
<code>LL_SPI_SR_BSY</code>	Busy flag
<code>LL_SPI_SR_CRCERR</code>	CRC error flag
<code>LL_SPI_SR_MODF</code>	Mode fault flag
<code>LL_SPI_SR_OVR</code>	Overrun flag
<code>LL_SPI_SR_FRE</code>	TI mode frame format error flag

IT Defines

<code>LL_SPI_CR2_RXNEIE</code>	Rx buffer not empty interrupt enable
<code>LL_SPI_CR2_TXEIE</code>	Tx buffer empty interrupt enable
<code>LL_SPI_CR2_ERRIE</code>	Error interrupt enable

Operation Mode

<code>LL_SPI_MODE_MASTER</code>	Master configuration
<code>LL_SPI_MODE_SLAVE</code>	Slave configuration

Slave Select Pin Mode

<code>LL_SPI_NSS_SOFT</code>	NSS managed internally. NSS pin not used and free
<code>LL_SPI_NSS_HARD_INPUT</code>	NSS pin used in Input. Only used in Master mode
<code>LL_SPI_NSS_HARD_OUTP</code>	NSS pin used in Output. Only used in Slave mode as chip select UT

Clock Phase

<code>LL_SPI_PHASE_1EDGE</code>	First clock transition is the first data capture edge
<code>LL_SPI_PHASE_2EDGE</code>	Second clock transition is the first data capture edge

Clock Polarity

<code>LL_SPI_POLARITY_LOW</code>	Clock to 0 when idle
<code>LL_SPI_POLARITY_HIGH</code>	Clock to 1 when idle

Serial Protocol

<code>LL_SPI_PROTOCOL_MOTO</code>	Motorola mode. Used as default value ROLA
<code>LL_SPI_PROTOCOL_TI</code>	TI mode

RX FIFO Level

LL_SPI_RX_FIFO_EMPTY FIFO reception empty

LL_SPI_RX_FIFO_QUARTE R_FULL FIFO reception 1/4

LL_SPI_RX_FIFO_HALF_F ULL FIFO reception 1/2

LL_SPI_RX_FIFO_FULL FIFO reception full

RX FIFO Threshold

LL_SPI_RX_FIFO_TH_HAL F RXNE event is generated if FIFO level is greater than or equal to 1/2 (16-bit)

LL_SPI_RX_FIFO_TH_QUA RTER 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 X 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_QUARTE R_FULL FIFO transmission 1/4

LL_SPI_TX_FIFO_HALF_F ULL 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

67 LL SYSTEM Generic Driver

67.1 SYSTEM Firmware driver API description

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

67.1.1 Detailed description of functions

`LL_SYSCFG_SetRemapMemory`

Function name	<code>__STATIC_INLINE void LL_SYSCFG_SetRemapMemory (uint32_t Memory)</code>
Function description	Set memory mapping at address 0x00000000.
Parameters	<ul style="list-style-type: none">Memory: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_SYSCFG_REMAP_FLASH</code>– <code>LL_SYSCFG_REMAP_SYSTEMFLASH</code>– <code>LL_SYSCFG_REMAP_SRAM</code>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">– <code>SYSCFG_CFGR1 MEM_MODE LL_SYSCFG_SetRemapMemory</code>

`LL_SYSCFG_GetRemapMemory`

Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_GetRemapMemory (void)</code>
Function description	Get memory mapping at address 0x00000000.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_SYSCFG_REMAP_FLASH</code>– <code>LL_SYSCFG_REMAP_SYSTEMFLASH</code>– <code>LL_SYSCFG_REMAP_SRAM</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">– <code>SYSCFG_CFGR1 MEM_MODE LL_SYSCFG_GetRemapMemory</code>

`LL_SYSCFG_SetIRModEnvelopeSignal`

Function name	<code>__STATIC_INLINE void LL_SYSCFG_SetIRModEnvelopeSignal (uint32_t Source)</code>
Function description	Set IR Modulation Envelope signal source.
Parameters	<ul style="list-style-type: none">Source: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_SYSCFG_IR_MOD_TIM16</code>– <code>LL_SYSCFG_IR_MOD_USART1</code>– <code>LL_SYSCFG_IR_MOD_USART4</code>
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL
API cross reference:

- SYSCFG_CFGR1 IR_MOD LL_SYSCFG_SetIRModEnvelopeSignal

LL_SYSCFG_GetIRModEnvelopeSignal

Function name **__STATIC_INLINE uint32_t LL_SYSCFG_GetIRModEnvelopeSignal (void)**

Function description Get IR Modulation Envelope signal source.

Return values

- Returned:** value can be one of the following values:
 - LL_SYSCFG_IR_MOD_TIM16
 - LL_SYSCFG_IR_MOD_USART1
 - LL_SYSCFG_IR_MOD_USART4

Reference Manual to LL
API cross reference:

- SYSCFG_CFGR1 IR_MOD LL_SYSCFG_SetIRModEnvelopeSignal

LL_SYSCFG_EnableFastModePlus

Function name **__STATIC_INLINE void LL_SYSCFG_EnableFastModePlus (uint32_t ConfigFastModePlus)**

Function description Enable the I2C fast mode plus driving capability.

Parameters

- ConfigFastModePlus:** This parameter can be a combination of the following values:
 - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
 - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
 - LL_SYSCFG_I2C_FASTMODEPLUS_PB8
 - LL_SYSCFG_I2C_FASTMODEPLUS_PB9
 - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1 (*)
 - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
 - LL_SYSCFG_I2C_FASTMODEPLUS_PA9 (*)
 - LL_SYSCFG_I2C_FASTMODEPLUS_PA10 (*)

(*) value not defined in all devices

Return values

- None:**

Reference Manual to LL
API cross reference:

- SYSCFG_CFGR1 I2C_FMP_PB6 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB7 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB8 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB9 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_I2C1 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_I2C2 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PA9 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PA10 LL_SYSCFG_EnableFastModePlus

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	<ul style="list-style-type: none">ConfigFastModePlus: This parameter can be a combination of the following values:<ul style="list-style-type: none">LL_SYSCFG_I2C_FASTMODEPLUS_PB6LL_SYSCFG_I2C_FASTMODEPLUS_PB7LL_SYSCFG_I2C_FASTMODEPLUS_PB8LL_SYSCFG_I2C_FASTMODEPLUS_PB9LL_SYSCFG_I2C_FASTMODEPLUS_I2C1 (*)LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)LL_SYSCFG_I2C_FASTMODEPLUS_PA9 (*)LL_SYSCFG_I2C_FASTMODEPLUS_PA10 (*) <p>(*) value not defined in all devices</p>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SYSCFG_CFGR1_I2C_FMP_PB6 LL_SYSCFG_DisableFastModePlusSYSCFG_CFGR1_I2C_FMP_PB7 LL_SYSCFG_DisableFastModePlusSYSCFG_CFGR1_I2C_FMP_PB8 LL_SYSCFG_DisableFastModePlusSYSCFG_CFGR1_I2C_FMP_PB9 LL_SYSCFG_DisableFastModePlusSYSCFG_CFGR1_I2C_FMP_I2C1 LL_SYSCFG_DisableFastModePlusSYSCFG_CFGR1_I2C_FMP_I2C2 LL_SYSCFG_DisableFastModePlusSYSCFG_CFGR1_I2C_FMP_PA9 LL_SYSCFG_DisableFastModePlusSYSCFG_CFGR1_I2C_FMP_PA10 LL_SYSCFG_DisableFastModePlus

LL_SYSCFG_SetEXTISource

Function name	<u>__STATIC_INLINE void LL_SYSCFG_SetEXTISource (uint32_t Port, uint32_t Line)</u>
Function description	Configure source input for the EXTI external interrupt.

Parameters	<ul style="list-style-type: none">• Port: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_SYSCFG EXTI_PORTA– LL_SYSCFG EXTI_PORTB– LL_SYSCFG EXTI_PORTC– LL_SYSCFG EXTI_PORTD (*)– LL_SYSCFG EXTI PORTE (*)– LL_SYSCFG EXTI PORTF(*) value not defined in all devices• Line: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_SYSCFG EXTI_LINE0– LL_SYSCFG EXTI_LINE1– LL_SYSCFG EXTI_LINE2– LL_SYSCFG EXTI_LINE3– LL_SYSCFG EXTI_LINE4– LL_SYSCFG EXTI_LINE5– LL_SYSCFG EXTI_LINE6– LL_SYSCFG EXTI_LINE7– LL_SYSCFG EXTI_LINE8– LL_SYSCFG EXTI_LINE9– LL_SYSCFG EXTI_LINE10– LL_SYSCFG EXTI_LINE11– LL_SYSCFG EXTI_LINE12– LL_SYSCFG EXTI_LINE13– LL_SYSCFG EXTI_LINE14– LL_SYSCFG EXTI_LINE15
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_EXTICR1 EXTI0 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR1 EXTI1 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR1 EXTI2 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR1 EXTI3 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR2 EXTI4 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR2 EXTI5 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR2 EXTI6 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR2 EXTI7 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR3 EXTI8 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR3 EXTI9 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR3 EXTI10 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR3 EXTI11 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR4 EXTI12 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR4 EXTI13 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR4 EXTI14 LL_SYSCFG_SetEXTISource• SYSCFG_EXTICR4 EXTI15 LL_SYSCFG_SetEXTISource

LL_SYSCFG_GetEXTISource

Function name [__STATIC_INLINE uint32_t LL_SYSCFG_GetEXTISource \(uint32_t Line\)](#)

Function description Get the configured defined for specific EXTI Line.

Parameters

- **Line:** This parameter can be one of the following values:
 - LL_SYSCFG_EXTI_LINE0
 - LL_SYSCFG_EXTI_LINE1
 - LL_SYSCFG_EXTI_LINE2
 - LL_SYSCFG_EXTI_LINE3
 - LL_SYSCFG_EXTI_LINE4
 - LL_SYSCFG_EXTI_LINE5
 - LL_SYSCFG_EXTI_LINE6
 - LL_SYSCFG_EXTI_LINE7
 - LL_SYSCFG_EXTI_LINE8
 - LL_SYSCFG_EXTI_LINE9
 - LL_SYSCFG_EXTI_LINE10
 - LL_SYSCFG_EXTI_LINE11
 - LL_SYSCFG_EXTI_LINE12
 - LL_SYSCFG_EXTI_LINE13
 - LL_SYSCFG_EXTI_LINE14
 - LL_SYSCFG_EXTI_LINE15

Return values

- **Returned:** value can be one of the following values:
 - LL_SYSCFG_EXTI_PORTA
 - LL_SYSCFG_EXTI_PORTB
 - LL_SYSCFG_EXTI_PORTC
 - LL_SYSCFG_EXTI_PORTD (*)
 - LL_SYSCFG_EXTI_PORTE (*)
 - LL_SYSCFG_EXTI_PORTF

(*) value not defined in all devices

Reference Manual to LL API cross reference:

- SYSCFG_EXTICR1 EXTI0 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR1 EXTI1 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR1 EXTI2 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR1 EXTI3 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR2 EXTI4 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR2 EXTI5 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR2 EXTI6 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR2 EXTI7 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR3 EXTI8 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR3 EXTI9 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR3 EXTI10 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR3 EXTI11 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR4 EXTI12 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR4 EXTI13 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR4 EXTI14 LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR4 EXTI15 LL_SYSCFG_SetEXTISource

LL_SYSCFG_IsActiveFlag_WWDG

Function name

_STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_WWDG (void)

Function description	Check if Window watchdog interrupt occurred or not.
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SYSCFG_ITLINE0 SR_EWDG LL_SYSCFG_IsActiveFlag_WWDG
LL_SYSCFG_IsActiveFlag_PVDOUT	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_PVDOUT (void)</code>
Function description	Check if PVD supply monitoring interrupt occurred or not (EXTI line 16).
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SYSCFG_ITLINE1 SR_PVDOUT LL_SYSCFG_IsActiveFlag_PVDOUT
LL_SYSCFG_IsActiveFlag_VDDIO2	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_VDDIO2 (void)</code>
Function description	Check if VDDIO2 supply monitoring interrupt occurred or not (EXTI line 31).
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SYSCFG_ITLINE1 SR_VDDIO2 LL_SYSCFG_IsActiveFlag_VDDIO2
LL_SYSCFG_IsActiveFlag_RTC_WAKEUP	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_RTC_WAKEUP (void)</code>
Function description	Check if RTC Wake Up interrupt occurred or not (EXTI line 20).
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SYSCFG_ITLINE2 SR_RTC_WAKEUP LL_SYSCFG_IsActiveFlag_RTC_WAKEUP
LL_SYSCFG_IsActiveFlag_RTC_TSTAMP	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_RTC_TSTAMP (void)</code>
Function description	Check if RTC Tamper and TimeStamp interrupt occurred or not (EXTI line 19).
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SYSCFG_ITLINE2 SR_RTC_TSTAMP LL_SYSCFG_IsActiveFlag_RTC_TSTAMP

`LL_SYSCFG_IsActiveFlag_RTC_ALRA`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_RTC_ALRA (void)`

Function description Check if RTC Alarm interrupt occurred or not (EXTI line 17).

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE2 SR_RTC_ALRA LL_SYSCFG_IsActiveFlag_RTC_ALRA

`LL_SYSCFG_IsActiveFlag_FLASH_ITF`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_FLASH_ITF (void)`

Function description Check if Flash interface interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE3 SR_FLASH_ITF LL_SYSCFG_IsActiveFlag_FLASH_ITF

`LL_SYSCFG_IsActiveFlag_CRS`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_CRS (void)`

Function description Check if Clock recovery system interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE4 SR_CRS LL_SYSCFG_IsActiveFlag_CRS

`LL_SYSCFG_IsActiveFlag_CLK_CTRL`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_CLK_CTRL (void)`

Function description Check if Reset and clock control interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE4 SR_CLK_CTRL LL_SYSCFG_IsActiveFlag_CLK_CTRL

`LL_SYSCFG_IsActiveFlag_EXTI0`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI0 (void)`

Function description Check if EXTI line 0 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE5 SR_EXTI0 LL_SYSCFG_IsActiveFlag_EXTI0

LL_SYSCFG_IsActiveFlag_EXTI1

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI1 (void)`

Function description Check if EXTI line 1 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE5 SR_EXTI1 LL_SYSCFG_IsActiveFlag_EXTI1

LL_SYSCFG_IsActiveFlag_EXTI2

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI2 (void)`

Function description Check if EXTI line 2 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE6 SR_EXTI2 LL_SYSCFG_IsActiveFlag_EXTI2

LL_SYSCFG_IsActiveFlag_EXTI3

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI3 (void)`

Function description Check if EXTI line 3 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE6 SR_EXTI3 LL_SYSCFG_IsActiveFlag_EXTI3

LL_SYSCFG_IsActiveFlag_EXTI4

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI4 (void)`

Function description Check if EXTI line 4 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE7 SR_EXTI4 LL_SYSCFG_IsActiveFlag_EXTI4

LL_SYSCFG_IsActiveFlag_EXTI5

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI5 (void)`

Function description Check if EXTI line 5 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI5 LL_SYSCFG_IsActiveFlag_EXTI5

LL_SYSCFG_IsActiveFlag_EXTI6

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI6 (void)

Function description

Check if EXTI line 6 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI6 LL_SYSCFG_IsActiveFlag_EXTI6

LL_SYSCFG_IsActiveFlag_EXTI7

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI7 (void)

Function description

Check if EXTI line 7 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI7 LL_SYSCFG_IsActiveFlag_EXTI7

LL_SYSCFG_IsActiveFlag_EXTI8

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI8 (void)

Function description

Check if EXTI line 8 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI8 LL_SYSCFG_IsActiveFlag_EXTI8

LL_SYSCFG_IsActiveFlag_EXTI9

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI9 (void)

Function description

Check if EXTI line 9 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI9 LL_SYSCFG_IsActiveFlag_EXTI9

LL_SYSCFG_IsActiveFlag_EXTI10

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI10 (void)

Function description Check if EXTI line 10 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI10 LL_SYSCFG_IsActiveFlag_EXTI10

LL_SYSCFG_IsActiveFlag_EXTI11

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI11 (void)`

Function description Check if EXTI line 11 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI11 LL_SYSCFG_IsActiveFlag_EXTI11

LL_SYSCFG_IsActiveFlag_EXTI12

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI12 (void)`

Function description Check if EXTI line 12 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI12 LL_SYSCFG_IsActiveFlag_EXTI12

LL_SYSCFG_IsActiveFlag_EXTI13

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI13 (void)`

Function description Check if EXTI line 13 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI13 LL_SYSCFG_IsActiveFlag_EXTI13

LL_SYSCFG_IsActiveFlag_EXTI14

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI14 (void)`

Function description Check if EXTI line 14 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI14 LL_SYSCFG_IsActiveFlag_EXTI14

`LL_SYSCFG_IsActiveFlag_EXTI15`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI15 (void)`

Function description Check if EXTI line 15 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI15 LL_SYSCFG_IsActiveFlag_EXTI15

`LL_SYSCFG_IsActiveFlag_TSC_EOA`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TSC_EOA (void)`

Function description Check if Touch sensing controller end of acquisition interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE8 SR_TSC_EOA LL_SYSCFG_IsActiveFlag_TSC_EOA

`LL_SYSCFG_IsActiveFlag_TSC_MCE`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TSC_MCE (void)`

Function description Check if Touch sensing controller max countererror interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE8 SR_TSC_MCE LL_SYSCFG_IsActiveFlag_TSC_MCE

`LL_SYSCFG_IsActiveFlag_DMA1_CH1`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH1 (void)`

Function description Check if DMA1 channel 1 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE9 SR_DMA1_CH1 LL_SYSCFG_IsActiveFlag_DMA1_CH1

`LL_SYSCFG_IsActiveFlag_DMA1_CH2`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH2 (void)`

Function description Check if DMA1 channel 2 interrupt occurred or not.

Return values

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

**Reference Manual to LL
API cross reference:**

- SYSCFG_ITLINE10 SR_DMA1_CH2 LL_SYSCFG_IsActiveFlag_DMA1_CH2

LL_SYSCFG_IsActiveFlag_DMA1_CH3

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH3 (void)`

Function description Check if DMA1 channel 3 interrupt occurred or not.

Return values

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

**Reference Manual to LL
API cross reference:**

LL_SYSCFG_IsActiveFlag_DMA2_CH1

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA2_CH1 (void)`

Function description Check if DMA2 channel 1 interrupt occurred or not.

Return values

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

**Reference Manual to LL
API cross reference:**

LL_SYSCFG_IsActiveFlag_DMA2_CH2

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA2_CH2 (void)`

Function description Check if DMA2 channel 2 interrupt occurred or not.

Return values

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

**Reference Manual to LL
API cross reference:**

LL_SYSCFG_IsActiveFlag_DMA1_CH4

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH4 (void)`

Function description Check if DMA1 channel 4 interrupt occurred or not.

Return values

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

**Reference Manual to LL
API cross reference:**

LL_SYSCFG_IsActiveFlag_DMA1_CH5

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH5 (void)`

Function description Check if DMA1 channel 5 interrupt occurred or not.

- Return values**
- **State:** of bit (1 or 0).
- Reference Manual to LL API cross reference:**
- SYSCFG_ITLINE11 SR_DMA1_CH5 LL_SYSCFG_IsActiveFlag_DMA1_CH5

LL_SYSCFG_IsActiveFlag_DMA1_CH6

- Function name**
- `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH6 (void)`
- Function description**
- Check if DMA1 channel 6 interrupt occurred or not.
- Return values**
- **State:** of bit (1 or 0).
- Reference Manual to LL API cross reference:**
- SYSCFG_ITLINE11 SR_DMA1_CH6 LL_SYSCFG_IsActiveFlag_DMA1_CH6

LL_SYSCFG_IsActiveFlag_DMA1_CH7

- Function name**
- `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH7 (void)`
- Function description**
- Check if DMA1 channel 7 interrupt occurred or not.
- Return values**
- **State:** of bit (1 or 0).
- Reference Manual to LL API cross reference:**
- SYSCFG_ITLINE11 SR_DMA1_CH7 LL_SYSCFG_IsActiveFlag_DMA1_CH7

LL_SYSCFG_IsActiveFlag_DMA2_CH3

- Function name**
- `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA2_CH3 (void)`
- Function description**
- Check if DMA2 channel 3 interrupt occurred or not.
- Return values**
- **State:** of bit (1 or 0).
- Reference Manual to LL API cross reference:**
- SYSCFG_ITLINE11 SR_DMA2_CH3 LL_SYSCFG_IsActiveFlag_DMA2_CH3

LL_SYSCFG_IsActiveFlag_DMA2_CH4

- Function name**
- `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA2_CH4 (void)`
- Function description**
- Check if DMA2 channel 4 interrupt occurred or not.
- Return values**
- **State:** of bit (1 or 0).
- Reference Manual to LL API cross reference:**
- SYSCFG_ITLINE11 SR_DMA2_CH4 LL_SYSCFG_IsActiveFlag_DMA2_CH4

LL_SYSCFG_IsActiveFlag_DMA2_CH5

- Function name**
- `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA2_CH5 (void)`

Function description	Check if DMA2 channel 5 interrupt occurred or not.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE11 SR_DMA2_CH5 LL_SYSCFG_IsActiveFlag_DMA2_CH5
LL_SYSCFG_IsActiveFlag_ADC	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_ADC (void)</code>
Function description	Check if ADC interrupt occurred or not.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE12 SR_ADC LL_SYSCFG_IsActiveFlag_ADC
LL_SYSCFG_IsActiveFlag_COMP1	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_COMP1 (void)</code>
Function description	Check if Comparator 1 interrupt occurred or not (EXTI line 21).
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE12 SR_COMP1 LL_SYSCFG_IsActiveFlag_COMP1
LL_SYSCFG_IsActiveFlag_COMP2	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_COMP2 (void)</code>
Function description	Check if Comparator 2 interrupt occurred or not (EXTI line 22).
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE12 SR_COMP2 LL_SYSCFG_IsActiveFlag_COMP2
LL_SYSCFG_IsActiveFlag_TIM1_BRK	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_BRK (void)</code>
Function description	Check if Timer 1 break interrupt occurred or not.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE13 SR_TIM1_BRK LL_SYSCFG_IsActiveFlag_TIM1_BRK

`LL_SYSCFG_IsActiveFlag_TIM1_UPD`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_UPD (void)`

Function description Check if Timer 1 update interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE13 SR_TIM1_UPD LL_SYSCFG_IsActiveFlag_TIM1_UPD

`LL_SYSCFG_IsActiveFlag_TIM1_TRG`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_TRG (void)`

Function description Check if Timer 1 trigger interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE13 SR_TIM1_TRG LL_SYSCFG_IsActiveFlag_TIM1_TRG

`LL_SYSCFG_IsActiveFlag_TIM1_CCU`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_CCU (void)`

Function description Check if Timer 1 commutation interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE13 SR_TIM1_CCU LL_SYSCFG_IsActiveFlag_TIM1_CCU

`LL_SYSCFG_IsActiveFlag_TIM1_CC`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_CC (void)`

Function description Check if Timer 1 capture compare interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE14 SR_TIM1_CC LL_SYSCFG_IsActiveFlag_TIM1_CC

`LL_SYSCFG_IsActiveFlag_TIM2`

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM2 (void)`

Function description Check if Timer 2 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE15 SR_TIM2_GLB LL_SYSCFG_IsActiveFlag_TIM2

LL_SYSCFG_IsActiveFlag_TIM3

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM3 (void)`

Function description Check if Timer 3 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE16 SR_TIM3_GLB LL_SYSCFG_IsActiveFlag_TIM3

LL_SYSCFG_IsActiveFlag_DAC

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DAC (void)`

Function description Check if DAC underrun interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE17 SR_DAC LL_SYSCFG_IsActiveFlag_DAC

LL_SYSCFG_IsActiveFlag_TIM6

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM6 (void)`

Function description Check if Timer 6 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE17 SR_TIM6_GLB LL_SYSCFG_IsActiveFlag_TIM6

LL_SYSCFG_IsActiveFlag_TIM7

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM7 (void)`

Function description Check if Timer 7 interrupt occurred or not.

Return values

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

[Reference Manual to LL API cross reference:](#)

- SYSCFG_ITLINE18 SR_TIM7_GLB LL_SYSCFG_IsActiveFlag_TIM7

LL_SYSCFG_IsActiveFlag_TIM14

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM14 (void)`

Function description Check if Timer 14 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE19 SR_TIM14_GLB LL_SYSCFG_IsActiveFlag_TIM14

LL_SYSCFG_IsActiveFlag_TIM15

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM15 (void)

Function description

Check if Timer 15 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE20 SR_TIM15_GLB LL_SYSCFG_IsActiveFlag_TIM15

LL_SYSCFG_IsActiveFlag_TIM16

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM16 (void)

Function description

Check if Timer 16 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE21 SR_TIM16_GLB LL_SYSCFG_IsActiveFlag_TIM16

LL_SYSCFG_IsActiveFlag_TIM17

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM17 (void)

Function description

Check if Timer 17 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE22 SR_TIM17_GLB LL_SYSCFG_IsActiveFlag_TIM17

LL_SYSCFG_IsActiveFlag_I2C1

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_I2C1 (void)

Function description

Check if I2C1 interrupt occurred or not, combined with EXTI line 23.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE23 SR_I2C1_GLB LL_SYSCFG_IsActiveFlag_I2C1

LL_SYSCFG_IsActiveFlag_I2C2

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_I2C2 (void)

Function description	Check if I2C2 interrupt occurred or not.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE24 SR_I2C2_GLB LL_SYSCFG_IsActiveFlag_I2C2
LL_SYSCFG_IsActiveFlag_SPI1	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_SPI1 (void)</code>
Function description	Check if SPI1 interrupt occurred or not.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE25 SR_SPI1 LL_SYSCFG_IsActiveFlag_SPI1
LL_SYSCFG_IsActiveFlag_SPI2	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_SPI2 (void)</code>
Function description	Check if SPI2 interrupt occurred or not.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE26 SR_SPI2 LL_SYSCFG_IsActiveFlag_SPI2
LL_SYSCFG_IsActiveFlag_USART1	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART1 (void)</code>
Function description	Check if USART1 interrupt occurred or not, combined with EXTI line 25.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE27 SR_USART1_GLB LL_SYSCFG_IsActiveFlag_USART1
LL_SYSCFG_IsActiveFlag_USART2	
Function name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART2 (void)</code>
Function description	Check if USART2 interrupt occurred or not, combined with EXTI line 26.
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SYSCFG_ITLINE28 SR_USART2_GLB LL_SYSCFG_IsActiveFlag_USART2

LL_SYSCFG_IsActiveFlag_USART3

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART3 (void)`

Function description Check if USART3 interrupt occurred or not, combined with EXTI line 28.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART3_GLB LL_SYSCFG_IsActiveFlag_USART3

LL_SYSCFG_IsActiveFlag_USART4

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART4 (void)`

Function description Check if USART4 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART4_GLB LL_SYSCFG_IsActiveFlag_USART4

LL_SYSCFG_IsActiveFlag_USART5

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART5 (void)`

Function description Check if USART5 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART5_GLB LL_SYSCFG_IsActiveFlag_USART5

LL_SYSCFG_IsActiveFlag_USART6

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART6 (void)`

Function description Check if USART6 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART6_GLB LL_SYSCFG_IsActiveFlag_USART6

LL_SYSCFG_IsActiveFlag_USART7

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART7 (void)`

Function description Check if USART7 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART7_GLB LL_SYSCFG_IsActiveFlag_USART7

LL_SYSCFG_IsActiveFlag_USART8

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART8 (void)`

Function description Check if USART8 interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART8_GLB LL_SYSCFG_IsActiveFlag_USART8

LL_SYSCFG_IsActiveFlag_CAN

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_CAN (void)`

Function description Check if CAN interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE30 SR_CAN LL_SYSCFG_IsActiveFlag_CAN

LL_SYSCFG_IsActiveFlag_CEC

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_CEC (void)`

Function description Check if CEC interrupt occurred or not, combined with EXTI line 27.

Return values

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

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE30 SR_CEC LL_SYSCFG_IsActiveFlag_CEC

LL_SYSCFG_SetTIMBreakInputs

Function name `__STATIC_INLINE void LL_SYSCFG_SetTIMBreakInputs (uint32_t Break)`

Function description Set connections to TIMx Break inputs.

Parameters

- Break:** This parameter can be a combination of the following values:

- `LL_SYSCFG_TIMBREAK_PVD` (*)
- `LL_SYSCFG_TIMBREAK_SRAM_PARITY`
- `LL_SYSCFG_TIMBREAK_LOCKUP`

(*) value not defined in all devices

Return values

- None:**

- Reference Manual to LL**
API cross reference:
- SYSCFG_CFGR2 LOCKUP_LOCK LL_SYSCFG_SetTIMBreakInputs
 - SYSCFG_CFGR2 SRAM_PARITY_LOCK LL_SYSCFG_SetTIMBreakInputs
 - SYSCFG_CFGR2 PVD_LOCK LL_SYSCFG_SetTIMBreakInputs

LL_SYSCFG_GetTIMBreakInputs

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_GetTIMBreakInputs (void)`

Function description Get connections to TIMx Break inputs.

- Return values**
- **Returned:** value can be can be a combination of the following values:
 - LL_SYSCFG_TIMBREAK_PVD (*)
 - LL_SYSCFG_TIMBREAK_SRAM_PARITY
 - LL_SYSCFG_TIMBREAK_LOCKUP(*) value not defined in all devices

- Reference Manual to LL**
API cross reference:
- SYSCFG_CFGR2 LOCKUP_LOCK LL_SYSCFG_GetTIMBreakInputs
 - SYSCFG_CFGR2 SRAM_PARITY_LOCK LL_SYSCFG_GetTIMBreakInputs
 - SYSCFG_CFGR2 PVD_LOCK LL_SYSCFG_GetTIMBreakInputs

LL_SYSCFG_IsActiveFlag_SP

Function name `__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_SP (void)`

Function description Check if SRAM parity error detected.

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

- Reference Manual to LL**
API cross reference:
- SYSCFG_CFGR2 SRAM_PEF LL_SYSCFG_IsActiveFlag_SP

LL_SYSCFG_ClearFlag_SP

Function name `__STATIC_INLINE void LL_SYSCFG_ClearFlag_SP (void)`

Function description Clear SRAM parity error flag.

- Return values**
- **None:**

- Reference Manual to LL**
API cross reference:
- SYSCFG_CFGR2 SRAM_PEF LL_SYSCFG_ClearFlag_SP

LL_DBGMCU_GetDeviceID

Function name `__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void)`

Function description Return the device identifier.

- Return values**
- **Values:** between Min_Data=0x00 and Max_Data=0xFFFF

Notes

- For STM32F03x devices, the device ID is 0x444
- For STM32F04x devices, the device ID is 0x445.
- For STM32F05x devices, the device ID is 0x440
- For STM32F07x devices, the device ID is 0x448
- For STM32F09x devices, the device ID is 0x442

**Reference Manual to LL
API cross reference:**

- DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

LL_DBGMCU_GetRevisionID**Function name** `__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)`**Function description** Return the device revision identifier.**Return values** • **Values:** between Min_Data=0x00 and Max_Data=0xFFFF**Notes**

- This field indicates the revision of the device. For example, it is read as 0x1000 for Revision 1.0.

**Reference Manual to LL
API cross reference:**

- DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableDBGStopMode**Function name** `__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void)`**Function description** Enable the Debug Module during STOP mode.**Return values** • **None:****Reference Manual to LL
API cross reference:** • DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode**LL_DBGMCU_DisableDBGStopMode****Function name** `__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void)`**Function description** Disable the Debug Module during STOP mode.**Return values** • **None:****Reference Manual to LL
API cross reference:** • DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode**LL_DBGMCU_EnableDBGStandbyMode****Function name** `__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void)`**Function description** Enable the Debug Module during STANDBY mode.**Return values** • **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode

LL_DBGMCU_DisableDBGStandbyMode

Function name	<u>__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void)</u>
Function description	Disable the Debug Module during STANDBY mode.
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode
	<u>LL_DBGMCU_APB1_GRP1_FreezePeriph</u>
Function name	<u>__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periph)</u>
Function description	Freeze APB1 peripherals (group1 peripherals)
Parameters	<ul style="list-style-type: none">• Periph: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_DBGMCU_APB1_GRP1_TIM2_STOP (*)– LL_DBGMCU_APB1_GRP1_TIM3_STOP– LL_DBGMCU_APB1_GRP1_TIM6_STOP (*)– LL_DBGMCU_APB1_GRP1_TIM7_STOP (*)– LL_DBGMCU_APB1_GRP1_TIM14_STOP– LL_DBGMCU_APB1_GRP1_RTC_STOP– LL_DBGMCU_APB1_GRP1_WWDG_STOP– LL_DBGMCU_APB1_GRP1_IWDG_STOP– LL_DBGMCU_APB1_GRP1_I2C1_STOP– LL_DBGMCU_APB1_GRP1_CAN_STOP (*)(*) value not defined in all devices
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DBGMCU_APB1FZ DBG_TIM2_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_TIM3_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_TIM6_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_TIM7_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_TIM14_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_I2C1_SMBUS_TIMEOUT LL_DBGMCU_APB1_GRP1_FreezePeriph• DBGMCU_APB1FZ DBG_CAN_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
	<u>LL_DBGMCU_APB1_GRP1_UnFreezePeriph</u>
Function name	<u>__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periph)</u>

Function description	Unfreeze APB1 peripherals (group1 peripherals)
Parameters	<ul style="list-style-type: none">Periph: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_DBGMCU_APB1_GRP1_TIM2_STOP (*)– LL_DBGMCU_APB1_GRP1_TIM3_STOP– LL_DBGMCU_APB1_GRP1_TIM6_STOP (*)– LL_DBGMCU_APB1_GRP1_TIM7_STOP (*)– LL_DBGMCU_APB1_GRP1_TIM14_STOP– LL_DBGMCU_APB1_GRP1_RTC_STOP– LL_DBGMCU_APB1_GRP1_WWDG_STOP– LL_DBGMCU_APB1_GRP1_IWDG_STOP– LL_DBGMCU_APB1_GRP1_I2C1_STOP– LL_DBGMCU_APB1_GRP1_CAN_STOP (*) <p>(*) value not defined in all devices</p>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DBGMCU_APB1FZ DBG_TIM2_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_TIM3_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_TIM6_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_TIM7_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_TIM14_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_I2C1_SMBUS_TIMEOUT LL_DBGMCU_APB1_GRP1_UnFreezePeriph• DBGMCU_APB1FZ DBG_CAN_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

LL_DBGMCU_APB1_GRP2_FreezePeriph

Function name	<u>__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_FreezePeriph (uint32_t Periph)</u>
Function description	Freeze APB1 peripherals (group2 peripherals)
Parameters	<ul style="list-style-type: none">Periph: This parameter can be a combination of the following values:<ul style="list-style-type: none">– LL_DBGMCU_APB1_GRP2_TIM1_STOP– LL_DBGMCU_APB1_GRP2_TIM15_STOP (*)– LL_DBGMCU_APB1_GRP2_TIM16_STOP– LL_DBGMCU_APB1_GRP2_TIM17_STOP <p>(*) value not defined in all devices</p>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• DBGMCU_APB2FZ DBG_TIM1_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph• DBGMCU_APB2FZ DBG_TIM15_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph• DBGMCU_APB2FZ DBG_TIM16_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph• DBGMCU_APB2FZ DBG_TIM17_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph

LL_DBGMCU_APB1_GRP2_UnFreezePeriph

Function name	<code>__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_UnFreezePeriph (uint32_t Periph)</code>
Function description	Unfreeze APB1 peripherals (group2 peripherals)
Parameters	<ul style="list-style-type: none">Periph: This parameter can be a combination of the following values:<ul style="list-style-type: none">– <code>LL_DBGMCU_APB1_GRP2_TIM1_STOP</code>– <code>LL_DBGMCU_APB1_GRP2_TIM15_STOP (*)</code>– <code>LL_DBGMCU_APB1_GRP2_TIM16_STOP</code>– <code>LL_DBGMCU_APB1_GRP2_TIM17_STOP</code>
	(*) value not defined in all devices
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• <code>DBGMCU_APB2FZ DBG_TIM1_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph</code>• <code>DBGMCU_APB2FZ DBG_TIM15_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph</code>• <code>DBGMCU_APB2FZ DBG_TIM16_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph</code>• <code>DBGMCU_APB2FZ DBG_TIM17_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph</code>

LL_FLASH_SetLatency

Function name	<code>__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)</code>
Function description	Set FLASH Latency.
Parameters	<ul style="list-style-type: none">Latency: This parameter can be one of the following values:<ul style="list-style-type: none">– <code>LL_FLASH_LATENCY_0</code>– <code>LL_FLASH_LATENCY_1</code>
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• <code>FLASH_ACR LATENCY LL_FLASH_SetLatency</code>

LL_FLASH_GetLatency

Function name	<code>__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)</code>
Function description	Get FLASH Latency.
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– <code>LL_FLASH_LATENCY_0</code>– <code>LL_FLASH_LATENCY_1</code>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• <code>FLASH_ACR LATENCY LL_FLASH_GetLatency</code>

LL_FLASH_EnablePrefetch

Function name `__STATIC_INLINE void LL_FLASH_EnablePrefetch (void)`

Function description Enable Prefetch.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR PRFTBE LL_FLASH_EnablePrefetch

LL_FLASH_DisablePrefetch

Function name `__STATIC_INLINE void LL_FLASH_DisablePrefetch (void)`

Function description Disable Prefetch.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR PRFTBE LL_FLASH_DisablePrefetch

LL_FLASH_IsPrefetchEnabled

Function name `__STATIC_INLINE uint32_t LL_FLASH_IsPrefetchEnabled (void)`

Function description Check if Prefetch buffer is enabled.

Return values

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

Reference Manual to LL API cross reference:

- FLASH_ACR PRFTBS LL_FLASH_IsPrefetchEnabled

67.2 SYSTEM Firmware driver defines

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

67.2.1 SYSTEM

SYSTEM

GRP2_STOP_IP DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2 TIM1 counter stopped when core is halted
_TIM1_STOP

LL_DBGMCU_APB1_GRP2 TIM15 counter stopped when core is halted
_TIM15_STOP

LL_DBGMCU_APB1_GRP2 TIM16 counter stopped when core is halted
_TIM16_STOP

LL_DBGMCU_APB1_GRP2 TIM17 counter stopped when core is halted
_TIM17_STOP

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1 TIM2 counter stopped when core is halted
_TIM2_STOP

LL_DBGMCU_APB1_GRP1 TIM3 counter stopped when core is halted
_TIM3_STOP

LL_DBGMCU_APB1_GRP1 TIM6 counter stopped when core is halted
_TIM6_STOP

LL_DBGMCU_APB1_GRP1 TIM7 counter stopped when core is halted
_TIM7_STOP

LL_DBGMCU_APB1_GRP1 TIM14 counter stopped when core is halted
_TIM14_STOP

LL_DBGMCU_APB1_GRP1 RTC Calendar frozen when core is halted
_RTC_STOP

LL_DBGMCU_APB1_GRP1 Debug Window Watchdog stopped when Core is halted
_WWDG_STOP

LL_DBGMCU_APB1_GRP1 Debug Independent Watchdog stopped when Core is halted
_IWDG_STOP

LL_DBGMCU_APB1_GRP1 I2C1 SMBUS timeout mode stopped when Core is halted
_I2C1_STOP

LL_DBGMCU_APB1_GRP1 CAN debug stopped when Core is halted
_CAN_STOP

SYSCFG EXTI LINE

LL_SYSCFG_EXTI_LINE0 EXTI_POSITION_0 | EXTICR[0]

LL_SYSCFG_EXTI_LINE1 EXTI_POSITION_4 | EXTICR[0]

LL_SYSCFG_EXTI_LINE2 EXTI_POSITION_8 | EXTICR[0]

LL_SYSCFG_EXTI_LINE3 EXTI_POSITION_12 | EXTICR[0]

LL_SYSCFG_EXTI_LINE4 EXTI_POSITION_0 | EXTICR[1]

LL_SYSCFG_EXTI_LINE5 EXTI_POSITION_4 | EXTICR[1]

LL_SYSCFG_EXTI_LINE6 EXTI_POSITION_8 | EXTICR[1]

LL_SYSCFG_EXTI_LINE7 EXTI_POSITION_12 | EXTICR[1]

LL_SYSCFG_EXTI_LINE8 EXTI_POSITION_0 | EXTICR[2]

LL_SYSCFG_EXTI_LINE9 EXTI_POSITION_4 | EXTICR[2]

LL_SYSCFG_EXTI_LINE10 EXTI_POSITION_8 | EXTICR[2]

LL_SYSCFG_EXTI_LINE11 EXTI_POSITION_12 | EXTICR[2]

LL_SYSCFG_EXTI_LINE12 EXTI_POSITION_0 | EXTICR[3]

LL_SYSCFG_EXTI_LINE13 EXTI_POSITION_4 | EXTICR[3]

LL_SYSCFG_EXTI_LINE14 EXTI_POSITION_8 | EXTICR[3]

LL_SYSCFG_EXTI_LINE15 EXTI_POSITION_12 | EXTICR[3]

SYSCFG EXTI PORT

LL_SYSCFG_EXTI_PORTA EXTI PORT A

LL_SYSCFG_EXTI_PORTB EXTI PORT B

LL_SYSCFG_EXTI_PORTC EXTI PORT C

LL_SYSCFG_EXTI_PORTD EXTI PORT D

LL_SYSCFG_EXTI_PORTE EXTI PORT E

LL_SYSCFG_EXTI_PORTF EXTI PORT F

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMO I2C PB6 Fast mode plus
DEPLUS_PB6

LL_SYSCFG_I2C_FASTMO I2C PB7 Fast mode plus
DEPLUS_PB7

LL_SYSCFG_I2C_FASTMO I2C PB8 Fast mode plus
DEPLUS_PB8

LL_SYSCFG_I2C_FASTMO I2C PB9 Fast mode plus
DEPLUS_PB9

LL_SYSCFG_I2C_FASTMO Enable Fast Mode Plus on PB10, PB11, PF6 and PF7
DEPLUS_I2C1

LL_SYSCFG_I2C_FASTMO Enable I2C2 Fast mode plus
DEPLUS_I2C2

LL_SYSCFG_I2C_FASTMO Enable Fast Mode Plus on PA9
DEPLUS_PA9

LL_SYSCFG_I2C_FASTMO Enable Fast Mode Plus on PA10
DEPLUS_PA10

SYSCFG IR Modulation

LL_SYSCFG_IR_MOD_TIM16 Timer16 is selected as IR Modulation enveloppe source

RT1

LL_SYSCFG_IR_MOD_USART1 USART1 is selected as IR Modulation enveloppe source

RT4

LL_SYSCFG_IR_MOD_USART4 USART4 is selected as IR Modulation enveloppe source

RT4

FLASH LATENCY

LL_FLASH_LATENCY_0 FLASH Zero Latency cycle

LL_FLASH_LATENCY_1 FLASH One Latency cycle

SYSCFG Remap

LL_SYSCFG_REMAP_FLA Main Flash memory mapped at 0x00000000

SH

LL_SYSCFG_REMAP_SYS System Flash memory mapped at 0x00000000

TEMFLASH

LL_SYSCFG_REMAP_SRA Embedded SRAM mapped at 0x00000000

M

SYSCFG TIMER BREAK

LL_SYSCFG_TIMBREAK_P Enables and locks the PVD connection with TIM1/15/16U/17 Break Input and also the PVDE

VD and PLS bits of the Power Control Interface

LL_SYSCFG_TIMBREAK_S Enables and locks the SRAM_PARITY error signal with Break Input of TIM1/15/16/17

RAM_PARITY

LL_SYSCFG_TIMBREAK_L Enables and locks the LOCKUP (Hardfault) output of CortexM0 with Break Input of

LOCKUP TIM1/15/16/17

68 LL TIM Generic Driver

68.1 TIM Firmware driver registers structures

68.1.1 LL_TIM_InitTypeDef

`LL_TIM_InitTypeDef` is defined in the `stm32f0xx_ll_tim.h`

Data Fields

- `uint16_t Prescaler`
- `uint32_t CounterMode`
- `uint32_t Autoreload`
- `uint32_t ClockDivision`
- `uint8_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()`.

- `uint8_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 This parameter must be a number between 0x00 and 0xFF.

This feature can be modified afterwards using unitary function `LL_TIM_SetRepetitionCounter()`.

68.1.2 LL_TIM_OC_InitTypeDef

`LL_TIM_OC_InitTypeDef` is defined in the `stm32f0xx_ll_tim.h`

Data Fields

- `uint32_t OCMode`
- `uint32_t OCState`
- `uint32_t OCNState`
- `uint32_t CompareValue`
- `uint32_t OCIdleState`
- `uint32_t OCNPolarity`
- `uint32_t OCIdleState`
- `uint32_t OCIdleState`

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()`.

68.1.3 `LL_TIM_IC_InitTypeDef`

`LL_TIM_IC_InitTypeDef` is defined in the `stm32f0xx_ll_tim.h`

Data Fields

- **`uint32_t IC_Polarity`**
- **`uint32_t IC_ActiveInput`**
- **`uint32_t IC_Prescaler`**
- **`uint32_t IC_Filter`**

Field Documentation

- **`uint32_t LL_TIM_IC_InitTypeDef::IC_Polarity`**
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::IC_ActiveInput`**
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::IC_Prescaler`**
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::IC_Filter`**
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()`.

68.1.4 LL_TIM_ENCODER_InitTypeDef

`LL_TIM_ENCODER_InitTypeDef` is defined in the `stm32f0xx_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_IC_PSC`. 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_IC_PSC`. 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()`.

68.1.5 LL_TIM_HALLSENSOR_InitTypeDef

`LL_TIM_HALLSENSOR_InitTypeDef` is defined in the `stm32f0xx_ll_tim.h`

Data Fields

- `uint32_t IC1Polarity`
- `uint32_t IC1Prescaler`
- `uint32_t IC1Filter`
- `uint32_t CommutationDelay`

Field Documentation

- **`uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity`**
Specifies the active edge of TI1 input. This parameter can be a value of `TIM_LL_EC_IC_POLARITY`.This feature can be modified afterwards using unitary function `LL_TIM_IC_SetPolarity()`.
- **`uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler`**
Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of `TIM_LL_EC_ICPSC`.This feature can be modified afterwards using unitary function `LL_TIM_IC_SetPrescaler()`.
- **`uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter`**
Specifies the TI1 input filter. This parameter can be a value of `TIM_LL_EC_IC_FILTER`.This feature can be modified afterwards using unitary function `LL_TIM_IC_SetFilter()`.
- **`uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay`**
Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function `LL_TIM_OC_SetCompareCH2()`.

68.1.6 LL_TIM_BDTR_InitTypeDef

`LL_TIM_BDTR_InitTypeDef` is defined in the `stm32f0xx_ll_tim.h`

Data Fields

- `uint32_t OSSRState`
- `uint32_t OSSISState`
- `uint32_t LockLevel`
- `uint8_t DeadTime`
- `uint16_t BreakState`
- `uint32_t BreakPolarity`
- `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::OSSISState`**
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::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.

68.2 TIM Firmware driver API description

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

68.2.1 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	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the timer counter is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name	<code>__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)</code>
Function description	Enable update event generation.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name	<code>__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)</code>
Function description	Disable update event generation.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether update event generation is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">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 (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 (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_REPEATIVE

[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.

[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 (TIM_TypeDef * TIMx)`

Function description Get actual counter mode.

Parameters

- **TIMx:** Timer instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR1 DIR LL_TIM_GetCounterMode• CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name	<code>__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)</code>
Function description	Enable auto-reload (ARR) preload.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name	<code>__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)</code>
Function description	Disable auto-reload (ARR) preload.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether auto-reload (ARR) preload is enabled.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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:

LL_TIM_GetClockDivision

Function name `__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (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:

LL_TIM_SetCounter

Function name `__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)`

Function description Set the counter value.

Parameters

- **TIMx:** Timer instance
- **Counter:** Counter value (between `Min_Data=0` and `Max_Data=0xFFFF or 0xFFFFFFFF`)

Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_GetCounter (TIM_TypeDef * TIMx)</code>
Function description	Get the counter value.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">Counter: value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)
Notes	<ul style="list-style-type: none">Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_GetDirection (TIM_TypeDef * TIMx)</code>
Function description	Get the current direction of the counter.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_TIM_COUNTERDIRECTION_UP– LL_TIM_COUNTERDIRECTION_DOWN
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name	<code>__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)</code>
Function description	Set the prescaler value.
Parameters	<ul style="list-style-type: none">TIMx: Timer instancePrescaler: between Min_Data=0 and Max_Data=65535
Return values	<ul style="list-style-type: none">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 (TIM_TypeDef * TIMx)`

Function description Get the prescaler value.

Parameters

- **TIMx:** Timer instance

Return values

- **Prescaler:** value between Min_Data=0 and Max_Data=65535

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name `__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)`

Function description Set the auto-reload value.

Parameters

- **TIMx:** Timer instance
- **AutoReload:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter is blocked while the auto-reload value is null.
- Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
- Helper macro `__LL_TIM_CALC_ARR` can be used to calculate the AutoReload parameter

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name `__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (TIM_TypeDef * TIMx)`

Function description Get the auto-reload value.

Parameters

- **TIMx:** Timer instance

Return values

- **Auto-reload:** value

- Notes**
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.

- Reference Manual to LL API cross reference:**
- ARR ARR LL_TIM_SetAutoReload

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

- Return values**
- None:**

- 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_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name `__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (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_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 (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	<ul style="list-style-type: none">• TIMx: Timer instance• Channels: This parameter can be a combination of the following values:<ul style="list-style-type: none">– 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
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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

LL_TIM_CC_DisableChannel

Function name	<code>_STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)</code>
Function description	Disable capture/compare channels.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channels: This parameter can be a combination of the following values:<ul style="list-style-type: none">– 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
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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

LL_TIM_CC_IsEnabledChannel

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (TIM_TypeDef * TIMx, uint32_t Channels)</code>
Function description	Indicate whether channel(s) is(are) enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceChannels: This parameter can be a combination of the following values:<ul style="list-style-type: none">– 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
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CCER CC1E LL_TIM_CC_IsEnabledChannelCCER CC1NE LL_TIM_CC_IsEnabledChannelCCER CC2E LL_TIM_CC_IsEnabledChannelCCER CC2NE LL_TIM_CC_IsEnabledChannelCCER CC3E LL_TIM_CC_IsEnabledChannelCCER CC3NE LL_TIM_CC_IsEnabledChannelCCER CC4E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name	<code>__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)</code>
Function description	Configure an output channel.
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceChannel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4Configuration: This parameter must be a combination of all the following values:<ul style="list-style-type: none">– LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW– LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH
Return values	<ul style="list-style-type: none">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
- CCER CC1P LL_TIM_OC_ConfigOutput
- CCER CC2P LL_TIM_OC_ConfigOutput
- CCER CC3P LL_TIM_OC_ConfigOutput
- CCER CC4P 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

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
- **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

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

LL_TIM_OC_GetMode

Function name `__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (TIM_TypeDef * TIMx, uint32_t Channel)`

Function description Get the output compare mode of an output channel.

- | | |
|--|---|
| Parameters | <ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4 |
| Return values | <ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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 |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none">• CCMR1 OC1M LL_TIM_OC_GetMode• CCMR1 OC2M LL_TIM_OC_GetMode• CCMR2 OC3M LL_TIM_OC_GetMode• CCMR2 OC4M LL_TIM_OC_GetMode |

LL_TIM_OC_SetPolarity

- | | |
|-----------------------------|---|
| Function name | <code>_STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)</code> |
| Function description | Set the polarity of an output channel. |
| Parameters | <ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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• Polarity: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_OCPOLARITY_HIGH– LL_TIM_OCPOLARITY_LOW |
| Return values | <ul style="list-style-type: none">• 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

LL_TIM_OC_GetPolarity

Function name `__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (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

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

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	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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• IdleState: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_OCIDLESTATE_LOW– LL_TIM_OCIDLESTATE_HIGH
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR2 OIS1 LL_TIM_OC_SetIdleState• CR2 OIS1N 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

LL_TIM_OC_GetIdleState

Function name	<code>_STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function description	Get the IDLE state of an output channel.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_TIM_OCIDLESTATE_LOW– LL_TIM_OCIDLESTATE_HIGH

**Reference Manual to LL
API cross reference:**

- CR2 OIS1 LL_TIM_OC_GetIdleState
- CR2 OIS1N 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

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

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

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

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

LL_TIM_OC_IsEnabledFast

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function description	Indicates whether fast mode is enabled for the output channel.
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceChannel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCMR1 OC1FE LL_TIM_OC_IsEnabledFast• CCMR1 OC2FE LL_TIM_OC_IsEnabledFast• CCMR2 OC3FE LL_TIM_OC_IsEnabledFast• CCMR2 OC4FE LL_TIM_OC_IsEnabledFast•

LL_TIM_OC_EnablePreload

Function name	<code>__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function description	Enable compare register (TIMx_CCRx) preload for the output channel.
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceChannel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCMR1 OC1PE LL_TIM_OC_EnablePreload• CCMR1 OC2PE LL_TIM_OC_EnablePreload• CCMR2 OC3PE LL_TIM_OC_EnablePreload• CCMR2 OC4PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name	<code>__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function description	Disable compare register (TIMx_CCRx) preload for the output channel.

Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCMR1 OC1PE LL_TIM_OC_DisablePreload• CCMR1 OC2PE LL_TIM_OC_DisablePreload• CCMR2 OC3PE LL_TIM_OC_DisablePreload• CCMR2 OC4PE LL_TIM_OC_DisablePreload
LL_TIM_OC_IsEnabledPreload	
Function name	<code>__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function description	Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload• CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload• CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload• CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload•
LL_TIM_OC_EnableClear	
Function name	<code>__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function description	Enable clearing the output channel on an external event.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4
Return values	<ul style="list-style-type: none">• 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

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

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

LL_TIM_OC_IsEnabledClear

Function name `__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (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

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
-

LL_TIM_OC_SetDeadTime**Function name**

`__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)`

Function description

Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge of the Ocx and OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter

Reference Manual to LL API cross reference:

- BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1**Function name**

`__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)`

Function description

Set compare value for output channel 1 (TIMx_CCR1).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.

- Reference Manual to LL API cross reference:**
- CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name `__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)`

Function description Set compare value for output channel 2 (TIMx_CCR2).

- Parameters**
- **TIMx:** Timer instance
 - **CompareValue:** between Min_Data=0 and Max_Data=65535

- Return values**
- **None:**

- Notes**
- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
 - Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
 - Macro `IS_TIM_CC2_INSTANCE(TIMx)` can be used to check whether or not output channel 2 is supported by a timer instance.

- Reference Manual to LL API cross reference:**
- CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name `__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)`

Function description Set compare value for output channel 3 (TIMx_CCR3).

- Parameters**
- **TIMx:** Timer instance
 - **CompareValue:** between Min_Data=0 and Max_Data=65535

- Return values**
- **None:**

- Notes**
- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
 - Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
 - Macro `IS_TIM_CC3_INSTANCE(TIMx)` can be used to check whether or not output channel is supported by a timer instance.

- Reference Manual to LL API cross reference:**
- CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name `__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)`

Function description Set compare value for output channel 4 (TIMx_CCR4).

Parameters	<ul style="list-style-type: none">TIMx: Timer instanceCompareValue: between Min_Data=0 and Max_Data=65535
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_GetCompareCH1

Function name	<u>__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (TIM_TypeDef * TIMx)</u>
Function description	Get compare value (TIMx_CCR1) set for output channel 1.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">CompareValue: (between Min_Data=0 and Max_Data=65535)
Notes	<ul style="list-style-type: none">In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name	<u>__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (TIM_TypeDef * TIMx)</u>
Function description	Get compare value (TIMx_CCR2) set for output channel 2.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">CompareValue: (between Min_Data=0 and Max_Data=65535)
Notes	<ul style="list-style-type: none">In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name `__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (TIM_TypeDef * TIMx)`

Function description Get compare value (TIMx_CCR3) set for output channel 3.

Parameters

- TIMx:** Timer instance

Return values

- CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro `IS_TIM_CC3_INSTANCE(TIMx)` can be used to check whether or not output channel 3 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name `__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (TIM_TypeDef * TIMx)`

Function description Get compare value (TIMx_CCR4) set for output channel 4.

Parameters

- TIMx:** Timer instance

Return values

- CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro `IS_TIM_CC4_INSTANCE(TIMx)` can be used to check whether or not output channel 4 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_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	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<code>_STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)</code>
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 (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	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– LL_TIM_ICPSC_DIV1– LL_TIM_ICPSC_DIV2– LL_TIM_ICPSC_DIV4– LL_TIM_ICPSC_DIV8
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function description	Get the current prescaler value acting on an input channel.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_TIM_ICPSC_DIV1– LL_TIM_ICPSC_DIV2– LL_TIM_ICPSC_DIV4– LL_TIM_ICPSC_DIV8
----------------------	---

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)</code>
Function description	Set the input filter duration.

Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• 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	<u>__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (TIM_TypeDef * TIMx, uint32_t Channel)</u>
Function description	Get the input filter duration.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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_SetPolarity
- CCMR1 IC2F LL_TIM_IC_SetPolarity
- CCMR2 IC3F LL_TIM_IC_SetPolarity
- CCMR2 IC4F LL_TIM_IC_SetPolarity

LL_TIM_IC_SetPolarity

Function name	<code>__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICpolarity)</code>
Function description	Set the input channel polarity.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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:<ul style="list-style-type: none">– LL_TIM_IC_POLARITY_RISING– LL_TIM_IC_POLARITY_FALLING– LL_TIM_IC_POLARITY_BOTHEDGE
Return values	<ul style="list-style-type: none">• 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 (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_DisableXORCombination

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 (TIM_TypeDef * TIMx)`

Function description Indicates whether the TIMx_CH1, CH2 and CH3 pins are connected 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:](#)

LL_TIM_IC_GetCaptureCH1

Function name `__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (TIM_TypeDef * TIMx)`

Function description Get captured value for input channel 1.

Parameters • **TIMx:** Timer instance

Return values • **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes • In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
• Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
• Macro `IS_TIM_CC1_INSTANCE(TIMx)` can be used to check whether or not input channel 1 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name `__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (TIM_TypeDef * TIMx)`

Function description Get captured value for input channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro `IS_TIM_CC2_INSTANCE(TIMx)` can be used to check whether or not input channel 2 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name `__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (TIM_TypeDef * TIMx)`

Function description Get captured value for input channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro `IS_TIM_32B_COUNTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro `IS_TIM_CC3_INSTANCE(TIMx)` can be used to check whether or not input channel 3 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name `__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (TIM_TypeDef * TIMx)`

Function description Get captured value for input channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock**Function name** [__STATIC_INLINE void LL_TIM_EnableExternalClock \(TIM_TypeDef * TIMx\)](#)**Function description** Enable external clock mode 2.**Parameters** • **TIMx:** Timer instance**Return values** • **None:****Notes**

- When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock**Function name** [__STATIC_INLINE void LL_TIM_DisableExternalClock \(TIM_TypeDef * TIMx\)](#)**Function description** Disable external clock mode 2.**Parameters** • **TIMx:** Timer instance**Return values** • **None:****Notes**

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock**Function name** [__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock \(TIM_TypeDef * TIMx\)](#)**Function description** Indicate whether external clock mode 2 is enabled.**Parameters** • **TIMx:** Timer instance

Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name	<code>__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)</code>
Function description	Set the clock source of the counter clock.
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceClockSource: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CLOCKSOURCE_INTERNAL– LL_TIM_CLOCKSOURCE_EXT_MODE1– LL_TIM_CLOCKSOURCE_EXT_MODE2
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">SMCR SMS LL_TIM_SetClockSourceSMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name	<code>__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)</code>
Function description	Set the encoder interface mode.
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceEncoderMode: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_ENCODERMODE_X2_TI1– LL_TIM_ENCODERMODE_X2_TI2– LL_TIM_ENCODERMODE_X4_TI12
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.

- Reference Manual to LL API cross reference:**
- SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name `__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)`

Function description Set the trigger output (TRGO) used for timer synchronization .

- Parameters**
- **TIMx:** Timer instance
 - **TimerSynchronization:** This parameter can be one of the following values:
 - LL_TIM_TRGO_RESET
 - LL_TIM_TRGO_ENABLE
 - LL_TIM_TRGO_UPDATE
 - LL_TIM_TRGO_CC1IF
 - LL_TIM_TRGO_OC1REF
 - LL_TIM_TRGO_OC2REF
 - LL_TIM_TRGO_OC3REF
 - LL_TIM_TRGO_OC4REF

- Return values**
- **None:**

- Notes**
- Macro `IS_TIM_MASTER_INSTANCE(TIMx)` can be used to check whether or not a timer instance can operate as a master timer.

- Reference Manual to LL API cross reference:**
- CR2 MMS `LL_TIM_SetTriggerOutput`

LL_TIM_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

- Return values**
- **None:**

- Notes**
- Macro `IS_TIM_SLAVE_INSTANCE(TIMx)` can be used to check whether or not a timer instance can operate as a slave timer.

- Reference Manual to LL API cross reference:**
- SMCR SMS `LL_TIM_SetSlaveMode`

LL_TIM_SetTriggerInput

Function name `__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)`

Function description Set the selects the trigger input to be used to synchronize the counter.

Parameters

- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
 - `LL_TIM_TS_ITR0`
 - `LL_TIM_TS_ITR1`
 - `LL_TIM_TS_ITR2`
 - `LL_TIM_TS_ITR3`
 - `LL_TIM_TS_TI1F_ED`
 - `LL_TIM_TS_TI1FP1`
 - `LL_TIM_TS_TI2FP2`
 - `LL_TIM_TS_ETRF`

Return values

- **None:**

Notes

- Macro `IS_TIM_SLAVE_INSTANCE(TIMx)` can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR TS `LL_TIM_SetTriggerInput`

LL_TIM_EnableMasterSlaveMode

Function name `__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)`

Function description Enable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro `IS_TIM_SLAVE_INSTANCE(TIMx)` can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM `LL_TIM_EnableMasterSlaveMode`

LL_TIM_DisableMasterSlaveMode

Function name `__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)`

Function description Disable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">SMCR MSM LL_TIM_DisableMasterSlaveMode
LL_TIM_IsEnabledMasterSlaveMode	
Function name	<u>__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (TIM_TypeDef * TIMx)</u>
Function description	Indicates whether the Master/Slave mode is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">SMCR MSM LL_TIM_IsEnabledMasterSlaveMode
LL_TIM_ConfigETR	
Function name	<u>__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)</u>
Function description	Configure the external trigger (ETR) input.

Parameters	<ul style="list-style-type: none">• TIMx: Timer instance• ETRPolarity: This parameter can be one of the following values:<ul style="list-style-type: none">- LL_TIM_ETR_POLARITY_NONINVERTED- LL_TIM_ETR_POLARITY_INVERTED• ETRPrescaler: This parameter can be one of the following values:<ul style="list-style-type: none">- 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:<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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)**

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

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 BKP LL_TIM_ConfigBRK

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	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• BDTR OSSI LL_TIM_SetOffStates• BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name	<u>__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)</u>
Function description	Enable automatic output (MOE can be set by software or automatically when a break input is active).
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name	<u>__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)</u>
Function description	Disable automatic output (MOE can be set only by software).
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name	<u>__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (TIM_TypeDef * TIMx)</u>
Function description	Indicate whether automatic output is enabled.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).

Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">BDTR AOE LL_TIM_IsEnabledAutomaticOutput
LL_TIM_EnableAllOutputs	
Function name	<code>__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)</code>
Function description	Enable the outputs (set the MOE bit in TIMx_BDTR register).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 eventMacro 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:	<ul style="list-style-type: none">BDTR MOE LL_TIM_EnableAllOutputs
LL_TIM_DisableAllOutputs	
Function name	<code>__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)</code>
Function description	Disable the outputs (reset the MOE bit in TIMx_BDTR register).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">BDTR MOE LL_TIM_DisableAllOutputs
LL_TIM_IsEnabledAllOutputs	
Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether outputs are enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">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_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	<ul style="list-style-type: none">• TIMx: Timer instance• DMABurstBaseAddress: This parameter can be one of the following values:<ul style="list-style-type: none">- 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• DMABurstLength: This parameter can be one of the following values:<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• DCR DBL LL_TIM_ConfigDMABurst• DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_SetRemap

Function name	<code>__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)</code>
Function description	Remap TIM inputs (input channel, internal/external triggers).
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceRemap: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_TIM14_TI1_RMP_GPIO– LL_TIM_TIM14_TI1_RMP_RTC_CLK– LL_TIM_TIM14_TI1_RMP_HSE– LL_TIM_TIM14_TI1_RMP_MCO
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">TIM14_OR TI1_RMP LL_TIM_SetRemap

LL_TIM_SetOCRefClearInputSource

Function name	<code>__STATIC_INLINE void LL_TIM_SetOCRefClearInputSource (TIM_TypeDef * TIMx, uint32_t OCRefClearInputSource)</code>
Function description	Set the OCREF clear input source.
Parameters	<ul style="list-style-type: none">TIMx: Timer instanceOCRefClearInputSource: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_OCREF_CLR_INT_OCREF_CLR– LL_TIM_OCREF_CLR_INT_ETR
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUTThis function can only be used in Output compare and PWM modes.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SMCR OCCS LL_TIM_SetOCRefClearInputSource

LL_TIM_ClearFlag_UPDATE

Function name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Clear the update interrupt flag (UIF).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance

Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR UIF LL_TIM_ClearFlag_UPDATE
LL_TIM_IsActiveFlag_UPDATE	
Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR UIF LL_TIM_IsActiveFlag_UPDATE
LL_TIM_ClearFlag_CC1	
Function name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)</code>
Function description	Clear the Capture/Compare 1 interrupt flag (CC1F).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR CC1IF LL_TIM_ClearFlag_CC1
LL_TIM_IsActiveFlag_CC1	
Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (TIM_TypeDef * TIMx)</code>
Function description	Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR CC1IF LL_TIM_IsActiveFlag_CC1
LL_TIM_ClearFlag_CC2	
Function name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)</code>
Function description	Clear the Capture/Compare 2 interrupt flag (CC2F).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance

Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR CC2IF LL_TIM_ClearFlag_CC2
LL_TIM_IsActiveFlag_CC2	
Function name	<u>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (TIM_TypeDef * TIMx)</u>
Function description	Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR CC2IF LL_TIM_IsActiveFlag_CC2
LL_TIM_ClearFlag_CC3	
Function name	<u>__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)</u>
Function description	Clear the Capture/Compare 3 interrupt flag (CC3F).
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR CC3IF LL_TIM_ClearFlag_CC3
LL_TIM_IsActiveFlag_CC3	
Function name	<u>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (TIM_TypeDef * TIMx)</u>
Function description	Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• SR CC3IF LL_TIM_IsActiveFlag_CC3
LL_TIM_ClearFlag_CC4	
Function name	<u>__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)</u>
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 (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 (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	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (TIM_TypeDef * TIMx)</code>
Function description	Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)</code>
Function description	Clear the break interrupt flag (BIF).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (TIM_TypeDef * TIMx)</code>
Function description	Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_CC1OVR

Function name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)</code>
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 (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 (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 (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 (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	<code>__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Enable update interrupt (UIE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name	<code>__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Disable update interrupt (UIE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the update interrupt (UIE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name	<code>__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)</code>
Function description	Enable capture/compare 1 interrupt (CC1IE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">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 (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	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">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	<code>__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)</code>
Function description	Enable capture/compare 3 interrupt (CC3IE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name	<code>__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)</code>
Function description	Disable capture/compare 3 interrupt (CC3IE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">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 (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 (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	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the trigger interrupt (TIE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name	<code>__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)</code>
Function description	Enable break interrupt (BIE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name	<code>__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)</code>
Function description	Disable break interrupt (BIE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the break interrupt (BIE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableDMAReq_UPDATE

Function name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Enable update DMA request (UDE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Disable update DMA request (UDE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the update DMA request (UDE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">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	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)</code>
Function description	Enable capture/compare 1 DMA request (CC1DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)</code>
Function description	Disable capture/compare 1 DMA request (CC1DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)</code>
Function description	Enable capture/compare 2 DMA request (CC2DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)</code>
Function description	Disable capture/compare 2 DMA request (CC2DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">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	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)</code>
Function description	Enable capture/compare 3 DMA request (CC3DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)</code>
Function description	Disable capture/compare 3 DMA request (CC3DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">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	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)</code>
Function description	Enable capture/compare 4 DMA request (CC4DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)</code>
Function description	Disable capture/compare 4 DMA request (CC4DE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)</code>
Function description	Enable commutation DMA request (COMDE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER COMDE LL_TIM_EnableDMAReq_COM

LL_TIM_DisableDMAReq_COM

Function name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)</code>
Function description	Disable commutation DMA request (COMDE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the commutation DMA request (COMDE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)</code>
Function description	Enable trigger interrupt (TDE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)</code>
Function description	Disable trigger interrupt (TDE).
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (TIM_TypeDef * TIMx)</code>
Function description	Indicates whether the trigger interrupt (TDE) is enabled.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)</code>
Function description	Generate an update event.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)</code>
Function description	Generate Capture/Compare 1 event.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)</code>
Function description	Generate Capture/Compare 2 event.
Parameters	<ul style="list-style-type: none">TIMx: Timer instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)</code>
Function description	Generate Capture/Compare 3 event.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)</code>
Function description	Generate Capture/Compare 4 event.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)</code>
Function description	Generate commutation event.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)</code>
Function description	Generate trigger event.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name	<code>__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)</code>
Function description	Generate break event.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_DeInit

Function name	<code>ErrorStatus LL_TIM_DeInit (TIM_TypeDef * TIMx)</code>
Function description	Set TIMx registers to their reset values.
Parameters	<ul style="list-style-type: none">• TIMx: Timer instance
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: TIMx registers are de-initialized– ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name	<code>void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)</code>
Function description	Set the fields of the time base unit configuration data structure to their default values.
Parameters	<ul style="list-style-type: none">• TIM_InitStruct: pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)
Return values	<ul style="list-style-type: none">• None:

LL_TIM_Init

Function name	<code>ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, LL_TIM_InitTypeDef * TIM_InitStruct)</code>
Function description	Configure the TIMx time base unit.
Parameters	<ul style="list-style-type: none">• TIMx: Timer Instance• TIM_InitStruct: pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: TIMx registers are de-initialized– ERROR: not applicable

LL_TIM_OC_StructInit

Function name	<code>void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)</code>
Function description	Set the fields of the TIMx output channel configuration data structure to their default values.
Parameters	<ul style="list-style-type: none">• TIM_OC_InitStruct: pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)
Return values	<ul style="list-style-type: none">• None:

LL_TIM_OC_Init

Function name	<code>ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)</code>
Function description	Configure the TIMx output channel.
Parameters	<ul style="list-style-type: none">• TIMx: Timer Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_TIM_CHANNEL_CH1– LL_TIM_CHANNEL_CH2– LL_TIM_CHANNEL_CH3– LL_TIM_CHANNEL_CH4• TIM_OC_InitStruct: pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: TIMx output channel is initialized– ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name	<code>void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)</code>
Function description	Set the fields of the TIMx input channel configuration data structure to their default values.
Parameters	<ul style="list-style-type: none">• TIM_ICInitStruct: pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)
Return values	<ul style="list-style-type: none">• None:

LL_TIM_IC_Init

Function name	<code>ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)</code>
Function description	Configure the TIMx input channel.

Parameters	<ul style="list-style-type: none">• TIMx: Timer Instance• Channel: This parameter can be one of the following values:<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: TIMx output channel is initialized– ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name	<code>void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)</code>
Function description	Fills each TIM_EncoderInitStruct field with its default value.
Parameters	<ul style="list-style-type: none">• TIM_EncoderInitStruct: pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)
Return values	<ul style="list-style-type: none">• None:

LL_TIM_ENCODER_Init

Function name	<code>ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)</code>
Function description	Configure the encoder interface of the timer instance.
Parameters	<ul style="list-style-type: none">• TIMx: Timer Instance• TIM_EncoderInitStruct: pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)
Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: TIMx registers are de-initialized– ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name	<code>void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)</code>
Function description	Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.
Parameters	<ul style="list-style-type: none">• TIM_HallSensorInitStruct: pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)
Return values	<ul style="list-style-type: none">• None:

LL_TIM_HALLSENSOR_Init

Function name	<code>ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)</code>
Function description	Configure the Hall sensor interface of the timer instance.
Parameters	<ul style="list-style-type: none">TIMx: Timer InstanceTIM_HallSensorInitStruct: pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)
Return values	<ul style="list-style-type: none">An: ErrorStatus enumeration value:<ul style="list-style-type: none">SUCCESS: TIMx registers are de-initializedERROR: not applicable
Notes	<ul style="list-style-type: none">TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channelTIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED.Channel 1 is configured as input, IC1 is mapped on TRC.Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed.Channel 2 is configured in output PWM 2 mode.Compare value stored in TIMx_CCR2 corresponds to the commutation delay.OC2REF is selected as trigger output on TRGO.LL_TIM_IC_POLARITY_BOTHEDGE must not be used for TI1 when it is used when TIMx operates in Hall sensor interface mode.

LL_TIM_BDTR_StructInit

Function name	<code>void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)</code>
Function description	Set the fields of the Break and Dead Time configuration data structure to their default values.
Parameters	<ul style="list-style-type: none">TIM_BDTRInitStruct: pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)
Return values	<ul style="list-style-type: none">None:

LL_TIM_BDTR_Init

Function name	<code>ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)</code>
Function description	Configure the Break and Dead Time feature of the timer instance.
Parameters	<ul style="list-style-type: none">TIMx: Timer InstanceTIM_BDTRInitStruct: pointer to a LL_TIM_BDTR_InitTypeDef structure(Break and Dead Time configuration data structure)

Return values	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: Break and Dead Time is initialized– ERROR: not applicable
Notes	<ul style="list-style-type: none">• As the bits AOE, BKP, BKE, OSSR, OSSI 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.

68.3 TIM Firmware driver defines

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

68.3.1 TIM

TIM

Active Input Selection

`LL_TIM_ACTIVEINPUT_DIR` ICx is mapped on TIx
ECTTI

`LL_TIM_ACTIVEINPUT_IND` ICx is mapped on Tly
IRECTTI

`LL_TIM_ACTIVEINPUT_TR` ICx is mapped on TRC
C

Automatic output enable

`LL_TIM_AUTOMATICOUTP` MOE can be set only by software
UT_DISABLE

`LL_TIM_AUTOMATICOUTP` MOE can be set by software or automatically at the next update event
UT_ENABLE

Break Enable

`LL_TIM_BREAK_DISABLE` Break function disabled

`LL_TIM_BREAK_ENABLE` Break function enabled

break polarity

`LL_TIM_BREAK_POLARIT` Break input BRK is active low
Y_LOW

`LL_TIM_BREAK_POLARIT` Break input BRK is active high
Y_HIGH

Capture Compare DMA Request

`LL_TIM_CCDMAREQUEST` CCx DMA request sent when CCx event occurs
_CC

LL_TIM_CCDMAREQUEST_UPDATE CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOUR_CE_COMG_ONLY Capture/compare control bits are updated by setting the COMG bit only

LL_TIM_CCUPDATESOUR_CE_COMG_AND_TRGI Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_TIM_CHANNEL_CH1 Timer input/output channel 1

LL_TIM_CHANNEL_CH1N Timer complementary output channel 1

LL_TIM_CHANNEL_CH2 Timer input/output channel 2

LL_TIM_CHANNEL_CH2N Timer complementary output channel 2

LL_TIM_CHANNEL_CH3 Timer input/output channel 3

LL_TIM_CHANNEL_CH3N Timer complementary output channel 3

LL_TIM_CHANNEL_CH4 Timer input/output channel 4

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_I The timer is clocked by the internal clock provided from the RCC INTERNAL

LL_TIM_CLOCKSOURCE_E Counter counts at each rising or falling edge on a selected input XT_MODE1

LL_TIM_CLOCKSOURCE_E Counter counts at each rising or falling edge on the external trigger input ETR XT_MODE2

Counter Direction

LL_TIM_COUNTERDIRECTI_ON_UP Timer counter counts up

LL_TIM_COUNTERDIRECTI Timer counter counts down
ON_DOWN

Counter Mode

LL_TIM_COUNTERMODE_UP Counter used as upcounter

LL_TIM_COUNTERMODE_DOWN Counter used as downcounter

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 down.

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 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_BASE_ADDR_CR1 TIMx_CR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_CR2 TIMx_CR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_SMCR TIMx_SMCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_DIER TIMx_DIER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_SR TIMx_SR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_EGR TIMx_EGR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_CCMR1 TIMx_CCMR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_CCMR2 TIMx_CCMR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_CCER TIMx_CCER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_CNT TIMx_CNT register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE_ADDR_PSC TIMx_PSC register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASE TIMx_ARR register is the DMA base address for DMA burst
ADDR_ARR

LL_TIM_DMABURST_BASE TIMx_RCR register is the DMA base address for DMA burst
ADDR_RCR

LL_TIM_DMABURST_BASE TIMx_CCR1 register is the DMA base address for DMA burst
ADDR_CCR1

LL_TIM_DMABURST_BASE TIMx_CCR2 register is the DMA base address for DMA burst
ADDR_CCR2

LL_TIM_DMABURST_BASE TIMx_CCR3 register is the DMA base address for DMA burst
ADDR_CCR3

LL_TIM_DMABURST_BASE TIMx_CCR4 register is the DMA base address for DMA burst
ADDR_CCR4

LL_TIM_DMABURST_BASE TIMx_BDTR register is the DMA base address for DMA burst
ADDR_BDTR

DMA Burst Length

LL_TIM_DMABURST LENG Transfer is done to 1 register starting from the DMA burst base address
TH_1TRANSFER

LL_TIM_DMABURST LENG Transfer is done to 2 registers starting from the DMA burst base address
TH_2TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 3 registers starting from the DMA burst base address
TH_3TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 4 registers starting from the DMA burst base address
TH_4TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 5 registers starting from the DMA burst base address
TH_5TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 6 registers starting from the DMA burst base address
TH_6TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 7 registers starting from the DMA burst base address
TH_7TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 1 registers starting from the DMA burst base address
TH_8TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 9 registers starting from the DMA burst base address
TH_9TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 10 registers starting from the DMA burst base address
TH_10TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 11 registers starting from the DMA burst base address
TH_11TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 12 registers starting from the DMA burst base address
TH_12TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 13 registers starting from the DMA burst base address
TH_13TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 14 registers starting from the DMA burst base address
TH_14TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 15 registers starting from the DMA burst base address
TH_15TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 16 registers starting from the DMA burst base address
TH_16TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 17 registers starting from the DMA burst base address
TH_17TRANSFERS

LL_TIM_DMABURST LENG Transfer is done to 18 registers starting from the DMA burst base address
TH_18TRANSFERS

Encoder Mode

LL_TIM_ENCODERMODE_ Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level
X2_TI1

LL_TIM_ENCODERMODE_ Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level
X2_TI2

LL_TIM_ENCODERMODE_ Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input I
X4_TI12

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV No filter, sampling is done at fDTS
1

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fCK_INT, N=2
1_N2

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fCK_INT, N=4
1_N4

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fCK_INT, N=8
1_N8

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/2, N=6
2_N6

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/2, N=8
2_N8

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/4, N=6
4_N6

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/4, N=8
4_N8

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/8, N=8
8_N6

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/16, N=5
8_N8

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/16, N=6
16_N5

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/16, N=8
16_N6

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/16, N=5
16_N8

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/32, N=5
32_N5

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/32, N=6
32_N6

LL_TIM_ETR_FILTER_FDIV fSAMPLING=fDTS/32, N=8
32_N8

External Trigger Polarity

LL_TIM_ETR_POLARITY_N ETR is non-inverted, active at high level or rising edge
ONINVERTED

LL_TIM_ETR_POLARITY_I ETR is inverted, active at low level or falling edge
NVERTED

External Trigger Prescaler

LL_TIM_ETR_PRESCALER ETR prescaler OFF
_DIV1

LL_TIM_ETR_PRESCALER ETR frequency is divided by 2
_DIV2

LL_TIM_ETR_PRESCALER ETR frequency is divided by 4
_DIV4

LL_TIM_ETR_PRESCALER ETR frequency is divided by 8
_DIV8

Get Flags Defines

<code>LL_TIM_SR_UIF</code>	Update interrupt flag
<code>LL_TIM_SR_CC1IF</code>	Capture/compare 1 interrupt flag
<code>LL_TIM_SR_CC2IF</code>	Capture/compare 2 interrupt flag
<code>LL_TIM_SR_CC3IF</code>	Capture/compare 3 interrupt flag
<code>LL_TIM_SR_CC4IF</code>	Capture/compare 4 interrupt flag
<code>LL_TIM_SR_COMIF</code>	COM interrupt flag
<code>LL_TIM_SR_TIF</code>	Trigger interrupt flag
<code>LL_TIM_SR_BIF</code>	Break interrupt flag
<code>LL_TIM_SR_CC1OF</code>	Capture/Compare 1 overcapture flag
<code>LL_TIM_SR_CC2OF</code>	Capture/Compare 2 overcapture flag
<code>LL_TIM_SR_CC3OF</code>	Capture/Compare 3 overcapture flag
<code>LL_TIM_SR_CC4OF</code>	Capture/Compare 4 overcapture flag

Input Configuration Prescaler

<code>LL_TIM_ICPSC_DIV1</code>	No prescaler, capture is done each time an edge is detected on the capture input
<code>LL_TIM_ICPSC_DIV2</code>	Capture is done once every 2 events
<code>LL_TIM_ICPSC_DIV4</code>	Capture is done once every 4 events
<code>LL_TIM_ICPSC_DIV8</code>	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_ fSAMPLING=fCK_INT, N=2`
`N2`

`LL_TIM_IC_FILTER_FDIV1_ fSAMPLING=fCK_INT, N=4`
`N4`

`LL_TIM_IC_FILTER_FDIV1_ fSAMPLING=fCK_INT, N=8`
`N8`

`LL_TIM_IC_FILTER_FDIV2_ fSAMPLING=fDTS/2, N=6`
`N6`

`LL_TIM_IC_FILTER_FDIV2_ fSAMPLING=fDTS/2, N=8`
`N8`

`LL_TIM_IC_FILTER_FDIV4_ fSAMPLING=fDTS/4, N=6`
`N6`

`LL_TIM_IC_FILTER_FDIV4_ fSAMPLING=fDTS/4, N=8`
`N8`

`LL_TIM_IC_FILTER_FDIV8_ fSAMPLING=fDTS/8, N=6`
`N6`

`LL_TIM_IC_FILTER_FDIV8_ fSAMPLING=fDTS/8, N=8`
`N8`

`LL_TIM_IC_FILTER_FDIV16 fSAMPLING=fDTS/16, N=5`
`_N5`

`LL_TIM_IC_FILTER_FDIV16 fSAMPLING=fDTS/16, N=6`
`_N6`

`LL_TIM_IC_FILTER_FDIV16 fSAMPLING=fDTS/16, N=8`
`_N8`

`LL_TIM_IC_FILTER_FDIV32 fSAMPLING=fDTS/32, N=5`
`_N5`

`LL_TIM_IC_FILTER_FDIV32 fSAMPLING=fDTS/32, N=6`
`_N6`

`LL_TIM_IC_FILTER_FDIV32 fSAMPLING=fDTS/32, N=8`
`_N8`

Input Configuration Polarity

`LL_TIM_IC_POLARITY_RIS` The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted
`ING`

`LL_TIM_IC_POLARITY_FAL` The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted
`LING`

`LL_TIM_IC_POLARITY_BO` The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted
`THEEDGE`

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_LO OCx=0 (after a dead-time if OC is implemented) when MOE=0
W

LL_TIM_OCIDLESTATE_HI OCx=1 (after a dead-time if OC is implemented) when MOE=0
GH

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_FORCE_D_INACTIVE OCyREF is forced low

LL_TIM_OCMODE_FORCE_D_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

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH OCxactive high

LL_TIM_OCPOLARITY_LO_W 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_DISABL_E OCx is not active

LL_TIM_OCSTATE_ENABL_E OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE Counter is not stopped at update event

LL_TIM_ONEPULSEMODE_REPEATIVE Counter stops counting at the next update event

OSSI

LL_TIM_OSSI_DISABLE When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSI_ENABLE When inactive, OCx/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, OCx/OCxN 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_GATE 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

Trigger Output

<code>LL_TIM_TRGO_RESET</code>	UG bit from the TIMx_EGR register is used as trigger output
<code>LL_TIM_TRGO_ENABLE</code>	Counter Enable signal (CNT_EN) is used as trigger output
<code>LL_TIM_TRGO_UPDATE</code>	Update event is used as trigger output
<code>LL_TIM_TRGO_CC1IF</code>	CC1 capture or a compare match is used as trigger output
<code>LL_TIM_TRGO_OC1REF</code>	OC1REF signal is used as trigger output
<code>LL_TIM_TRGO_OC2REF</code>	OC2REF signal is used as trigger output
<code>LL_TIM_TRGO_OC3REF</code>	OC3REF signal is used as trigger output
<code>LL_TIM_TRGO_OC4REF</code>	OC4REF signal is used as trigger output

Trigger Selection

<code>LL_TIM_TS_ITR0</code>	Internal Trigger 0 (ITR0) is used as trigger input
<code>LL_TIM_TS_ITR1</code>	Internal Trigger 1 (ITR1) is used as trigger input
<code>LL_TIM_TS_ITR2</code>	Internal Trigger 2 (ITR2) is used as trigger input
<code>LL_TIM_TS_ITR3</code>	Internal Trigger 3 (ITR3) is used as trigger input
<code>LL_TIM_TS_TI1F_ED</code>	TI1 Edge Detector (TI1F_ED) is used as trigger input
<code>LL_TIM_TS_TI1FP1</code>	Filtered Timer Input 1 (TI1FP1) is used as trigger input
<code>LL_TIM_TS_TI2FP2</code>	Filtered Timer Input 2 (TI12P2) is used as trigger input
<code>LL_TIM_TS_ETRF</code>	Filtered external Trigger (ETRF) is used as trigger input

Update Source

<code>LL_TIM_UPDATESOURCE_REGULAR</code>	Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request
<code>LL_TIM_UPDATESOURCE_COUNTER</code>	Only counter overflow/underflow generates an update request

Exported Macros

__LL_TIM_CALC_DEADTIM Description:

E

- 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_RA

TIO

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 ());

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 Constants

LL_TIM_TIM14_TI1_RMP_G TIM14_TI1 is connected to Ored GPIO
PIO

LL_TIM_TIM14_TI1_RMP_R TIM14_TI1 is connected to RTC clock
TC_CLK

LL_TIM_TIM14_TI1_RMP_H TIM14_TI1 is connected to HSE/32 clock
SE

LL_TIM_TIM14_TI1_RMP_M TIM14_TI1 is connected to MCO
CO

69 LL USART Generic Driver

69.1 USART Firmware driver registers structures

69.1.1 LL_USART_InitTypeDef

`LL_USART_InitTypeDef` is defined in the `stm32f0xx_ll_usart.h`

Data Fields

- `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::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()`.

69.1.2 LL_USART_ClockInitTypeDef

`LL_USART_ClockInitTypeDef` is defined in the `stm32f0xx_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.

69.2 USART Firmware driver API description

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

69.2.1 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 (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_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	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (USART_TypeDef * USARTx)</code>
Function description	Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name	<code>__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)</code>
Function description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name	<code>__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)</code>
Function description	Receiver Disable.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name	<code>__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)</code>
Function description	Transmitter Enable.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance

Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 TE LL_USART_EnableDirectionTx
LL_USART_DisableDirectionTx	
Function name	<u>__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)</u>
Function description	Transmitter Disable.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 TE LL_USART_DisableDirectionTx
LL_USART_SetTransferDirection	
Function name	<u>__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)</u>
Function description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance• TransferDirection: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_DIRECTION_NONE– LL_USART_DIRECTION_RX– LL_USART_DIRECTION_TX– LL_USART_DIRECTION_TX_RX
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 RE LL_USART_SetTransferDirection• CR1 TE LL_USART_SetTransferDirection
LL_USART_GetTransferDirection	
Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (USART_TypeDef * USARTx)</u>
Function description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– 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_SetParity
 - CR1 TE LL_USART_SetParity

LL_USART_SetParity

Function name	<code>__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)</code>
Function description	Configure Parity (enabled/disabled and parity mode if enabled).
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance• Parity: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_PARITY_NONE– LL_USART_PARITY_EVEN– LL_USART_PARITY_ODD
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR1 PS LL_USART_SetParity• CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetParity (USART_TypeDef * USARTx)</code>
Function description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_USART_PARITY_NONE– LL_USART_PARITY_EVEN– LL_USART_PARITY_ODD
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 PS LL_USART_SetParity• CR1 PCE LL_USART_SetParity

LL_USART_SetWakeUpMethod

Function name	<code>__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)</code>
Function description	Set Receiver Wake Up method from Mute mode.

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance• Method: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_WAKEUP_IDLELINE– LL_USART_WAKEUP_ADDRESSMARK
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 WAKE LL_USART_SetWakeUpMethod
LL_USART_GetWakeUpMethod	
Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (USART_TypeDef * USARTx)</u>
Function description	Return Receiver Wake Up method from Mute mode.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_USART_WAKEUP_IDLELINE– LL_USART_WAKEUP_ADDRESSMARK
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 WAKE LL_USART_GetWakeUpMethod
LL_USART_SetDataWidth	
Function name	<u>__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)</u>
Function description	Set Word length (i.e.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance• DataWidth: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_DATAWIDTH_7B (*)– LL_USART_DATAWIDTH_8B– LL_USART_DATAWIDTH_9B
(*) Values not available on all devices	
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 M0 LL_USART_SetDataWidth• CR1 M1 LL_USART_SetDataWidth
LL_USART_GetDataWidth	
Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetDataWidth (USART_TypeDef * USARTx)</u>
Function description	Return Word length (i.e.
Parameters	<ul style="list-style-type: none">• USARTTx: 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
- (*) Values not available on all devices

- 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 (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	<ul style="list-style-type: none">• USARTTx: USART Instance• OverSampling: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_OVERSAMPLING_16– LL_USART_OVERSAMPLING_8
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 OVER8 LL_USART_SetOverSampling
	LL_USART_GetOverSampling
Function name	<u>_STATIC_INLINE uint32_t LL_USART_GetOverSampling (USART_TypeDef * USARTTx)</u>
Function description	Return Oversampling mode.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_USART_OVERSAMPLING_16– LL_USART_OVERSAMPLING_8
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 OVER8 LL_USART_GetOverSampling
	LL_USART_SetLastClkPulseOutput
Function name	<u>_STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTTx, uint32_t LastBitClockPulse)</u>
Function description	Configure if Clock pulse of the last data bit is output to the SCLK pin or not.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance• LastBitClockPulse: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_LASTCLKPULSE_NO_OUTPUT– LL_USART_LASTCLKPULSE_OUTPUT
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_USART_INSTANCE(USARTTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 LBCL LL_USART_SetLastClkPulseOutput
	LL_USART_GetLastClkPulseOutput
Function name	<u>_STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (USART_TypeDef * USARTTx)</u>

Function description Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)

Parameters

- **USARTTx:** 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

- **USARTTx:** 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 (USART_TypeDef * USARTx)**

Function description Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTTx:** 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 (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	<ul style="list-style-type: none">• USARTTx: USART Instance• Phase: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_PHASE_1EDGE– LL_USART_PHASE_2EDGE• Polarity: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_POLARITY_LOW– LL_USART_POLARITY_HIGH• LBCPOutput: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_LASTCLKPULSE_NO_OUTPUT– LL_USART_LASTCLKPULSE_OUTPUT
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_USART_INSTANCE(USARTTx) can be used to check whether or not Synchronous mode is supported by the USARTTx 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:	<ul style="list-style-type: none">• CR2 CPHA LL_USART_ConfigClock• CR2 CPOL LL_USART_ConfigClock• CR2 LBCL LL_USART_ConfigClock

LL_USART_EnableSCLKOutput

Function name	<u>__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTTx)</u>
Function description	Enable Clock output on SCLK pin.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_USART_INSTANCE(USARTTx) can be used to check whether or not Synchronous mode is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name	<u>__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTTx)</u>
Function description	Disable Clock output on SCLK pin.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_USART_INSTANCE(USARTTx) can be used to check whether or not Synchronous mode is supported by the USARTTx 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 (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:](#)

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

(*) Values not available on all devices

Return values • **None:**

[Reference Manual to LL](#)
[API cross reference:](#)

LL_USART_GetStopBitsLength

Function name `__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (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
- (*) Values not available on all devices

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
- (*) Values not available on all devices

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**
- **USARTTx:** USART Instance
 - **SwapConfig:** This parameter can be one of the following values:
 - LL_USART_RXRX_STANDARD
 - LL_USART_RXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

`LL_USART_SetTXRXSwap`

Function name `__STATIC_INLINE uint32_t LL_USART_SetTXRXSwap (USART_TypeDef * USARTTx)`

Function description Retrieve TX/RX pins swapping configuration.

- Parameters**
- **USARTTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
 - LL_USART_RXRX_STANDARD
 - LL_USART_RXRX_SWAPPED

Reference Manual to LL API cross reference:

`LL_USART_SetRXPinLevel`

Function name `__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTTx, uint32_t PinInvMethod)`

Function description Configure RX pin active level logic.

- Parameters**
- **USARTTx:** 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:

`LL_USART_GetRXPinLevel`

Function name `__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (USART_TypeDef * USARTTx)`

Function description Retrieve RX pin active level logic configuration.

- Parameters**
- **USARTTx:** USART Instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_USART_RXPIN_LEVEL_STANDARD– LL_USART_RXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 RXINV LL_USART_GetRXPinLevel
LL_USART_SetTXPinLevel	
Function name	<code>__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)</code>
Function description	Configure TX pin active level logic.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance• PinInvMethod: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_TXPIN_LEVEL_STANDARD– LL_USART_TXPIN_LEVEL_INVERTED
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 TXINV LL_USART_SetTXPinLevel
LL_USART_GetTXPinLevel	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (USART_TypeDef * USARTx)</code>
Function description	Retrieve TX pin active level logic configuration.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_USART_TXPIN_LEVEL_STANDARD– LL_USART_TXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 TXINV LL_USART_GetTXPinLevel
LL_USART_SetBinaryDataLogic	
Function name	<code>__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)</code>
Function description	Configure Binary data logic.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance• DataLogic: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_BINARY_LOGIC_POSITIVE– LL_USART_BINARY_LOGIC_NEGATIVE
Return values	<ul style="list-style-type: none">• None:

Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 DATAINV LL_USART_SetBinaryDataLogic
LL_USART_GetBinaryDataLogic	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (USART_TypeDef * USARTx)</code>
Function description	Retrieve Binary data configuration.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_USART_BINARY_LOGIC_POSITIVELL_USART_BINARY_LOGIC_NEGATIVE
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 DATAINV LL_USART_SetBinaryDataLogic
LL_USART_SetTransferBitOrder	
Function name	<code>__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)</code>
Function description	Configure transfer bit order (either Less or Most Significant Bit First)
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceBitOrder: This parameter can be one of the following values:<ul style="list-style-type: none">LL_USART_BITORDER_LSBFIRSTLL_USART_BITORDER_MSBFIRST
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 MSBFIRST LL_USART_SetTransferBitOrder
LL_USART_GetTransferBitOrder	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (USART_TypeDef * USARTx)</code>
Function description	Return transfer bit order (either Less or Most Significant Bit First)
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">Returned: value can be one of the following values:<ul style="list-style-type: none">LL_USART_BITORDER_LSBFIRSTLL_USART_BITORDER_MSBFIRST

Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 MSBFIRST LL_USART_GetTransferBitOrder
LL_USART_EnableAutoBaudRate	
Function name	<code>__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)</code>
Function description	Enable Auto Baud-Rate Detection.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 ABREN LL_USART_EnableAutoBaudRate
LL_USART_DisableAutoBaudRate	
Function name	<code>__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)</code>
Function description	Disable Auto Baud-Rate Detection.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 ABREN LL_USART_DisableAutoBaudRate
LL_USART_IsEnabledAutoBaud	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (USART_TypeDef * USARTx)</code>
Function description	Indicate if Auto Baud-Rate Detection mechanism is enabled.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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 (*)

(*) Values not available on all devices

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:****LL_USART_GetAutoBaudRateMode**

Function name `__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (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 (*)

(*) Values not available on all devices

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_EnableRxTimeout

Function name	<code>__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)</code>
Function description	Enable Receiver Timeout.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name	<code>__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)</code>
Function description	Disable Receiver Timeout.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (USART_TypeDef * USARTx)</code>
Function description	Indicate if Receiver Timeout feature is enabled.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name	<code>__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)</code>
Function description	Set Address of the USART node.
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceAddressLen: This parameter can be one of the following values:<ul style="list-style-type: none">LL_USART_ADDRESS_DETECT_4BLL_USART_ADDRESS_DETECT_7BNodeAddress: 4 or 7 bit Address of the USART node.

Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR2 ADD LL_USART_ConfigNodeAddress• CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (USART_TypeDef * USARTx)</u>
Function description	Return 8 bit Address of the USART node as set in ADD field of CR2.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• Address: of the USART node (Value between Min_Data=0 and Max_Data=255)
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (USART_TypeDef * USARTx)</u>
Function description	Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_USART_ADDRESS_DETECT_4B– LL_USART_ADDRESS_DETECT_7B
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name	<u>__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)</u>
Function description	Enable RTS HW Flow Control.

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_UART_HWFLOW_INSTANCE(USARTTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name	<code>__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)</code>
Function description	Disable RTS HW Flow Control.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_UART_HWFLOW_INSTANCE(USARTTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name	<code>__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)</code>
Function description	Enable CTS HW Flow Control.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_UART_HWFLOW_INSTANCE(USARTTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name	<code>__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)</code>
Function description	Disable CTS HW Flow Control.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• 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 (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	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 ONEBIT LL_USART_EnableOneBitSamp
	LL_USART_DisableOneBitSamp
Function name	__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTTx)
Function description	Disable One bit sampling method.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 ONEBIT LL_USART_DisableOneBitSamp
	LL_USART_IsEnabledOneBitSamp
Function name	__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (USART_TypeDef * USARTTx)
Function description	Indicate if One bit sampling method is enabled.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 ONEBIT LL_USART_IsEnabledOneBitSamp
	LL_USART_EnableOverrunDetect
Function name	__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTTx)
Function description	Enable Overrun detection.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 OVRDIS LL_USART_EnableOverrunDetect
	LL_USART_DisableOverrunDetect
Function name	__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTTx)

Function description	Disable Overrun detection.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 OVRDIS LL_USART_DisableOverrunDetect
	LL_USART_IsEnabledOverrunDetect
Function name	<u>__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (USART_TypeDef * USARTTx)</u>
Function description	Indicate if Overrun detection is enabled.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 OVRDIS LL_USART_IsEnabledOverrunDetect
	LL_USART_SetWKUPType
Function name	<u>__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTTx, uint32_t Type)</u>
Function description	Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	<ul style="list-style-type: none">USARTTx: USART InstanceType: This parameter can be one of the following values:<ul style="list-style-type: none">LL_USART_WAKEUP_ON_ADDRESSLL_USART_WAKEUP_ON_STARTBITLL_USART_WAKEUP_ON_RXNE
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 WUS LL_USART_SetWKUPType
	LL_USART_GetWKUPType
Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetWKUPType (USART_TypeDef * USARTTx)</u>
Function description	Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_USART_WAKEUP_ON_ADDRESS– LL_USART_WAKEUP_ON_STARTBIT– LL_USART_WAKEUP_ON_RXNE
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR3 WUS LL_USART_GetWKUPTYPE

LL_USART_SetBaudRate

Function name	<code>__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t OverSampling, uint32_t BaudRate)</code>
Function description	Configure USART BRR register for achieving expected Baud Rate value.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance• PeriphClk: Peripheral Clock• OverSampling: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_OVERSAMPLING_16– LL_USART_OVERSAMPLING_8• BaudRate: Baud Rate
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t OverSampling)</code>
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	<ul style="list-style-type: none">• USARTx: USART Instance• PeriphClk: Peripheral Clock• OverSampling: This parameter can be one of the following values:<ul style="list-style-type: none">– LL_USART_OVERSAMPLING_16– LL_USART_OVERSAMPLING_8
Return values	<ul style="list-style-type: none">• 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_SetBaudRate

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 (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_SetRxTimeout

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 (USART_TypeDef * USARTx)`

Function description	Get Block Length value in reception.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">Value: between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">RTOR BLEN LL_USART_GetBlockLength
LL_USART_EnableIrda	
Function name	<u>__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)</u>
Function description	Enable IrDA mode.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 IREN LL_USART_EnableIrda
LL_USART_DisableIrda	
Function name	<u>__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)</u>
Function description	Disable IrDA mode.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 IREN LL_USART_DisableIrda
LL_USART_IsEnabledIrda	
Function name	<u>__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (USART_TypeDef * USARTx)</u>
Function description	Indicate if IrDA mode is enabled.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">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 (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	<ul style="list-style-type: none">• USARTTx: USART Instance• PrescalerValue: Value between Min_Data=0x00 and Max_Data=0xFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_IRDA_INSTANCE(USARTTx) can be used to check whether or not IrDA feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (USART_TypeDef * USARTTx)</u>
Function description	Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• Irda: prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	<ul style="list-style-type: none">• Macro IS_IRDA_INSTANCE(USARTTx) can be used to check whether or not IrDA feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name	<u>__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTTx)</u>
Function description	Enable Smartcard NACK transmission.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_SMARTCARD_INSTANCE(USARTTx) can be used to check whether or not Smartcard feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name	<u>__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTTx)</u>
Function description	Disable Smartcard NACK transmission.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance

Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (USART_TypeDef * USARTx)</code>
----------------------	---

Function description	Indicate if Smartcard NACK transmission is enabled.
-----------------------------	---

Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
-------------------	---

Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
----------------------	--

Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR3 NACK LL_USART_IsEnabledSmartcardNACK
--	--

LL_USART_EnableSmartcard

Function name	<code>__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)</code>
----------------------	--

Function description	Enable Smartcard mode.
-----------------------------	------------------------

Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
-------------------	---

Return values	<ul style="list-style-type: none">• None:
----------------------	--

Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR3 SCEN LL_USART_EnableSmartcard
--	---

LL_USART_DisableSmartcard

Function name	<code>__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)</code>
----------------------	---

Function description	Disable Smartcard mode.
-----------------------------	-------------------------

Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
-------------------	---

Return values	<ul style="list-style-type: none">• None:
----------------------	--

Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 SCEN LL_USART_DisableSmartcard
LL_USART_IsEnabledSmartcard	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (USART_TypeDef * USARTx)</code>
Function description	Indicate if Smartcard mode is enabled.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 SCEN LL_USART_IsEnabledSmartcard
LL_USART_SetSmartcardAutoRetryCount	
Function name	<code>__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)</code>
Function description	Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceAutoRetryCount: Value between Min_Data=0 and Max_Data=7
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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 of erroneous reception trials, before generating a reception error (RXNE and PE bits set)
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount
LL_USART_GetSmartcardAutoRetryCount	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (USART_TypeDef * USARTx)</code>
Function description	Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	<ul style="list-style-type: none">USARTx: USART Instance

Return values	<ul style="list-style-type: none">• Smartcard: Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount
LL_USART_SetSmartcardPrescaler	
Function name	<code>__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)</code>
Function description	Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance• PrescalerValue: Value between Min_Data=0 and Max_Data=31
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• GTPR PSC LL_USART_SetSmartcardPrescaler
LL_USART_GetSmartcardPrescaler	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (USART_TypeDef * USARTx)</code>
Function description	Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• Smartcard: prescaler value (Value between Min_Data=0 and Max_Data=31)
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• GTPR PSC LL_USART_GetSmartcardPrescaler
LL_USART_SetSmartcardGuardTime	
Function name	<code>__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)</code>
Function description	Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance• GuardTime: Value between Min_Data=0x00 and Max_Data=0xFF
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_SMARTCARD_INSTANCE(USARTTx) can be used to check whether or not Smartcard feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (USART_TypeDef * USARTx)</u>
Function description	Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• Smartcard: Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	<ul style="list-style-type: none">• Macro IS_SMARTCARD_INSTANCE(USARTTx) can be used to check whether or not Smartcard feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name	<u>__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)</u>
Function description	Enable Single Wire Half-Duplex mode.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTTx) can be used to check whether or not Half-Duplex mode is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name	<u>__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)</u>
Function description	Disable Single Wire Half-Duplex mode.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance

Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 HDSEL LL_USART_DisableHalfDuplex
LL_USART_IsEnabledHalfDuplex	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (USART_TypeDef * USARTx)</code>
Function description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 HDSEL LL_USART_IsEnabledHalfDuplex
LL_USART_SetLINBrkDetectionLen	
Function name	<code>__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)</code>
Function description	Set LIN Break Detection Length.
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceLINBDLength: This parameter can be one of the following values:<ul style="list-style-type: none">LL_USART_LINBREAK_DETECT_10BLL_USART_LINBREAK_DETECT_11B
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 LBDL LL_USART_SetLINBrkDetectionLen
LL_USART_GetLINBrkDetectionLen	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (USART_TypeDef * USARTx)</code>
Function description	Return LIN Break Detection Length.
Parameters	<ul style="list-style-type: none">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 (USART_TypeDef * USARTx)**
- Function description**
- Indicate if LIN mode is enabled.
- Parameters**
- **USARTx:** USART Instance
- Return values**
- **State:** of bit (1 or 0).

Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR2 LINEN LL_USART_IsEnabledLIN
LL_USART_SetDEDeassertionTime	
Function name	<u>__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)</u>
Function description	Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceTime: Value between Min_Data=0 and Max_Data=31
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR1 DEDT LL_USART_SetDEDeassertionTime
LL_USART_GetDEDeassertionTime	
Function name	<u>__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (USART_TypeDef * USARTx)</u>
Function description	Return DEDT (Driver Enable De-Assertion Time)
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">Time: value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR1 DEDT LL_USART_GetDEDeassertionTime
LL_USART_SetDEAssertionTime	
Function name	<u>__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)</u>
Function description	Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceTime: Value between Min_Data=0 and Max_Data=31
Return values	<ul style="list-style-type: none">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 (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 (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 (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 (if LIN feature is supported),CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register (if Smartcard feature is supported),IREN bit in the USART_CR3 register (if Irda feature is supported),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() function (if LIN feature is supported)Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() function (if Smartcard feature is supported)Clear IREN in CR3 using LL_USART_DisableIrda() function (if Irda feature is supported)Clear 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 (if LIN feature is supported),SCEN bit in the USART_CR3 register (if Smartcard feature is supported),IREN bit in the USART_CR3 register (if Irda feature is supported),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() function (if LIN feature is supported)Clear IREN in CR3 using LL_USART_DisableIrda() function (if Irda feature is supported)Clear SCEN in CR3 using LL_USART_DisableSmartcard() function (if Smartcard feature is supported)Clear 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	<u>__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)</u>
----------------------	--

Function description	Perform basic configuration of USART for enabling use in LIN Mode.
-----------------------------	--

Parameters	• USARTx: USART Instance
-------------------	---------------------------------

Return values	• None:
----------------------	----------------

Notes	<ul style="list-style-type: none">• 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 (if Smartcard feature is supported),IREN bit in the USART_CR3 register (if Irda feature is supported),HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode.• Macro IS_USART_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() function (if Smartcard feature is supported)Clear IREN in CR3 using LL_USART_DisableIrda() function (if Irda feature is supported)Clear 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:	<ul style="list-style-type: none">• 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	<code>__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)</code>
Function description	Perform basic configuration of USART for enabling use in Half Duplex Mode.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register (if LIN feature is supported),CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register (if Smartcard feature is supported),IREN bit in the USART_CR3 register (if Irda feature is supported). This function also sets the UART/USART in Half Duplex mode.Macro <code>IS_UART_HALFDUPLEX_INSTANCE(USARTx)</code> 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 <code>LL_USART_DisableLIN()</code> function (if LIN feature is supported)Clear CLKEN in CR2 using <code>LL_USART_DisableSCLKOutput()</code> functionClear SCEN in CR3 using <code>LL_USART_DisableSmartcard()</code> function (if Smartcard feature is supported)Clear IREN in CR3 using <code>LL_USART_DisableIrda()</code> function (if Irda feature is supported)Set HDSEL in CR3 using <code>LL_USART_EnableHalfDuplex()</code> functionOther 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:	<ul style="list-style-type: none">CR2 LINEN <code>LL_USART_ConfigHalfDuplexMode</code>CR2 CLKEN <code>LL_USART_ConfigHalfDuplexMode</code>CR3 HDSEL <code>LL_USART_ConfigHalfDuplexMode</code>CR3 SCEN <code>LL_USART_ConfigHalfDuplexMode</code>CR3 IREN <code>LL_USART_ConfigHalfDuplexMode</code>

LL_USART_ConfigSmartcardMode

Function name	<code>__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)</code>
Function description	Perform basic configuration of USART for enabling use in Smartcard Mode.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register (if LIN feature is supported),IREN bit in the USART_CR3 register (if Irda feature is supported),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() function (if LIN feature is supported)Clear IREN in CR3 using LL_USART_DisableIrda() function (if Irda feature is supported)Clear 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 (if LIN feature is supported),STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register (if Smartcard feature is supported),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() function (if LIN feature is supported)Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() function (if Smartcard feature is supported)Clear 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	<code>__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)</code>
----------------------	--

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	<ul style="list-style-type: none">• USARTx: USART Instance
-------------------	---

Return values	<ul style="list-style-type: none">• None:
----------------------	--

Notes	<ul style="list-style-type: none">• In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register (if LIN feature is supported),CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register (if Smartcard feature is supported),IREN bit in the USART_CR3 register (if Irda feature is supported),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() function (if LIN feature is supported)Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() function (if Smartcard feature is supported)Clear IREN in CR3 using LL_USART_DisableIrda() function (if Irda feature is supported)Clear 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	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (USART_TypeDef * USARTx)</code>
----------------------	---

Function description	Check if the USART Parity Error Flag is set or not.
-----------------------------	---

Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
-------------------	---

Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
----------------------	--

Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR PE LL_USART_IsActiveFlag_PE
--	---

LL_USART_IsActiveFlag_FE

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Framing Error Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Noise error detected Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR NF LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (USART_TypeDef * USARTx)</code>
Function description	Check if the USART OverRun Error Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (USART_TypeDef * USARTx)</code>
Function description	Check if the USART IDLE line detected Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Read Data Register Not Empty Flag is set or not.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR RXNE LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TC

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Transmission Complete Flag is set or not.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Transmit Data Register Empty Flag is set or not.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR TXE LL_USART_IsActiveFlag_TXE

LL_USART_IsActiveFlag_LBD

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (USART_TypeDef * USARTx)</code>
Function description	Check if the USART LIN Break Detection Flag is set or not.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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 (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 (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 (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 (USART_TypeDef * USARTx)`

Function description	Check if the USART End Of Block Flag is set or not.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">Macro IS_SMARTCARD_INSTANCE(USARTTx) can be used to check whether or not Smartcard feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR EOBF LL_USART_IsActiveFlag_EOB
	LL_USART_IsActiveFlag_ABRE
Function name	<u>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (USART_TypeDef * USARTx)</u>
Function description	Check if the USART Auto-Baud Rate Error Flag is set or not.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR ABRE LL_USART_IsActiveFlag_ABRE
	LL_USART_IsActiveFlag_ABR
Function name	<u>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (USART_TypeDef * USARTx)</u>
Function description	Check if the USART Auto-Baud Rate Flag is set or not.
Parameters	<ul style="list-style-type: none">USARTTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">ISR ABRF LL_USART_IsActiveFlag_ABR
	LL_USART_IsActiveFlag_BUSY
Function name	<u>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (USART_TypeDef * USARTx)</u>
Function description	Check if the USART Busy Flag is set or not.

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR BUSY LL_USART_IsActiveFlag_BUSY
LL_USART_IsActiveFlag_CM	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Character Match Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR CMF LL_USART_IsActiveFlag_CM
LL_USART_IsActiveFlag_SBK	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Send Break Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR SBKF LL_USART_IsActiveFlag_SBK
LL_USART_IsActiveFlag_RWU	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Receive Wake Up from mute mode Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR RWU LL_USART_IsActiveFlag_RWU
LL_USART_IsActiveFlag_WKUP	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Wake Up from stop mode Flag is set or not.

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTTx instance.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR WUF LL_USART_IsActiveFlag_WKUP

LL_USART_IsActiveFlag_TEACK

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (USART_TypeDef * USARTTx)</code>
Function description	Check if the USART Transmit Enable Acknowledge Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (USART_TypeDef * USARTTx)</code>
Function description	Check if the USART Receive Enable Acknowledge Flag is set or not.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_ClearFlag_PE

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTTx)</code>
Function description	Clear Parity Error Flag.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)</code>
Function description	Clear Framing Error Flag.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)</code>
Function description	Clear Noise detected Flag.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- ICR NCF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)</code>
Function description	Clear OverRun Error Flag.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)</code>
Function description	Clear IDLE line detected Flag.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- ICR IDLECF LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TC

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)</code>
Function description	Clear Transmission Complete Flag.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_LBD

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)</code>
Function description	Clear LIN Break Detection Flag.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)</code>
Function description	Clear CTS Interrupt Flag.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name	<code>__STATIC_INLINE void LL_USART_ClearFlag_RTO (USART_TypeDef * USARTx)</code>
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_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	<code>__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)</code>
Function description	Enable IDLE Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE

Function name	<code>__STATIC_INLINE void LL_USART_EnableIT_RXNE (USART_TypeDef * USARTx)</code>
Function description	Enable RX Not Empty Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 RXNEIE LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TC

Function name	<code>__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)</code>
Function description	Enable Transmission Complete Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE

Function name	<code>__STATIC_INLINE void LL_USART_EnableIT_TXE (USART_TypeDef * USARTx)</code>
Function description	Enable TX Empty Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 TXEIE LL_USART_EnableIT_TXE

LL_USART_EnableIT_PE

Function name	<code>__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)</code>
Function description	Enable Parity Error Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name	<code>__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)</code>
Function description	Enable Character Match Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name	<code>__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)</code>
Function description	Enable Receiver Timeout Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name	<code>__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)</code>
Function description	Enable End Of Block Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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 EOBIIE LL_USART_EnableIT_EOB

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	<code>__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)</code>
Function description	Enable Wake Up from Stop Mode Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR3 WUFIE LL_USART_EnableIT_WKUP

LL_USART_DisableIT_IDLE

Function name	<code>__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)</code>
Function description	Disable IDLE Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE

Function name	<code>__STATIC_INLINE void LL_USART_DisableIT_RXNE (USART_TypeDef * USARTx)</code>
Function description	Disable RX Not Empty Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 RXNEIE LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TC

Function name	<code>__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)</code>
Function description	Disable Transmission Complete Interrupt.
Parameters	<ul style="list-style-type: none">• USARTx: USART Instance
Return values	<ul style="list-style-type: none">• None:

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE

Function name `__STATIC_INLINE void LL_USART_DisableIT_TXE (USART_TypeDef * USARTx)`

Function description Disable TX Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE LL_USART_DisableIT_TXE

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_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_IsEnabledIT_IDLE

Function name `__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (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

Function name `__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE (USART_TypeDef * USARTx)`

Function description Check if the USART RX Not Empty Interrupt is enabled or disabled.

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 RXNEIE LL_USART_IsEnabledIT_RXNE
LL_USART_IsEnabledIT_TC	
Function name	__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (USART_TypeDef * USARTTx)
Function description	Check if the USART Transmission Complete Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 TCIE LL_USART_IsEnabledIT_TC
LL_USART_IsEnabledIT_TXE	
Function name	__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE (USART_TypeDef * USARTTx)
Function description	Check if the USART TX Empty Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 TXEIE LL_USART_IsEnabledIT_TXE
LL_USART_IsEnabledIT_PE	
Function name	__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (USART_TypeDef * USARTTx)
Function description	Check if the USART Parity Error Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 PEIE LL_USART_IsEnabledIT_PE
LL_USART_IsEnabledIT_CM	
Function name	__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (USART_TypeDef * USARTTx)
Function description	Check if the USART Character Match Interrupt is enabled or disabled.

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 CMIE LL_USART_IsEnabledIT_CM
LL_USART_IsEnabledIT_RTO	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Receiver Timeout Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR1 RTOIE LL_USART_IsEnabledIT_RTO
LL_USART_IsEnabledIT_EOB	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (USART_TypeDef * USARTx)</code>
Function description	Check if the USART End Of Block Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR1 EOBIE LL_USART_IsEnabledIT_EOB
LL_USART_IsEnabledIT_LBD	
Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (USART_TypeDef * USARTx)</code>
Function description	Check if the USART LIN Break Detection Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Error Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (USART_TypeDef * USARTx)</code>
Function description	Check if the USART CTS Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_WKUP

Function name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (USART_TypeDef * USARTx)</code>
Function description	Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">CR3 WUFIE LL_USART_IsEnabledIT_WKUP

LL_USART_EnableDMAReq_RX

Function name	<code>__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)</code>
Function description	Enable DMA Mode for reception.

Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 DMAR LL_USART_EnableDMAReq_RX
LL_USART_DisableDMAReq_RX	
Function name	__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTTx)
Function description	Disable DMA Mode for reception.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 DMAR LL_USART_DisableDMAReq_RX
LL_USART_IsEnabledDMAReq_RX	
Function name	__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (USART_TypeDef * USARTTx)
Function description	Check if DMA Mode is enabled for reception.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 DMAR LL_USART_IsEnabledDMAReq_RX
LL_USART_EnableDMAReq_TX	
Function name	__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTTx)
Function description	Enable DMA Mode for transmission.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CR3 DMAT LL_USART_EnableDMAReq_TX
LL_USART_DisableDMAReq_TX	
Function name	__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTTx)
Function description	Disable DMA Mode for transmission.

Parameters • **USARTTx:** 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 (USART_TypeDef * USARTTx)**

Function description Check if DMA Mode is enabled for transmission.

Parameters • **USARTTx:** 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 * USARTTx)**

Function description Enable DMA Disabling on Reception Error.

Parameters • **USARTTx:** 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 * USARTTx)**

Function description Disable DMA Disabling on Reception Error.

Parameters • **USARTTx:** 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 (USART_TypeDef * USARTTx)**

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_IsEnabledDMAOnRxErr

LL_USART_DMA_GetRegAddr

Function name `__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (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 (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 (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	<code>__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)</code>
Function description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceValue: between Min_Data=0x00 and Max_Data=0xFF
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name	<code>__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)</code>
Function description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceValue: between Min_Data=0x00 and Max_Data=0x1FF
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name	<code>__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)</code>
Function description	Request an Automatic Baud Rate measurement on next received data frame.
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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:	<ul style="list-style-type: none">RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name	<code>__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)</code>
---------------	---

Function description	Request Break sending.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RQR SBKRQ LL_USART_RequestBreakSending
LL_USART_RequestEnterMuteMode	
Function name	<u>__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)</u>
Function description	Put USART in mute mode and set the RWU flag.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RQR MMRQ LL_USART_RequestEnterMuteMode
LL_USART_RequestRxDataFlush	
Function name	<u>__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)</u>
Function description	Request a Receive Data flush.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• RQR RXFRQ LL_USART_RequestRxDataFlush
LL_USART_RequestTxDataFlush	
Function name	<u>__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)</u>
Function description	Request a Transmit data flush.
Parameters	<ul style="list-style-type: none">• USARTTx: USART Instance
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• 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:	<ul style="list-style-type: none">• RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_Delnit

Function name	ErrorStatus LL_USART_Delnit (USART_TypeDef * USARTx)
Function description	De-initialize USART registers (Registers restored to their default values).
Parameters	<ul style="list-style-type: none">USARTx: USART Instance
Return values	<ul style="list-style-type: none">An: ErrorStatus enumeration value:<ul style="list-style-type: none">SUCCESS: USART registers are de-initializedERROR: USART registers are not de-initialized

LL_USART_Init

Function name	ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, LL_USART_InitTypeDef * USART_InitStruct)
Function description	Initialize USART registers according to the specified parameters in USART_InitStruct.
Parameters	<ul style="list-style-type: none">USARTx: USART InstanceUSART_InitStruct: pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.
Return values	<ul style="list-style-type: none">An: ErrorStatus enumeration value:<ul style="list-style-type: none">SUCCESS: USART registers are initialized according to USART_InitStruct contentERROR: Problem occurred during USART Registers initialization
Notes	<ul style="list-style-type: none">As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART IP 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	<ul style="list-style-type: none">USART_InitStruct: pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.
Return values	<ul style="list-style-type: none">None:

LL_USART_ClockInit

Function name	ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, LL_USART_ClockInitTypeDef * USART_ClockInitStruct)
Function description	Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.

Parameters	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• An: ErrorStatus enumeration value:<ul style="list-style-type: none">– SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content– ERROR: Problem occurred during USART Registers initialization
Notes	<ul style="list-style-type: none">• As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART IP 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	<ul style="list-style-type: none">• USART_ClockInitStruct: pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.
Return values	<ul style="list-style-type: none">• None:

69.3 USART Firmware driver defines

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

69.3.1 USART

USART

Address Length Detection

LL_USART_ADDRESS_DE 4-bit address detection method selected
TECT_4B

LL_USART_ADDRESS_DE 7-bit address detection (in 8-bit data mode) method selected
TECT_7B

Autobaud Detection

LL_USART_AUTOBAUD_D Measurement of the start bit is used to detect the baud rate
ETECT_ON_STARTBIT

LL_USART_AUTOBAUD_D Falling edge to falling edge measurement. Received frame must start with a single bit = 1 ->
ETECT_ON_FALLINGEDGE Frame = Start1xxxxxxxx

LL_USART_AUTOBAUD_D 0x7F frame detection
ETECT_ON_7F_FRAME

LL_USART_AUTOBAUD_D 0x55 frame detection
ETECT_ON_55_FRAME

Binary Data Inversion

LL_USART_BINARY_LOGI_C_POSITIVE Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_USART_BINARY_LOGI_C_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_LS_BFIRST data is transmitted/received with data bit 0 first, following the start bit

LL_USART_BITORDER_MS_BFIRST data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF Parity error flag

LL_USART_ICR_FECF Framing error flag

LL_USART_ICR_NCF Noise detected flag

LL_USART_ICR_ORECF Overrun error flag

LL_USART_ICR_IDLECF Idle line detected flag

LL_USART_ICR_TCCF Transmission complete flag

LL_USART_ICR_LBDCF LIN break detection flag

LL_USART_ICR_CTSCF CTS flag

LL_USART_ICR_RTOCF Receiver timeout flag

LL_USART_ICR_EOBCF End of block flag

LL_USART_ICR_CMCF Character match flag

LL_USART_ICR_WUCF Wakeup from Stop mode flag

Clock Signal

LL_USART_CLOCK_DISAB_LE Clock signal not provided

LL_USART_CLOCK_ENAB LE Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7_B 7 bits word length : Start bit, 7 data bits, n stop bits

LL_USART_DATAWIDTH_8 8 bits word length : Start bit, 8 data bits, n stop bits
B

LL_USART_DATAWIDTH_9 9 bits word length : Start bit, 9 data bits, n stop bits
B

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_BOTH Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DA_TA_TRANSMIT Get address of data register used for transmission

LL_USART_DMA_REG_DA_TA_RECEIVE Get address of data register used for reception

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 Read data register not empty flag

LL_USART_ISR_TC Transmission complete flag

LL_USART_ISR_TXE Transmit data register empty flag

LL_USART_ISR_LBDF LIN break detection flag

<code>LL_USART_ISR_CTSIF</code>	CTS interrupt flag
<code>LL_USART_ISR_CTS</code>	CTS flag
<code>LL_USART_ISR_RTOF</code>	Receiver timeout flag
<code>LL_USART_ISR_EOBF</code>	End of block flag
<code>LL_USART_ISR_ABRE</code>	Auto baud rate error flag
<code>LL_USART_ISR_ABRF</code>	Auto baud rate flag
<code>LL_USART_ISR_BUSY</code>	Busy flag
<code>LL_USART_ISR_CMF</code>	Character match flag
<code>LL_USART_ISR_SBKF</code>	Send break flag
<code>LL_USART_ISR_RWU</code>	Receiver wakeup from Mute mode flag
<code>LL_USART_ISR_WUF</code>	Wakeup from Stop mode flag
<code>LL_USART_ISR_TEACK</code>	Transmit enable acknowledge flag
<code>LL_USART_ISR_REACK</code>	Receive enable acknowledge flag

Hardware Control

`LL_USART_HWCONTROL_` CTS and RTS hardware flow control disabled
`NONE`

`LL_USART_HWCONTROL_` RTS output enabled, data is only requested when there is space in the receive buffer
`RTS`

`LL_USART_HWCONTROL_` CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)
`CTS`

`LL_USART_HWCONTROL_` CTS and RTS hardware flow control enabled
`RTS_CTS`

IrDA Power

`LL_USART_IRDA_POWER_` IrDA normal power mode
`NORMAL`

`LL_USART_IRDA_POWER_` IrDA low power mode
`LOW`

IT Defines

`LL_USART_CR1_IDLEIE` IDLE interrupt enable

`LL_USART_CR1_RXNEIE` Read data register not empty interrupt enable

<code>LL_USART_CR1_TCIE</code>	Transmission complete interrupt enable
<code>LL_USART_CR1_TXEIE</code>	Transmit data register empty interrupt enable
<code>LL_USART_CR1_PEIE</code>	Parity error
<code>LL_USART_CR1_CMIE</code>	Character match interrupt enable
<code>LL_USART_CR1_RTOIE</code>	Receiver timeout interrupt enable
<code>LL_USART_CR1_EOBIE</code>	End of Block interrupt enable
<code>LL_USART_CR2_LBDIE</code>	LIN break detection interrupt enable
<code>LL_USART_CR3_EIE</code>	Error interrupt enable
<code>LL_USART_CR3_CTSIE</code>	CTS interrupt enable
<code>LL_USART_CR3_WUFIE</code>	Wakeup from Stop mode interrupt enable

Last Clock Pulse

`LL_USART_LASTCLKPULS` The clock pulse of the last data bit is not output to the SCLK pin
`E_NO_OUTPUT`

`LL_USART_LASTCLKPULS` The clock pulse of the last data bit is output to the SCLK pin
`E_OUTPUT`

LIN Break Detection Length

`LL_USART_LINBREAK_DE` 10-bit break detection method selected
`TECT_10B`

`LL_USART_LINBREAK_DE` 11-bit break detection method selected
`TECT_11B`

Oversampling

`LL_USART_OVERSAMPLIN` Oversampling by 16
`G_16`

`LL_USART_OVERSAMPLIN` Oversampling by 8
`G_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_1EDG_E The first clock transition is the first data capture edge

LL_USART_PHASE_2EDG_E The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LO_W Steady low value on SCLK pin outside transmission window

LL_USART_POLARITY_HI_GH Steady high value on SCLK pin outside transmission window

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_STANDA_RD TX/RX pins are used as defined in standard pinout

LL_USART_TXRX_SWAPP_ED TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLE_LINE 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
ADDRESS

LL_USART_WAKEUP_ON_STARTBIT Wake up active on Start bit detection
STARTBIT

LL_USART_WAKEUP_ON_RXNE Wake up active on RXNE
RXNE

Exported Macros Helper

__LL_USART_DIV_SAMPLI Description:

NG8

- 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
- BAUDRATE: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLI Description:

NG16

- 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
- 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

70 LL UTILS Generic Driver

70.1 UTILS Firmware driver registers structures

70.1.1 LL_UTILS_PLLInitTypeDef

`LL_UTILS_PLLInitTypeDef` is defined in the `stm32f0xx_ll_utils.h`

Data Fields

- `uint32_t PLLMul`
- `uint32_t PLLDiv`

Field Documentation

- `uint32_t LL_UTILS_PLLInitTypeDef::PLLMul`

Multiplication factor for PLL VCO input clock. This parameter can be a value of `RCC_LL_EC_PLL_MUL`This feature can be modified afterwards using unitary function `LL_RCC_PLL_ConfigDomain_SYS()`.

- `uint32_t LL_UTILS_PLLInitTypeDef::PLLDiv`

Division factor for PLL VCO output clock. This parameter can be a value of `RCC_LL_EC_PREDIV_DIV`This feature can be modified afterwards using unitary function `LL_RCC_PLL_ConfigDomain_SYS()`.

70.1.2 LL_UTILS_ClkInitTypeDef

`LL_UTILS_ClkInitTypeDef` is defined in the `stm32f0xx_ll_utils.h`

Data Fields

- `uint32_t AHCLKDivider`
- `uint32_t APB1CLKDivider`

Field Documentation

- `uint32_t LL_UTILS_ClkInitTypeDef::AHCLKDivider`

The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of `RCC_LL_EC_SYSCLK_DIV`This feature can be modified afterwards using unitary function `LL_RCC_SetAHBPrescaler()`.

- `uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider`

The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of `RCC_LL_EC_APB1_DIV`This feature can be modified afterwards using unitary function `LL_RCC_SetAPB1Prescaler()`.

70.2 UTILS Firmware driver API description

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

70.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 48000000 Hz.

This section contains the following APIs:

- `LL_SetSystemCoreClock`
- `LL_PLL_ConfigSystemClock_HSI`
- `LL_PLL_ConfigSystemClock_HSI48`
- `LL_PLL_ConfigSystemClock_HSE`

70.2.2 Detailed description of functions

`LL_GetUID_Word0`

Function name

`__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)`

Function description	Get Word0 of the unique device identifier (UID based on 96 bits)
Return values	<ul style="list-style-type: none">UID[31:0]: X and Y coordinates on the wafer expressed in BCD format
LL_GetUID_Word1	
Function name	<code>__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)</code>
Function description	Get Word1 of the unique device identifier (UID based on 96 bits)
Return values	<ul style="list-style-type: none">UID[63:32]: Wafer number (UID[39:32]) & LOT_NUM[23:0] (UID[63:40])
LL_GetUID_Word2	
Function name	<code>__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)</code>
Function description	Get Word2 of the unique device identifier (UID based on 96 bits)
Return values	<ul style="list-style-type: none">UID[95:64]: Lot number (ASCII encoded) - LOT_NUM[55:24]
LL_GetFlashSize	
Function name	<code>__STATIC_INLINE uint32_t LL_GetFlashSize (void)</code>
Function description	Get Flash memory size.
Return values	<ul style="list-style-type: none">FLASH_SIZE[15:0]: Flash memory size
Notes	<ul style="list-style-type: none">This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.
LL_InitTick	
Function name	<code>__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)</code>
Function description	This function configures the Cortex-M SysTick source of the time base.
Parameters	<ul style="list-style-type: none">HCLKFrequency: HCLK frequency in Hz (can be calculated thanks to RCC helper macro)Ticks: Number of ticks
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">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	<code>void LL_Init1msTick (uint32_t HCLKFrequency)</code>
Function description	This function configures the Cortex-M SysTick source to have 1ms time base.

Parameters	<ul style="list-style-type: none">HCLKFrequency: HCLK frequency in Hz
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.HCLK frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_mDelay

Function name	void LL_mDelay (uint32_t Delay)
Function description	This function provides accurate delay (in milliseconds) based on SysTick counter flag.
Parameters	<ul style="list-style-type: none">Delay: specifies the delay time length, in milliseconds.
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service.To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name	void LL_SetSystemCoreClock (uint32_t HCLKFrequency)
Function description	This function sets directly SystemCoreClock CMSIS variable.
Parameters	<ul style="list-style-type: none">HCLKFrequency: HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">Variable can be calculated also through SystemCoreClockUpdate function.

LL_PLL_ConfigSystemClock_HSI

Function name	ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)
Function description	This function configures system clock with HSI as clock source of the PLL.
Parameters	<ul style="list-style-type: none">UTILS_PLLInitStruct: pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.UTILS_ClkInitStruct: pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.
Return values	<ul style="list-style-type: none">An: ErrorStatus enumeration value:<ul style="list-style-type: none">SUCCESS: Max frequency configuration doneERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL is disabled.
- Function is based on the following formula: $\text{PLL output frequency} = ((\text{HSI frequency} / \text{PREDIV}) * \text{PLLMUL}) \text{PREDIV}$: Set to 2 for few devices
- PLLMUL: The application software must set correctly the PLL multiplication factor to be in the range 16-48MHz
- FLASH latency can be modified through this function.

LL_PLL_ConfigSystemClock_HSI48

Function name `ErrorStatus LL_PLL_ConfigSystemClock_HSI48 (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)`

Function description This function configures system clock with HSI48 as clock source of the PLL.

Parameters

- **UTILS_PLLInitStruct**: pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct**: pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An**: ErrorStatus enumeration value:
 - SUCCESS: Max frequency configuration done
 - ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL is disabled.
- Function is based on the following formula: $\text{PLL output frequency} = ((\text{HSI48 frequency} / \text{PREDIV}) * \text{PLLMUL}) \text{PLLMUL}$: The application software must set correctly the PLL multiplication factor to be in the range 16-48MHz

LL_PLL_ConfigSystemClock_HSE

Function name `ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)`

Function description This function configures system clock with HSE as clock source of the PLL.

Parameters

- **HSEFrequency**: Value between Min_Data = 4000000 and Max_Data = 32000000
- **HSEBypass**: This parameter can be one of the following values:
 - LL_UTILS_HSEBYPASS_ON
 - LL_UTILS_HSEBYPASS_OFF
- **UTILS_PLLInitStruct**: pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct**: pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An**: ErrorStatus enumeration value:
 - SUCCESS: Max frequency configuration done
 - ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL is disabled.
- Function is based on the following formula: $\text{PLL output frequency} = ((\text{HSE frequency} / \text{PREDIV}) * \text{PLLMUL}) \text{PLLMUL}$: The application software must set correctly the PLL multiplication factor to be in the range 16-48MHz
- FLASH latency can be modified through this function.

70.3 UTILS Firmware driver defines

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

70.3.1 UTILS

UTILS

HSE Bypass activation

`LL_UTILS_HSEBYPASS_O` HSE Bypass is not enabled
FF

`LL_UTILS_HSEBYPASS_O` HSE Bypass is enabled
N

71 LL WWDG Generic Driver

71.1 WWDG Firmware driver API description

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

71.1.1 Detailed description of functions

`LL_WWDG_Enable`

Function name	<code>__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)</code>
Function description	Enable Window Watchdog.
Parameters	<ul style="list-style-type: none">WWDGx: WWDG Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR WDGA LL_WWDG_Enable

`LL_WWDG_IsEnabled`

Function name	<code>__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (WWDG_TypeDef * WWDGx)</code>
Function description	Checks if Window Watchdog is enabled.
Parameters	<ul style="list-style-type: none">WWDGx: WWDG Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CR WDGA LL_WWDG_IsEnabled

`LL_WWDG_SetCounter`

Function name	<code>__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)</code>
Function description	Set the Watchdog counter value to provided value (7-bits T[6:0])
Parameters	<ul style="list-style-type: none">WWDGx: WWDG InstanceCounter: 0..0x7F (7 bit counter value)
Return values	<ul style="list-style-type: none">None:

Notes

- When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset. This counter is decremented every $(4096 \times 2^{\text{exp}}\text{WDGTB})$ PCLK cycles. A reset is produced when it rolls over from 0x40 to 0x3F (bit T6 becomes cleared). Setting the counter lower than 0x40 causes an immediate reset (if WWDG enabled).

Reference Manual to LL API cross reference:

- CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name `__STATIC_INLINE uint32_t LL_WWDG_GetCounter (WWDG_TypeDef * WWDGx)`

Function description Return current Watchdog Counter Value (7 bits counter value)

Parameters

- WWDGx:** WWDG Instance

Return values

- 7: bit Watchdog Counter value

Reference Manual to LL API cross reference:

- CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name `__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)`

Function description Set the time base of the prescaler (WDGTB).

Parameters

- WWDGx:** WWDG Instance
- Prescaler:** This parameter can be one of the following values:
 - LL_WWDG_PRESCALER_1
 - LL_WWDG_PRESCALER_2
 - LL_WWDG_PRESCALER_4
 - LL_WWDG_PRESCALER_8

Return values

- None:**

Notes

- Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every $(4096 \times 2^{\text{exp}}\text{WDGTB})$ PCLK cycles

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name `__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (WWDG_TypeDef * WWDGx)`

Function description Return current Watchdog Prescaler Value.

Parameters

- WWDGx:** WWDG Instance

Return values	<ul style="list-style-type: none">• Returned: value can be one of the following values:<ul style="list-style-type: none">– LL_WWDG_PRESCALER_1– LL_WWDG_PRESCALER_2– LL_WWDG_PRESCALER_4– LL_WWDG_PRESCALER_8
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFR WDGTB LL_WWDG_GetPrescaler
LL_WWDG_SetWindow	
Function name	<code>__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)</code>
Function description	Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).
Parameters	<ul style="list-style-type: none">• WWDGx: WWDG Instance• Window: 0x00..0x7F (7 bit Window value)
Return values	<ul style="list-style-type: none">• None:
Notes	<ul style="list-style-type: none">• This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower than 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFR W LL_WWDG_SetWindow
LL_WWDG_GetWindow	
Function name	<code>__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)</code>
Function description	Return current Watchdog Window Value (7 bits value)
Parameters	<ul style="list-style-type: none">• WWDGx: WWDG Instance
Return values	<ul style="list-style-type: none">• 7: bit Watchdog Window value
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">• CFR W LL_WWDG_GetWindow
LL_WWDG_IsActiveFlag_EWKUP	
Function name	<code>__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)</code>
Function description	Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.
Parameters	<ul style="list-style-type: none">• WWDGx: WWDG Instance

Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Notes	<ul style="list-style-type: none">This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR EWIF LL_WWDG_IsActiveFlag_EWKUP
	LL_WWDG_ClearFlag_EWKUP
Function name	<code>__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)</code>
Function description	Clear WWDG Early Wakeup Interrupt Flag (EWIF)
Parameters	<ul style="list-style-type: none">WWDGx: WWDG Instance
Return values	<ul style="list-style-type: none">None:
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">SR EWIF LL_WWDG_ClearFlag_EWKUP
	LL_WWDG_EnableIT_EWKUP
Function name	<code>__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)</code>
Function description	Enable the Early Wakeup Interrupt.
Parameters	<ul style="list-style-type: none">WWDGx: WWDG Instance
Return values	<ul style="list-style-type: none">None:
Notes	<ul style="list-style-type: none">When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CFR EWI LL_WWDG_EnableIT_EWKUP
	LL_WWDG_IsEnabledIT_EWKUP
Function name	<code>__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (WWDG_TypeDef * WWDGx)</code>
Function description	Check if Early Wakeup Interrupt is enabled.
Parameters	<ul style="list-style-type: none">WWDGx: WWDG Instance
Return values	<ul style="list-style-type: none">State: of bit (1 or 0).
Reference Manual to LL API cross reference:	<ul style="list-style-type: none">CFR EWI LL_WWDG_IsEnabledIT_EWKUP

71.2 WWDG Firmware driver defines

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

71.2.1 WWDG

WWDG

IT Defines

[LL_WWDG_CFR_EWI](#)

PRESCALER

[LL_WWDG_PRESCALER_1](#) WWDG counter clock = (PCLK1/4096)/1

[LL_WWDG_PRESCALER_2](#) WWDG counter clock = (PCLK1/4096)/2

[LL_WWDG_PRESCALER_4](#) WWDG counter clock = (PCLK1/4096)/4

[LL_WWDG_PRESCALER_8](#) WWDG counter clock = (PCLK1/4096)/8

Common Write and read registers macros

[LL_WWDG_WriteReg](#)

Description:

- Write a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

[LL_WWDG_ReadReg](#)

Description:

- Read a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

72 Correspondence between API registers and API low-layer driver functions

This annex contains correspondance table between the register or bit name, as mentionned inside the reference manual of the component, and the name of the LL functions to modify or read this register or bit.

72.1 ADC

Table 25. Correspondence between ADC registers and ADC low-layer driver functions

Register	Field	Function
CCR	TSEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
	VBATEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
	VREFEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
CFG1	ALIGN	LL_ADC.GetDataAlignment
		LL_ADC_SetDataAlignment
	AUTOFF	LL_ADC.GetLowPowerMode
		LL_ADC_SetLowPowerMode
	AWDCH	LL_ADC.GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	AWDEN	LL_ADC.GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	AWDSGL	LL_ADC.GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	CONT	LL_ADC_REG.GetContinuousMode
		LL_ADC_REG_SetContinuousMode
	DISCEN	LL_ADC_REG.GetSequencerDiscont
		LL_ADC_REG_SetSequencerDiscont
	DMACFG	LL_ADC_REG.GetDMATransfer
		LL_ADC_REG_SetDMATransfer
	DMAEN	LL_ADC_REG.GetDMATransfer
		LL_ADC_REG_SetDMATransfer
	EXTEN	LL_ADC_REG.GetTriggerEdge
		LL_ADC_REG.GetTriggerSource
		LL_ADC_REG_IsTriggerSourceSWStart
		LL_ADC_REG_SetTriggerEdge
		LL_ADC_REG_SetTriggerSource
	EXTSEL	LL_ADC_REG.GetTriggerSource
		LL_ADC_REG_SetTriggerSource

Register	Field	Function
CFGREG1	OVRMOD	LL_ADC_REG_GetOverrun
		LL_ADC_REG_SetOverrun
	RES	LL_ADC_GetResolution
		LL_ADC_SetResolution
	SCANDIR	LL_ADC_REG_GetSequencerScanDirection
		LL_ADC_REG_SetSequencerScanDirection
	WAIT	LL_ADC_GetLowPowerMode
		LL_ADC_SetLowPowerMode
CFGREG2	CKMODE	LL_ADC_GetClock
		LL_ADC_SetClock
CHSELREG	CHSEL0	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL1	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL10	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL11	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL12	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL13	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL14	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL15	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd

Register	Field	Function
CHSELR	CHSEL15	LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL16	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL17	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL18	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL2	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL3	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL4	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL5	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL6	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL7	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL8	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd

Register	Field	Function
CHSELR	CHSEL8	LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL9	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
CR	ADCAL	LL_ADC_IsCalibrationOnGoing
		LL_ADC_StartCalibration
	ADDIS	LL_ADC_Disable
		LL_ADC_IsDisableOngoing
	ADEN	LL_ADC_Enable
		LL_ADC_IsEnabled
	ADSTART	LL_ADC_REG_IsConversionOngoing
		LL_ADC_REG_StartConversion
	ADSTP	LL_ADC_REG_IsStopConversionOngoing
		LL_ADC_REG_StopConversion
DR	DATA	LL_ADC_DMA_GetRegAddr
		LL_ADC_REG_ReadConversionData10
		LL_ADC_REG_ReadConversionData12
		LL_ADC_REG_ReadConversionData32
		LL_ADC_REG_ReadConversionData6
		LL_ADC_REG_ReadConversionData8
IER	ADRDYIE	LL_ADC_DisableIT_ADRDY
		LL_ADC_EnableIT_ADRDY
		LL_ADC_IsEnabledIT_ADRDY
	AWDIE	LL_ADC_DisableIT_AWD1
		LL_ADC_EnableIT_AWD1
		LL_ADC_IsEnabledIT_AWD1
	EOCIE	LL_ADC_DisableIT_EOC
		LL_ADC_EnableIT_EOC
		LL_ADC_IsEnabledIT_EOC
	EOSEQIE	LL_ADC_DisableIT_EOS
		LL_ADC_EnableIT_EOS
		LL_ADC_IsEnabledIT_EOS
	EOSMPIE	LL_ADC_DisableIT_EOSMP
		LL_ADC_EnableIT_EOSMP
		LL_ADC_IsEnabledIT_EOSMP
	OVRIE	LL_ADC_DisableIT_OVR
		LL_ADC_EnableIT_OVR
		LL_ADC_IsEnabledIT_OVR

Register	Field	Function
ISR	ADRDY	LL_ADC_ClearFlag_ADRDY
		LL_ADC_IsActiveFlag_ADRDY
	AWD	LL_ADC_ClearFlag_AWD1
		LL_ADC_IsActiveFlag_AWD1
	EOC	LL_ADC_ClearFlag_EOC
		LL_ADC_IsActiveFlag_EOC
	EOSEQ	LL_ADC_ClearFlag_EOS
		LL_ADC_IsActiveFlag_EOS
	EOSMP	LL_ADC_ClearFlag_EOSMP
		LL_ADC_IsActiveFlag_EOSMP
	OVR	LL_ADC_ClearFlag_OVR
		LL_ADC_IsActiveFlag_OVR
SMPR	SMP	LL_ADC_GetSamplingTimeCommonChannels LL_ADC_SetSamplingTimeCommonChannels
TR	HT	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
TR1	HT1	LL_ADC_GetAnalogWDThresholds
	LT1	LL_ADC_GetAnalogWDThresholds
TR2	HT2	LL_ADC_GetAnalogWDThresholds
	LT2	LL_ADC_GetAnalogWDThresholds
TR3	HT3	LL_ADC_GetAnalogWDThresholds
	LT3	LL_ADC_GetAnalogWDThresholds

72.2 BUS

Table 26. Correspondence between BUS registers and BUS low-layer driver functions

Register	Field	Function
AHBENR	CRCEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA2EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FLITFEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock

Register	Field	Function
AHBENR	FLITFEN	LL_AHB1_GRP1_IsEnabledClock
	GPIOAEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
	GPIOBEN	LL_AHB1_GRP1_IsEnabledClock
		LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
	GPIOCEN	LL_AHB1_GRP1_IsEnabledClock
		LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
	GPIODEN	LL_AHB1_GRP1_IsEnabledClock
		LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
AHBRSTR	GPIOARST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPIOBRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPIOCRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPIODRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPIOERST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPIOFRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	TSCRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
APB1ENR	CANEN	LL_APB1_GRP1_DisableClock

Register	Field	Function
APB1ENR	CANEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_IsEnabledClock
	CECEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CRSEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	DACEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C1EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	PWREN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	SPI2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM14EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM3EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM6EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
APB1ENR	TIM7EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock

Register	Field	Function
APB1ENR	USART2EN	LL_APB1_GRP1_IsEnabledClock
		LL_APB1_GRP1_DisableClock
	USART3EN	LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART4EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART5EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
APB1RSTR	USBN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	WWDGREN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CANRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	CECRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	CRSRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	DACRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
I2C1RST	I2C1RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	I2C2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	PWRRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	SPI2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM14RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
TIM2RST		LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM3RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
TIM6RST		LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset

Register	Field	Function
APB1RSTR	TIM7RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART3RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART4RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
APB2ENR	USART5RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USBRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	WWDGRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	ADC1EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	DBGMCUEN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	SPI1EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	SYSCFGEN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	TIM15EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	TIM16EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	TIM17EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	TIM1EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	USART1EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock

Register	Field	Function
APB2ENR	USART1EN	LL_APB1_GRP2_IsEnabledClock
		LL_APB1_GRP2_DisableClock
	USART6EN	LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	USART7EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	USART8EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
APB2RSTR	ADC1RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	DBGMCURST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	SPI1RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	SYSCFGRST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	TIM15RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	TIM16RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	TIM17RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	TIM1RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	USART1RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	USART6RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	USART7RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	USART8RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset

72.3 COMP

Table 27. Correspondence between COMP registers and COMP low-layer driver functions

Register	Field	Function
CSR	COMP1EN	LL_COMP_Disable

Register	Field	Function
CSR	COMP1EN	LL_COMP_Enable
		LL_COMP_IsEnabled
	COMP1HYST	LL_COMP_GetInputHysteresis
		LL_COMP_SetInputHysteresis
	COMP1INSEL	LL_COMP_ConfigInputs
		LL_COMP_GetInputPlus
		LL_COMP_SetInputPlus
	COMP1LOCK	LL_COMP_IsLocked
		LL_COMP_Lock
	COMP1MODE	LL_COMP_GetPowerMode
		LL_COMP_SetPowerMode
	COMP1OUT	LL_COMP_ReadOutputLevel
	COMP1OUTSEL	LL_COMP_GetOutputSelection
		LL_COMP_SetOutputSelection
	COMP1POL	LL_COMP_GetOutputPolarity
		LL_COMP_SetOutputPolarity
	COMP1SW1	LL_COMP_ConfigInputs
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	COMP2EN	LL_COMP_Disable
		LL_COMP_Enable
		LL_COMP_IsEnabled
	COMP2INSEL	LL_COMP_ConfigInputs
		LL_COMP_GetInputPlus
		LL_COMP_SetInputPlus
	COMP2LOCK	LL_COMP_IsLocked
		LL_COMP_Lock
	COMP2OUT	LL_COMP_ReadOutputLevel
	WNDWEN	LL_COMP_GetCommonWindowMode
		LL_COMP_SetCommonWindowMode

72.4

CORTEX

Table 28. Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
SCB_CPUID	ARCHITECTURE	LL_CPUID_GetArchitecture
	IMPLEMENTER	LL_CPUID_GetImplementer
	PARTNO	LL_CPUID_GetParNo
	REVISION	LL_CPUID_GetRevision
	VARIANT	LL_CPUID_GetVariant

Register	Field	Function
SCB_SCR	SEVEONPEND	LL_LPM_DisableEventOnPend
		LL_LPM_EnableEventOnPend
	SLEEPDEEP	LL_LPM_EnableDeepSleep
		LL_LPM_EnableSleep
	SLEEPONEXIT	LL_LPM_DisableSleepOnExit
		LL_LPM_EnableSleepOnExit
STK_CTRL	CLKSOURCE	LL_SYSTICK_GetClkSource
		LL_SYSTICK_SetClkSource
	COUNTFLAG	LL_SYSTICK_IsActiveCounterFlag
	TICKINT	LL_SYSTICK_DisableIT
		LL_SYSTICK_EnableIT
		LL_SYSTICK_IsEnabledIT

72.5 CRC

Table 29. Correspondence between CRC registers and CRC low-layer driver functions

Register	Field	Function
CR	POLYSIZE	LL_CRC_GetPolynomialSize
		LL_CRC_SetPolynomialSize
	RESET	LL_CRC_ResetCRCCalculationUnit
	REV_IN	LL_CRC_GetInputDataReverseMode
		LL_CRC_SetInputDataReverseMode
	REV_OUT	LL_CRC_GetOutputDataReverseMode
		LL_CRC_SetOutputDataReverseMode
DR	DR	LL_CRC_FeedData16
		LL_CRC_FeedData32
		LL_CRC_FeedData8
		LL_CRC_ReadData16
		LL_CRC_ReadData32
		LL_CRC_ReadData7
		LL_CRC_ReadData8
IDR	IDR	LL_CRC_Read_IDR
		LL_CRC_Write_IDR
INIT	INIT	LL_CRC_GetInitialData
		LL_CRC_SetInitialData
POL	POL	LL_CRC_GetPolynomialCoef
		LL_CRC_SetPolynomialCoef

72.6 CRS

Table 30. Correspondence between CRS registers and CRS low-layer driver functions

Register	Field	Function
CFG_R	FELIM	LL_CRS_ConfigSynchronization
		LL_CRS_GetFreqErrorLimit
		LL_CRS_SetFreqErrorLimit
	RELOAD	LL_CRS_ConfigSynchronization
		LL_CRS_GetReloadCounter
		LL_CRS_SetReloadCounter
	SYNCDIV	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncDivider
		LL_CRS_SetSyncDivider
	SYNCPOL	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncPolarity
		LL_CRS_SetSyncPolarity
CR	SYNCSRC	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncSignalSource
		LL_CRS_SetSyncSignalSource
	AUTOTRIMEN	LL_CRS_DisableAutoTrimming
		LL_CRS_EnableAutoTrimming
		LL_CRS_IsEnabledAutoTrimming
	CEN	LL_CRS_DisableFreqErrorCounter
		LL_CRS_EnableFreqErrorCounter
		LL_CRS_IsEnabledFreqErrorCounter
	ERRIE	LL_CRS_DisableIT_ERR
		LL_CRS_EnableIT_ERR
		LL_CRS_IsEnabledIT_ERR
	EYNCIE	LL_CRS_DisableIT_ESYNC
		LL_CRS_EnableIT_ESYNC
		LL_CRS_IsEnabledIT_ESYNC
	SWSYNC	LL_CRS_GenerateEvent_SWSYNC
	SYNCOKIE	LL_CRS_DisableIT_SYNCOK
		LL_CRS_EnableIT_SYNCOK
		LL_CRS_IsEnabledIT_SYNCOK
	SYNCWARNIE	LL_CRS_DisableIT_SYNCWARN
		LL_CRS_EnableIT_SYNCWARN
		LL_CRS_IsEnabledIT_SYNCWARN
	TRIM	LL_CRS_ConfigSynchronization
		LL_CRS_GetHSI48SmoothTrimming
		LL_CRS_SetHSI48SmoothTrimming

Register	Field	Function
ICR	ERRC	LL_CRS_ClearFlag_ERR
	ESYNCC	LL_CRS_ClearFlag_ESYNC
	SYNCOKC	LL_CRS_ClearFlag_SYNCOK
	SYNCWARNC	LL_CRS_ClearFlag_SYNCWARN
ISR	ERRF	LL_CRS_IsActiveFlag_ERR
	ESYNCF	LL_CRS_IsActiveFlag_ESYNC
	FECAP	LL_CRS_GetFreqErrorCapture
	FEDIR	LL_CRS_GetFreqErrorDirection
	SYNCERR	LL_CRS_IsActiveFlag_SYNCERR
	SYNCMISS	LL_CRS_IsActiveFlag_SYNCMISS
	SYNCOKF	LL_CRS_IsActiveFlag_SYNCOK
	SYNCWARNF	LL_CRS_IsActiveFlag_SYNCWARN
	TRIMOVF	LL_CRS_IsActiveFlag_TRIMOVF

72.7 DAC

Table 31. Correspondence between DAC registers and DAC low-layer driver functions

Register	Field	Function
CR	BOFF1	LL_DAC_GetOutputBuffer
		LL_DAC_SetOutputBuffer
	BOFF2	LL_DAC_GetOutputBuffer
		LL_DAC_SetOutputBuffer
	DMAEN1	LL_DAC_DisableDMAReq
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAEN2	LL_DAC_DisableDMAReq
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAUDRIE1	LL_DAC_DisableIT_DMAUDR1
		LL_DAC_EnableIT_DMAUDR1
		LL_DAC_IsEnabledIT_DMAUDR1
	DMAUDRIE2	LL_DAC_DisableIT_DMAUDR2
		LL_DAC_EnableIT_DMAUDR2
		LL_DAC_IsEnabledIT_DMAUDR2
	EN1	LL_DAC_Disable
		LL_DAC_Enable
		LL_DAC_IsEnabled
	EN2	LL_DAC_Disable
		LL_DAC_Enable
		LL_DAC_IsEnabled

Register	Field	Function
CR	MAMP1	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MAMP2	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	TEN1	LL_DAC_DisableTrigger
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TEN2	LL_DAC_DisableTrigger
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
CR	TSEL1	LL_DAC_GetTriggerSource
		LL_DAC_SetTriggerSource
	TSEL2	LL_DAC_GetTriggerSource
		LL_DAC_SetTriggerSource
	WAVE1	LL_DAC_GetWaveAutoGeneration
		LL_DAC_SetWaveAutoGeneration
	WAVE2	LL_DAC_GetWaveAutoGeneration
		LL_DAC_SetWaveAutoGeneration
DHR12L1	DACC1DHR	LL_DAC_ConvertData12LeftAligned
		LL_DAC_DMA_GetRegAddr
DHR12L2	DACC2DHR	LL_DAC_ConvertData12LeftAligned
		LL_DAC_DMA_GetRegAddr
DHR12LD	DACC1DHR	LL_DAC_ConvertDualData12LeftAligned
	DACC2DHR	LL_DAC_ConvertDualData12LeftAligned
DHR12R1	DACC1DHR	LL_DAC_ConvertData12RightAligned
		LL_DAC_DMA_GetRegAddr
DHR12R2	DACC2DHR	LL_DAC_ConvertData12RightAligned
		LL_DAC_DMA_GetRegAddr
DHR12RD	DACC1DHR	LL_DAC_ConvertDualData12RightAligned
	DACC2DHR	LL_DAC_ConvertDualData12RightAligned
DHR8R1	DACC1DHR	LL_DAC_ConvertData8RightAligned
		LL_DAC_DMA_GetRegAddr
DHR8R2	DACC2DHR	LL_DAC_ConvertData8RightAligned
		LL_DAC_DMA_GetRegAddr
DHR8RD	DACC1DHR	LL_DAC_ConvertDualData8RightAligned
	DACC2DHR	LL_DAC_ConvertDualData8RightAligned

Register	Field	Function
DOR1	DACC1DOR	LL_DAC_RetrieveOutputData
DOR2	DACC2DOR	LL_DAC_RetrieveOutputData
SR	DMAUDR1	LL_DAC_ClearFlag_DMAUDR1
		LL_DAC_IsActiveFlag_DMAUDR1
	DMAUDR2	LL_DAC_ClearFlag_DMAUDR2
		LL_DAC_IsActiveFlag_DMAUDR2
SWTRIGR	SWTRIG1	LL_DAC_TrigSWConversion
	SWTRIG2	LL_DAC_TrigSWConversion

72.8 DMA

Table 32. 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

Register	Field	Function
CCR	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
		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
CSELR	C1S	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	C2S	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	C3S	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	C4S	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	C5S	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	C6S	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	C7S	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

Register	Field	Function
IFCR	CGIF6	LL_DMA_ClearFlag_GI6
	CGIF7	LL_DMA_ClearFlag_GI7
	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
	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
	CTEIF1	LL_DMA_ClearFlag_TE1
	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
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
	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
	TCIF1	LL_DMA_IsActiveFlag_TC1
	TCIF2	LL_DMA_IsActiveFlag_TC2
	TCIF3	LL_DMA_IsActiveFlag_TC3

Register	Field	Function
ISR	TCIF4	LL_DMA_IsActiveFlag_TC4
	TCIF5	LL_DMA_IsActiveFlag_TC5
	TCIF6	LL_DMA_IsActiveFlag_TC6
	TCIF7	LL_DMA_IsActiveFlag_TC7
	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

72.9 EXTI

Table 33. Correspondence between EXTI registers and EXTI low-layer driver functions

Register	Field	Function
EMR	EMx	LL_EXTI_DisableEvent_0_31
		LL_EXTI_EnableEvent_0_31
		LL_EXTI_IsEnabledEvent_0_31
FTSR	FTx	LL_EXTI_DisableFallingTrig_0_31
		LL_EXTI_EnableFallingTrig_0_31
		LL_EXTI_IsEnabledFallingTrig_0_31
IMR	IMx	LL_EXTI_DisableIT_0_31
		LL_EXTI_EnableIT_0_31
		LL_EXTI_IsEnabledIT_0_31
PR	PIFx	LL_EXTI_ClearFlag_0_31
		LL_EXTI_IsActiveFlag_0_31
		LL_EXTI_ReadFlag_0_31
RTSR	RTx	LL_EXTI_DisableRisingTrig_0_31
		LL_EXTI_EnableRisingTrig_0_31
		LL_EXTI_IsEnabledRisingTrig_0_31
SWIER	SWIx	LL_EXTI_GenerateSWI_0_31

72.10 GPIO

Table 34. 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

Register	Field	Function
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

72.11

I2C

Table 35. 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
	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

Register	Field	Function
CR1	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
CR1	SBC	LL_I2C_DisableSlaveByteControl
		LL_I2C_EnableSlaveByteControl
		LL_I2C_IsEnabledSlaveByteControl
	SMBDEN	LL_I2C_GetMode
		LL_I2C_SetMode
	SMBHEN	LL_I2C_GetMode
		LL_I2C_SetMode
	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

Register	Field	Function
CR1	WUPEN	LL_I2C_DisableWakeUpFromStop
		LL_I2C_EnableWakeUpFromStop
		LL_I2C_IsEnabledWakeUpFromStop
	ADD10	LL_I2C_GetMasterAddressingMode
		LL_I2C_HandleTransfer
		LL_I2C_SetMasterAddressingMode
	AUTOEND	LL_I2C_DisableAutoEndMode
		LL_I2C_EnableAutoEndMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAutoEndMode
CR2	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
ICR	SADD	LL_I2C_GetSlaveAddr
		LL_I2C_HandleTransfer
		LL_I2C_SetSlaveAddr
	START	LL_I2C_GenerateStartCondition
		LL_I2C_HandleTransfer
	STOP	LL_I2C_GenerateStopCondition
		LL_I2C_HandleTransfer
	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

Register	Field	Function
ICR	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
		LL_I2C_DisableOwnAddress1
	OA1EN	LL_I2C_EnableOwnAddress1
		LL_I2C_IsEnabledOwnAddress1
OAR2	OA1MODE	LL_I2C_SetOwnAddress1
	OA2	LL_I2C_SetOwnAddress2
		LL_I2C_DisableOwnAddress2
	OA2EN	LL_I2C_EnableOwnAddress2
PECR		LL_I2C_IsEnabledOwnAddress2
	PEC	LL_I2C_GetSMBusPEC
RXDR	RXDATA	LL_I2C_DMA_GetRegAddr
		LL_I2C_ReceiveData8
TIMEOUTR	TEXTEN	LL_I2C_DisableSMBusTimeout
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
	TIDLE	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutAMode
		LL_I2C_SetSMBusTimeoutAMode

Register	Field	Function
TIMEOUTR	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

72.12 I2S

Table 36. 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
	TXEIE	LL_I2S_DisableIT_TXE
		LL_I2S_EnableIT_TXE
		LL_I2S_IsEnabledIT_TXE
DR	DR	LL_I2S_ReceiveData16
		LL_I2S_TransmitData16
I2SCFGR	CHLEN	LL_I2S_GetDataFormat

Register	Field	Function
I2SCFGR	CHLEN	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
	I2STD	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

72.13 IWDG

Table 37. Correspondence between IWDG registers and IWDG low-layer driver functions

Register	Field	Function
KR	KEY	LL_IWDG_DisableWriteAccess
		LL_IWDG_Enable
		LL_IWDG_EnableWriteAccess

Register	Field	Function
KR	KEY	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

72.14 PWR

Table 38. Correspondence between PWR registers and PWR low-layer driver functions

Register	Field	Function
CR	CSBF	LL_PWR_ClearFlag_SB
	CWUF	LL_PWR_ClearFlag_WU
	DBP	LL_PWR_DisableBkUpAccess
		LL_PWR_EnableBkUpAccess
		LL_PWR_IsEnabledBkUpAccess
	LPDS	LL_PWR_GetPowerMode
		LL_PWR_GetRegulModeDS
		LL_PWR_SetPowerMode
		LL_PWR_SetRegulModeDS
	PDDS	LL_PWR_GetPowerMode
		LL_PWR_SetPowerMode
	PLS	LL_PWR_GetPVDLevel
		LL_PWR_SetPVDLevel
	PVDE	LL_PWR_DisablePVD
		LL_PWR_EnablePVD
		LL_PWR_IsEnabledPVD
CSR	EWUP1	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP2	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin

Register	Field	Function
CSR	EWUP3	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP4	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP5	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP6	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP7	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP8	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	PVDO	LL_PWR_IsActiveFlag_PVDO
	SBF	LL_PWR_IsActiveFlag_SB
	VREFINTRDYF	LL_PWR_IsActiveFlag_VREFINTRDY
	WUF	LL_PWR_IsActiveFlag_WU

72.15 RCC

Table 39. Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
BDCR	BDRST	LL_RCC_ForceBackupDomainReset
		LL_RCC_ReleaseBackupDomainReset
	LSEBYP	LL_RCC_LSE_DisableBypass
		LL_RCC_LSE_EnableBypass
	LSEDRV	LL_RCC_LSE_GetDriveCapability
		LL_RCC_LSE_SetDriveCapability
	LSEON	LL_RCC_LSE_Disable
		LL_RCC_LSE_Enable
	LSERDY	LL_RCC_LSE_IsReady
	RTCEN	LL_RCC_DisableRTC
		LL_RCC_EnableRTC
		LL_RCC_IsEnabledRTC
	RTCSEL	LL_RCC_GetRTCClockSource

Register	Field	Function
BDCR	RTCSEL	LL_RCC_SetRTCClockSource
CFGR	HPRE	LL_RCC_GetAHBPrescaler
		LL_RCC_SetAHBPrescaler
	MCO	LL_RCC_ConfigMCO
	MCOPRE	LL_RCC_ConfigMCO
	PLLMUL	LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetMultiplicator
	PLLNODIV	LL_RCC_ConfigMCO
	PLLSRC	LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetMainSource
CFG2	PPRE	LL_RCC_GetAPB1Prescaler
		LL_RCC_SetAPB1Prescaler
	SW	LL_RCC_SetSysClkSource
	SWS	LL_RCC_GetSysClkSource
	PREDIV	LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetPrediv
CFG3	CECSW	LL_RCC_GetCECClockSource
		LL_RCC_SetCECClockSource
	I2C1SW	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	USART1SW	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART2SW	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART3SW	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
CIR	CSSC	LL_RCC_ClearFlag_HSECSS
	CSSF	LL_RCC_IsActiveFlag_HSECSS
	HSERDYC	LL_RCC_ClearFlag_HSERDY
	HSERDYF	LL_RCC_IsActiveFlag_HSERDY
	HSERDYIE	LL_RCC_DisableIT_HSERDY
		LL_RCC_EnableIT_HSERDY
		LL_RCC_IsEnabledIT_HSERDY
	HSI14RDYC	LL_RCC_ClearFlag_HSI14RDY
	HSI14RDYF	LL_RCC_IsActiveFlag_HSI14RDY
	HSI14RDYIE	LL_RCC_DisableIT_HSI14RDY
		LL_RCC_EnableIT_HSI14RDY
		LL_RCC_IsEnabledIT_HSI14RDY
	HSI48RDYC	LL_RCC_ClearFlag_HSI48RDY
	HSI48RDYF	LL_RCC_IsActiveFlag_HSI48RDY

Register	Field	Function
CIR	HSI48RDYIE	LL_RCC_DisableIT_HSI48RDY
		LL_RCC_EnableIT_HSI48RDY
		LL_RCC_IsEnabledIT_HSI48RDY
	HSIRDYC	LL_RCC_ClearFlag_HSIRDY
	HSIRDYF	LL_RCC_IsActiveFlag_HSIRDY
	HSIRDYIE	LL_RCC_DisableIT_HSIRDY
		LL_RCC_EnableIT_HSIRDY
		LL_RCC_IsEnabledIT_HSIRDY
	LSERDYC	LL_RCC_ClearFlag_LSERDY
	LSERDYF	LL_RCC_IsActiveFlag_LSERDY
	LSERDYIE	LL_RCC_DisableIT_LSERDY
		LL_RCC_EnableIT_LSERDY
		LL_RCC_IsEnabledIT_LSERDY
	LSIRDYC	LL_RCC_ClearFlag_LSIRDY
	LSIRDYF	LL_RCC_IsActiveFlag_LSIRDY
	LSIRDYIE	LL_RCC_DisableIT_LSIRDY
		LL_RCC_EnableIT_LSIRDY
		LL_RCC_IsEnabledIT_LSIRDY
CR	PLLRDYC	LL_RCC_ClearFlag_PLLRDY
	PLLRDYF	LL_RCC_IsActiveFlag_PLLRDY
	PLLRDYIE	LL_RCC_DisableIT_PLLRDY
		LL_RCC_EnableIT_PLLRDY
		LL_RCC_IsEnabledIT_PLLRDY
	CSSON	LL_RCC_HSE_DisableCSS
		LL_RCC_HSE_EnableCSS
	HSEBYP	LL_RCC_HSE_DisableBypass
		LL_RCC_HSE_EnableBypass
	HSEON	LL_RCC_HSE_Disable
		LL_RCC_HSE_Enable
	HSERDY	LL_RCC_HSE_IsReady
	HSICAL	LL_RCC_HSI_GetCalibration
	HSION	LL_RCC_HSI_Disable
		LL_RCC_HSI_Enable
	HSIRDY	LL_RCC_HSI_IsReady
	HSITRIM	LL_RCC_HSI_GetCalibTrimming
		LL_RCC_HSI_SetCalibTrimming
	PLLON	LL_RCC_PLL_Disable
		LL_RCC_PLL_Enable
	PLLRDY	LL_RCC_PLL_IsReady
CR2	HSI14CAL	LL_RCC_HSI14_GetCalibration

Register	Field	Function
CR2	HSI14DIS	LL_RCC_HSI14_DisableADCControl
		LL_RCC_HSI14_EnableADCControl
	HSI14ON	LL_RCC_HSI14_Disable
		LL_RCC_HSI14_Enable
	HSI14RDY	LL_RCC_HSI14_IsReady
	HSI14TRIM	LL_RCC_HSI14_GetCalibTrimming
		LL_RCC_HSI14_SetCalibTrimming
	HSI48CAL	LL_RCC_HSI48_GetCalibration
	HSI48ON	LL_RCC_HSI48_Disable
		LL_RCC_HSI48_Enable
	HSI48RDY	LL_RCC_HSI48_IsReady
CSR	IWDGRSTF	LL_RCC_IsActiveFlag_IWDGRST
	LPWRRSTF	LL_RCC_IsActiveFlag_LPWRST
	LSION	LL_RCC_LSI_Disable
		LL_RCC_LSI_Enable
	LSIRDY	LL_RCC_LSI_IsReady
	OBLRSTF	LL_RCC_IsActiveFlag_OBLRST
	PINRSTF	LL_RCC_IsActiveFlag_PINRST
	PORRSTF	LL_RCC_IsActiveFlag_PORRST
	RMVF	LL_RCC_ClearResetFlags
	SFTRSTF	LL_RCC_IsActiveFlag_SFTRST
	V18PWRRSTF	LL_RCC_IsActiveFlag_V18PWRRST
	WWDGRSTF	LL_RCC_IsActiveFlag_WWDGRST

72.16 RTC

Table 40. 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

Register	Field	Function
ALRMAR	HU	LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	MNT	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MNU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MSK1	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	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

Register	Field	Function
CALR	CALP	LL_RTC_CAL_SetPulse
	CALW16	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	CALW8	LL_RTC_CAL_GetPeriod
		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
	REFCKON	LL_RTC_DisableRefClock
		LL_RTC_EnableRefClock
	SUB1H	LL_RTC_TIME_DecHour
	TSE	LL_RTC_TS_Disable
		LL_RTC_TS_Enable
	TSEDGE	LL_RTC_TS_GetActiveEdge
		LL_RTC_TS_SetActiveEdge
	TSIE	LL_RTC_DisableIT_TS
		LL_RTC_EnableIT_TS
		LL_RTC_IsEnabledIT_TS
	WUCKSEL	LL_RTC_WAKEUP_GetClock
		LL_RTC_WAKEUP_SetClock
	WUTE	LL_RTC_WAKEUP_Disable

Register	Field	Function
CR	WUTE	LL_RTC_WAKEUP_Enable
		LL_RTC_WAKEUP_IsEnabled
	WUTIE	LL_RTC_DisableIT_WUT
		LL_RTC_EnableIT_WUT
		LL_RTC_IsEnabledIT_WUT
	DT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	DU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	MT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
DR	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

Register	Field	Function
ISR	RECALPF	<code>LL_RTC_IsActiveFlag_RECALP</code>
	RSF	<code>LL_RTC_ClearFlag_RS</code>
		<code>LL_RTC_IsActiveFlag_RS</code>
	SHPF	<code>LL_RTC_IsActiveFlag_SHP</code>
	TAMP1F	<code>LL_RTC_ClearFlag_TAMP1</code>
		<code>LL_RTC_IsActiveFlag_TAMP1</code>
	TAMP2F	<code>LL_RTC_ClearFlag_TAMP2</code>
		<code>LL_RTC_IsActiveFlag_TAMP2</code>
	TAMP3F	<code>LL_RTC_ClearFlag_TAMP3</code>
		<code>LL_RTC_IsActiveFlag_TAMP3</code>
	TSF	<code>LL_RTC_ClearFlag_TS</code>
		<code>LL_RTC_IsActiveFlag_TS</code>
	TSOVF	<code>LL_RTC_ClearFlag_TSOV</code>
		<code>LL_RTC_IsActiveFlag_TSOV</code>
	WUTF	<code>LL_RTC_ClearFlag_WUT</code>
		<code>LL_RTC_IsActiveFlag_WUT</code>
	WUTWF	<code>LL_RTC_IsActiveFlag_WUTW</code>
PRER	PREDIV_A	<code>LL_RTC_GetAsynchPrescaler</code>
		<code>LL_RTC_SetAsynchPrescaler</code>
	PREDIV_S	<code>LL_RTC_GetSynchPrescaler</code>
		<code>LL_RTC_SetSynchPrescaler</code>
SHIFTR	ADD1S	<code>LL_RTC_TIME_Synchronize</code>
	SUBFS	<code>LL_RTC_TIME_Synchronize</code>
SSR	SS	<code>LL_RTC_TIME_GetSubSecond</code>
TAFCR	ALARMOUTTYPE	<code>LL_RTC_GetAlarmOutputType</code>
		<code>LL_RTC_SetAlarmOutputType</code>
	PC13MODE	<code>LL_RTC_DisablePushPullMode</code>
		<code>LL_RTC_EnablePushPullMode</code>
	PC14MODE	<code>LL_RTC_DisablePushPullMode</code>
		<code>LL_RTC_EnablePushPullMode</code>
	PC14VALUE	<code>LL_RTC_ResetOutputPin</code>
		<code>LL_RTC_SetOutputPin</code>
	PC15MODE	<code>LL_RTC_DisablePushPullMode</code>
		<code>LL_RTC_EnablePushPullMode</code>
	PC15VALUE	<code>LL_RTC_ResetOutputPin</code>
		<code>LL_RTC_SetOutputPin</code>
	TAMP1E	<code>LL_RTC_TAMPER_Disable</code>
		<code>LL_RTC_TAMPER_Enable</code>
	TAMP1TRG	<code>LL_RTC_TAMPER_DisableActiveLevel</code>
		<code>LL_RTC_TAMPER_EnableActiveLevel</code>

Register	Field	Function
TAFCR	TAMP2E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP2TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMP3E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP3TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMPFLT	LL_RTC_TAMPER_GetFilterCount
		LL_RTC_TAMPER_SetFilterCount
	TAMPFREQ	LL_RTC_TAMPER_GetSamplingFreq
		LL_RTC_TAMPER_SetSamplingFreq
	TAMPIE	LL_RTC_DisableIT_TAMP
		LL_RTC_EnableIT_TAMP
		LL_RTC_IsEnabledIT_TAMP
TR	HT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	HU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	MNT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	MNU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	PM	LL_RTC_TIME_Config
		LL_RTC_TIME_GetFormat
		LL_RTC_TIME_SetFormat

Register	Field	Function
TR	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
TSDR	DT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	DU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	MT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	MU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	WDU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetWeekDay
TSSSR	SS	LL_RTC_TS_GetSubSecond
TSTR	HT	LL_RTC_TS_GetHour
		LL_RTC_TS_GetTime
	HU	LL_RTC_TS_GetHour
		LL_RTC_TS_GetTime
	MNT	LL_RTC_TS_GetMinute
		LL_RTC_TS_GetTime
	MNU	LL_RTC_TS_GetMinute
		LL_RTC_TS_GetTime
	PM	LL_RTC_TS_GetTimeFormat
	ST	LL_RTC_TS_GetSecond
		LL_RTC_TS_GetTime
WPR	KEY	LL_RTC_DisableWriteProtection
		LL_RTC_EnableWriteProtection
WUTR	WUT	LL_RTC_WAKEUP_GetAutoReload
		LL_RTC_WAKEUP_SetAutoReload

72.17 SPI

Table 41. Correspondence between SPI registers and SPI low-layer driver functions

Register	Field	Function
CR1	BIDIMODE	<code>LL_SPI_GetTransferDirection</code>
		<code>LL_SPI_SetTransferDirection</code>
	BIDIOE	<code>LL_SPI_GetTransferDirection</code>
		<code>LL_SPI_SetTransferDirection</code>
	BR	<code>LL_SPI_GetBaudRatePrescaler</code>
		<code>LL_SPI_SetBaudRatePrescaler</code>
	CPHA	<code>LL_SPI_GetClockPhase</code>
		<code>LL_SPI_SetClockPhase</code>
	CPOL	<code>LL_SPI_GetClockPolarity</code>
		<code>LL_SPI_SetClockPolarity</code>
	CRCEN	<code>LL_SPI_DisableCRC</code>
		<code>LL_SPI_EnableCRC</code>
		<code>LL_SPI_IsEnabledCRC</code>
	CRCL	<code>LL_SPI_GetCRCWidth</code>
		<code>LL_SPI_SetCRCWidth</code>
	CRCNEXT	<code>LL_SPI_SetCRCNext</code>
	LSBFIRST	<code>LL_SPI_GetTransferBitOrder</code>
		<code>LL_SPI_SetTransferBitOrder</code>
	MSTR	<code>LL_SPI_GetMode</code>
		<code>LL_SPI_SetMode</code>
	RXONLY	<code>LL_SPI_GetTransferDirection</code>
		<code>LL_SPI_SetTransferDirection</code>
	SPE	<code>LL_SPI_Disable</code>
		<code>LL_SPI_Enable</code>
		<code>LL_SPI_IsEnabled</code>
	SSI	<code>LL_SPI_GetMode</code>
		<code>LL_SPI_SetMode</code>
	SSM	<code>LL_SPI_GetNSSMode</code>
		<code>LL_SPI_SetNSSMode</code>
CR2	DS	<code>LL_SPI_GetDataWidth</code>
		<code>LL_SPI_SetDataWidth</code>
	ERRIE	<code>LL_SPI_DisableIT_ERR</code>
		<code>LL_SPI_EnableIT_ERR</code>
		<code>LL_SPI_IsEnabledIT_ERR</code>
	FRF	<code>LL_SPI_GetStandard</code>
		<code>LL_SPI_SetStandard</code>
	FRXTH	<code>LL_SPI_GetRxFIFOThreshold</code>

Register	Field	Function
CR2	FRXTH	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
		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
CRCPR		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

Register	Field	Function
SR	OVR	LL_SPI_IsActiveFlag_OVR
	RXNE	LL_SPI_IsActiveFlag_RXNE
	TXE	LL_SPI_IsActiveFlag_TXE
TXCRCR	TXCRC	LL_SPI_GetTxCRC

72.18 SYSTEM

Table 42. Correspondence between SYSTEM registers and SYSTEM low-layer driver functions

Register	Field	Function
DBGMCU_APB1FZ	DBG_CAN_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_I2C1_SMBUS_TIMEOUT	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_IWDG_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_RTC_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM14_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM2_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM3_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM6_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM7_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_WWDG_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
DBGMCU_APB2FZ	DBG_TIM15_STOP	LL_DBGMCU_APB1_GRP2_FreezePeriph
		LL_DBGMCU_APB1_GRP2_UnFreezePeriph
	DBG_TIM16_STOP	LL_DBGMCU_APB1_GRP2_FreezePeriph
		LL_DBGMCU_APB1_GRP2_UnFreezePeriph
DBGMCU_CR	DBG_TIM17_STOP	LL_DBGMCU_APB1_GRP2_FreezePeriph
		LL_DBGMCU_APB1_GRP2_UnFreezePeriph
	DBG_TIM1_STOP	LL_DBGMCU_APB1_GRP2_FreezePeriph
		LL_DBGMCU_APB1_GRP2_UnFreezePeriph
	DBG_STANDBY	LL_DBGMCU_DisableDBGStandbyMode
		LL_DBGMCU_EnableDBGStandbyMode
	DBG_STOP	LL_DBGMCU_DisableDBGStopMode

Register	Field	Function
DBGMCU_CR	DBG_STOP	LL_DBGMCU_EnableDBGStopMode
DBGMCU_IDCODE	DEV_ID	LL_DBGMCU_GetDeviceID
	REV_ID	LL_DBGMCU_GetRevisionID
FLASH_ACR	LATENCY	LL_FLASH_GetLatency LL_FLASH_SetLatency
	PRFTBE	LL_FLASH_DisablePrefetch LL_FLASH_EnablePrefetch
	PRFTBS	LL_FLASH_IsPrefetchEnabled
SYSCFG_CFGR1	I2C_FMP_I2C1	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	I2C_FMP_I2C2	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	I2C_FMP_PA10	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	I2C_FMP_PA9	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	I2C_FMP_PB6	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	I2C_FMP_PB7	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	I2C_FMP_PB8	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	I2C_FMP_PB9	LL_SYSCFG_DisableFastModePlus LL_SYSCFG_EnableFastModePlus
	IR_MOD	LL_SYSCFG_GetIRModEnvelopeSignal LL_SYSCFG_SetIRModEnvelopeSignal
	MEM_MODE	LL_SYSCFG_GetRemapMemory LL_SYSCFG_SetRemapMemory
	LOCKUP_LOCK	LL_SYSCFG_GetTIMBreakInputs LL_SYSCFG_SetTIMBreakInputs
	PVD_LOCK	LL_SYSCFG_GetTIMBreakInputs LL_SYSCFG_SetTIMBreakInputs
SYSCFG_CFGR2	SRAM_PARITY_LOCK	LL_SYSCFG_GetTIMBreakInputs LL_SYSCFG_SetTIMBreakInputs
	SRAM_PEF	LL_SYSCFG_ClearFlag_SP LL_SYSCFG_IsActiveFlag_SP
	EXTI0	LL_SYSCFG_SetEXTISource
	EXTI1	LL_SYSCFG_SetEXTISource
SYSCFG_EXTICR1	EXTI2	LL_SYSCFG_SetEXTISource
	EXTI3	LL_SYSCFG_SetEXTISource

Register	Field	Function
SYSCFG_EXTICR2	EXTI4	LL_SYSCFG_SetEXTISource
	EXTI5	LL_SYSCFG_SetEXTISource
	EXTI6	LL_SYSCFG_SetEXTISource
	EXTI7	LL_SYSCFG_SetEXTISource
SYSCFG_EXTICR3	EXTI10	LL_SYSCFG_SetEXTISource
	EXTI11	LL_SYSCFG_SetEXTISource
	EXTI18	LL_SYSCFG_SetEXTISource
	EXTI9	LL_SYSCFG_SetEXTISource
SYSCFG_EXTICR4	EXTI12	LL_SYSCFG_SetEXTISource
	EXTI13	LL_SYSCFG_SetEXTISource
	EXTI14	LL_SYSCFG_SetEXTISource
	EXTI15	LL_SYSCFG_SetEXTISource
SYSCFG_ITLINE0	SR_EWDG	LL_SYSCFG_IsActiveFlag_WWDG
SYSCFG_ITLINE1	SR_PVDOUT	LL_SYSCFG_IsActiveFlag_PVDOUT
	SR_VDDIO2	LL_SYSCFG_IsActiveFlag_VDDIO2
SYSCFG_ITLINE10	SR_DMA1_CH2	LL_SYSCFG_IsActiveFlag_DMA1_CH2
	SR_DMA1_CH3	LL_SYSCFG_IsActiveFlag_DMA1_CH3
	SR_DMA2_CH1	LL_SYSCFG_IsActiveFlag_DMA2_CH1
	SR_DMA2_CH2	LL_SYSCFG_IsActiveFlag_DMA2_CH2
SYSCFG_ITLINE11	SR_DMA1_CH4	LL_SYSCFG_IsActiveFlag_DMA1_CH4
	SR_DMA1_CH5	LL_SYSCFG_IsActiveFlag_DMA1_CH5
	SR_DMA1_CH6	LL_SYSCFG_IsActiveFlag_DMA1_CH6
	SR_DMA1_CH7	LL_SYSCFG_IsActiveFlag_DMA1_CH7
	SR_DMA2_CH3	LL_SYSCFG_IsActiveFlag_DMA2_CH3
	SR_DMA2_CH4	LL_SYSCFG_IsActiveFlag_DMA2_CH4
	SR_DMA2_CH5	LL_SYSCFG_IsActiveFlag_DMA2_CH5
SYSCFG_ITLINE12	SR_ADC	LL_SYSCFG_IsActiveFlag_ADC
	SR_COMP1	LL_SYSCFG_IsActiveFlag_COMP1
	SR_COMP2	LL_SYSCFG_IsActiveFlag_COMP2
SYSCFG_ITLINE13	SR_TIM1_BRK	LL_SYSCFG_IsActiveFlag_TIM1_BRK
	SR_TIM1_CCU	LL_SYSCFG_IsActiveFlag_TIM1_CCU
	SR_TIM1_TRG	LL_SYSCFG_IsActiveFlag_TIM1_TRG
	SR_TIM1_UPD	LL_SYSCFG_IsActiveFlag_TIM1_UPD
SYSCFG_ITLINE14	SR_TIM1_CC	LL_SYSCFG_IsActiveFlag_TIM1_CC
SYSCFG_ITLINE15	SR_TIM2_GLB	LL_SYSCFG_IsActiveFlag_TIM2
SYSCFG_ITLINE16	SR_TIM3_GLB	LL_SYSCFG_IsActiveFlag_TIM3
SYSCFG_ITLINE17	SR_DAC	LL_SYSCFG_IsActiveFlag_DAC
	SR_TIM6_GLB	LL_SYSCFG_IsActiveFlag_TIM6
SYSCFG_ITLINE18	SR_TIM7_GLB	LL_SYSCFG_IsActiveFlag_TIM7
SYSCFG_ITLINE19	SR_TIM14_GLB	LL_SYSCFG_IsActiveFlag_TIM14

Register	Field	Function
SYSCFG_ITLINE2	SR_RTC_ALRA	LL_SYSCFG_IsActiveFlag_RTC_ALRA
	SR_RTC_TSTAMP	LL_SYSCFG_IsActiveFlag_RTC_TSTAMP
	SR_RTC_WAKEUP	LL_SYSCFG_IsActiveFlag_RTC_WAKEUP
SYSCFG_ITLINE20	SR_TIM15_GLB	LL_SYSCFG_IsActiveFlag_TIM15
SYSCFG_ITLINE21	SR_TIM16_GLB	LL_SYSCFG_IsActiveFlag_TIM16
SYSCFG_ITLINE22	SR_TIM17_GLB	LL_SYSCFG_IsActiveFlag_TIM17
SYSCFG_ITLINE23	SR_I2C1_GLB	LL_SYSCFG_IsActiveFlag_I2C1
SYSCFG_ITLINE24	SR_I2C2_GLB	LL_SYSCFG_IsActiveFlag_I2C2
SYSCFG_ITLINE25	SR_SPI1	LL_SYSCFG_IsActiveFlag_SPI1
SYSCFG_ITLINE26	SR_SPI2	LL_SYSCFG_IsActiveFlag_SPI2
SYSCFG_ITLINE27	SR_USART1_GLB	LL_SYSCFG_IsActiveFlag_USART1
SYSCFG_ITLINE28	SR_USART2_GLB	LL_SYSCFG_IsActiveFlag_USART2
SYSCFG_ITLINE29	SR_USART3_GLB	LL_SYSCFG_IsActiveFlag_USART3
	SR_USART4_GLB	LL_SYSCFG_IsActiveFlag_USART4
	SR_USART5_GLB	LL_SYSCFG_IsActiveFlag_USART5
	SR_USART6_GLB	LL_SYSCFG_IsActiveFlag_USART6
	SR_USART7_GLB	LL_SYSCFG_IsActiveFlag_USART7
	SR_USART8_GLB	LL_SYSCFG_IsActiveFlag_USART8
SYSCFG_ITLINE3	SR_FLASH_ITF	LL_SYSCFG_IsActiveFlag_FLASH_ITF
SYSCFG_ITLINE30	SR_CAN	LL_SYSCFG_IsActiveFlag_CAN
	SR_CEC	LL_SYSCFG_IsActiveFlag_CEC
SYSCFG_ITLINE4	SR_CLK_CTRL	LL_SYSCFG_IsActiveFlag_CLK_CTRL
	SR_CRS	LL_SYSCFG_IsActiveFlag_CRS
SYSCFG_ITLINE5	SR EXTI0	LL_SYSCFG_IsActiveFlag_EXTI0
	SR EXTI1	LL_SYSCFG_IsActiveFlag_EXTI1
SYSCFG_ITLINE6	SR EXTI2	LL_SYSCFG_IsActiveFlag_EXTI2
	SR EXTI3	LL_SYSCFG_IsActiveFlag_EXTI3
SYSCFG_ITLINE7	SR EXTI10	LL_SYSCFG_IsActiveFlag_EXTI10
	SR EXTI11	LL_SYSCFG_IsActiveFlag_EXTI11
	SR EXTI12	LL_SYSCFG_IsActiveFlag_EXTI12
	SR EXTI13	LL_SYSCFG_IsActiveFlag_EXTI13
	SR EXTI14	LL_SYSCFG_IsActiveFlag_EXTI14
	SR EXTI15	LL_SYSCFG_IsActiveFlag_EXTI15
	SR EXTI4	LL_SYSCFG_IsActiveFlag_EXTI4
	SR EXTI5	LL_SYSCFG_IsActiveFlag_EXTI5
	SR EXTI6	LL_SYSCFG_IsActiveFlag_EXTI6
	SR EXTI7	LL_SYSCFG_IsActiveFlag_EXTI7
SYSCFG_ITLINE8	SR EXTI8	LL_SYSCFG_IsActiveFlag_EXTI8
	SR EXTI9	LL_SYSCFG_IsActiveFlag_EXTI9
SYSCFG_ITLINE8	SR_TSC_EOA	LL_SYSCFG_IsActiveFlag_TSC_EOA

Register	Field	Function
SYSCFG_ITLINE8	SR_TSC_MCE	LL_SYSCFG_IsActiveFlag_TSC_MCE
SYSCFG_ITLINE9	SR_DMA1_CH1	LL_SYSCFG_IsActiveFlag_DMA1_CH1

72.19 TIM

Table 43. Correspondence between TIM registers and TIM low-layer driver functions

Register	Field	Function
ARR	ARR	LL_TIM_SetAutoReload
		LL_TIM_SetAutoReload
BDTR	AOE	LL_TIM_DisableAutomaticOutput
		LL_TIM_EnableAutomaticOutput
		LL_TIM_IsEnabledAutomaticOutput
	BKE	LL_TIM_DisableBRK
		LL_TIM_EnableBRK
	BKP	LL_TIM_ConfigBRK
	DTG	LL_TIM_OC_SetDeadTime
	LOCK	LL_TIM_CC_SetLockLevel
	MOE	LL_TIM_DisableAllOutputs
		LL_TIM_EnableAllOutputs
		LL_TIM_IsEnabledAllOutputs
CCER	OSSI	LL_TIM_SetOffStates
	OSSR	LL_TIM_SetOffStates
	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
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2E	LL_TIM_CC_DisableChannel

Register	Field	Function
CCER	CC2E	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
	CC2P	LL_TIM_OC_SetPolarity
		LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
	CC3E	LL_TIM_OC_SetPolarity
		LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
CCER	CC3NE	LL_TIM_CC_IsEnabledChannel
		LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
	CC3NP	LL_TIM_CC_IsEnabledChannel
		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
	CC4NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
	CC4P	LL_TIM_IC_Config

Register	Field	Function
CCER	CC4P	LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	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
CCMR1	IC2PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC1CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC1FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC1M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC1PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC2CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC2FE	LL_TIM_OC_DisableFast

Register	Field	Function
CCMR1	OC2FE	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
		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

Register	Field	Function
CCMR2	OC4CE	LL_TIM_OC_IsEnabledClear
		LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC4M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
		LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
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
CR1	CNT	LL_TIM_GetCounter
		LL_TIM_SetCounter
	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

Register	Field	Function
CR1	UDIS	<code>LL_TIM_IsEnabledUpdateEvent</code>
	URS	<code>LL_TIM_GetUpdateSource</code> <code>LL_TIM_SetUpdateSource</code>
CR2	CCDS	<code>LL_TIM_CC_GetDMAReqTrigger</code> <code>LL_TIM_CC_SetDMAReqTrigger</code>
	CCPC	<code>LL_TIM_CC_DisablePreload</code> <code>LL_TIM_CC_EnablePreload</code>
	CCUS	<code>LL_TIM_CC_SetUpdate</code>
	MMS	<code>LL_TIM_SetTriggerOutput</code>
	OIS1	<code>LL_TIM_OC_ConfigOutput</code> <code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
		<code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
		<code>LL_TIM_OC_ConfigOutput</code> <code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
	OIS2	<code>LL_TIM_OC_ConfigOutput</code> <code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
	OIS2N	<code>LL_TIM_OC_ConfigOutput</code> <code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
	OIS3	<code>LL_TIM_OC_ConfigOutput</code> <code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
		<code>LL_TIM_OC_ConfigOutput</code> <code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
		<code>LL_TIM_OC_ConfigOutput</code> <code>LL_TIM_OC_GetIdleState</code> <code>LL_TIM_OC_SetIdleState</code>
DCR	DBA	<code>LL_TIM_ConfigDMAburst</code>
	DBL	<code>LL_TIM_ConfigDMAburst</code>
	DIER	<code>LL_TIM_DisableIT_BRK</code> <code>LL_TIM_EnableIT_BRK</code> <code>LL_TIM_IsEnabledIT_BRK</code>
DIER	CC1DE	<code>LL_TIM_DisableDMAReq_CC1</code> <code>LL_TIM_EnableDMAReq_CC1</code> <code>LL_TIM_IsEnabledDMAReq_CC1</code>
		<code>LL_TIM_DisableIT_CC1</code> <code>LL_TIM_EnableIT_CC1</code>

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
	COMDE	LL_TIM_DisableDMAReq_COM
		LL_TIM_EnableDMAReq_COM
		LL_TIM_IsEnabledDMAReq_COM
	COMIE	LL_TIM_DisableIT_COM
		LL_TIM_EnableIT_COM
		LL_TIM_IsEnabledIT_COM
	TDE	LL_TIM_DisableDMAReq_TRIG
		LL_TIM_EnableDMAReq_TRIG
		LL_TIM_IsEnabledDMAReq_TRIG
DIER	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

Register	Field	Function
EGR	CC3G	<code>LL_TIM_GenerateEvent_CC3</code>
	CC4G	<code>LL_TIM_GenerateEvent_CC4</code>
	COMG	<code>LL_TIM_GenerateEvent_COM</code>
	TG	<code>LL_TIM_GenerateEvent_TRIG</code>
	UG	<code>LL_TIM_GenerateEvent_UPDATE</code>
PSC	PSC	<code>LL_TIM_GetPrescaler</code>
		<code>LL_TIM_SetPrescaler</code>
RCR	REP	<code>LL_TIM_GetRepetitionCounter</code>
		<code>LL_TIM_SetRepetitionCounter</code>
SMCR	ECE	<code>LL_TIM_DisableExternalClock</code>
		<code>LL_TIM_EnableExternalClock</code>
		<code>LL_TIM_IsEnabledExternalClock</code>
		<code>LL_TIM_SetClockSource</code>
	ETF	<code>LL_TIM_ConfigETR</code>
	ETP	<code>LL_TIM_ConfigETR</code>
	ETPS	<code>LL_TIM_ConfigETR</code>
	MSM	<code>LL_TIM_DisableMasterSlaveMode</code>
		<code>LL_TIM_EnableMasterSlaveMode</code>
		<code>LL_TIM_IsEnabledMasterSlaveMode</code>
	OCCS	<code>LL_TIM_SetOCRefClearInputSource</code>
	SMS	<code>LL_TIM_SetClockSource</code>
		<code>LL_TIM_SetEncoderMode</code>
		<code>LL_TIM_SetSlaveMode</code>
	TS	<code>LL_TIM_SetTriggerInput</code>
SR	BIF	<code>LL_TIM_ClearFlag_BRK</code>
		<code>LL_TIM_IsActiveFlag_BRK</code>
	CC1IF	<code>LL_TIM_ClearFlag_CC1</code>
		<code>LL_TIM_IsActiveFlag_CC1</code>
	CC1OF	<code>LL_TIM_ClearFlag_CC1OVR</code>
		<code>LL_TIM_IsActiveFlag_CC1OVR</code>
	CC2IF	<code>LL_TIM_ClearFlag_CC2</code>
		<code>LL_TIM_IsActiveFlag_CC2</code>
	CC2OF	<code>LL_TIM_ClearFlag_CC2OVR</code>
		<code>LL_TIM_IsActiveFlag_CC2OVR</code>
	CC3IF	<code>LL_TIM_ClearFlag_CC3</code>
		<code>LL_TIM_IsActiveFlag_CC3</code>
	CC3OF	<code>LL_TIM_ClearFlag_CC3OVR</code>
		<code>LL_TIM_IsActiveFlag_CC3OVR</code>
	CC4IF	<code>LL_TIM_ClearFlag_CC4</code>
		<code>LL_TIM_IsActiveFlag_CC4</code>

Register	Field	Function
SR	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
TIM14_OR	TI1_RMP	LL_TIM_SetRemap

72.20 USART

Table 44. Correspondence between USART registers and USART low-layer driver functions

Register	Field	Function
BRR	BRR	LL_USART_GetBaudRate
		LL_USART_SetBaudRate
	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
CR1	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

Register	Field	Function
CR1	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
	RE	LL_USART_DisableDirectionRx
		LL_USART_EnableDirectionRx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
CR1	RTOIE	LL_USART_DisableIT_RTO
		LL_USART_EnableIT_RTO
		LL_USART_IsEnabledIT_RTO
	RXNEIE	LL_USART_DisableIT_RXNE
		LL_USART_EnableIT_RXNE
		LL_USART_IsEnabledIT_RXNE
	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
CR1	TXEIE	LL_USART_DisableIT_TXE
		LL_USART_EnableIT_TXE
		LL_USART_IsEnabledIT_TXE
	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

Register	Field	Function
CR2	ABRMODE	LL_USART_GetAutoBaudRateMode
		LL_USART_SetAutoBaudRateMode
	ADD	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddress
	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
CR2	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

Register	Field	Function
CR2	LINEN	LL_USART_EnableLIN
		LL_USART_IsEnabledLIN
	MSBFIRST	LL_USART_GetTransferBitOrder
		LL_USART_SetTransferBitOrder
	RTOEN	LL_USART_DisableRxTimeout
		LL_USART_EnableRxTimeout
		LL_USART_IsEnabledRxTimeout
	RXINV	LL_USART_GetRXPinLevel
		LL_USART_SetRXPinLevel
	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
CR3	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
	SCARCNT	LL_USART_GetSmartcardAutoRetryCount
		LL_USART_SetSmartcardAutoRetryCount
	SCEN	LL_USART_ConfigAsyncMode

Register	Field	Function
CR3	SCEN	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
	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
	NCF	LL_USART_ClearFlag_NE
	ORECF	LL_USART_ClearFlag_ORE
	PECF	LL_USART_ClearFlag_PE
	RTOCF	LL_USART_ClearFlag_RTO
	TCCF	LL_USART_ClearFlag_TC
	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

Register	Field	Function
ISR	IDLE	LL_USART_IsActiveFlag_IDLE
	LBDF	LL_USART_IsActiveFlag_LBD
	NF	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
	RXNE	LL_USART_IsActiveFlag_RXNE
	SBKF	LL_USART_IsActiveFlag_SBK
	TC	LL_USART_IsActiveFlag_TC
	TEACK	LL_USART_IsActiveFlag_TEACK
	TXE	LL_USART_IsActiveFlag_TXE
	WUF	LL_USART_IsActiveFlag_WKUP
RDR	RDR	LL_USART_DMA_GetRegAddr
		LL_USART_ReceiveData8
		LL_USART_ReceiveData9
RQR	ABRRQ	LL_USART_RequestAutoBaudRate
	MMRQ	LL_USART_RequestEnterMuteMode
	RXFRQ	LL_USART_RequestRxDataFlush
	SBKRQ	LL_USART_RequestBreakSending
	TXFRQ	LL_USART_RequestTxDataFlush
RTOR	BLEN	LL_USART_GetBlockLength
		LL_USART_SetBlockLength
	RTO	LL_USART_GetRxTimeout
		LL_USART_SetRxTimeout
TDR	TDR	LL_USART_DMA_GetRegAddr
		LL_USART_TransmitData8
		LL_USART_TransmitData9

72.21 WWDG

Table 45. Correspondence between WWDG registers and WWDG low-layer driver functions

Register	Field	Function
CFR	EWI	LL_WWDG_EnableIT_EWKUP
		LL_WWDG_IsEnabledIT_EWKUP
	W	LL_WWDG_GetWindow
		LL_WWDG_SetWindow
	WDGTB	LL_WWDG_GetPrescaler
		LL_WWDG_SetPrescaler

Register	Field	Function
CR	T	LL_WWDG_GetCounter
		LL_WWDG_SetCounter
	WDGA	LL_WWDG_Enable
		LL_WWDG_IsEnabled
SR	EWIF	LL_WWDG_ClearFlag_EWKUP
		LL_WWDG_IsActiveFlag_EWKUP

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 STM32F0 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: stm32f0xx_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 (stm32f0xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32f0xx_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 (stm32f0xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines

- The extension APIs (stm32f0xx_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_stm32f0xx.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_MspDelInit()** 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 stm32f0xx_hal_msp.c. A template is provided in the HAL driver folders (stm32f0xx_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() instm32f0xx_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 stm32f0xx_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 stm32f0xx_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 stm32f0xx_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 46. Document revision history

Date	Revision	Changes
15-Oct-2014	1	Initial release.
09-Nov-2015	2	Macros and functions renamed for all APIs. Updated common macros in HAL common resources. Added LSE_STARTUP_TIMEOUT in Table 1
06-Jun-2016	3	Changed GPIO_SPEED_LOW, GPIO_SPEED_MEDIUM, GPIO_SPEED_HIGH into GPIO_SPEED_FREQ_LOW, GPIO_SPEED_FREQ_MEDIUM, GPIO_SPEED_FREQ_HIGH in GPIOs. Added low-layer driver APIs. HAL CRC <ul style="list-style-type: none">Updated devices supporting Programmable Polynomial features: defines and functions prototypes are available only for STM32F071xB, STM32F072xB, STM32F078xx, STM32F091xC, STM32F098x devices.Updated HAL_CRC_Delnit() function (restored IDR Register to Reset value). HAL DMA <ul style="list-style-type: none">Added __HAL_DMA_GET_COUNTER() macro returning the number of remaining data units in the current DMA Channel transfer.Provided new function HAL_DMA_Abort_IT() to abort current DMA transfer under interrupt mode without polling for DMA enable bit.Added __HAL_DMA_GET_GI_FLAG_INDEX macro returning current DMA Channel Global interrupt flag HAL FLASH <ul style="list-style-type: none">Suppress function FLASH_PageErase – become internal HAL I2C <ul style="list-style-type: none">Added support of repeated start feature in case of multimaster mode (allow master to keep I2C communication with slave). HAL PWR <ul style="list-style-type: none">Aligned EWUPx pins and PWR functions with CMSIS definitions. HAL RCC <ul style="list-style-type: none">Added missing HAL IRQHandler and callbacks API for CRS management.Added missing RCC_CFGR_PLLNODIV definition for STM32F030x4/6 devices.Removed HSI48State from structure RCC_OsclInitTypeDef when device does not support HSI48.Removed RCC_HSI48_OFF.Removed flag RCC_FLAG_RMV which is write only.Renamed RCC_CRS_SYNCWARM to RCC_CRS_SYNCWARN.Renamed RCC_CRS_TRIMOV to RCC_CRS_TRIMOVF. HAL SMARTCARD <ul style="list-style-type: none">Aligned SMARTCARD Stop Bits with others STM32 series. HAL TIM <ul style="list-style-type: none">Updated HAL_TIM_ConfigOCrefClear() function to allow TIM_CLEARINPUTSOURCE_OCREFCLR as new ClearInputSource. HAL UART-USART <ul style="list-style-type: none">Aligned UART-USART Stop Bits with others STM32 series.
08-Nov-2016	4	Update for release V1.5.0 of the HAL and LL drivers. Refer to the release note provided within the firmware package for details.

Date	Revision	Changes
04-May-2017	5	Update for release V1.6.0 of the HAL and LL drivers. Refer to the release note provided within the firmware package for details.
xx-Jul-2017	6	Update for release V1.7.0 of the HAL and LL drivers. Refer to the release note provided within the firmware package for details.
03-Feb-2020	7	Minor update of Section Introduction . Added Section 1 General information . List of acronyms made generic in Section 2 Acronyms and definitions . Updated Section 73 FAQs .

Contents

1	General information	3
2	Acronyms and definitions	4
3	Overview of HAL drivers	7
3.1	HAL and user-application files	7
3.1.1	HAL driver files	7
3.1.2	User-application files	8
3.2	HAL data structures	9
3.2.1	Peripheral handle structures	9
3.2.2	Initialization and configuration structure	10
3.2.3	Specific process structures	11
3.3	API classification	12
3.4	Devices supported by HAL drivers	13
3.5	HAL driver rules	16
3.5.1	HAL API naming rules	16
3.5.2	HAL general naming rules	17
3.5.3	HAL interrupt handler and callback functions	18
3.6	HAL generic APIs	19
3.7	HAL extension APIs	20
3.7.1	HAL extension model overview	20
3.7.2	HAL extension model cases	20
3.8	File inclusion model	23
3.9	HAL common resources	23
3.10	HAL configuration	24
3.11	HAL system peripheral handling	25
3.11.1	Clock	25
3.11.2	GPIOs	26
3.11.3	Cortex® NVIC and SysTick timer	27
3.11.4	PWR	27
3.11.5	EXTI	28
3.11.6	DMA	29

3.12	How to use HAL drivers	30
3.12.1	HAL usage models	30
3.12.2	HAL initialization	30
3.12.3	HAL I/O operation process	32
3.12.4	Timeout and error management	35
4	Overview of low-layer drivers	39
4.1	Low-layer files	39
4.2	Overview of low-layer APIs and naming rules	41
4.2.1	Peripheral initialization functions	41
4.2.2	Peripheral register-level configuration functions	43
5	Cohabiting of HAL and LL	45
5.1	Low-layer driver used in Standalone mode	45
5.2	Mixed use of low-layer APIs and HAL drivers	45
6	HAL System Driver	46
6.1	HAL Firmware driver API description	46
6.1.1	How to use this driver	46
6.1.2	Initialization and de-initialization functions	46
6.1.3	HAL Control functions	46
6.1.4	Detailed description of functions	47
6.2	HAL Firmware driver defines	51
6.2.1	HAL	51
7	HAL ADC Generic Driver	59
7.1	ADC Firmware driver registers structures	59
7.1.1	ADC_InitTypeDef	59
7.1.2	ADC_ChannelConfTypeDef	60
7.1.3	ADC_AnalogWDGConfTypeDef	61
7.1.4	ADC_HandleTypeDef	62
7.2	ADC Firmware driver API description	62
7.2.1	ADC peripheral features	62
7.2.2	How to use this driver	62
7.2.3	Initialization and de-initialization functions	64

7.2.4	IO operation functions	65
7.2.5	Peripheral Control functions	65
7.2.6	Peripheral State and Errors functions	65
7.2.7	Detailed description of functions	65
7.3	ADC Firmware driver defines	71
7.3.1	ADC	71
8	HAL ADC Extension Driver	80
8.1	ADCEx Firmware driver API description	80
8.1.1	IO operation functions	80
8.1.2	Detailed description of functions	80
8.2	ADCEx Firmware driver defines	80
8.2.1	ADCEx	80
9	HAL CAN Generic Driver	81
9.1	CAN Firmware driver registers structures	81
9.1.1	CAN_InitTypeDef	81
9.1.2	CAN_FilterConfTypeDef	81
9.1.3	CanTxMsgTypeDef	82
9.1.4	CanRxMsgTypeDef	83
9.1.5	CAN_HandleTypeDef	84
9.2	CAN Firmware driver API description	84
9.2.1	How to use this driver	84
9.2.2	Initialization and de-initialization functions	85
9.2.3	Peripheral State and Error functions	85
9.2.4	Detailed description of functions	85
9.3	CAN Firmware driver defines	89
9.3.1	CAN	89
10	HAL CEC Generic Driver	98
10.1	CEC Firmware driver registers structures	98
10.1.1	CEC_InitTypeDef	98
10.1.2	CEC_HandleTypeDef	99
10.2	CEC Firmware driver API description	99

10.2.1	How to use this driver	100
10.2.2	Initialization and Configuration functions	100
10.2.3	IO operation functions	100
10.2.4	Peripheral Control function	100
10.2.5	Detailed description of functions	100
10.3	CEC Firmware driver defines	104
10.3.1	CEC	104
11	HAL COMP Generic Driver	113
11.1	COMP Firmware driver registers structures	113
11.1.1	COMP_InitTypeDef	113
11.1.2	COMP_HandleTypeDef	113
11.2	COMP Firmware driver API description	114
11.2.1	COMP Peripheral features	114
11.2.2	How to use this driver	114
11.2.3	Initialization and Configuration functions	115
11.2.4	IO operation functions	115
11.2.5	Peripheral Control functions	115
11.2.6	Peripheral State functions	115
11.2.7	Detailed description of functions	115
11.3	COMP Firmware driver defines	118
11.3.1	COMP	118
12	HAL CORTEX Generic Driver	126
12.1	CORTEX Firmware driver API description	126
12.1.1	How to use this driver	126
12.1.2	Initialization and de-initialization functions	126
12.1.3	Peripheral Control functions	127
12.1.4	Detailed description of functions	127
12.2	CORTEX Firmware driver defines	130
12.2.1	CORTEX	130
13	HAL CRC Generic Driver	131
13.1	CRC Firmware driver registers structures	131
13.1.1	CRC_InitTypeDef	131

13.1.2	CRC_HandleTypeDef	131
13.2	CRC Firmware driver API description	132
13.2.1	How to use this driver	132
13.2.2	Initialization and de-initialization functions	132
13.2.3	Peripheral Control functions	132
13.2.4	Peripheral State functions	133
13.2.5	Detailed description of functions	133
13.3	CRC Firmware driver defines	135
13.3.1	CRC	135
14	HAL CRC Extension Driver.....	138
14.1	CRCEx Firmware driver API description	138
14.1.1	How to use this driver	138
14.1.2	Initialization and Configuration functions	138
14.1.3	Detailed description of functions	138
14.2	CRCEx Firmware driver defines	139
14.2.1	CRCEx	139
15	HAL DAC Generic Driver.....	142
15.1	DAC Firmware driver registers structures	142
15.1.1	DAC_HandleTypeDef	142
15.1.2	DAC_ChannelConfTypeDef	142
15.2	DAC Firmware driver API description	142
15.2.1	DAC Peripheral features	142
15.2.2	How to use this driver	144
15.2.3	Initialization and de-initialization functions	144
15.2.4	IO operation functions	144
15.2.5	Peripheral Control functions	145
15.2.6	Peripheral State and Errors functions	145
15.2.7	Detailed description of functions	145
15.3	DAC Firmware driver defines	150
15.3.1	DAC	150
16	HAL DAC Extension Driver.....	154

16.1	DACEEx Firmware driver API description	154
16.1.1	How to use this driver	154
16.1.2	Extended features functions	154
16.1.3	Detailed description of functions	154
16.2	DACEEx Firmware driver defines	158
16.2.1	DACEEx	158
17	HAL DMA Generic Driver	160
17.1	DMA Firmware driver registers structures	160
17.1.1	DMA_InitTypeDef	160
17.1.2	__DMA_HandleTypeDef	160
17.2	DMA Firmware driver API description	161
17.2.1	How to use this driver	161
17.2.2	Initialization and de-initialization functions	162
17.2.3	IO operation functions	162
17.2.4	State and Errors functions	162
17.2.5	Detailed description of functions	162
17.3	DMA Firmware driver defines	166
17.3.1	DMA	166
18	HAL DMA Extension Driver	171
18.1	DMAEEx Firmware driver defines	171
18.1.1	DMAEEx	171
19	HAL FLASH Generic Driver	182
19.1	FLASH Firmware driver registers structures	182
19.1.1	FLASH_ProcessTypeDef	182
19.2	FLASH Firmware driver API description	182
19.2.1	FLASH peripheral features	182
19.2.2	How to use this driver	182
19.2.3	Peripheral Control functions	183
19.2.4	Peripheral Errors functions	183
19.2.5	Detailed description of functions	183
19.3	FLASH Firmware driver defines	186

19.3.1	FLASH	186
20	HAL FLASH Extension Driver	189
20.1	FLASHEx Firmware driver registers structures	189
20.1.1	FLASH_EraseInitTypeDef	189
20.1.2	FLASH_OBProgramInitTypeDef	189
20.2	FLASHEx Firmware driver API description	190
20.2.1	FLASH Erasing Programming functions	190
20.2.2	Option Bytes Programming functions	190
20.2.3	Detailed description of functions	190
20.3	FLASHEx Firmware driver defines	192
20.3.1	FLASHEx	192
21	HAL GPIO Generic Driver	196
21.1	GPIO Firmware driver registers structures	196
21.1.1	GPIO_InitTypeDef	196
21.2	GPIO Firmware driver API description	196
21.2.1	GPIO Peripheral features	196
21.2.2	How to use this driver	197
21.2.3	Initialization and de-initialization functions	197
21.2.4	IO operation functions	197
21.2.5	Detailed description of functions	197
21.3	GPIO Firmware driver defines	199
21.3.1	GPIO	199
22	HAL GPIO Extension Driver	203
22.1	GPIOEx Firmware driver defines	203
22.1.1	GPIOEx	203
23	HAL I2C Generic Driver	207
23.1	I2C Firmware driver registers structures	207
23.1.1	I2C_InitTypeDef	207
23.1.2	__I2C_HandleTypeDef	207
23.2	I2C Firmware driver API description	208
23.2.1	How to use this driver	208

23.2.2	Initialization and de-initialization functions	212
23.2.3	IO operation functions	213
23.2.4	Peripheral State, Mode and Error functions	214
23.2.5	Detailed description of functions	214
23.3	I2C Firmware driver defines	227
23.3.1	I2C	227
24	HAL I2C Extension Driver	234
24.1	I2CEx Firmware driver API description	234
24.1.1	I2C peripheral Extended features	234
24.1.2	How to use this driver	234
24.1.3	Extended features functions	234
24.1.4	Detailed description of functions	234
24.2	I2CEx Firmware driver defines	236
24.2.1	I2CEx	236
25	HAL I2S Generic Driver	238
25.1	I2S Firmware driver registers structures	238
25.1.1	I2S_InitTypeDef	238
25.1.2	I2S_HandleTypeDef	238
25.2	I2S Firmware driver API description	239
25.2.1	How to use this driver	239
25.2.2	Initialization and de-initialization functions	241
25.2.3	IO operation functions	241
25.2.4	Peripheral State and Errors functions	242
25.2.5	Detailed description of functions	242
25.3	I2S Firmware driver defines	248
25.3.1	I2S	248
26	HAL IRDA Generic Driver	253
26.1	IRDA Firmware driver registers structures	253
26.1.1	IRDA_InitTypeDef	253
26.1.2	IRDA_HandleTypeDef	253
26.2	IRDA Firmware driver API description	254

26.2.1	How to use this driver	254
26.2.2	Initialization and Configuration functions	256
26.2.3	IO operation functions	256
26.2.4	Peripheral State and Error functions	258
26.2.5	Detailed description of functions	258
26.3	IRDA Firmware driver defines	266
26.3.1	IRDA	266
27	HAL IRDA Extension Driver	275
27.1	IRDAEx Firmware driver defines	275
27.1.1	IRDAEx	275
28	HAL IWDG Generic Driver	276
28.1	IWDG Firmware driver registers structures	276
28.1.1	IWDG_InitTypeDef	276
28.1.2	IWDG_HandleTypeDef	276
28.2	IWDG Firmware driver API description	276
28.2.1	IWDG Generic features	276
28.2.2	How to use this driver	277
28.2.3	Initialization and Start functions	277
28.2.4	IO operation functions	277
28.2.5	Detailed description of functions	277
28.3	IWDG Firmware driver defines	278
28.3.1	IWDG	278
29	HAL PCD Generic Driver	280
29.1	PCD Firmware driver registers structures	280
29.1.1	PCD_InitTypeDef	280
29.1.2	PCD_EPTTypeDef	280
29.1.3	PCD_HandleTypeDef	281
29.2	PCD Firmware driver API description	282
29.2.1	How to use this driver	282
29.2.2	Initialization and de-initialization functions	282
29.2.3	IO operation functions	282
29.2.4	Peripheral Control functions	283

29.2.5	Peripheral State functions	283
29.2.6	Detailed description of functions	283
29.3	PCD Firmware driver defines	290
29.3.1	PCD	290
30	HAL PCD Extension Driver	293
30.1	PCDEx Firmware driver API description	293
30.1.1	Extended Peripheral Control functions	293
30.1.2	Detailed description of functions	293
31	HAL PWR Generic Driver	294
31.1	PWR Firmware driver API description	294
31.1.1	Initialization and de-initialization functions	294
31.1.2	Peripheral Control functions	294
31.1.3	Detailed description of functions	296
31.2	PWR Firmware driver defines	299
31.2.1	PWR	299
32	HAL PWR Extension Driver	301
32.1	PWREx Firmware driver registers structures	301
32.1.1	PWR_PVDTTypeDef	301
32.2	PWREx Firmware driver API description	301
32.2.1	Peripheral extended control functions	301
32.2.2	Detailed description of functions	301
32.3	PWREx Firmware driver defines	303
32.3.1	PWREx	303
33	HAL RCC Generic Driver	308
33.1	RCC Firmware driver registers structures	308
33.1.1	RCC_PLLInitTypeDef	308
33.1.2	RCC_OscInitTypeDef	308
33.1.3	RCC_ClkInitTypeDef	309
33.2	RCC Firmware driver API description	309
33.2.1	RCC specific features	309
33.2.2	RCC Limitations	310

33.2.3	Initialization and de-initialization functions	310
33.2.4	Peripheral Control functions	311
33.2.5	Detailed description of functions	311
33.3	RCC Firmware driver defines	315
33.3.1	RCC	315
34	HAL RCC Extension Driver	338
34.1	RCCEEx Firmware driver registers structures	338
34.1.1	RCC_PeriphCLKInitTypeDef	338
34.1.2	RCC_CRSInitTypeDef	338
34.1.3	RCC_CRSSynchroInfoTypeDef	339
34.2	RCCEEx Firmware driver API description	339
34.2.1	Extended Peripheral Control functions	339
34.2.2	Extended Clock Recovery System Control functions	339
34.2.3	Detailed description of functions	340
34.3	RCCEEx Firmware driver defines	343
34.3.1	RCCEEx	343
35	HAL RTC Generic Driver	360
35.1	RTC Firmware driver registers structures	360
35.1.1	RTC_InitTypeDef	360
35.1.2	RTC_TimeTypeDef	360
35.1.3	RTC_DateTypeDef	361
35.1.4	RTC_AlarmTypeDef	361
35.1.5	RTC_HandleTypeDef	362
35.2	RTC Firmware driver API description	362
35.2.1	How to use RTC Driver	362
35.2.2	RTC and low power modes	363
35.2.3	Initialization and de-initialization functions	363
35.2.4	RTC Time and Date functions	363
35.2.5	RTC Alarm functions	363
35.2.6	Peripheral Control functions	364
35.2.7	Peripheral State functions	364
35.2.8	Detailed description of functions	364

35.3	RTC Firmware driver defines	369
35.3.1	RTC	369
36	HAL RTC Extension Driver.....	379
36.1	RTCEEx Firmware driver registers structures	379
36.1.1	RTC_TamperTypeDef	379
36.2	RTCEEx Firmware driver API description.....	379
36.2.1	How to use this driver	379
36.2.2	RTC TimeStamp and Tamper functions.....	380
36.2.3	RTC Wake-up functions	380
36.2.4	Extended Peripheral Control functions	380
36.2.5	Detailed description of functions	381
36.3	RTCEEx Firmware driver defines	389
36.3.1	RTCEEx	389
37	HAL SMARTCARD Generic Driver.....	404
37.1	SMARTCARD Firmware driver registers structures	404
37.1.1	SMARTCARD_InitTypeDef	404
37.1.2	SMARTCARD_AdvFeatureInitTypeDef	405
37.1.3	SMARTCARD_HandleTypeDef.....	406
37.2	SMARTCARD Firmware driver API description.....	407
37.2.1	How to use this driver	407
37.2.2	Initialization and Configuration functions	408
37.2.3	IO operation functions	409
37.2.4	Peripheral State and Errors functions	410
37.2.5	Detailed description of functions	411
37.3	SMARTCARD Firmware driver defines	417
37.3.1	SMARTCARD	418
38	HAL SMARTCARD Extension Driver.....	429
38.1	SMARTCARDEEx Firmware driver API description	429
38.1.1	SMARTCARD peripheral extended features	429
38.1.2	Peripheral Control functions	429
38.1.3	Detailed description of functions	429

39 HAL SMBUS Generic Driver.....	431
39.1 SMBUS Firmware driver registers structures.....	431
39.1.1 SMBUS_InitTypeDef.....	431
39.1.2 SMBUS_HandleTypeDef.....	432
39.2 SMBUS Firmware driver API description	432
39.2.1 How to use this driver	432
39.2.2 Initialization and de-initialization functions.....	434
39.2.3 IO operation functions	434
39.2.4 Peripheral State and Errors functions	435
39.2.5 Detailed description of functions	435
39.3 SMBUS Firmware driver defines.....	442
39.3.1 SMBUS.....	442
40 HAL SPI Generic Driver	450
40.1 SPI Firmware driver registers structures	450
40.1.1 SPI_InitTypeDef	450
40.1.2 __SPI_HandleTypeDef	451
40.2 SPI Firmware driver API description	452
40.2.1 How to use this driver	452
40.2.2 Initialization and de-initialization functions.....	453
40.2.3 IO operation functions	453
40.2.4 Peripheral State and Errors functions	454
40.2.5 Detailed description of functions	454
40.3 SPI Firmware driver defines.....	462
40.3.1 SPI	462
41 HAL SPI Extension Driver.....	469
41.1 SPIEx Firmware driver API description.....	469
41.1.1 IO operation functions	469
41.1.2 Detailed description of functions	469
42 HAL TIM Generic Driver	470
42.1 TIM Firmware driver registers structures	470
42.1.1 TIM_Base_InitTypeDef	470

42.1.2	TIM_OC_InitTypeDef	470
42.1.3	TIM_OnePulse_InitTypeDef	471
42.1.4	TIM_IC_InitTypeDef	472
42.1.5	TIM_Encoder_InitTypeDef	472
42.1.6	TIM_ClockConfigTypeDef	473
42.1.7	TIM_ClearInputConfigTypeDef	473
42.1.8	TIM_SlaveConfigTypeDef	474
42.1.9	TIM_HandleTypeDef	474
42.2	TIM Firmware driver API description	475
42.2.1	TIMER Generic features	475
42.2.2	How to use this driver	475
42.2.3	Time Base functions	476
42.2.4	Time Output Compare functions	476
42.2.5	Time PWM functions	477
42.2.6	Time Input Capture functions	477
42.2.7	Time One Pulse functions	477
42.2.8	Time Encoder functions	478
42.2.9	IRQ handler management	478
42.2.10	Peripheral Control functions	478
42.2.11	TIM Callbacks functions	479
42.2.12	Peripheral State functions	479
42.2.13	Detailed description of functions	479
42.3	TIM Firmware driver defines	508
42.3.1	TIM	508
43	HAL TIM Extension Driver	530
43.1	TIMEx Firmware driver registers structures	530
43.1.1	TIM_HallSensor_InitTypeDef	530
43.1.2	TIM_MasterConfigTypeDef	530
43.1.3	TIM_BreakDeadTimeConfigTypeDef	530
43.2	TIMEx Firmware driver API description	531
43.2.1	TIMER Extended features	531
43.2.2	How to use this driver	531

43.2.3	Timer Hall Sensor functions	532
43.2.4	Timer Complementary Output Compare functions.	532
43.2.5	Timer Complementary PWM functions	532
43.2.6	Timer Complementary One Pulse functions	533
43.2.7	Peripheral Control functions	533
43.2.8	Extension Callbacks functions.	534
43.2.9	Extension Peripheral State functions.	534
43.2.10	Detailed description of functions	534
43.3	TIMEx Firmware driver defines	544
43.3.1	TIMEx	544
44	HAL TSC Generic Driver	546
44.1	TSC Firmware driver registers structures.	546
44.1.1	TSC_InitTypeDef.	546
44.1.2	TSC_IOConfigTypeDef	547
44.1.3	TSC_HandleTypeDef	547
44.2	TSC Firmware driver API description	547
44.2.1	TSC specific features	547
44.2.2	How to use this driver	548
44.2.3	Initialization and de-initialization functions.	548
44.2.4	Peripheral Control functions	548
44.2.5	State functions	548
44.2.6	Detailed description of functions	549
44.3	TSC Firmware driver defines	552
44.3.1	TSC	552
45	HAL UART Generic Driver	564
45.1	UART Firmware driver registers structures	564
45.1.1	UART_InitTypeDef	564
45.1.2	UART_AdvFeatureInitTypeDef	564
45.1.3	UART_HandleTypeDef	565
45.2	UART Firmware driver API description	566
45.2.1	How to use this driver	566
45.2.2	Initialization and Configuration functions	568

45.2.3	IO operation functions	569
45.2.4	Peripheral Control functions	569
45.2.5	Peripheral State and Error functions	570
45.2.6	Detailed description of functions	570
45.3	UART Firmware driver defines	581
45.3.1	UART	581
46	HAL UART Extension Driver	596
46.1	UARTEx Firmware driver registers structures	596
46.1.1	UART_WakeUpTypeDef	596
46.2	UARTEx Firmware driver API description	596
46.2.1	UART peripheral extended features	596
46.2.2	Initialization and Configuration functions	596
46.2.3	IO operation function	597
46.2.4	Peripheral Control functions	597
46.2.5	Detailed description of functions	597
46.3	UARTEx Firmware driver defines	599
46.3.1	UARTEx	599
47	HAL USART Generic Driver	601
47.1	USART Firmware driver registers structures	601
47.1.1	USART_InitTypeDef	601
47.1.2	USART_HandleTypeDef	601
47.2	USART Firmware driver API description	602
47.2.1	How to use this driver	602
47.2.2	Initialization and Configuration functions	604
47.2.3	IO operation functions	604
47.2.4	Peripheral State and Error functions	606
47.2.5	Detailed description of functions	606
47.3	USART Firmware driver defines	614
47.3.1	USART	614
48	HAL USART Extension Driver	622
48.1	USARTEx Firmware driver defines	622

48.1.1	USARTEx	622
49	HAL WWDG Generic Driver	623
49.1	WWDG Firmware driver registers structures	623
49.1.1	WWDG_InitTypeDef	623
49.1.2	WWDG_HandleTypeDef	623
49.2	WWDG Firmware driver API description	623
49.2.1	WWDG specific features	623
49.2.2	How to use this driver	624
49.2.3	Initialization and Configuration functions	624
49.2.4	IO operation functions	624
49.2.5	Detailed description of functions	625
49.3	WWDG Firmware driver defines	626
49.3.1	WWDG	626
50	LL ADC Generic Driver	629
50.1	ADC Firmware driver registers structures	629
50.1.1	LL_ADC_InitTypeDef	629
50.1.2	LL_ADC_REG_InitTypeDef	629
50.2	ADC Firmware driver API description	630
50.2.1	Detailed description of functions	630
50.3	ADC Firmware driver defines	670
50.3.1	ADC	670
51	LL BUS Generic Driver	692
51.1	BUS Firmware driver API description	692
51.1.1	Detailed description of functions	692
51.2	BUS Firmware driver defines	705
51.2.1	BUS	705
52	LL COMP Generic Driver	709
52.1	COMP Firmware driver registers structures	709
52.1.1	LL_COMP_InitTypeDef	709
52.2	COMP Firmware driver API description	709
52.2.1	Detailed description of functions	709

52.3	COMP Firmware driver defines	718
52.3.1	COMP	718
53	LL CORTEX Generic Driver	722
53.1	CORTEX Firmware driver API description	722
53.1.1	Detailed description of functions	722
53.2	CORTEX Firmware driver defines	725
53.2.1	CORTEX	726
54	LL CRC Generic Driver	727
54.1	CRC Firmware driver API description	727
54.1.1	Detailed description of functions	727
54.2	CRC Firmware driver defines	734
54.2.1	CRC	734
55	LL CRS Generic Driver	736
55.1	CRS Firmware driver API description	736
55.1.1	Detailed description of functions	736
55.2	CRS Firmware driver defines	747
55.2.1	CRS	747
56	LL DAC Generic Driver	750
56.1	DAC Firmware driver registers structures	750
56.1.1	LL_DAC_InitTypeDef	750
56.2	DAC Firmware driver API description	750
56.2.1	Detailed description of functions	750
56.3	DAC Firmware driver defines	768
56.3.1	DAC	768
57	LL DMA Generic Driver	773
57.1	DMA Firmware driver registers structures	773
57.1.1	LL_DMA_InitTypeDef	773
57.2	DMA Firmware driver API description	774
57.2.1	Detailed description of functions	774
57.3	DMA Firmware driver defines	811
57.3.1	DMA	811

58	LL EXTI Generic Driver	818
58.1	EXTI Firmware driver registers structures	818
58.1.1	LL_EXTI_InitTypeDef	818
58.2	EXTI Firmware driver API description.....	818
58.2.1	Detailed description of functions	818
58.3	EXTI Firmware driver defines	835
58.3.1	EXTI	835
59	LL GPIO Generic Driver	838
59.1	GPIO Firmware driver registers structures.....	838
59.1.1	LL_GPIO_InitTypeDef.....	838
59.2	GPIO Firmware driver API description	838
59.2.1	Detailed description of functions	838
59.3	GPIO Firmware driver defines	856
59.3.1	GPIO	856
60	LL I2C Generic Driver	860
60.1	I2C Firmware driver registers structures	860
60.1.1	LL_I2C_InitTypeDef	860
60.2	I2C Firmware driver API description	860
60.2.1	Detailed description of functions	860
60.3	I2C Firmware driver defines	902
60.3.1	I2C	902
61	LL I2S Generic Driver	908
61.1	I2S Firmware driver registers structures.....	908
61.1.1	LL_I2S_InitTypeDef.....	908
61.2	I2S Firmware driver API description	908
61.2.1	Detailed description of functions	908
61.3	I2S Firmware driver defines	922
61.3.1	I2S	922
62	LL IWDG Generic Driver	925
62.1	IWDG Firmware driver API description	925
62.1.1	Detailed description of functions	925

62.2	IWDG Firmware driver defines	929
62.2.1	IWDG	929
63	LL PWR Generic Driver	931
63.1	PWR Firmware driver API description	931
63.1.1	Detailed description of functions	931
63.2	PWR Firmware driver defines	938
63.2.1	PWR	938
64	LL RCC Generic Driver	941
64.1	RCC Firmware driver registers structures	941
64.1.1	LL_RCC_ClocksTypeDef	941
64.2	RCC Firmware driver API description	941
64.2.1	Detailed description of functions	941
64.3	RCC Firmware driver defines	971
64.3.1	RCC	971
65	LL RTC Generic Driver	982
65.1	RTC Firmware driver registers structures	982
65.1.1	LL_RTC_InitTypeDef	982
65.1.2	LL_RTC_TimeTypeDef	982
65.1.3	LL_RTC_DateTypeDef	982
65.1.4	LL_RTC_AlarmTypeDef	983
65.2	RTC Firmware driver API description	983
65.2.1	Detailed description of functions	983
65.3	RTC Firmware driver defines	1039
65.3.1	RTC	1039
66	LL SPI Generic Driver	1049
66.1	SPI Firmware driver registers structures	1049
66.1.1	LL_SPI_InitTypeDef	1049
66.2	SPI Firmware driver API description	1050
66.2.1	Detailed description of functions	1050
66.3	SPI Firmware driver defines	1073
66.3.1	SPI	1073

67	LL SYSTEM Generic Driver	1078
67.1	SYSTEM Firmware driver API description	1078
67.1.1	Detailed description of functions	1078
67.2	SYSTEM Firmware driver defines	1103
67.2.1	SYSTEM	1103
68	LL TIM Generic Driver.....	1107
68.1	TIM Firmware driver registers structures	1107
68.1.1	LL_TIM_InitTypeDef	1107
68.1.2	LL_TIM_OC_InitTypeDef	1107
68.1.3	LL_TIM_IC_InitTypeDef	1108
68.1.4	LL_TIM_ENCODER_InitTypeDef	1108
68.1.5	LL_TIM_HALLSENSOR_InitTypeDef	1109
68.1.6	LL_TIM_BDTR_InitTypeDef	1110
68.2	TIM Firmware driver API description.....	1111
68.2.1	Detailed description of functions	1111
68.3	TIM Firmware driver defines	1179
68.3.1	TIM	1179
69	LL USART Generic Driver.....	1193
69.1	USART Firmware driver registers structures	1193
69.1.1	LL_USART_InitTypeDef	1193
69.1.2	LL_USART_ClockInitTypeDef	1193
69.2	USART Firmware driver API description	1194
69.2.1	Detailed description of functions	1194
69.3	USART Firmware driver defines	1262
69.3.1	USART	1262
70	LL UTILS Generic Driver	1269
70.1	UTILS Firmware driver registers structures	1269
70.1.1	LL_UTILS_PLLInitTypeDef	1269
70.1.2	LL_UTILS_ClkInitTypeDef	1269
70.2	UTILS Firmware driver API description	1269
70.2.1	System Configuration functions	1269

70.2.2	Detailed description of functions	1269
70.3	UTILS Firmware driver defines	1273
70.3.1	UTILS	1273
71	LL WWDG Generic Driver	1274
71.1	WWDG Firmware driver API description	1274
71.1.1	Detailed description of functions	1274
71.2	WWDG Firmware driver defines	1277
71.2.1	WWDG	1278
72	Correspondence between API registers and API low-layer driver functions	1279
72.1	ADC	1279
72.2	BUS.....	1283
72.3	COMP.....	1288
72.4	CORTEX	1289
72.5	CRC	1290
72.6	CRS	1290
72.7	DAC	1292
72.8	DMA	1294
72.9	EXTI	1297
72.10	GPIO.....	1297
72.11	I2C	1298
72.12	I2S.....	1302
72.13	IWDG	1303
72.14	PWR.....	1304
72.15	RCC	1305
72.16	RTC.....	1308
72.17	SPI	1314
72.18	SYSTEM.....	1317
72.19	TIM	1321
72.20	USART	1329
72.21	WWDG	1335
73	FAQs	1337

Revision history	1340
------------------------	------

List of tables

Table 1. Acronyms and definitions	4
Table 2. HAL driver files	7
Table 3. User-application files	8
Table 4. API classification	12
Table 5. List of devices supported by HAL drivers.	14
Table 6. HAL API naming rules	16
Table 7. Macros handling interrupts and specific clock configurations	18
Table 8. Callback functions	19
Table 9. HAL generic APIs	19
Table 10. HAL extension APIs	20
Table 11. Define statements used for HAL configuration	24
Table 12. Description of GPIO_InitTypeDef structure	26
Table 13. Description of EXTI configuration macros	28
Table 14. MSP functions	32
Table 15. Timeout values	35
Table 16. LL driver files.	39
Table 17. Common peripheral initialization functions.	41
Table 18. Optional peripheral initialization functions	42
Table 19. Specific Interrupt, DMA request and status flags management.	43
Table 20. Available function formats	43
Table 21. Peripheral clock activation/deactivation management	43
Table 22. Peripheral activation/deactivation management	44
Table 23. Peripheral configuration management.	44
Table 24. Peripheral register management	44
Table 25. Correspondence between ADC registers and ADC low-layer driver functions.	1279
Table 26. Correspondence between BUS registers and BUS low-layer driver functions	1283
Table 27. Correspondence between COMP registers and COMP low-layer driver functions	1288
Table 28. Correspondence between CORTEX registers and CORTEX low-layer driver functions	1289
Table 29. Correspondence between CRC registers and CRC low-layer driver functions	1290
Table 30. Correspondence between CRS registers and CRS low-layer driver functions.	1291
Table 31. Correspondence between DAC registers and DAC low-layer driver functions.	1292
Table 32. Correspondence between DMA registers and DMA low-layer driver functions	1294
Table 33. Correspondence between EXTI registers and EXTI low-layer driver functions	1297
Table 34. Correspondence between GPIO registers and GPIO low-layer driver functions	1297
Table 35. Correspondence between I2C registers and I2C low-layer driver functions	1298
Table 36. Correspondence between I2S registers and I2S low-layer driver functions.	1302
Table 37. Correspondence between IWDG registers and IWDG low-layer driver functions.	1303
Table 38. Correspondence between PWR registers and PWR low-layer driver functions	1304
Table 39. Correspondence between RCC registers and RCC low-layer driver functions	1305
Table 40. Correspondence between RTC registers and RTC low-layer driver functions	1308
Table 41. Correspondence between SPI registers and SPI low-layer driver functions	1315
Table 42. Correspondence between SYSTEM registers and SYSTEM low-layer driver functions	1317
Table 43. Correspondence between TIM registers and TIM low-layer driver functions	1321
Table 44. Correspondence between USART registers and USART low-layer driver functions.	1329
Table 45. Correspondence between WWDG registers and WWDG low-layer driver functions.	1335
Table 46. Document revision history	1340

List of figures

Figure 1.	Example of project template	9
Figure 2.	Adding device-specific functions	21
Figure 3.	Adding family-specific functions	21
Figure 4.	Adding new peripherals	22
Figure 5.	Updating existing APIs.	22
Figure 6.	File inclusion model.	23
Figure 7.	HAL driver model	30
Figure 8.	Low-layer driver folders	40
Figure 9.	Low-layer driver CMSIS files	41

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved