UM1749
User Manual

Description of STM32L0 HAL and Low Layer drivers

Introduction

STMCube^{TM} is STMicroelectronics's original initiative to ease developers' life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL0 forSTM32L0 Series)
  - The STM32Cube Hardware Abstraction Layer (HAL), an STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. The HAL is available for all peripherals.
  - The Low Layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS, USB.
  - All embedded software utilities coming with a full set of examples.

The HAL driver layer provides a generic multi-instance simple 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 which simplify the user application implementation. As an 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. As an example, the timer APIs are split into several categories following the IP functions: basic timer, capture, pulse width modulation (PWM), etc.. The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking contributes to enhance 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 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 associated peripheral registers content. Contrary to the HAL, the 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 complex upper level stack (such as USB).

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

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

The source code of HAL and LL drivers is developed in Strict ANSI-C which makes it independent from the development tools. It is checked with CodeSonar™ static analysis tool. It is fully documented and is MISRA-C 2004 compliant.

Contents

List of tables

List of figures

1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
COMP	Comparator
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
LPTIM	Low Power Timer
MSP	MCU Specific Package
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer

Acronym	Definition
TSC	Touch Sensing Controller
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

2 Overview of HAL drivers

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

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

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

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully re-entrant 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.

2.1 HAL and user-application files

2.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2: HAL driver files

File	Description
stm32l0xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32l0xx_hal_adc.c, stm32l0xx_hal_irda.c, …*
stm32l0xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example: stm32l0xx_hal_adc.h, stm32l0xx_hal_irda.h, …*

File	Description
stm32l0xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example: stm32l0xx_hal_adc_ex.c, stm32l0xx_hal_dma_ex.c, …*
stm32l0xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32l0xx_hal_adc_ex.h, stm32l0xx_hal_dma_ex.h, …*
stm32l0xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
stm32l0xx_hal.h	*stm32l0*xx_hal.c header file
stm32l0xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32l0xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32l0xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

2.1.2 User-application files

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

Table 3: User-application files

File	Description
system_stm32l0xx.c	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.
startup_stm32l0xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32l0xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
stm32l0xx_FLASH.ld (optional)	Linker file for SW4STM32 toolchain.
stm32l0xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32l0xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

File	Description
stm32l0xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32l0xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, in particular call to HAL_Init(), assert_failed() implementation, system clock configuration, peripheral HAL initialization and user application code.

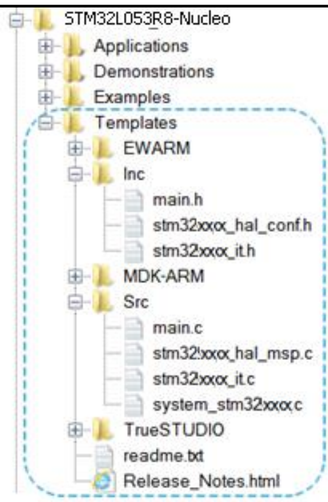
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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 frequency of the device

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

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

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

2.2.1 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 uint32_t ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

1) The multi-instance feature implies that all the APIs used in the application are re-entrant and avoid using global variables because subroutines can fail to be re-entrant 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:

- Re-entrant code does not hold any static (or global) non-constant data: re-entrant functions can work with global data. For example, a re-entrant 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.

2.2.2 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 wether the Receive or Transmit mode is enabled or
disabled.*/
uint32_t HwFlowCtl; /*!< Specifies wether the hardware flow control mode is enabled
or disabled.*/
uint32_t OverSampling; /*!< Specifies wether the Over sampling 8 is enabled or
disabled,
to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```

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)
```

2.2.3 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)
```

2.3 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(STM32L051xx) && !defined(STM32L061xx) void
HAL_RCCEx_CRSConfig(RCC_CRSInitTypeDef *pInit); void
HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void); RCC_CRSStatusTypeDef
HAL_RCCEx_CRSWaitSynchronization(uint32_t Timeout); #endif /* !(STM32L051xx) &&
!(STM32L061xx) */
```

  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

Notes:

(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

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

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

2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_cryp.c	No	Yes	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
stm32l0xx_hal_cryp_ex.c	No	Yes	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
stm32l0xx_hal_dac.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_dac_ex.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_firewall.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash_ramfunc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2s.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_lcd.c	No	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes	No	No	Yes
stm32l0xx_hal_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_pcd.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_pcd_ex.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rng.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_tsc.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.5 HAL driver rules

2.5.1 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	*stm32l0xx_hal_ppp (c/h)*	*stm32l0xx_hal_ppp_ex (c/h)*	*stm32l0xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function HAL_PPP_FeatureFunction _MODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_M ODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_M ODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameType Def*	*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 STM32L0 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in stm32l0xxx.h header file.stm32l0xxx.h corresponds to stm32l051xx.h, stm32l052xx.h, stm32l053xx.h, stm32l061xx.h, stm32l062xx.h or stm32l063xx.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_xxxxConfTypeDef (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 PPP_DeInit, e.g. TIM_DeInit.
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA ().*

- The **Feature** prefix should refer to the new feature.
  Example: *HAL_ADC_Start*() refers to the injection mode

2.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  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 */
  }
  - GPIO
  - SYSTICK
  - NVIC
  - RCC
  - FLASH.
- 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: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7: Macros handling interrupts and specific clock configurations

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

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the stm32l0xx_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)
```

2.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32l0xx_it.c
- User callback functions.

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

There are three types of user callbacks functions:

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

Table 8: Callback functions

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

2.6 HAL generic APIs

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

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

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

Example: in the timer driver, the API grouping is based on timer features (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_DeInit()*function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

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

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

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

Table 9: HAL generic APIs

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

Function group	Common API name	Description
	HAL_ADC_GetError()	This fuction allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 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, stm32l0xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32l0xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

Function Group	Common API Name
HAL_ADCEx_Calibration_Start()	This function is used to start the automatic ADC calibration
HAL_ADCEx_Calibration_GetValue()	This function is used to get the ADC calibration factor
HAL_ADCEx_Calibration_SetValue()	This function is used to set the calibration factor to overwrite automatic conversion result

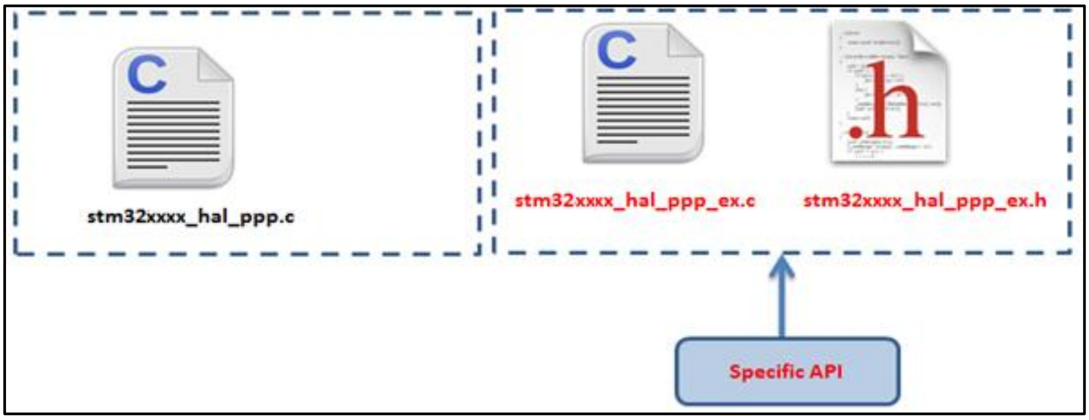
2.7.2 HAL extension model cases

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

Case 1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32l0xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

Figure 2: Adding device-specific functions



Example: stm32l0xx_hal_rcc_ex.c/h

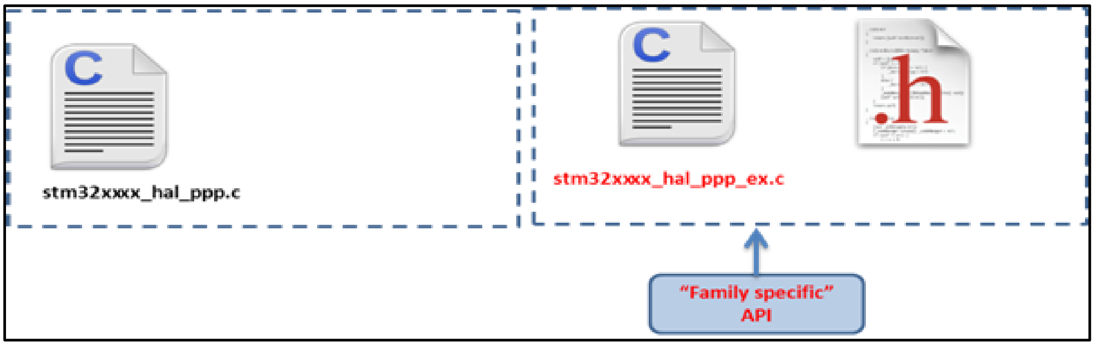
```
#if !defined(STM32L051xx) && !defined(STM32L061xx)
void HAL_RCCEx_CRSConfig(RCC_CRSInitTypeDef *pInit);
void HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void);
```

```
void HAL_RCCEx_CRSGetSynchronizationInfo(RCC_CRSSynchroInfoTypeDef *pSynchroInfo);
RCC_CRSStatusTypeDef HAL_RCCEx_CRSWaitSynchronization(uint32_t Timeout);
#endif /* !(STM32L051xx) && !(STM32L061xx) */
```

Case 2: 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

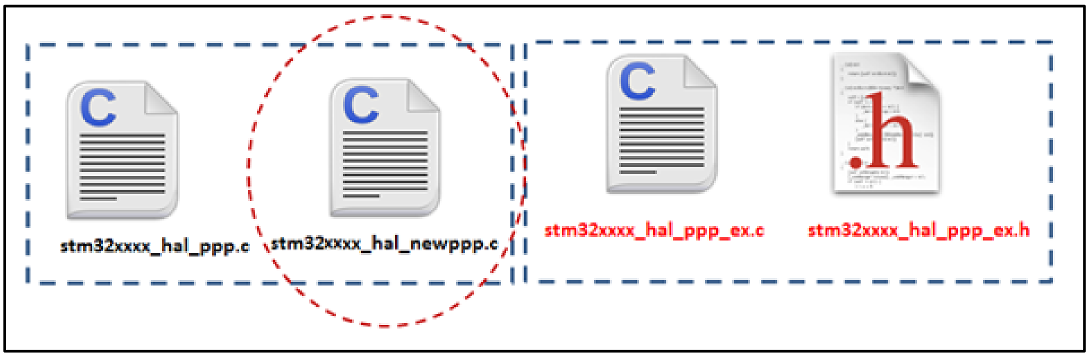


Case 3: Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in stm32l0xx_hal_newppp.c. However the inclusion of this file is selected in the stm32lxx_hal_conf.h using the macro:

#define HAL_NEWPPP_MODULE_ENABLED

Figure 4: Adding new peripherals

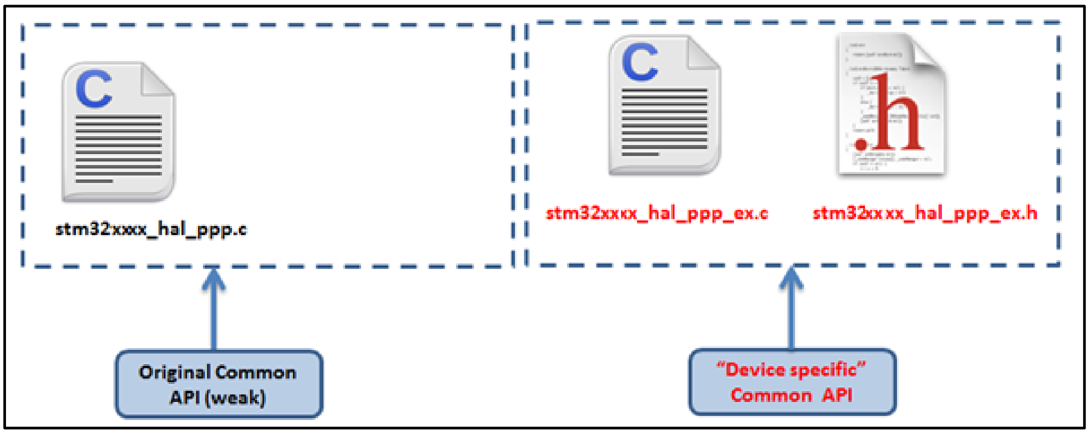


Example:stm32l0xx_hal_lcd.c/h

Case 4: Updating existing common APIs

In this case, the routines are defined with the same names in the stm32l0xx_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



Case 5: Updating existing data structures

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

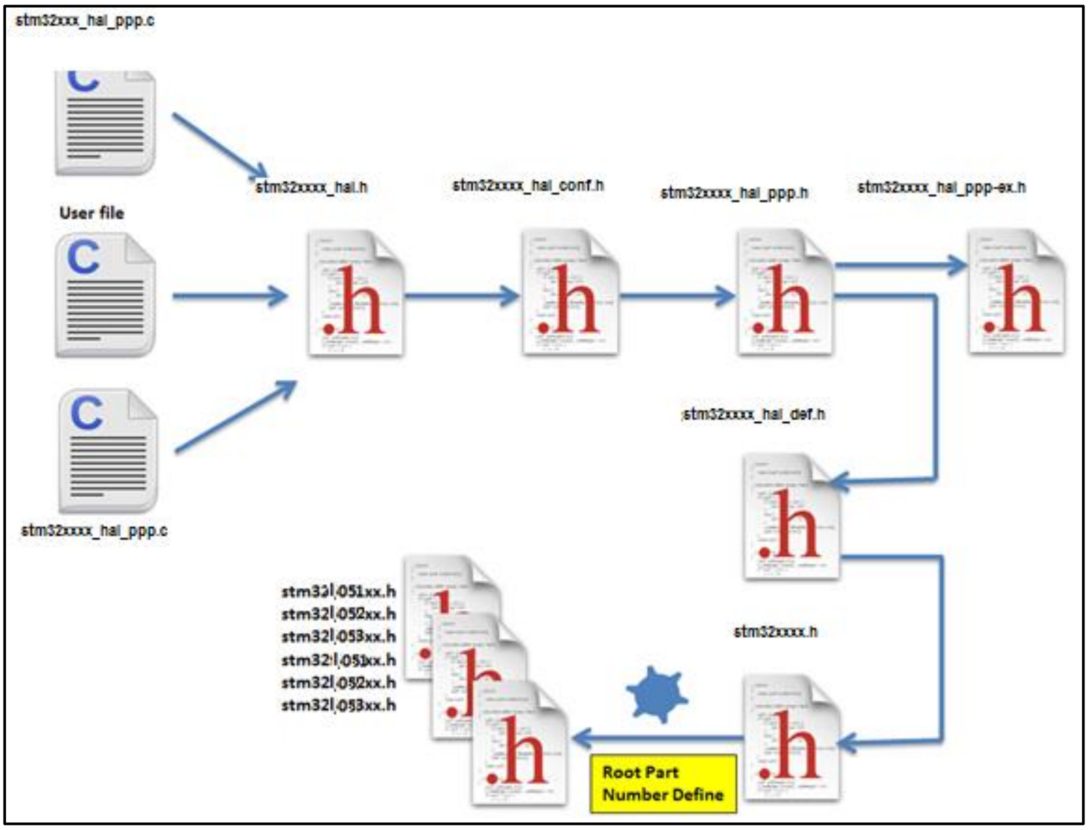
Example:

```
#if defined (STM32L051xx)
typedef struct
{
(…)
}PPP_InitTypeDef;
#endif /* STM32L051xx */
```

2.8 File inclusion model

The header of the common HAL driver file (stm32l0xx_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 stm32l0xx_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
(…)
```

2.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32l0xx_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 stm32l0xx_hal_def.h file calls the stm32l0xx.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 HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

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

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

2.10 HAL configuration

The configuration file, *stm32l0xx_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)
MSI_VALUE	Defines the Internal Multiple Speed oscillator (MSI) value expressed in Hz.	2 000 000 (Hz)
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE
BUFFER_CACHE_ENABLE	Enables buffer cache	FALSE

The stm32l0xx_hal_conf_template.h file is located in the HAL drivers *Inc* folder. It should be copied to the user folder, renamed and modified as described above.

By default, the values defined in the stm32l0xx_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.

2.11 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.

2.11.1 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, APB1 and APB2 clock dividers
  - configures the number of Flash memory wait states
  - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, USB…). In this case, the clock configuration is performed by an extended API defined in stm32l0xx_hal_ppp_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

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

A set of macros are defined in stm32l0xx_hal_rcc.h. They allows executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __PPP_CLK_ENABLE/__PPP_CLK_DISABLE to enable/disable the peripheral clock
- __PPP_FORCE_RESET/__PPP_RELEASE_RESET to force/release peripheral reset
- __PPP_CLK_SLEEP_ENABLE/__PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during low power (Sleep) mode.

2.11.2 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 stm32l0xx_it.c and implement HAL_GPIO_EXTI_Callback().

The table below describes the GPIO_InitTypeDef structure field.

Table 12: Description of GPIO_InitTypeDef structure

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

Structure field	Description
Alternate	Peripheral to be connected to the selected pins. Possible values: GPIO_AFx_PPP, where AFx: is the alternate function index PPP: is the peripheral instance Example: use GPIO_AF1_TIM1 to connect TIM1 IOs on AF1. These values are defined in the GPIO extended driver, since the AF mapping may change between product lines. Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.

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_HIGH; 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);
```

2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, stm32l0xx_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()

2.11.4 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_PVDConfig()
  - 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()

Depending on the STM32 Series, extension functions are available in stm32l0xx_hal_pwr_ex. Here are a few examples (the list is not exhaustive)

- Ultra low-power mode control
  - HAL_PWREx_EnableUltraLowPower() / HAL_PWREx_DisableUltraLowPower()
  - HAL_PWREx_EnableLowPowerRunMode() / HAL_PWREx_DisableLowPowerRunMode()

2.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral. As a result there are no EXTI APIs but each peripheral HAL driver implements the associated EXTI configuration and EXTI function are implemented 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: #define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!<External interrupt line 16 Connected to the PVD EXTI Line */
__HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)	Enables a given EXTI line Example: __HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)

Macros	Description
__HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)	Disables a given EXTI line. Example: __HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_GENERATE_SWIT (__EXTI_LINE__)	Generates a software interrupt for a given EXTI line. Example: __HAL_PVD_EXTI_ GENERATE_SWIT (PWR_EXTI_LINE_PVD)

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32l0xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

2.11.6 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 stream, 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
- Channels 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
  a. 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.
  b. 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
  a. Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
  b. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
  c. 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 confgured. In this case the DMA interrupt is configured.

d. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
e. 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: enablse the specified DMA Channels.
- __HAL_DMA_DISABLE: disables the specified DMA Channels.
- __HAL_DMA_GET_FLAG: gets the DMA Channels pending flags.
- __HAL_DMA_CLEAR_FLAG: clears the DMA Channels pending flags.
- __HAL_DMA_ENABLE_IT: enables the specified DMA Channels interrupts.
- __HAL_DMA_DISABLE_IT: disables the specified DMA Channels interrupts.
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA stream interrupt has occurred or not.

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”).

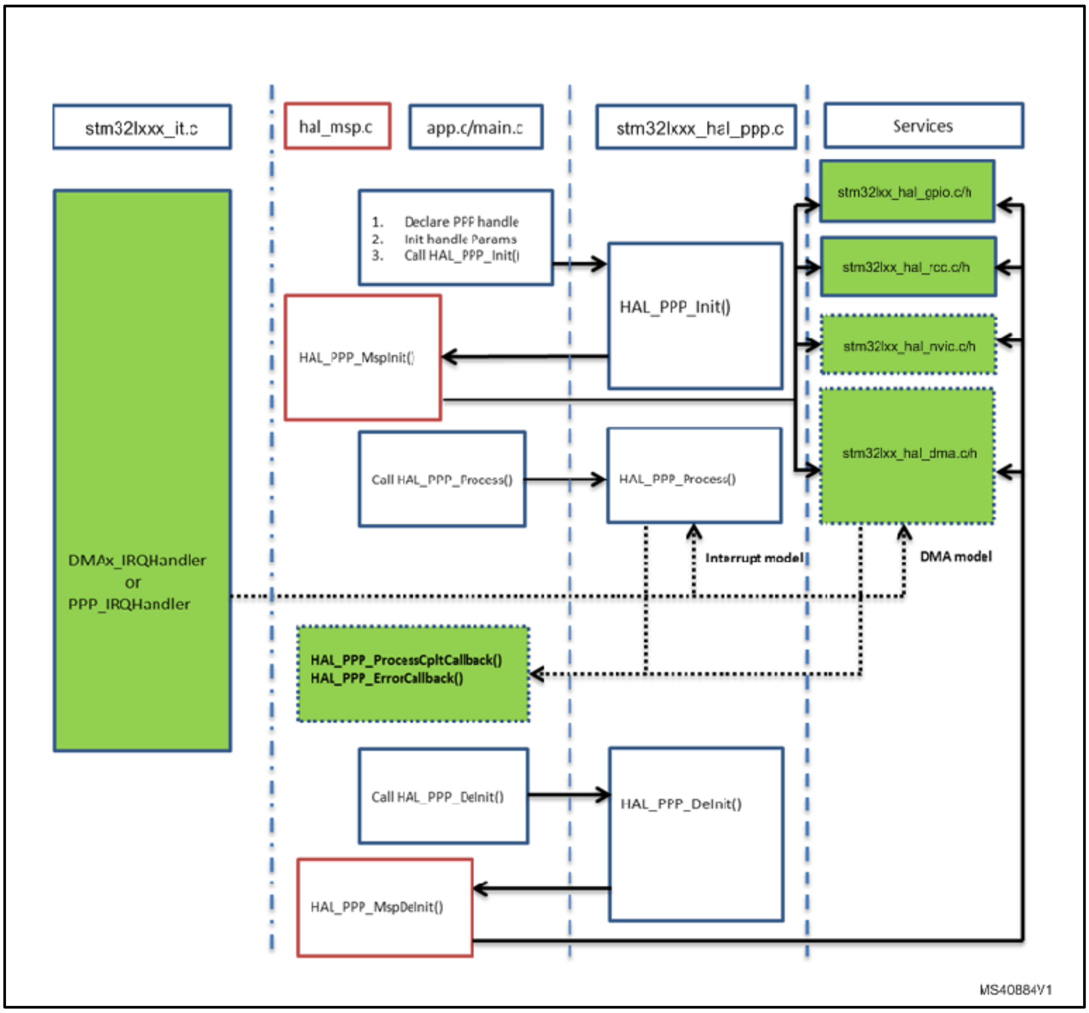
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

2.12 How to use HAL drivers

2.12.1 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



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 (see green blocks in the above schematics).

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 (see blue and red blocks in the above schematics).

2.12.2 HAL initialization

2.12.2.1 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 stm32l0xx_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 a is user callback function to do system level De-Initalizations.
- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.
  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

2.12.2.2 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 the application code. Please find below the typical Clock configuration sequence:

```
void SystemClock_Config(void)
{
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
RCC_OscInitTypeDef RCC_OscInitStruct = {0};

/* Enable MSI Oscillator         */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
RCC_OscInitStruct.MSIState = RCC_MSI_ON;
RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
RCC_OscInitStruct.MSICalibrationValue=0x00;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
if  (HAL_RCC_OscConfig(&RCC_OscInitStruct)!= HAL_OK)
{
  /* Initialization Error        */
  while(1);
  }

/* Select MSI as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks
dividers */
RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0)!= HAL_OK)
{
  /* Initialization Error   */
  while(1);
```

```
}

 /* Enable Power Control clock */
__HAL_RCC_PWR_CLK_ENABLE();
 /* The voltage scaling allows optimizing the power consumption when the device is
clocked below the
  maximum system frequency, to update the voltage scaling value regarding system
  frequency refer to product datasheet.  */
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
 /* Disable Power Control clock       */
__HAL_RCC_PWR_CLK_DISABLE();
  }
```

2.12.2.3 HAL MSP initialization process

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

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

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

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

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

The HAL MSP callback is implemented inside the *stm32l0xx_hal_msp.c* file in the user folders. An *stm32l0xx_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.

stm32l0xx_hal_msp.c file contains the following functions:

Table 14: MSP functions

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

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

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

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

2.12.3 HAL IO 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

2.12.3.1 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 HELIAC; }
```

2.12.3.2 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 *stm32l0xx_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)
{
}
```

*stm32l0xx_it.c*file:

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

2.12.3.3 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 *stm32l0xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

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

The initialization is done as follows (UART example):

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

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

An example of use is illustrated below:

main.c file:

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

stm32l0xx_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 ;
(…)
}
```

2.12.4 Timeout and error management

2.12.4.1 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

Notes:

[1]HAL_MAX_DELAY is defined in the stm32l0xx_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)
{
uint32_t tickstart = 0;
(…)
tickstart = HAL_GetTick();
(…)
while(ProcessOngoing)
{
(…)
if((HAL_GetTick()- tickstart)> LOCAL_PROCESS_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)
{
uint32_t tickstart = 0;
(…)
tickstart = HAL_GetTick();
(…)
while(ProcessOngoing)
{
(…)
if(Timeout != HAL_MAX_DELAY)
{
if((HAL_GetTick()- tickstart )> Timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return hppp->State;
}
}
(…)
}
```

2.12.4.2 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 */
}
```

2.12.4.3 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 drivers' 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 *stm32l0xx_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 stm32l0xx_hal_conf.h:

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

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

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

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.

3 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.

3.1 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

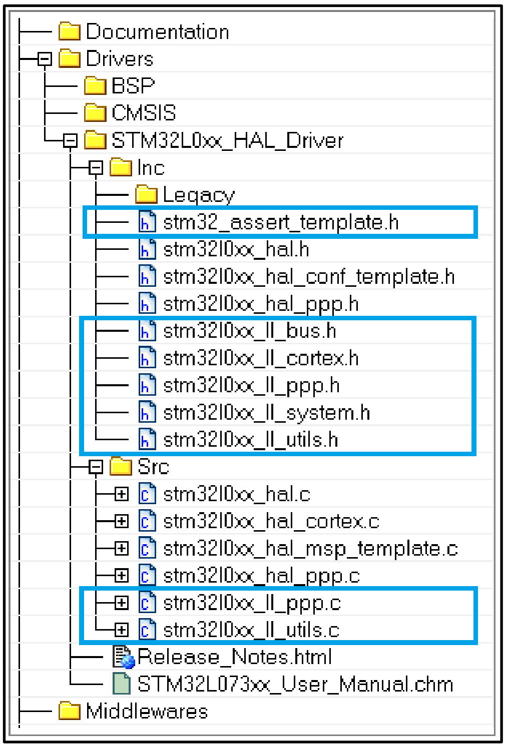
File	Description
stm32l0xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB1_GRP1_EnableClock*
stm32l0xx_ll_ppp.h/.c	stm32l0xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are definined within stm32l0xx_ll_ppp.h file. The Low Layer PPP driver is a standalone module. To use it, the application must include it in the *stm32l0*xx_ll_ppp.h file.
stm32l0xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (LL_SYSTICK_xxxxx, LL_LPM_xxxxx "Low Power Mode" ...)
stm32l0xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
*stm32l0*xx_ll_system.h	System related operations (LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx)

File	Description
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

There is no configuration file for the LL drivers.

The Low Layer files are located in the same HAL driver folder.

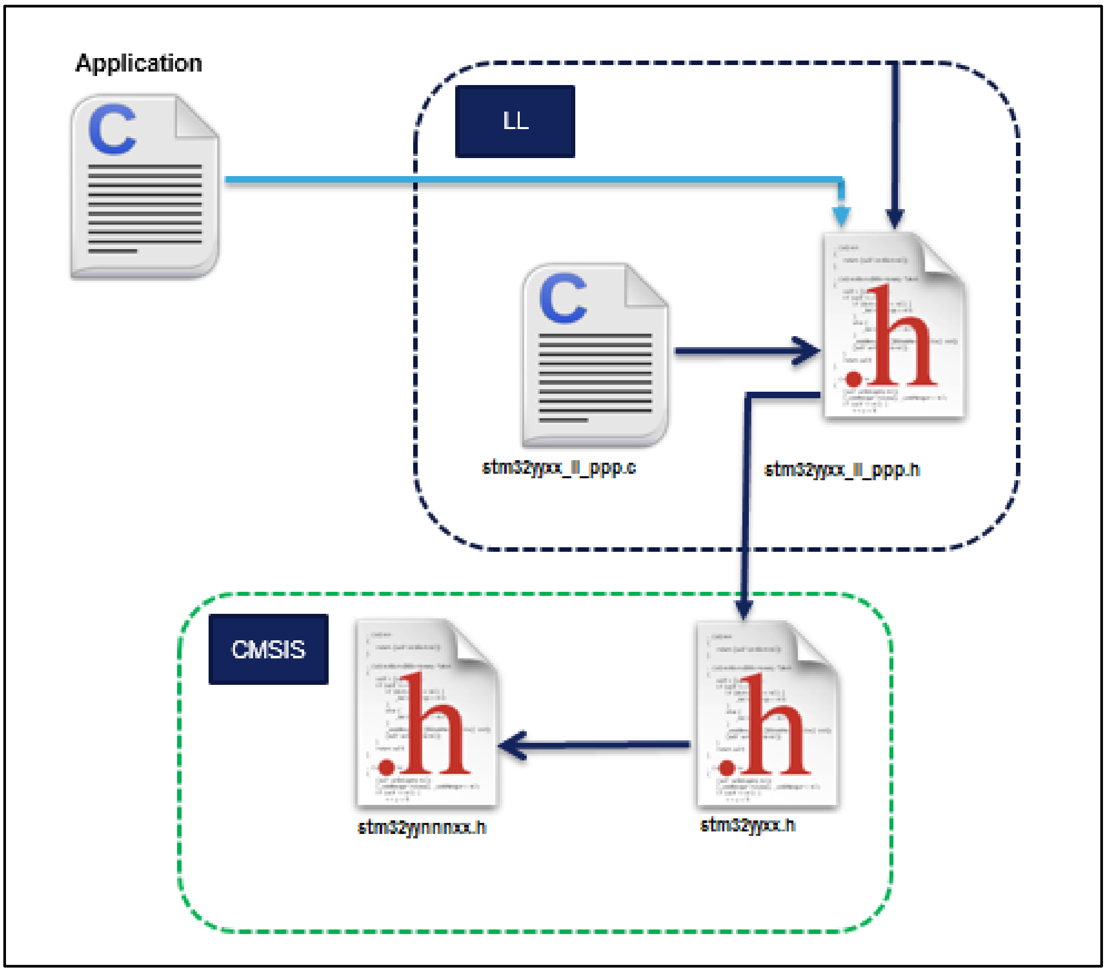
Figure 8: Low Layer driver folders



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 drivers header files.

3.2 Overview of Low Layer APIs and naming rules

3.2.1 Peripheral initialization functions

The LL drivers offer three sets of initialization functions. They are defined in stm32l0xx_ll_ppp.c file:

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

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

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

Table 17: Common peripheral initialization functions

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

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

Table 18: Optional peripheral initialization functions

Functions	Return type	Parameters	Examples
LL_PPP{_ CATEGORY}_Init	*ErrorStatus*	• *PPP_TypeDef * PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef* TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef * TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_ CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_RTC_TIME_StructInit(LL_RTC_TimeTypeDef *RTC_TimeStruct);
LL_PPP_CommonInit	*ErrorStatus*	• *PPP_TypeDef * PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_CommonStructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each PPP_CommonInitStruct member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)

Functions	Return type	Parameters	Examples
LL_PPP_ClockInit	*ErrorStatus*	• *PPP_TypeDef * PPPx* • *LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Initializes the peripheral clock configuration in synchronous mode. Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

3.2.1.1 Run-time checking

Like HAL drivers, LL initialization functions implement run-time failure detection by checking the input values of all LL drivers functions. For more details please refer to *Section 2.12.4.3: "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.

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

3.2.2 Register-level peripheral configuration functions

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

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

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

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

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

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

Table 20: Available function formats

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

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

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

Table 21: Peripheral clock activation/deactivation management

Name	Examples
LL_bus_GRPx_ActionClock{Mode}	• *LL_IOP_GRP1_EnableClock (LL_IOP_GRP1_PERIPH_GPIOA|LL_IOP_GRP1_PERIPH_GPIOB)* • *LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

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

'bus' refers to the bus name (eg APB1).

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

Table 22: Peripheral activation/deactivation management

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

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

Table 23: Peripheral configuration management

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, 16000000, LL_USART_OVERSAMPLING_16, 9600)*

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

Table 24: Peripheral register management

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

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

4 Cohabiting of HAL and LL

The Low Layer is designed to be used in standalone mode or combined with the HAL. It 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 might overwrite some registers which content is mirrored in the HAL handles.

4.1 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 stm32l0xx_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 STM32CubeL0 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.

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.

4.2 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 stm32l0 firmware package (refer to Examples_MIX projects).

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.

5 HAL System Driver

5.1 HAL Firmware driver API description

5.1.1 How to use this driver

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

The HAL contains two APIs categories:

- Common HAL APIs
- Services HAL APIs

5.1.2 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 timebase also when timeout is needed and the backup domain when enabled.
- de-Initializes common part of the HAL.
- Configure The timebase source to have 1ms timebase with a dedicated Tick interrupt priority.
  - Systick timer is used by default as source of timebase, but user can eventually implement his proper timebase 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 timebase 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 timebase configurations are declared as __weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

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

5.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the timebase source interrupt
- Resume the timebase source interrupt
- Get the HAL API driver version
- Get the device identifier

- Get the device revision identifier
- Configure low power mode behavior when the MCU is in Debug mode
- Manage the VEREFINT feature (activation, lock, output selection)

This section contains the following APIs:

- *HAL_IncTick()*
- *HAL_GetTick()*
- *HAL_Delay()*
- *HAL_SuspendTick()*
- *HAL_ResumeTick()*
- *HAL_GetHalVersion()*
- *HAL_GetREVID()*
- *HAL_GetDEVID()*
- *HAL_DBGMCU_EnableDBGSleepMode()*
- *HAL_DBGMCU_DisableDBGSleepMode()*
- *HAL_DBGMCU_EnableDBGStopMode()*
- *HAL_DBGMCU_DisableDBGStopMode()*
- *HAL_DBGMCU_EnableDBGStandbyMode()*
- *HAL_DBGMCU_DisableDBGStandbyMode()*
- *HAL_DBGMCU_DBG_EnableLowPowerConfig()*
- *HAL_DBGMCU_DBG_DisableLowPowerConfig()*
- *HAL_SYSCFG_GetBootMode()*
- *HAL_SYSCFG_VREFINT_OutputSelect()*
- *HAL_SYSCFG_Enable_Lock_VREFINT()*
- *HAL_SYSCFG_Disable_Lock_VREFINT()*

5.1.4 Detailed description of functions

HAL_Init

Function name	**HAL_StatusTypeDef HAL_Init (void)**
Function description	This function configures the Flash prefetch, Flash preread and Buffer cache, Configures timebase source, NVIC and Low level hardware.
Return values	• **HAL:** status
Notes	• This function is called at the beginning of program after reset and before the clock configuration • The timebase configuration is based on MSI clock when exiting from Reset. Once done, timebase tick start incrementing. In the default implementation,Systick is used as source of timebase. the tick variable is incremented each 1ms in its ISR.

HAL_DeInit

Function name	**HAL_StatusTypeDef HAL_DeInit (void)**
Function description	This function de-Initializes common part of the HAL and stops the source of timebase.
Return values	• **HAL:** status
Notes	• This function is optional.

HAL_MspInit

Function name	**void HAL_MspInit (void)**
Function description	Initializes the MSP.
Return values	• **None**

HAL_MspDeInit

Function name	**void HAL_MspDeInit (void)**
Function description	DeInitializes the MSP.
Return values	• **None**

HAL_InitTick

Function name	**HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)**
Function description	This function configures the source of the timebase.
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 timebase. 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 timebase.
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 ms) based on a 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 timebase. 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	Suspends the Tick increment.
Return values	• **None**
Notes	• In the default implementation , SysTick timer is the source of timebase. 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	Resumes the Tick increment.
Return values	• **None**
Notes	• In the default implementation , SysTick timer is the source of timebase. 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	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_DBGMCU_EnableDBGSleepMode

Function name	**void HAL_DBGMCU_EnableDBGSleepMode (void)**
Function description	Enables the Debug Module during SLEEP mode.
Return values	• **None**

HAL_DBGMCU_DisableDBGSleepMode

Function name	**void HAL_DBGMCU_DisableDBGSleepMode (void)**
Function description	Disables the Debug Module during SLEEP mode.
Return values	• **None**

HAL_DBGMCU_EnableDBGStopMode

Function name	**void HAL_DBGMCU_EnableDBGStopMode (void)**
Function description	Enables the Debug Module during STOP mode.
Return values	• **None**

HAL_DBGMCU_DisableDBGStopMode

Function name	**void HAL_DBGMCU_DisableDBGStopMode (void)**
Function description	Disables the Debug Module during STOP mode.
Return values	• **None**

HAL_DBGMCU_EnableDBGStandbyMode

Function name	**void HAL_DBGMCU_EnableDBGStandbyMode (void)**
Function description	Enables the Debug Module during STANDBY mode.
Return values	• **None**

HAL_DBGMCU_DisableDBGStandbyMode

Function name	**void HAL_DBGMCU_DisableDBGStandbyMode (void)**
Function description	Disables the Debug Module during STANDBY mode.

Return values	• **None**

HAL_DBGMCU_DBG_EnableLowPowerConfig

Function name	**void HAL_DBGMCU_DBG_EnableLowPowerConfig (uint32_t Periph)**
Function description	Enable low power mode behavior when the MCU is in Debug mode.
Parameters	• **Periph:** specifies the low power mode. This parameter can be any combination of the following values: – DBGMCU_SLEEP: Keep debugger connection during SLEEP mode – DBGMCU_STOP: Keep debugger connection during STOP mode – DBGMCU_STANDBY: Keep debugger connection during STANDBY mode
Return values	• **None**

HAL_DBGMCU_DBG_DisableLowPowerConfig

Function name	**void HAL_DBGMCU_DBG_DisableLowPowerConfig (uint32_t Periph)**
Function description	Disable low power mode behavior when the MCU is in Debug mode.
Parameters	• **Periph:** specifies the low power mode. This parameter can be any combination of the following values: – DBGMCU_SLEEP: Keep debugger connection during SLEEP mode – DBGMCU_STOP: Keep debugger connection during STOP mode – DBGMCU_STANDBY: Keep debugger connection during STANDBY mode
Return values	• **None**

HAL_SYSCFG_GetBootMode

Function name	**uint32_t HAL_SYSCFG_GetBootMode (void)**
Function description	Returns the boot mode as configured by user.
Return values	• **The:** boot mode as configured by user. The returned value can be one of the following values: – 0x00000000 : Boot is configured in Main Flash memory – 0x00000100 : Boot is configured in System Flash memory – 0x00000300 : Boot is configured in Embedded SRAM memory

HAL_SYSCFG_Enable_Lock_VREFINT

Function name	**void HAL_SYSCFG_Enable_Lock_VREFINT (void)**
Function description	Lock the SYSCFG VREF register values.
Return values	• **None**

HAL_SYSCFG_Disable_Lock_VREFINT

Function name	**void HAL_SYSCFG_Disable_Lock_VREFINT (void)**
Function description	Unlock the overall SYSCFG VREF register values.
Return values	• **None**

HAL_SYSCFG_VREFINT_OutputSelect

Function name	**void HAL_SYSCFG_VREFINT_OutputSelect (uint32_t SYSCFG_Vrefint_OUTPUT)**
Function description	Selects the output of internal reference voltage (VREFINT).
Parameters	• **SYSCFG_Vrefint_OUTPUT:** new state of the Vrefint output. This parameter can be one of the following values: – SYSCFG_VREFINT_OUT_NONE – SYSCFG_VREFINT_OUT_PB0 – SYSCFG_VREFINT_OUT_PB1 – SYSCFG_VREFINT_OUT_PB0_PB1
Return values	• **None**

5.2 HAL Firmware driver defines

5.2.1 HAL

DBGMCU Low Power Configuration

DBGMCU_SLEEP

DBGMCU_STOP

DBGMCU_STANDBY

IS_DBGMCU_PERIPH

HAL Exported Macros

__HAL_DBGMCU_FREEZE_TIM2	TIM2 Peripherals Debug mode
__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_RTC	
__HAL_DBGMCU_UNFREEZE_RTC	
__HAL_DBGMCU_FREEZE_WWDG	
__HAL_DBGMCU_UNFREEZE_WWDG	
__HAL_DBGMCU_FREEZE_IWDG	
__HAL_DBGMCU_UNFREEZE_IWDG	
__HAL_DBGMCU_FREEZE_I2C1_TIMEOUT	
__HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT_DBGMCU	
__HAL_DBGMCU_FREEZE_I2C2_TIMEOUT_DBGMCU	
__HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT_DBGMCU	
__HAL_DBGMCU_FREEZE_I2C3_TIMEOUT	
__HAL_DBGMCU_UNFREEZE_I2C3_TIMEOUT	
__HAL_DBGMCU_FREEZE_LPTIMER	
__HAL_DBGMCU_UNFREEZE_LPTIMER	
__HAL_DBGMCU_FREEZE_TIM22	
__HAL_DBGMCU_UNFREEZE_TIM22	
__HAL_DBGMCU_FREEZE_TIM21	
__HAL_DBGMCU_UNFREEZE_TIM21	
__HAL_SYSCFG_REMAPMEMORY_FLASH	
__HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH	
__HAL_SYSCFG_REMAPMEMORY_SRAM	
__HAL_SYSCFG_DBG_LP_CONFIG	**Description:** • Configuration of the DBG Low Power mode. **Parameters:** • __DBGLPMODE__: bit field to indicate in wich Low Power mode DBG is still active. This parameter can be a value of – DBGMCU_SLEEP – DBGMCU_STOP – DBGMCU_STANDBY
__HAL_SYSCFG_VLCD_CAPA_CONFIG	**Description:** • Macro to configure the VLCD Decoupling capacitance connection.

Parameters:

- __SYSCFG_VLCD_CAPA__: specifies the decoupling of LCD capacitance for rails connection on GPIO. This parameter can be a combination of following values (when available):
  - SYSCFG_VLCD_PB2_EXT_CAPA_ON: Connection on PB2
  - SYSCFG_VLCD_PB12_EXT_CAPA_ON: Connection on PB12
  - SYSCFG_VLCD_PB0_EXT_CAPA_ON: Connection on PB0
  - SYSCFG_VLCD_PE11_EXT_CAPA_ON: Connection on PE11
  - SYSCFG_VLCD_PE12_EXT_CAPA_ON: Connection on PE12

Return value:

- None

__HAL_SYSCFG_GET_VLCD_CAPA_CONFIG

Description:

- Returns the decoupling of LCD capacitance configured by user.

Return value:

- The: LCD capacitance connection as configured by user. The returned can be a combination of : SYSCFG_VLCD_PB2_EXT_CAPA_ON: Connection on PB2 SYSCFG_VLCD_PB12_EXT_CAPA_ON: Connection on PB12 SYSCFG_VLCD_PB0_EXT_CAPA_ON: Connection on PB0 SYSCFG_VLCD_PE11_EXT_CAPA_ON: Connection on PE11 SYSCFG_VLCD_PE12_EXT_CAPA_ON: Connection on PE12

__HAL_SYSCFG_GET_BOOT_MODE

Description:

- Returns the boot mode as configured by user.

Return value:

- The: boot mode as configured by user. The returned can be a value of :
  - SYSCFG_BOOT_MAINFLASH
  - SYSCFG_BOOT_SYSTEMFLASH
  - SYSCFG_BOOT_SRAM

__HAL_SYSCFG_GET_FLAG	**Description:** • Check whether the specified SYSCFG flag is set or not. **Parameters:** • __FLAG__: specifies the flag to check. The only parameter supported is SYSCFG_FLAG_VREFINT_READY **Return value:** • The: new state of __FLAG__ (TRUE or FALSE).
__HAL_SYSCFG_FASTMODEPLUS_ENABLE	**Description:** • Fast mode Plus driving capability enable macro. **Parameters:** • __FASTMODEPLUS__: This parameter can be a value of : – SYSCFG_FASTMODEPLUS_PB6 – SYSCFG_FASTMODEPLUS_PB7 – SYSCFG_FASTMODEPLUS_PB8 – SYSCFG_FASTMODEPLUS_PB9
__HAL_SYSCFG_FASTMODEPLUS_DISABLE	**Description:** • Fast mode Plus driving capability disable macro. **Parameters:** • __FASTMODEPLUS__: This parameter can be a value of : – SYSCFG_FASTMODEPLUS_PB6 – SYSCFG_FASTMODEPLUS_PB7 – SYSCFG_FASTMODEPLUS_PB8 – SYSCFG_FASTMODEPLUS_PB9

HAL state definition

HAL_SMBUS_STATE_RESET	SMBUS not yet initialized or disabled
HAL_SMBUS_STATE_READY	SMBUS initialized and ready for use
HAL_SMBUS_STATE_BUSY	SMBUS internal process is ongoing
HAL_SMBUS_STATE_MASTER_BUSY_TX	Master Data Transmission process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_RX	Master Data Reception process is ongoing
HAL_SMBUS_STATE_SLAVE_BUSY_TX	Slave Data Transmission process is ongoing
HAL_SMBUS_STATE_SLAVE_BUSY_RX	Slave Data Reception process is ongoing
HAL_SMBUS_STATE_TIMEOUT	Timeout state
HAL_SMBUS_STATE_ERROR	Reception process is ongoing
HAL_SMBUS_STATE_LISTEN	Address Listen Mode is ongoing

HAL Version

__STM32L0xx_HAL_VERSION_MAIN	[31:24] main version
__STM32L0xx_HAL_VERSION_SUB1	[23:16] sub1 version
__STM32L0xx_HAL_VERSION_SUB2	[15:8] sub2 version
__STM32L0xx_HAL_VERSION_RC	[7:0] release candidate
__STM32L0xx_HAL_VERSION	
IDCODE_DEVID_MASK	

Boot Mode

SYSCFG_BOOT_MAINFLASH

SYSCFG_BOOT_SYSTEMFLASH

SYSCFG_BOOT_SRAM

Fast Mode Plus on GPIO

SYSCFG_FASTMODEPLUS_PB6

SYSCFG_FASTMODEPLUS_PB7

SYSCFG_FASTMODEPLUS_PB8

SYSCFG_FASTMODEPLUS_PB9

IS_SYSCFG_FASTMODEPLUS

SYSCFG Flags Definition

SYSCFG_FLAG_VREFINT_READY

IS_SYSCFG_FLAG

SYSCFG LCD External Capacitors

SYSCFG_LCD_EXT_CAPA	Connection of internal Vlcd rail to external capacitors
SYSCFG_VLCD_PB2_EXT_CAPA_ON	Connection on PB2
SYSCFG_VLCD_PB12_EXT_CAPA_ON	Connection on PB12
SYSCFG_VLCD_PB0_EXT_CAPA_ON	Connection on PB0
SYSCFG_VLCD_PE11_EXT_CAPA_ON	Connection on PE11
SYSCFG_VLCD_PE12_EXT_CAPA_ON	Connection on PE12

SYSCFG VREFINT Out Selection

SYSCFG_VREFINT_OUT_NONE

SYSCFG_VREFINT_OUT_PB0

SYSCFG_VREFINT_OUT_PB1

SYSCFG_VREFINT_OUT_PB0_PB1

IS_SYSCFG_VREFINT_OUT_SELECT

6 HAL ADC Generic Driver

6.1 ADC Firmware driver registers structures

6.1.1 ADC_OversamplingTypeDef

Data Fields

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

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value of ***ADC_Oversampling_Ratio***
- ***uint32_t ADC_OversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_Right_Bit_Shift***
- ***uint32_t ADC_OversamplingTypeDef::TriggeredMode***
  Selects the regular triggered oversampling mode. This parameter can be a value of ***ADC_Triggered_Oversampling_Mode***

6.1.2 ADC_InitTypeDef

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 LowPowerFrequencyMode***
- ***uint32_t SamplingTime***
- ***uint32_t OversamplingMode***
- ***ADC_OversamplingTypeDef Oversample***

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) and clock prescaler. This parameter can be a value of ***ADC_ClockPrescaler***. Note: In case of synchronous clock mode based on HCLK/1, the configuration must be enabled only if the system clock has a 50% duty clock cycle (APB prescaler configured inside RCC must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference manual for

details. Note: In case of usage of 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***
  Configure the ADC resolution. This parameter can be a value of ***ADC_Resolution***
- ***uint32_t ADC_InitTypeDef::DataAlign***
  Specify ADC data alignment in conversion data register (right or left). Refer to reference manual for alignments formats versus resolutions. This parameter can be a value of ***ADC_Data_align***
- ***uint32_t ADC_InitTypeDef::ScanConvMode***
  Configure 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***
  Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC_EOCSelection***.
- ***uint32_t ADC_InitTypeDef::LowPowerAutoWait***
  Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) has been retrieved by user software, using function **HAL_ADC_GetValue()**. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: Do not use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since they 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 ADC conversion data is needed: use **HAL_ADC_PollForConversion()** to ensure that conversion is completed and **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion start.
- ***uint32_t ADC_InitTypeDef::LowPowerAutoPowerOff***
  Select the auto-off mode: the ADC automatically powers-off after a conversion and automatically wakes-up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with automatic wait mode (parameter 'LowPowerAutoWait'). This parameter can be set to ENABLE or DISABLE. Note: If enabled, this feature also turns off the ADC dedicated 14 MHz RC oscillator (HSI14)
- ***uint32_t ADC_InitTypeDef::ContinuousConvMode***
  Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::DiscontinuousConvMode***
  Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to

ENABLE or DISABLE. Note: On this STM32 serie, ADC group regular number of discontinuous ranks increment is fixed to one-by-one.

- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
  Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_regular_external_trigger_source***. Caution: external trigger source is common to all ADC instances.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
  Select the external event edge used to trigger ADC group regular conversion start. If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADC_regular_external_trigger_edge***
- ***uint32_t ADC_InitTypeDef::DMAContinuousRequests***
  Specify whether the DMA requests are performed in one shot mode (DMA transfer stops when number of conversions is reached) or in continuous mode (DMA transfer unlimited, whatever number of conversions). This parameter can be set to ENABLE or DISABLE. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached.
- ***uint32_t ADC_InitTypeDef::Overrun***
  Select the behavior in case of overrun: data overwritten or preserved (default). This parameter can be a value of ***ADC_Overrun***. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear). Note: Error reporting with respect to the conversion mode:Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).
- ***uint32_t ADC_InitTypeDef::LowPowerFrequencyMode***
  When selecting an analog ADC clock frequency lower than 2.8MHz, it is mandatory to first enable the Low Frequency Mode. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32_t ADC_InitTypeDef::SamplingTime***
  The sample time common to all channels. Unit: ADC clock cycles This parameter can be a value of ***ADC_sampling_times*** Note: This parameter can be modified only if there is no conversion ongoing.
- ***uint32_t ADC_InitTypeDef::OversamplingMode***
  Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC group regular.
- ***ADC_OversamplingTypeDef ADC_InitTypeDef::Oversample***
  Specify the Oversampling parameters Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.

6.1.3 ADC_ChannelConfTypeDef

Data Fields

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

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specify 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 device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Add or remove the channel from ADC regular group sequencer. On STM32L0 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***

6.1.4 ADC_AnalogWDGConfTypeDef

Data Fields

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

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
  Configure the ADC analog watchdog mode: single/all channels. This parameter can be a value of ***ADC_analog_watchdog_mode***
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
  Select which ADC channel to monitor by analog watchdog. This parameter has an effect only if watchdog mode is configured on single channel (parameter WatchdogMode) This parameter can be a value of ***ADC_channels***
- ***uint32_t ADC_AnalogWDGConfTypeDef::ITMode***
  Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
  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 = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  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 = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.

6.1.5 ADC_HandleTypeDef

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

6.2 ADC Firmware driver API description

6.2.1 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)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

6.2.2 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 STM32L0, ADC clock frequency max is 16MHz (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 16MHz. If asynchronous clock is selected, parameter "HSIState" must be set either: - to "...HSIState = RCC_HSI_ON" to maintain the HSI16 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) HSI enable (optional: if asynchronous clock selected)
     - RCC_OscInitTypeDef RCC_OscInitStructure;
     - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI;
     - RCC_OscInitStructure.HSI16CalibrationValue = RCC_HSICALIBRATION_DEFAULT;
     - RCC_OscInitStructure.HSIState = RCC_HSI_ON;
     - RCC_OscInitStructure.PLL... (optional if used for system clock)
     - HAL_RCC_OscConfig(&RCC_OscInitStructure);
   - 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().
4. When device is in mode low-power (low-power run, low-power sleep or stop mode), function "HAL_ADCEx_EnableVREFINT()" must be called before function HAL_ADC_Init(). In case of internal temperature sensor to be measured: function "HAL_ADCEx_EnableVREFINTTempSensor()" must be called similarilly

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()

Callback functions must be implemented in user program:

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

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI;
     - RCC_OscInitStructure.HSIState = RCC_HSI_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)

6.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

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

6.2.4 Peripheral state and errors functions

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

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- ***HAL_ADC_GetState()***
- ***HAL_ADC_GetError()***

6.2.5 Detailed description of functions

HAL_ADC_Init

Function name	**HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)**
Function description	Initialize the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".
Parameters	• **hadc:** ADC handle

Return values	• **HAL:** status
Notes	• As prerequisite, ADC clock must be configured at RCC top level depending on possible clock sources: APB clock of 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_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef". • This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef". • When device is in mode low-power (low-power run, low-power sleep or stop mode), function "HAL_ADCEx_EnableVREFINT()" must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_DeInit() must be called first). In case of internal temperature sensor to be measured: function "HAL_ADCEx_EnableVREFINTTempSensor()" must be called similarilly.

HAL_ADC_DeInit

Function name	**HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)**
Function description	Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behavior in case of reset of a single ADC while the other ADCs sharing the same common group is still running.

HAL_ADC_MspInit

Function name	**void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)**
Function description	Initialize the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• **None**

HAL_ADC_MspDeInit

Function name	**void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)**
Function description	DeInitialize the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• **None**

HAL_ADC_Start

Function name	**HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)**
Function description	Enable ADC, start conversion of regular group.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• Interruptions enabled in this function: None.

HAL_ADC_Stop

Function name	**HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.

HAL_ADC_PollForConversion

Function name	**HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)**
Function description	Wait for regular group conversion to be completed.
Parameters	• **hadc:** ADC handle • **Timeout:** Timeout value in millisecond.
Return values	• **HAL:** status
Notes	• ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue(). • This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

HAL_ADC_PollForEvent

Function name	**HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)**
Function description	Poll for ADC event.
Parameters	• **hadc:** ADC handle • **EventType:** the ADC event type. This parameter can be one of the following values: – ADC_AWD_EVENT: ADC Analog watchdog event – ADC_OVR_EVENT: ADC Overrun event • **Timeout:** Timeout value in millisecond.
Return values	• **HAL:** status
Notes	• The relevant flag is cleared if found to be set, except for ADC_FLAG_OVR. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_OVR_DATA_OVERWRITTEN. Otherwise, data register may be potentially overwritten by a new converted data as soon as OVR is cleared. To reset OVR flag once the preserved data is retrieved, the user can resort to macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_OVR);

HAL_ADC_Start_IT

Function name	**HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)**
Function description	Enable ADC, start conversion of regular group with interruption.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• Interruptions enabled in this function according to initialization setting : EOC (end of conversion), EOS (end of sequence), OVR overrun. Each of these interruptions has its dedicated callback function. • To guarantee a proper reset of all interruptions once all the needed conversions are obtained, HAL_ADC_Stop_IT() must be called to ensure a correct stop of the IT-based conversions. • By default, HAL_ADC_Start_IT() doesn't enable the End Of Sampling interruption. If required (e.g. in case of oversampling with trigger mode), the user must: 1. first clear the EOSMP flag if set with macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_EOSMP) 2. then enable the EOSMP interrupt with macro __HAL_ADC_ENABLE_IT(hadc, ADC_IT_EOSMP) before calling HAL_ADC_Start_IT().

HAL_ADC_Stop_IT

Function name	**HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.

HAL_ADC_Start_DMA

Function name	**HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)**
Function description	Enable ADC, start conversion of regular group and transfer result through DMA.
Parameters	• **hadc:** ADC handle • **pData:** Destination Buffer address. • **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes)
Return values	• **HAL:** status.
Notes	• Interruptions enabled in this function: overrun (if applicable), DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.

HAL_ADC_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.

HAL_ADC_GetValue

Function name	**uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)**
Function description	Get ADC regular group conversion result.
Parameters	• **hadc:** ADC handle
Return values	• **ADC:** group regular conversion data
Notes	• Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion). • This function does not clear ADC flag EOS (ADC group

regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC.If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_IRQHandler

Function name	**void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)**
Function description	Handle ADC interrupt request.
Parameters	• **hadc:** ADC handle
Return values	• **None**

HAL_ADC_ConvCpltCallback

Function name	**void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)**
Function description	Conversion complete callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None**

HAL_ADC_ConvHalfCpltCallback

Function name	**void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)**
Function description	Conversion DMA half-transfer callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None**

HAL_ADC_LevelOutOfWindowCallback

Function name	**void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)**
Function description	Analog watchdog 1 callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None**

HAL_ADC_ErrorCallback

Function name	**void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)**
Function description	ADC error callback in non-blocking mode (ADC conversion with interruption or transfer by DMA).

Parameters	• **hadc:** ADC handle
Return values	• **None**
Notes	• In case of error due to overrun when using ADC with DMA transfer (HAL ADC handle paramater "ErrorCode" to state "HAL_ADC_ERROR_OVR"): Reinitialize the DMA using function "HAL_ADC_Stop_DMA()".If needed, restart a new ADC conversion using function "HAL_ADC_Start_DMA()" (this function is also clearing overrun flag)

HAL_ADC_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)**
Function description	Configure a channel to be assigned to ADC group regular.
Parameters	• **hadc:** ADC handle • **sConfig:** Structure of ADC channel assigned to ADC group regular.
Return values	• **HAL:** status
Notes	• In case of usage of internal measurement channels: VrefInt/Vlcd(STM32L0x3xx only)/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_vlcd (STM32L0x3xx only), TS_temp (values rough order: 5us to 17us). These internal paths can be be disabled using function HAL_ADC_DeInit(). • Possibility to update parameters on the fly: This function initializes channel into ADC group regular, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name	**HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)**
Function description	Configure the analog watchdog.
Parameters	• **hadc:** ADC handle • **AnalogWDGConfig:** Structure of ADC analog watchdog configuration
Return values	• **HAL:** status
Notes	• Possibility to update parameters on the fly: This function initializes the selected analog watchdog, successive calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without

resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".
- Analog watchdog thresholds can be modified while ADC conversion is on going. In this case, some constraints must be taken into account: the programmed threshold values are effective from the next ADC EOC (end of unitary conversion). Considering that registers write delay may happen due to bus activity, this might cause an uncertainty on the effective timing of the new programmed threshold values.

HAL_ADC_GetState

Function name	**uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)**
Function description	Return the ADC handle state.
Parameters	• **hadc:** ADC handle
Return values	• **ADC:** handle state (bitfield on 32 bits)
Notes	• ADC state machine is managed by bitfields, ADC status must be compared with states bits. For example: " if (HAL_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	• **hadc:** ADC handle
Return values	• **ADC:** error code (bitfield on 32 bits)

6.3 ADC Firmware driver defines

6.3.1 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_18

ADC_CHANNEL_VLCD

ADC_CHANNEL_VREFINT

ADC_CHANNEL_TEMPSENSOR

ADC Channel Masks

ADC_CHANNEL_MASK

ADC_CHANNEL_AWD_MASK

ADC Clock Prescaler

ADC_CLOCK_ASYNC_DIV1	ADC Asynchronous clock mode divided by 1
ADC_CLOCK_ASYNC_DIV2	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV4	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV6	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV8	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV10	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV12	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV16	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV32	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV64	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV128	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV256	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_SYNC_PCLK_DIV1	Synchronous clock mode divided by 1 This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle)

ADC_CLOCK_SYNC_PCLK_DIV2	Synchronous clock mode divided by 2
ADC_CLOCK_SYNC_PCLK_DIV4	Synchronous clock mode divided by 4

ADC Conversion Group

ADC_REGULAR_GROUP

ADC conversion data alignment

ADC_DATAALIGN_RIGHT

ADC_DATAALIGN_LEFT

ADC EOC Selection

ADC_EOC_SINGLE_CONV

ADC_EOC_SEQ_CONV

ADC_EOC_SINGLE_SEQ_CONV reserved for future use

ADC Error Code

HAL_ADC_ERROR_NONE	No error
HAL_ADC_ERROR_INTERNAL	ADC IP internal error (problem of clocking, enable/disable, erroneous state, ...)
HAL_ADC_ERROR_OVR	Overrun error
HAL_ADC_ERROR_DMA	DMA transfer error

ADC Event

ADC_AWD_EVENT

ADC_OVR_EVENT

ADC Exported Macros

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE

Description:

- Enable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

ADC_ENABLING_CONDITIONS

Description:

- Verification of hardware constraints before ADC can be enabled.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (ADC can be enabled) or RESET (ADC cannot be enabled)

__HAL_ADC_DISABLE

Description:

- Disable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

ADC_DISABLING_CONDITIONS

Description:

- Verification of hardware constraints before ADC can be disabled.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (ADC can be disabled) or RESET (ADC cannot be disabled)

ADC_IS_ENABLE

Description:

- Verification of ADC state: enabled or disabled.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (ADC enabled) or RESET (ADC disabled)

ADC_GET_RESOLUTION

Description:

- Returns resolution bits in CFGR register: RES[1:0].

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

ADC_IS_SOFTWARE_START_REGULAR

Description:

- Test if conversion trigger of regular group is software start or external trigger.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (software start) or RESET (external trigger)

ADC_IS_CONVERSION_ONGOING_REGULAR

Description:

- Check if no conversion on going on regular group.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (conversion is on going) or RESET (no conversion is on going)

ADC_CONTINUOUS

Description:

- Enable ADC continuous conversion mode.

Parameters:

- _CONTINUOUS_MODE_: Continuous mode.

Return value:

- None

ADC_SCANDIR

Description:

- Enable ADC scan mode to convert multiple ranks with sequencer.

Parameters:

- _SCAN_MODE_: Scan conversion mode.

Return value:

- None

__HAL_ADC_CFGR1_DISCONTINUOUS_NUM

Description:

- Configures the number of discontinuous conversions for the regular group channels.

Parameters:

- _NBR_DISCONTINUOUS_CONV_: Number of discontinuous conversions.

Return value:

- None

ADC_DMACONTREQ	**Description:** • Enable the ADC DMA continuous request. **Parameters:** • _DMAContReq_MODE_: DMA continuous request mode. **Return value:** • None
__HAL_ADC_CFGR1_AutoDelay	**Description:** • Enable the ADC Auto Delay. **Parameters:** • _AutoDelay_: Auto delay bit enable or disable. **Return value:** • None
__HAL_ADC_CFGR1_AUTOFF	**Description:** • Enable the ADC LowPowerAutoPowerOff. **Parameters:** • _AUTOFF_: AutoOff bit enable or disable. **Return value:** • None
ADC_TRX_HIGHTHRESHOLD	**Description:** • Configure the analog watchdog high threshold into registers TR1, TR2 or TR3. **Parameters:** • _Threshold_: Threshold value **Return value:** • None
__HAL_ADC_CCR_LOWFREQUENCY	**Description:** • Enable the ADC Low Frequency mode. **Parameters:** • _LOW_FREQUENCY_MODE_: Low Frequency mode. **Return value:** • None

ADC_OFFSET_SHIFT_RESOLUTION	**Description:** • Shift the offset in function of the selected ADC resolution. **Parameters:** • __HANDLE__: ADC handle. • _Offset_: Value to be shifted **Return value:** • None
ADC_AWD1THRESHOLD_SHIFT_RESOLUTION	**Description:** • Shift the AWD1 threshold in function of the selected ADC resolution. **Parameters:** • __HANDLE__: ADC handle. • _Threshold_: Value to be shifted **Return value:** • None
__HAL_ADC_Value_Shift_left	**Description:** • Shift the value on the left, less significant are set to 0. **Parameters:** • _Value_: Value to be shifted • _Shift_: Number of shift to be done **Return value:** • None
__HAL_ADC_ENABLE_IT	**Description:** • Enable the ADC end of conversion interrupt. **Parameters:** • __HANDLE__: ADC handle. • __INTERRUPT__: ADC Interrupt. **Return value:** • None
__HAL_ADC_DISABLE_IT	**Description:** • Disable the ADC end of conversion interrupt. **Parameters:** • __HANDLE__: ADC handle. • __INTERRUPT__: ADC interrupt.

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check
  - ...
  - ...

Return value:

- State: of interruption (TRUE or FALSE)

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the ADC's pending flags.

Parameters:

- __HANDLE__: ADC handle.
- __FLAG__: ADC flag.

Return value:

- None

__HAL_ADC_GET_FLAG

Description:

- Get the selected ADC's flag status.

Parameters:

- __HANDLE__: ADC handle.
- __FLAG__: ADC flag.

Return value:

- None

ADC_STATE_CLR_SET

Description:

- Simultaneously clears and sets specific bits of the handle State.

Return value:

- None

Notes:

- : ADC_STATE_CLR_SET() macro is merely aliased to generic macro MODIFY_REG(), the first parameter is the ADC handle State, the second parameter is the bit field to clear, the third and last parameter is the bit field to set.

ADC_CLEAR_ERRORCODE

Description:

- Clear ADC error code (set it to error code: "no error")

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_CLOCK_PRESCALER

Description:

- Configuration of ADC clock & prescaler: clock source PCLK or Asynchronous with selectable prescaler.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

IS_ADC_CLOCKPRESCALER

IS_ADC_RESOLUTION

IS_ADC_RESOLUTION_8_6_BITS

IS_ADC_DATA_ALIGN

IS_ADC_EXTTRIG_EDGE

IS_ADC_EOC_SELECTION

IS_ADC_OVERRUN

IS_ADC_RANK

IS_ADC_CHANNEL

IS_ADC_SAMPLE_TIME

IS_ADC_SCAN_MODE

IS_ADC_OVERSAMPLING_RATIO

IS_ADC_RIGHT_BIT_SHIFT

IS_ADC_TRIGGERED_OVERSAMPLING_MODE

IS_ADC_ANALOG_WATCHDOG_MODE

IS_ADC_CONVERSION_GROUP

IS_ADC_EVENT_TYPE

ADC Exported Types

HAL_ADC_STATE_RESET	**Notes:** • 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
HAL_ADC_STATE_READY	ADC peripheral ready for use
HAL_ADC_STATE_BUSY_INTERNAL	ADC is busy due to an internal process (initialization, calibration)
HAL_ADC_STATE_TIMEOUT	TimeOut occurrence
HAL_ADC_STATE_ERROR_INTERNAL	Internal error occurrence
HAL_ADC_STATE_ERROR_CONFIG	Configuration error occurrence
HAL_ADC_STATE_ERROR_DMA	DMA error occurrence
HAL_ADC_STATE_REG_BUSY	A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))
HAL_ADC_STATE_REG_EOC	Conversion data available on group regular
HAL_ADC_STATE_REG_OVR	Overrun occurrence
HAL_ADC_STATE_REG_EOSMP	Not available on this STM32 serie: End Of Sampling flag raised
HAL_ADC_STATE_INJ_BUSY	Not available on this STM32 serie: A conversion on group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))
HAL_ADC_STATE_INJ_EOC	Not available on this STM32 serie: Conversion data available on group injected
HAL_ADC_STATE_INJ_JQOVF	Not available on this STM32 serie: Injected queue overflow occurrence
HAL_ADC_STATE_AWD1	Out-of-window occurrence of ADC analog watchdog 1
HAL_ADC_STATE_AWD2	Not available on this STM32 serie: Out-of-window occurrence of ADC analog watchdog 2
HAL_ADC_STATE_AWD3	Not available on this STM32 serie: Out-of-

	window occurrence of ADC analog watchdog 3
HAL_ADC_STATE_MULTIMODE_SLAVE	Not available on this STM32 serie: ADC in multimode slave state, controlled by another ADC master (when feature available)

ADC External Trigger Source

IS_ADC_EXTTRIG

ADC flags definition

ADC_FLAG_RDY	ADC Ready flag
ADC_FLAG_EOSMP	ADC End of Sampling flag
ADC_FLAG_EOC	ADC End of Regular Conversion flag
ADC_FLAG_EOS	ADC End of Regular sequence of Conversions flag
ADC_FLAG_OVR	ADC overrun flag
ADC_FLAG_AWD	ADC Analog watchdog flag
ADC_FLAG_EOCAL	ADC Enf Of Calibration flag
ADC_FLAG_ALL	

ADC Interrupts Definition

ADC_IT_RDY	ADC Ready (ADRDY) interrupt source
ADC_IT_EOSMP	ADC End of Sampling interrupt source
ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_EOS	ADC End of Regular sequence of Conversions interrupt source
ADC_IT_OVR	ADC overrun interrupt source
ADC_IT_AWD	ADC Analog watchdog 1 interrupt source
ADC_IT_EOCAL	ADC End of Calibration interrupt source

ADC Overrun

ADC_OVR_DATA_PRESERVED

ADC_OVR_DATA_OVERWRITTEN

ADC Oversampling Ratio

ADC_OVERSAMPLING_RATIO_2	ADC Oversampling ratio 2x
ADC_OVERSAMPLING_RATIO_4	ADC Oversampling ratio 4x
ADC_OVERSAMPLING_RATIO_8	ADC Oversampling ratio 8x
ADC_OVERSAMPLING_RATIO_16	ADC Oversampling ratio 16x
ADC_OVERSAMPLING_RATIO_32	ADC Oversampling ratio 32x
ADC_OVERSAMPLING_RATIO_64	ADC Oversampling ratio 64x
ADC_OVERSAMPLING_RATIO_128	ADC Oversampling ratio 128x
ADC_OVERSAMPLING_RATIO_256	ADC Oversampling ratio 256x

ADC Range Verification

IS_ADC_RANGE

ADC rank

ADC_RANK_CHANNEL_NUMBER	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 External Trigger Source Edge for Regular Group

ADC_EXTERNALTRIGCONVEDGE_NONE

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC External Trigger Source

ADC_EXTERNALTRIGCONV_T6_TRGO

ADC_EXTERNALTRIGCONV_T21_CC2

ADC_EXTERNALTRIGCONV_T2_TRGO

ADC_EXTERNALTRIGCONV_T2_CC4

ADC_EXTERNALTRIGCONV_T22_TRGO

ADC_EXTERNALTRIGCONV_T3_TRGO

ADC_EXTERNALTRIGCONV_EXT_IT11

ADC_SOFTWARE_START

ADC_EXTERNALTRIGCONV_T21_TRGO

ADC_EXTERNALTRIGCONV_T2_CC3

ADC Regular Nb Conversion Verification

IS_ADC_REGULAR_NB_CONV

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 Right Bit Shift

ADC_RIGHTBITSHIFT_NONE	ADC No bit shift for oversampling
ADC_RIGHTBITSHIFT_1	ADC 1 bit shift for oversampling
ADC_RIGHTBITSHIFT_2	ADC 2 bits shift for oversampling

ADC_RIGHTBITSHIFT_3	ADC 3 bits shift for oversampling
ADC_RIGHTBITSHIFT_4	ADC 4 bits shift for oversampling
ADC_RIGHTBITSHIFT_5	ADC 5 bits shift for oversampling
ADC_RIGHTBITSHIFT_6	ADC 6 bits shift for oversampling
ADC_RIGHTBITSHIFT_7	ADC 7 bits shift for oversampling
ADC_RIGHTBITSHIFT_8	ADC 8 bits shift for oversampling

ADC Sampling Cycles

ADC_SAMPLETIME_1CYCLE_5	ADC sampling time 1.5 cycle
ADC_SAMPLETIME_3CYCLES_5	ADC sampling time 3.5 CYCLES
ADC_SAMPLETIME_7CYCLES_5	ADC sampling time 7.5 CYCLES
ADC_SAMPLETIME_12CYCLES_5	ADC sampling time 12.5 CYCLES
ADC_SAMPLETIME_19CYCLES_5	ADC sampling time 19.5 CYCLES
ADC_SAMPLETIME_39CYCLES_5	ADC sampling time 39.5 CYCLES
ADC_SAMPLETIME_79CYCLES_5	ADC sampling time 79.5 CYCLES
ADC_SAMPLETIME_160CYCLES_5	ADC sampling time 160.5 CYCLES

ADC Scan mode

ADC_SCAN_DIRECTION_FORWARD	Scan direction forward: from channel 0 to channel 18
ADC_SCAN_DIRECTION_BACKWARD	Scan direction backward: from channel 18 to channel 0
ADC_SCAN_ENABLE	

ADC SYSCFG internal paths Flags Definition

ADC_FLAG_SENSOR

ADC_FLAG_VREFINT

ADC TimeOut Values

ADC_ENABLE_TIMEOUT

ADC_DISABLE_TIMEOUT

ADC_STOP_CONVERSION_TIMEOUT

ADC_DELAY_10US_MIN_CPU_CYCLES

ADC Triggered Oversampling Mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER	ADC No bit shift for oversampling
ADC_TRIGGEREDMODE_MULTI_TRIGGER	ADC No bit shift for oversampling

7 HAL ADC Extension Driver

7.1 ADCEx Firmware driver API description

7.1.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC calibration.

This section contains the following APIs:

- *HAL_ADCEx_Calibration_Start()*
- *HAL_ADCEx_Calibration_GetValue()*
- *HAL_ADCEx_Calibration_SetValue()*
- *HAL_ADCEx_EnableVREFINT()*
- *HAL_ADCEx_DisableVREFINT()*
- *HAL_ADCEx_EnableVREFINTTempSensor()*
- *HAL_ADCEx_DisableVREFINTTempSensor()*

7.1.2 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name	**HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)**
Function description	Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).
Parameters	• **hadc:** ADC handle • **SingleDiff:** Selection of single-ended or differential input This parameter can be only of the following values: – ADC_SINGLE_ENDED: Channel in mode input single ended
Return values	• **HAL:** status
Notes	• Calibration factor can be read after calibration, using function HAL_ADC_GetValue() (value on 7 bits: from DR[6;0]).

HAL_ADCEx_Calibration_GetValue

Function name	**uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)**
Function description	Get the calibration factor.
Parameters	• **hadc:** ADC handle. • **SingleDiff:** This parameter can be only: – ADC_SINGLE_ENDED: Channel in mode input single ended.
Return values	• **Calibration:** value.

HAL_ADCEx_Calibration_SetValue

Function name	**HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)**
Function description	Set the calibration factor to overwrite automatic conversion result.
Parameters	• **hadc:** ADC handle • **SingleDiff:** This parameter can be only: – ADC_SINGLE_ENDED: Channel in mode input single ended. • **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)
Return values	• **HAL:** state

HAL_ADCEx_EnableVREFINT

Function name	**HAL_StatusTypeDef HAL_ADCEx_EnableVREFINT (void)**
Function description	Enables the buffer of Vrefint for the ADC, required when device is in mode low-power (low-power run, low-power sleep or stop mode) This function must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_DeInit() must be called first) For more details on procedure and buffer current consumption, refer to device reference manual.
Return values	• **None**
Notes	• This is functional only if the LOCK is not set.

HAL_ADCEx_DisableVREFINT

Function name	**void HAL_ADCEx_DisableVREFINT (void)**
Function description	Disables the Buffer Vrefint for the ADC.
Return values	• **None**
Notes	• This is functional only if the LOCK is not set.

HAL_ADCEx_EnableVREFINTTempSensor

Function name	**HAL_StatusTypeDef HAL_ADCEx_EnableVREFINTTempSensor (void)**
Function description	Enables the buffer of temperature sensor for the ADC, required when device is in mode low-power (low-power run, low-power sleep or stop mode) This function must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_DeInit() must be called first) For more details on procedure and buffer current consumption, refer to device reference manual.
Return values	• **None**
Notes	• This is functional only if the LOCK is not set.

HAL_ADCEx_DisableVREFINTTempSensor

Function name	**void HAL_ADCEx_DisableVREFINTTempSensor (void)**
Function description	Disables the VREFINT and Sensor for the ADC.
Return values	• **None**
Notes	• This is functional only if the LOCK is not set.

7.2 ADCEx Firmware driver defines

7.2.1 ADCEx

ADC Calibration Factor Length Verification

IS_ADC_CALFACT

Description:

- Calibration factor length verification (7 bits maximum)

Parameters:

- _Calibration_Factor_: Calibration factor value

Return value:

- None

ADC Single Ended

ADC_SINGLE_ENDED

8 HAL COMP Generic Driver

8.1 COMP Firmware driver registers structures

8.1.1 COMP_InitTypeDef

Data Fields

- ***uint32_t WindowMode***
- ***uint32_t Mode***
- ***uint32_t NonInvertingInput***
- ***uint32_t InvertingInput***
- ***uint32_t OutputPol***
- ***uint32_t LPTIMConnection***
- ***uint32_t TriggerMode***

Field Documentation

- ***uint32_t COMP_InitTypeDef::WindowMode***
  Set window mode of a pair of comparators instances (2 consecutive instances odd and even COMP<x> and COMP<x+1>). Note: HAL COMP driver allows to set window mode from any COMP instance of the pair of COMP instances composing window mode. This parameter can be a value of ***COMP_WindowMode***
- ***uint32_t COMP_InitTypeDef::Mode***
  Set comparator operating mode to adjust power and speed. Note: For the characteritics of comparator power modes (propagation delay and power consumption), refer to device datasheet. This parameter can be a value of ***COMP_PowerMode***
- ***uint32_t COMP_InitTypeDef::NonInvertingInput***
  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_InputPlus***
- ***uint32_t COMP_InitTypeDef::InvertingInput***
  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_InputMinus***
- ***uint32_t COMP_InitTypeDef::OutputPol***
  Set comparator output polarity. This parameter can be a value of ***COMP_OutputPolarity***
- ***uint32_t COMP_InitTypeDef::LPTIMConnection***
  Set comparator output connection to LPTIM peripheral. This parameter can be a value of ***COMP_LPTIMConnection***
- ***uint32_t COMP_InitTypeDef::TriggerMode***
  Set the comparator output triggering External Interrupt Line (EXTI). This parameter can be a value of ***COMP_EXTI_TriggerMode***

8.1.2 COMP_HandleTypeDef

Data Fields

- ***COMP_TypeDef * Instance***
- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_COMP_StateTypeDef 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 HAL_COMP_StateTypeDef COMP_HandleTypeDef::State***
  COMP communication state

8.2 COMP Firmware driver API description

8.2.1 COMP Peripheral features

The STM32L0xx device family integrates two analog comparators instances COMP1 and COMP2:

1. The COMP input minus (inverting input) and input plus (non inverting input) can be set to internal references or to GPIO pins (refer to GPIO list in reference manual).
2. The COMP output level is available using HAL_COMP_GetOutputLevel() and can be redirected to other peripherals: GPIO pins (in mode alternate functions for comparator), timers. (refer to GPIO list in reference manual).
3. Pairs of comparators instances can be combined in window mode (2 consecutive instances odd and even COMP<x> and COMP<x+1>).
4. The comparators have interrupt capability through the EXTI controller with wake-up from sleep and stop modes:
   - 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 using macro __HAL_COMP_COMP1_EXTI_GET_FLAG() and __HAL_COMP_COMP2_EXTI_GET_FLAG().

8.2.2 How to use this driver

This driver provides functions to configure and program the comparator instances of STM32L0xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the HAL_COMP_MspInit():
   - Configure the GPIO connected to comparator inputs plus and minus in analog mode using HAL_GPIO_Init().
   - If needed, configure the GPIO connected to comparator output in alternate function mode using HAL_GPIO_Init().
   - If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL_GPIO_Init() function. After that enable the comparator interrupt vector using HAL_NVIC_EnableIRQ() function.
2. Configure the comparator using HAL_COMP_Init() function:
   - Select the input minus (inverting input)
   - Select the input plus (non-inverting input)
   - Select the output polarity
   - Select the power mode
   - Select the window mode HAL_COMP_Init() calls internally __HAL_RCC_SYSCFG_CLK_ENABLE() to enable internal control clock of the comparators. However, this is a legacy strategy. In future STM32 families, COMP

clock enable must be implemented by user in "HAL_COMP_MspInit()". Therefore, for compatibility anticipation, it is recommended to implement __HAL_RCC_SYSCFG_CLK_ENABLE() in "HAL_COMP_MspInit()".

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

8.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- ***HAL_COMP_Init()***
- ***HAL_COMP_DeInit()***
- ***HAL_COMP_MspInit()***
- ***HAL_COMP_MspDeInit()***

8.2.4 IO operation functions

This section provides functions allowing to:

- Start a comparator instance.
- Stop a comparator instance.

This section contains the following APIs:

- ***HAL_COMP_Start()***
- ***HAL_COMP_Stop()***
- ***HAL_COMP_IRQHandler()***

8.2.5 Peripheral Control functions

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

This section contains the following APIs:

- ***HAL_COMP_Lock()***
- ***HAL_COMP_GetOutputLevel()***
- ***HAL_COMP_TriggerCallback()***

8.2.6 Peripheral State functions

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

This section contains the following APIs:

- ***HAL_COMP_GetState()***

8.2.7 Detailed description of functions

HAL_COMP_Init

Function name	**HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)**
Function description	Initialize the COMP according to the specified parameters in the COMP_InitTypeDef and initialize the associated handle.
Parameters	• **hcomp:** COMP handle
Return values	• **HAL:** status
Notes	• If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset. • When the LPTIM connection is enabled, the following pins LPTIM_IN1(PB5, PC0) and LPTIM_IN2(PB7, PC2) should not be configured in alternate function.

HAL_COMP_DeInit

Function name	**HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)**
Function description	DeInitialize the COMP peripheral.
Parameters	• **hcomp:** COMP handle
Return values	• **HAL:** status
Notes	• Deinitialization cannot be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_COMP_MspInit

Function name	**void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)**
Function description	Initialize the COMP MSP.
Parameters	• **hcomp:** COMP handle
Return values	• **None**

HAL_COMP_MspDeInit

Function name	**void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)**
Function description	DeInitialize the COMP MSP.
Parameters	• **hcomp:** COMP handle
Return values	• **None**

HAL_COMP_Start

Function name	**HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)**
Function description	Start the comparator.
Parameters	• **hcomp:** COMP handle
Return values	• **HAL:** status

HAL_COMP_Stop

Function name	**HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)**
Function description	Stop the comparator.
Parameters	• **hcomp:** COMP handle
Return values	• **HAL:** status

HAL_COMP_IRQHandler

Function name	**void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)**
Function description	Comparator IRQ handler.
Parameters	• **hcomp:** COMP handle
Return values	• **None**

HAL_COMP_Lock

Function name	**HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)**
Function description	Lock the selected comparator configuration.
Parameters	• **hcomp:** COMP handle
Return values	• **HAL:** status
Notes	• A system reset is required to unlock the comparator configuration. • Locking the comparator from reset state is possible if __HAL_RCC_SYSCFG_CLK_ENABLE() is being called before.

HAL_COMP_GetOutputLevel

Function name	**uint32_t HAL_COMP_GetOutputLevel (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	**HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)**
Function description	Return the COMP handle state.
Parameters	• **hcomp:** COMP handle
Return values	• **HAL:** state

8.3 COMP Firmware driver defines

8.3.1 COMP

COMP Exported Types

COMP_STATE_BITFIELD_LOCK

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_EXTI_IT	EXTI line event with interruption
COMP_EXTI_EVENT	EXTI line event only (without interruption)
COMP_EXTI_RISING	EXTI line event on rising edge
COMP_EXTI_FALLING	EXTI line event on falling edge

COMP external interrupt line management

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_EDGE	**Description:** • Enable the COMP1 EXTI line rising edge trigger. **Return value:** • None
__HAL_COMP_COMP1_EXTI_DISABLE_RISING_EDGE	**Description:** • Disable the COMP1 EXTI line rising edge trigger. **Return value:** • None

__HAL_COMP_COMP1_EXTI_ENABLE_FALLING_EDGE	**Description:** • Enable the COMP1 EXTI line falling edge trigger. **Return value:** • None
__HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable the COMP1 EXTI line falling edge trigger. **Return value:** • None
__HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • Enable the COMP1 EXTI line rising & falling edge trigger. **Return value:** • None
__HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable the COMP1 EXTI line rising & falling edge trigger. **Return value:** • None
__HAL_COMP_COMP1_EXTI_ENABLE_IT	**Description:** • Enable the COMP1 EXTI line in interrupt mode. **Return value:** • None
__HAL_COMP_COMP1_EXTI_DISABLE_IT	**Description:** • Disable the COMP1 EXTI line in interrupt mode. **Return value:** • None
__HAL_COMP_COMP1_EXTI_GENERATE_SWIT	**Description:** • Generate a software interrupt on the COMP1 EXTI line. **Return value:**

- None

__HAL_COMP_COMP1_EXTI_ENABLE_EVENT

Description:

- Enable the COMP1 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_EVENT

Description:

- Disable the COMP1 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GET_FLAG

Description:

- Check whether the COMP1 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP1_EXTI_CLEAR_FLAG

Description:

- Clear the COMP1 EXTI flag.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP2 EXTI line falling edge trigger.

	Return value: • None
__HAL_COMP_COMP2_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable the COMP2 EXTI line falling edge trigger. **Return value:** • None
__HAL_COMP_COMP2_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • Enable the COMP2 EXTI line rising & falling edge trigger. **Return value:** • None
__HAL_COMP_COMP2_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable the COMP2 EXTI line rising & falling edge trigger. **Return value:** • None
__HAL_COMP_COMP2_EXTI_ENABLE_IT	**Description:** • Enable the COMP2 EXTI line in interrupt mode. **Return value:** • None
__HAL_COMP_COMP2_EXTI_DISABLE_IT	**Description:** • Disable the COMP2 EXTI line in interrupt mode. **Return value:** • None
__HAL_COMP_COMP2_EXTI_GENERATE_SWIT	**Description:** • Generate a software interrupt on the COMP2 EXTI line. **Return value:** • None
__HAL_COMP_COMP2_EXTI_ENABLE_EVENT	**Description:** • Enable the COMP2 EXTI line in event

	mode. **Return value:** • None
__HAL_COMP_COMP2_EXTI_DISABLE_EVENT	**Description:** • Disable the COMP2 EXTI line in event mode. **Return value:** • None
__HAL_COMP_COMP2_EXTI_GET_FLAG	**Description:** • Check whether the COMP2 EXTI line flag is set. **Return value:** • RESET: or SET
__HAL_COMP_COMP2_EXTI_CLEAR_FLAG	**Description:** • Clear the COMP2 EXTI flag. **Return value:** • None

COMP output to EXTI

COMP_TRIGGERMODE_NONE	Comparator output triggering no External Interrupt Line
COMP_TRIGGERMODE_IT_RISING	Comparator output triggering External Interrupt Line event with interruption, on rising edge
COMP_TRIGGERMODE_IT_FALLING	Comparator output triggering External Interrupt Line event with interruption, on falling edge
COMP_TRIGGERMODE_IT_RISING_FALLING	Comparator output triggering External Interrupt Line event with interruption, on both rising and falling edges
COMP_TRIGGERMODE_EVENT_RISING	Comparator output triggering External Interrupt Line event only (without interruption), on rising edge
COMP_TRIGGERMODE_EVENT_FALLING	Comparator output triggering External Interrupt Line event only (without interruption), on falling edge
COMP_TRIGGERMODE_EVENT_RISING_FALLING	Comparator output triggering External Interrupt Line event only (without interruption), on both

rising and falling edges

COMP private macros to get EXTI line associated with comparators

COMP_GET_EXTI_LINE

Description:

- Get the specified EXTI line for a comparator instance.

Parameters:

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

COMP Handle Management

__HAL_COMP_RESET_HANDLE_STATE

Description:

- Reset COMP handle state.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_ENABLE

Description:

- Enable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_DISABLE

Description:

- Disable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_LOCK

Description:

- Lock the specified comparator configuration.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

Notes:

- Using this macro induce HAL COMP handle state machine being no more in

	line with COMP instance state. To keep HAL COMP handle state machine updated, it is recommended to use function "HAL_COMP_Lock')".
__HAL_COMP_IS_LOCKED	**Description:** • Check whether the specified comparator is locked. **Parameters:** • __HANDLE__: COMP handle **Return value:** • Value: 0 if COMP instance is not locked, value 1 if COMP instance is locked

COMP input minus (inverting input)

COMP_INPUT_MINUS_1_4VREFINT	Comparator input minus connected to 1/4 VREFINT (only for COMP instance: COMP2)
COMP_INPUT_MINUS_1_2VREFINT	Comparator input minus connected to 1/2 VREFINT (only for COMP instance: COMP2)
COMP_INPUT_MINUS_3_4VREFINT	Comparator input minus connected to 3/4 VREFINT (only for COMP instance: COMP2)
COMP_INPUT_MINUS_VREFINT	Comparator input minus connected to VrefInt
COMP_INPUT_MINUS_DAC1_CH1	Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)
COMP_INPUT_MINUS_DAC1_CH2	Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)
COMP_INPUT_MINUS_IO1	Comparator input minus connected to IO1 (pin PA0 for COMP1, pin PA2 for COMP2)
COMP_INPUT_MINUS_IO2	Comparator input minus connected to IO2 (pin PB3 for COMP2) (only for COMP instance: COMP2)

COMP input plus (non-inverting input)

COMP_INPUT_PLUS_IO1	Comparator input plus connected to IO1 (pin PA1 for COMP1, pin PA3 for COMP2)
COMP_INPUT_PLUS_IO2	Comparator input plus connected to IO2 (pin PB4 for COMP2) (only for COMP instance: COMP2)
COMP_INPUT_PLUS_IO3	Comparator input plus connected to IO3 (pin PA5 for COMP2) (only for COMP instance: COMP2)
COMP_INPUT_PLUS_IO4	Comparator input plus connected to IO4 (pin PB6 for COMP2) (only for COMP instance: COMP2)
COMP_INPUT_PLUS_IO5	Comparator input plus connected to IO5 (pin PB7 for COMP2) (only for COMP instance: COMP2)

COMP private macros to check input parameters

IS_COMP_WINDOWMODE

IS_COMP_POWERMODE

IS_COMP_WINDOWMODE_INSTANCE

IS_COMP_INPUT_PLUS

IS_COMP_INPUT_MINUS

IS_COMP1_LPTIMCONNECTION

IS_COMP2_LPTIMCONNECTION

IS_COMP2_LPTIMCONNECTION_RESTRICTED

IS_COMP_OUTPUTPOL

IS_COMP_TRIGGERMODE

IS_COMP_OUTPUT_LEVEL

COMP Low power timer connection definition

COMP_LPTIMCONNECTION_DISABLED	COMPx signal is gated
COMP_LPTIMCONNECTION_IN1_ENABLED	COMPx signal is connected to LPTIM input 1
COMP_LPTIMCONNECTION_IN2_ENABLED	COMPx signal is connected to LPTIM input 2

COMP Output Level

COMP_OUTPUT_LEVEL_LOW

COMP_OUTPUT_LEVEL_HIGH

COMP output Polarity

COMP_OUTPUTPOL_NONINVERTED	COMP output on GPIO isn't inverted
COMP_OUTPUTPOL_INVERTED	COMP output on GPIO is inverted

COMP power mode

COMP_POWERMODE_MEDIUMSPEED	COMP power mode to low power (indicated as "high speed" in reference manual) (only for COMP instance: COMP2)
COMP_POWERMODE_ULTRALOWPOWER	COMP power mode to ultra low power (indicated as "low speed" in reference manual) (only for COMP instance: COMP2)

COMP Window Mode

COMP_WINDOWMODE_DISABLE	Window mode disable: Comparators instances pair COMP1 and COMP2 are independent
COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON	Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

9 HAL COMP Extension Driver

9.1 COMPEx Firmware driver API description

9.1.1 COMP peripheral Extended features

Comparing to other previous devices, the COMP interface for STM32L0XX devices contains the following additional features

- Possibility to enable or disable the VREFINT which is used as input to the comparator.

9.1.2 Detailed description of functions

HAL_COMPEx_EnableVREFINT

Function name	**void HAL_COMPEx_EnableVREFINT (void)**
Function description	Enable Vrefint and path to comparator, used by comparator instance COMP2 input based on VrefInt or subdivision of VrefInt.
Return values	• **None**
Notes	• The equivalent of this function is managed automatically when using function "HAL_COMP_Init()". • VrefInt requires a startup time (refer to device datasheet, parameter "TVREFINT"). This function waits for the startup time (alternative solution: poll for bit SYSCFG_CFGR3_VREFINT_RDYF set).

HAL_COMPEx_DisableVREFINT

Function name	**void HAL_COMPEx_DisableVREFINT (void)**
Function description	Disable Vrefint and path to comparator, used by comparator instance COMP2 input based on VrefInt or subdivision of VrefInt.
Return values	• **None**

10 HAL CORTEX Generic Driver

10.1 CORTEX Firmware driver registers structures

10.1.1 MPU_Region_InitTypeDef

Data Fields

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

Field Documentation

- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
  Specifies the base address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::Enable***
  Specifies the status of the region. This parameter can be a value of ***CORTEX_MPU_Region_Enable***
- ***uint8_t MPU_Region_InitTypeDef::Number***
  Specifies the number of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Number***
- ***uint8_t MPU_Region_InitTypeDef::Size***
  Specifies the size of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Size***
- ***uint8_t MPU_Region_InitTypeDef::SubRegionDisable***
  Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint8_t MPU_Region_InitTypeDef::TypeExtField***
  This parameter is NOT used but is kept to keep API unified through all families
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
  Specifies the region access permission type. This parameter can be a value of ***CORTEX_MPU_Region_Permission_Attributes***
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
  Specifies the instruction access status. This parameter can be a value of ***CORTEX_MPU_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
  Specifies the shareability status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Shareable***
- ***uint8_t MPU_Region_InitTypeDef::IsCacheable***
  Specifies the cacheable status of the region protected. This parameter can be a value of ***CORTEX_MPU_Access_Cacheable***
- ***uint8_t MPU_Region_InitTypeDef::IsBufferable***
  Specifies the bufferable status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Bufferable***

10.2 CORTEX Firmware driver API description

10.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provide 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. Lower priority values gives higher priority. Priority Order: Lowest priority.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()

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for timebase

- 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 function HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The HAL_SYSTICK_CLKSourceConfig() function is defined inside the stm32l0xx_hal_cortex.c 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 timebase, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

10.2.2 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()***
- ***HAL_MPU_Disable()***
- ***HAL_MPU_Enable()***

10.2.3 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()*
- *HAL_MPU_ConfigRegion()*

10.2.4 Detailed description of functions

HAL_NVIC_SetPriority

Function name	**void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)**
Function description	Sets the priority of an interrupt.
Parameters	• **IRQn:** External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file) • **PreemptPriority:** The pre-emption 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 stm32l0xx 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 stm32l0xx.h file)
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 stm32l0xx.h file)
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_MPU_Disable

Function name	**__STATIC_INLINE void HAL_MPU_Disable (void)**
Function description	Disable the MPU.
Return values	• **None**

HAL_MPU_Enable

Function name	**__STATIC_INLINE void HAL_MPU_Enable (uint32_t MPU_Control)**
Function description	Enable the MPU.
Parameters	• **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values: – MPU_HFNMI_PRIVDEF_NONE – MPU_HARDFAULT_NMI – MPU_PRIVILEGED_DEFAULT – MPU_HFNMI_PRIVDEF

Return values	• **None**

HAL_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 (stm32l0xxxx.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 stm32l0xx.h file)
Return values	• **status:** - 0 Interrupt status is not pending. – 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name	**void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)**
Function description	Sets Pending bit of an external interrupt.
Parameters	• **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)
Return values	• **None**

HAL_NVIC_ClearPendingIRQ

Function name	**void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)**
Function description	Clears the pending bit of an external interrupt.
Parameters	• **IRQn:** External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)
Return values	• **None**

HAL_SYSTICK_CLKSourceConfig

Function name	**void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)**
Function description	Configures the SysTick clock source.
Parameters	• **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values: – SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source. – SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
Return values	• **None**

HAL_SYSTICK_IRQHandler

Function name	**void HAL_SYSTICK_IRQHandler (void)**
Function description	This function handles SYSTICK interrupt request.
Return values	• **None**

HAL_SYSTICK_Callback

Function name	**void HAL_SYSTICK_Callback (void)**
Function description	SYSTICK callback.
Return values	• **None**

HAL_MPU_ConfigRegion

Function name	**void HAL_MPU_ConfigRegion (MPU_Region_InitTypeDef * MPU_Init)**
Function description	Initialize and configure the Region and the memory to be protected.
Parameters	• **MPU_Init:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.
Return values	• **None**

10.3 CORTEX Firmware driver defines

10.3.1 CORTEX

CORTEx Exported Constants

IS_NVIC_PREEMPTION_PRIORITY

IS_NVIC_DEVICE_IRQ

CORTEX MPU Instruction Access Bufferable

MPU_ACCESS_BUFFERABLE

MPU_ACCESS_NOT_BUFFERABLE

CORTEX MPU Instruction Access Cacheable

MPU_ACCESS_CACHEABLE

MPU_ACCESS_NOT_CACHEABLE

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_SHAREABLE

MPU_ACCESS_NOT_SHAREABLE

CORTEX MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU_HARDFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_NO_ACCESS

MPU_REGION_PRIV_RW

MPU_REGION_PRIV_RW_URO

MPU_REGION_FULL_ACCESS

MPU_REGION_PRIV_RO

MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_32B

MPU_REGION_SIZE_64B

MPU_REGION_SIZE_128B

MPU_REGION_SIZE_256B

MPU_REGION_SIZE_512B

MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB

MPU_REGION_SIZE_4KB

MPU_REGION_SIZE_8KB

MPU_REGION_SIZE_16KB

MPU_REGION_SIZE_32KB

MPU_REGION_SIZE_64KB

MPU_REGION_SIZE_128KB

MPU_REGION_SIZE_256KB

MPU_REGION_SIZE_512KB

MPU_REGION_SIZE_1MB

MPU_REGION_SIZE_2MB

MPU_REGION_SIZE_4MB

MPU_REGION_SIZE_8MB

MPU_REGION_SIZE_16MB

MPU_REGION_SIZE_32MB

MPU_REGION_SIZE_64MB

MPU_REGION_SIZE_128MB

MPU_REGION_SIZE_256MB

MPU_REGION_SIZE_512MB

MPU_REGION_SIZE_1GB

MPU_REGION_SIZE_2GB

MPU_REGION_SIZE_4GB

CORTEX SysTick Clock Source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

IS_SYSTICK_CLK_SOURCE

11 HAL CRC Generic Driver

11.1 CRC Firmware driver registers structures

11.1.1 CRC_InitTypeDef

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 ***CRC_Polynomial_Sizes*** and indicates CRC length. Value can be either one of CRC_POLYLENGTH_32B (32-bit CRC) CRC_POLYLENGTH_16B (16-bit CRC) CRC_POLYLENGTH_8B (8-bit CRC) CRC_POLYLENGTH_7B (7-bit CRC)
- ***uint32_t CRC_InitTypeDef::InitValue***
  Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE
- ***uint32_t CRC_InitTypeDef::InputDataInversionMode***
  This parameter is a value of ***CRCEx_Input_Data_Inversion*** and specifies input data inversion mode. Can be either one of the following values CRC_INPUTDATA_INVERSION_NONE no input data inversion CRC_INPUTDATA_INVERSION_BYTE byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2 CRC_INPUTDATA_INVERSION_HALFWORD halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C CRC_INPUTDATA_INVERSION_WORD word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458
- ***uint32_t CRC_InitTypeDef::OutputDataInversionMode***
  This parameter is a value of ***CRCEx_Output_Data_Inversion*** and specifies output

data (i.e. CRC) inversion mode. Can be either CRC_OUTPUTDATA_INVERSION_DISABLE no CRC inversion, or CRC_OUTPUTDATA_INVERSION_ENABLE CRC 0x11223344 is converted into 0x22CC4488

11.1.2 CRC_HandleTypeDef

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

11.2 CRC Firmware driver API description

11.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

1. Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle
2. DeInitialize the CRC peripheral
3. Initialize the CRC MSP
4. DeInitialize CRC MSP

This section contains the following APIs:

- ***HAL_CRC_Init()***
- ***HAL_CRC_DeInit()***
- ***HAL_CRC_MspInit()***
- ***HAL_CRC_MspDeInit()***

11.2.2 Peripheral Control functions

This section provides functions allowing to:

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

2. Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.

This section contains the following APIs:

- *HAL_CRC_Accumulate()*
- *HAL_CRC_Calculate()*

11.2.3 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()*

11.2.4 Detailed description of functions

HAL_CRC_Init

Function name	**HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)**
Function description	Initializes the CRC according to the specified parameters in the CRC_InitTypeDef and creates 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	DeInitializes 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	DeInitializes 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	Returns the CRC state.
Parameters	• **hcrc:** CRC handle
Return values	• **HAL:** state

11.3 CRC Firmware driver defines

11.3.1 CRC

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

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

DEFAULT_POLYNOMIAL_DISABLE

Default CRC generating polynomial

DEFAULT_CRC32_POLY

CRC Exported Constants

HAL_CRC_Input_Data_Reverse

HAL_CRC_Output_Data_Reverse

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

CRC input buffer format

CRC_INPUTDATA_FORMAT_UNDEFINED

CRC_INPUTDATA_FORMAT_BYTES

CRC_INPUTDATA_FORMAT_HALFWORDS

CRC_INPUTDATA_FORMAT_WORDS

Polynomial sizes to configure the IP

CRC_POLYLENGTH_32B

CRC_POLYLENGTH_16B

CRC_POLYLENGTH_8B

CRC_POLYLENGTH_7B

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

HAL_CRC_LENGTH_16B

HAL_CRC_LENGTH_8B

HAL_CRC_LENGTH_7B

12 HAL CRC Extension Driver

12.1 CRCEx Firmware driver API description

12.1.1 CRC Extended features functions

This subsection provides function allowing to:

- Set CRC polynomial if different from default one.

This section contains the following APIs:

- *HAL_CRCEx_Polynomial_Set()*
- *HAL_CRCEx_Input_Data_Reverse()*
- *HAL_CRCEx_Output_Data_Reverse()*

12.1.2 Detailed description of functions

HAL_CRCEx_Polynomial_Set

Function name	**HAL_StatusTypeDef HAL_CRCEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)**
Function description	Initializes the CRC polynomial if different from default one.
Parameters	• **hcrc:** CRC handle • **Pol:** CRC generating polynomial (7, 8, 16 or 32-bit long) This parameter is written in normal representation, e.g. for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65 for a polynomial of degree 16, X^16 + X^12 + X^5 + 1 is written 0x1021 • **PolyLength:** CRC polynomial length This parameter can be one of the following values: – CRC_POLYLENGTH_7B: 7-bit long CRC (generating polynomial of degree 7) – CRC_POLYLENGTH_8B: 8-bit long CRC (generating polynomial of degree 8) – CRC_POLYLENGTH_16B: 16-bit long CRC (generating polynomial of degree 16) – CRC_POLYLENGTH_32B: 32-bit long CRC (generating polynomial of degree 32)
Return values	• **HAL:** status

HAL_CRCEx_Input_Data_Reverse

Function name	**HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)**
Function description	Set the Reverse Input data mode.
Parameters	• **hcrc:** CRC handle • **InputReverseMode:** Input Data inversion mode This parameter can be one of the following values: – CRC_INPUTDATA_INVERSION_NONE no change in bit

order (default value)
- CRC_INPUTDATA_INVERSION_BYTE: Byte-wise bit reversal
- CRC_INPUTDATA_INVERSION_HALFWORD: HalfWord-wise bit reversal
- CRC_INPUTDATA_INVERSION_WORD: Word-wise bit reversal

Return values	• **HAL:** status

HAL_CRCEx_Output_Data_Reverse

Function name	**HAL_StatusTypeDef HAL_CRCEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)**
Function description	Set the Reverse Output data mode.
Parameters	• **hcrc:** CRC handle • **OutputReverseMode:** Output Data inversion mode This parameter can be one of the following values: – CRC_OUTPUTDATA_INVERSION_DISABLE: no CRC inversion (default value) – CRC_OUTPUTDATA_INVERSION_ENABLE: bit-level inversion (e.g for a 8-bit CRC: 0xB5 becomes 0xAD)
Return values	• **HAL:** status

12.2 CRCEx Firmware driver defines

12.2.1 CRCEx

CRCEx Exported Macros

__HAL_CRC_OUTPUTREVERSAL_ENABLE	**Description:** • Set CRC output reversal. **Parameters:** • __HANDLE__: : CRC handle **Return value:** • None.
__HAL_CRC_OUTPUTREVERSAL_DISABLE	**Description:** • Unset CRC output reversal. **Parameters:** • __HANDLE__: : CRC handle **Return value:** • None.
__HAL_CRC_POLYNOMIAL_CONFIG	**Description:** • Set CRC non-default polynomial.

Parameters:

- __HANDLE__: : CRC handle
- __POLYNOMIAL__: 7, 8, 16 or 32-bit polynomial

Return value:

- None.

CRC Extended input data inversion modes

CRC_INPUTDATA_INVERSION_NONE

CRC_INPUTDATA_INVERSION_BYTE

CRC_INPUTDATA_INVERSION_HALFWORD

CRC_INPUTDATA_INVERSION_WORD

CRC Extended output data inversion modes

CRC_OUTPUTDATA_INVERSION_DISABLE

CRC_OUTPUTDATA_INVERSION_ENABLE

13 HAL CRYP Generic Driver

13.1 CRYP Firmware driver registers structures

13.1.1 CRYP_InitTypeDef

Data Fields

- ***uint32_t DataType***
- ***uint8_t * pKey***
- ***uint8_t * pInitVect***

Field Documentation

- ***uint32_t CRYP_InitTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of ***CRYP_Data_Type***
- ***uint8_t* CRYP_InitTypeDef::pKey***
  The key used for encryption/decryption
- ***uint8_t* CRYP_InitTypeDef::pInitVect***
  The initialization vector used also as initialization counter in CTR mode

13.1.2 CRYP_HandleTypeDef

Data Fields

- ***AES_TypeDef * Instance***
- ***CRYP_InitTypeDef Init***
- ***uint8_t * pCrypInBuffPtr***
- ***uint8_t * pCrypOutBuffPtr***
- ***__IO uint16_t CrypInCount***
- ***__IO uint16_t CrypOutCount***
- ***HAL_StatusTypeDef Status***
- ***HAL_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***

Field Documentation

- ***AES_TypeDef* CRYP_HandleTypeDef::Instance***
  Register base address
- ***CRYP_InitTypeDef CRYP_HandleTypeDef::Init***
  CRYP required parameters
- ***uint8_t* CRYP_HandleTypeDef::pCrypInBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint8_t* CRYP_HandleTypeDef::pCrypOutBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***__IO uint16_t CRYP_HandleTypeDef::CrypInCount***
  Counter of inputed data
- ***__IO uint16_t CRYP_HandleTypeDef::CrypOutCount***
  Counter of outputed data
- ***HAL_StatusTypeDef CRYP_HandleTypeDef::Status***
  CRYP peripheral status

- ***HAL_PhaseTypeDef CRYP_HandleTypeDef::Phase***
  CRYP peripheral phase
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmain***
  CRYP In DMA handle parameters
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmaout***
  CRYP Out DMA handle parameters
- ***HAL_LockTypeDef CRYP_HandleTypeDef::Lock***
  CRYP locking object
- ***__IO HAL_CRYP_STATETypeDef CRYP_HandleTypeDef::State***
  CRYP peripheral state

13.2 CRYP Firmware driver API description

13.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYP according to the specified parameters in the CRYP_InitTypeDef and creates the associated handle
- DeInitialize the CRYP peripheral
- Initialize the CRYP MSP
- DeInitialize CRYP MSP

This section contains the following APIs:

- ***HAL_CRYP_Init()***
- ***HAL_CRYP_DeInit()***
- ***HAL_CRYP_MspInit()***
- ***HAL_CRYP_MspDeInit()***

13.2.2 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES algorithm in different chaining modes
- Decrypt cyphertext using AES algorithm in different chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode

This section contains the following APIs:

- ***HAL_CRYP_AESECB_Encrypt()***
- ***HAL_CRYP_AESCBC_Encrypt()***
- ***HAL_CRYP_AESCTR_Encrypt()***
- ***HAL_CRYP_AESECB_Decrypt()***
- ***HAL_CRYP_AESCBC_Decrypt()***
- ***HAL_CRYP_AESCTR_Decrypt()***
- ***HAL_CRYP_AESECB_Encrypt_IT()***
- ***HAL_CRYP_AESCBC_Encrypt_IT()***
- ***HAL_CRYP_AESCTR_Encrypt_IT()***
- ***HAL_CRYP_AESECB_Decrypt_IT()***
- ***HAL_CRYP_AESCBC_Decrypt_IT()***
- ***HAL_CRYP_AESCTR_Decrypt_IT()***
- ***HAL_CRYP_AESECB_Encrypt_DMA()***

- ***HAL_CRYP_AESCBC_Encrypt_DMA()***
- ***HAL_CRYP_AESCTR_Encrypt_DMA()***
- ***HAL_CRYP_AESECB_Decrypt_DMA()***
- ***HAL_CRYP_AESCBC_Decrypt_DMA()***
- ***HAL_CRYP_AESCTR_Decrypt_DMA()***

13.2.3 CRYP IRQ handler management

This section provides CRYP IRQ handler function.

This section contains the following APIs:

- ***HAL_CRYP_IRQHandler()***

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_CRYP_GetState()***

13.2.5 DMA callback functions

This section provides DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA error

This section contains the following APIs:

- ***HAL_CRYP_ErrorCallback()***
- ***HAL_CRYP_InCpltCallback()***
- ***HAL_CRYP_OutCpltCallback()***

13.2.6 Detailed description of functions

HAL_CRYP_Init

Function name	**HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)**
Function description	Initializes the CRYP according to the specified parameters in the CRYP_InitTypeDef and creates the associated handle.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **HAL:** status

HAL_CRYP_DeInit

Function name	**HAL_StatusTypeDef HAL_CRYP_DeInit (CRYP_HandleTypeDef * hcryp)**
Function description	DeInitializes the CRYP peripheral.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values
- **HAL:** status

HAL_CRYP_MspInit

Function name: **void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)**

Function description: Initializes the CRYP MSP.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values
- **None**

HAL_CRYP_MspDeInit

Function name: **void HAL_CRYP_MspDeInit (CRYP_HandleTypeDef * hcryp)**

Function description: DeInitializes CRYP MSP.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values
- **None**

HAL_CRYP_AESECB_Encrypt

Function name: **HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**

Function description: Initializes the CRYP peripheral in AES ECB encryption mode then encrypt pPlainData.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- **Size:** Length of the plaintext buffer, must be a multiple of 16.
- **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- **Timeout:** Specify Timeout value

Return values
- **HAL:** status

HAL_CRYP_AESECB_Decrypt

Function name: **HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**

Function description: Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- **Size:** Length of the plaintext buffer, must be a multiple of 16.
- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- **Timeout:** Specify Timeout value

Return values	• **HAL:** status

HAL_CRYP_AESCBC_Encrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function description	Initializes the CRYP peripheral in AES CBC encryption mode then encrypt pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Timeout:** Specify Timeout value
Return values	• **HAL:** status

HAL_CRYP_AESCBC_Decrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function description	Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Timeout:** Specify Timeout value
Return values	• **HAL:** status

HAL_CRYP_AESCTR_Encrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function description	Initializes the CRYP peripheral in AES CTR encryption mode then encrypt pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Timeout:** Specify Timeout value
Return values	• **HAL:** status

HAL_CRYP_AESCTR_Decrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function description	Initializes the CRYP peripheral in AES CTR decryption mode then decrypted pCypherData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Timeout:** Specify Timeout value
Return values	• **HAL:** status

HAL_CRYP_AESECB_Encrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Initializes the CRYP peripheral in AES ECB encryption mode using Interrupt.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCBC_Encrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Initializes the CRYP peripheral in AES CBC encryption mode using Interrupt.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCTR_Encrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Initializes the CRYP peripheral in AES CTR encryption mode using Interrupt.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESECB_Decrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Initializes the CRYP peripheral in AES ECB decryption mode using Interrupt.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCTR_Decrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Initializes the CRYP peripheral in AES CTR decryption mode using Interrupt.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 • **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCBC_Decrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Initializes the CRYP peripheral in AES CBC decryption mode using IT.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 • **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESECB_Encrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Initializes the CRYP peripheral in AES ECB encryption mode using DMA.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESECB_Decrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Initializes the CRYP peripheral in AES ECB decryption mode using DMA.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes • **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCBC_Encrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Initializes the CRYP peripheral in AES CBC encryption mode using DMA.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCBC_Decrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Initializes the CRYP peripheral in AES CBC encryption mode using DMA.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes • **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCTR_Encrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Initializes the CRYP peripheral in AES CTR encryption mode using DMA.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_AESCTR_Decrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Initializes the CRYP peripheral in AES CTR decryption mode using DMA.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer (aligned on u32) • **Size:** Length of the plaintext buffer, must be a multiple of 16 • **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
Return values	• **HAL:** status

HAL_CRYP_InCpltCallback

Function name	**void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)**
Function description	Input transfer completed callback.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None**

HAL_CRYP_OutCpltCallback

Function name	**void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)**
Function description	Output transfer completed callback.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None**

HAL_CRYP_ErrorCallback

Function name	**void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)**
Function description	CRYP error callback.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None**

HAL_CRYP_IRQHandler

Function name	**void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)**
Function description	This function handles CRYP interrupt request.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None**

HAL_CRYP_GetState

Function name	**HAL_CRYP_STATETypeDef HAL_CRYP_GetState (CRYP_HandleTypeDef * hcryp)**
Function description	Returns the CRYP state.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **HAL:** state

13.3 CRYP Firmware driver defines

13.3.1 CRYP

AES Clear Flags

CRYP_CLEARFLAG_CCF	Computation Complete Flag Clear
CRYP_CLEARFLAG_RDERR	Read Error Clear
CRYP_CLEARFLAG_WRERR	Write Error Clear

AES Flags

CRYP_FLAG_CCF	Computation Complete Flag
CRYP_FLAG_RDERR	Read Error Flag
CRYP_FLAG_WRERR	Write Error Flag

AES Interrupts

CRYP_IT_CC	Computation Complete interrupt
CRYP_IT_ERR	Error interrupt

CRYP Algo Mode Direction

CRYP_CR_ALGOMODE_DIRECTION

CRYP_CR_ALGOMODE_AES_ECB_ENCRYPT

CRYP_CR_ALGOMODE_AES_ECB_KEYDERDECRYPT

CRYP_CR_ALGOMODE_AES_CBC_ENCRYPT

CRYP_CR_ALGOMODE_AES_CBC_KEYDERDECRYPT

CRYP_CR_ALGOMODE_AES_CTR_ENCRYPT

CRYP_CR_ALGOMODE_AES_CTR_DECRYPT

CRYP Data Type

CRYP_DATATYPE_32B

CRYP_DATATYPE_16B

CRYP_DATATYPE_8B

CRYP_DATATYPE_1B

IS_CRYP_DATATYPE

CRYP Exported Macros

__HAL_CRYP_RESET_HANDLE_STATE

Description:

- Reset CRYP handle state.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_ENABLE

Description:

- Enable/Disable the CRYP peripheral.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_DISABLE

__HAL_CRYP_SET_MODE

Description:

- Set the algorithm mode: AES-ECB, AES-CBC, AES-CTR, DES-ECB, DES-CBC,...

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __MODE__: The algorithm mode.

Return value:

- None

__HAL_CRYP_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:

  - CRYP_FLAG_CCF : Computation Complete Flag
  - CRYP_FLAG_RDERR : Read Error Flag
  - CRYP_FLAG_WRERR : Write Error Flag

Return value:

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

__HAL_CRYP_CLEAR_FLAG

Description:

- Clear the CRYP pending flag.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - CRYP_CLEARFLAG_CCF : Computation Complete Clear Flag
  - CRYP_CLEARFLAG_RDERR : Read Error Clear
  - CRYP_CLEARFLAG_WRERR : Write Error Clear

Return value:

- None

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt.

Return value:

- None

__HAL_CRYP_DISABLE_IT

Description:

- Disable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP interrupt.

Return value:

- None

__HAL_CRYP_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP interrupt source to check This parameter can be one of the following values:
  - CRYP_IT_CC : Computation Complete interrupt
  - CRYP_IT_ERR : Error interrupt (used for RDERR and WRERR)

Return value:

- State: of interruption (SET or RESET)

__HAL_CRYP_CLEAR_IT

Description:

- Clear the CRYP pending IT.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __IT__: specifies the IT to clear. This parameter can be one of the following values:
  - CRYP_CLEARFLAG_CCF : Computation Complete Clear Flag
  - CRYP_CLEARFLAG_RDERR : Read Error Clear
  - CRYP_CLEARFLAG_WRERR : Write Error Clear

Return value:

- None

14 HAL CRYP Extension Driver

14.1 CRYPEx Firmware driver API description

14.1.1 Extended features functions

This section provides callback functions:

- Computation completed.

This section contains the following APIs:

- *HAL_CRYPEx_ComputationCpltCallback()*

14.1.2 Detailed description of functions

HAL_CRYPEx_ComputationCpltCallback

Function name	**void HAL_CRYPEx_ComputationCpltCallback (CRYP_HandleTypeDef * hcryp)**
Function description	Computation completed callbacks.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None**

15 HAL DAC Generic Driver

15.1 DAC Firmware driver registers structures

15.1.1 DAC_HandleTypeDef

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

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

15.2.1 DAC Peripheral features

DAC Channels

STM32F0 devices integrates no, one or two 12-bit Digital Analog Converters. STM32L05x & STM32L06x devices have one converter (channel1) STM32L07x & STM32L08x 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 (STM32L07x/STM32L08x only)
3. Channel1 & channel2 can be used independently or simultaneously in dual mode (STM32L07x/STM32L08x only)

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_Pin9) using DAC_Trigger_Ext_IT9. The used pin (GPIOx_Pin9) must be configured in input mode.
2. Timers TRGO: STM32L05x/STM32L06x : TIM2, TIM6 and TIM21 STM32L07x/STM32L08x : TIM2, TIM3, TIM6, TIM7 and TIM21 (DAC_Trigger_T2_TRGO, DAC_Trigger_T6_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;

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

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave using HAL_DACEx_NoiseWaveGenerate()
2. Triangle wave using HAL_DACEx_TriangleWaveGenerate()

DAC data format

The DAC data format can be:

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

3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation: DAC_OUTx = VREF+ * DOR / 4095 with DOR is the Data Output Register VEF+ 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()

DMA1 requests are mapped as following:

1. DAC channel1 : mapped on DMA1 Request9 channel2 which must be already configured
2. DAC channel2 : mapped on DMA1 Request15 channel4 which must be already configured (STM32L07x/STM32L08x only) For Dual mode (STM32L07x/STM32L08x only) and specific signal Triangle and noise generation please refer to Extension Features Driver description

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_OUT1: PA4 in analog mode.
- Configure DAC_OUT2: PA5 in analog mode (STM32L07x/STM32L08x only).
- 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 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_DAC_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1 or HAL_DAC_ConvHalfCpltCallbackCh2
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1()or HAL_DAC_ConvCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1 or HAL_DAC_ConvCpltCallbackCh2
- 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_DAC_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1

or HAL_DAC_DMAUnderrunCallbackCh2 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

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_DeInit()***
- ***HAL_DAC_MspInit()***
- ***HAL_DAC_MspDeInit()***

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.
- Get result of conversion.
- Get result of dual mode conversion (STM32L07xx/STM32L08xx only)

This section contains the following APIs:

- ***HAL_DAC_Start()***
- ***HAL_DAC_Stop()***
- ***HAL_DAC_Start_DMA()***
- ***HAL_DAC_Stop_DMA()***
- ***HAL_DAC_GetValue()***
- ***HAL_DAC_IRQHandler()***
- ***HAL_DAC_ConvCpltCallbackCh1()***
- ***HAL_DAC_ConvHalfCpltCallbackCh1()***
- ***HAL_DAC_ErrorCallbackCh1()***
- ***HAL_DAC_DMAUnderrunCallbackCh1()***
- ***HAL_DAC_SetValue()***

15.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- *HAL_DAC_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()*
- *HAL_DAC_SetValue()*

15.2.7 Detailed description of functions

HAL_DAC_Init

Function name	**HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)**
Function description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **HAL:** status

HAL_DAC_DeInit

Function name	**HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)**
Function description	Deinitializes 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	Initializes 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	DeInitializes the DAC MSP.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None**

HAL_DAC_Start

Function name	**HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function description	Enables DAC and starts conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected
Return values	• **HAL:** status

HAL_DAC_Stop

Function name	**HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function description	Disables DAC and stop conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)
Return values	• **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 using DMA.
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 (STM32L07x/STM32L08x only) • **pData:** The destination peripheral Buffer address.

- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values
- **HAL:** status

HAL_DAC_Stop_DMA

Function name
HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description
Disables DAC and stop conversion of channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)

Return values
- **HAL:** status

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 (STM32L07x/STM32L08x only)
- **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.
- **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:** status

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
- **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 (STM32L07x/STM32L08x only)

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

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_ConvCpltCallbackCh1

Function name: **void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)**

Function description: Conversion complete callback in non blocking mode for Channel1.

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

Return values
- **None**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name: **void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)**

Function description: Conversion half DMA transfer callback in non blocking mode for Channel1.

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

Return values	• **None**

HAL_DAC_ErrorCallbackCh1

Function name	**void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)**
Function description	Error DAC callback for Channel1.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None**

HAL_DAC_DMAUnderrunCallbackCh1

Function name	**void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)**
Function description	DMA underrun DAC callback for channel1.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None**

HAL_DAC_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	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **sConfig:** DAC configuration structure. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)
Return values	• **HAL:** status

HAL_DAC_GetState

Function name	**HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)**
Function description	return the DAC state
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **HAL:** state

HAL_DAC_GetError

Function name	**uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)**
Function description	Return the DAC error code.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **DAC:** Error Code

15.3 DAC Firmware driver defines

15.3.1 DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

IS_DAC_CHANNEL

DAC data

IS_DAC_DATA

DAC data alignement

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

IS_DAC_ALIGN

DAC Error Code

HAL_DAC_ERROR_NONE	No error
HAL_DAC_ERROR_DMAUNDERRUNCH1	DAC channel1 DAM underrun error
HAL_DAC_ERROR_DMAUNDERRUNCH2	DAC channel2 DAM underrun error
HAL_DAC_ERROR_DMA	DMA error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE	**Description:** • Reset DAC handle state. **Parameters:** • __HANDLE__: specifies the DAC handle. **Return value:** • None
__HAL_DAC_ENABLE	**Description:** • Enable the DAC channel. **Parameters:** • __HANDLE__: specifies the DAC handle. • __DAC_CHANNEL__: specifies the DAC

channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_CHANNEL__: specifies the DAC channel.

Return value:

- None

__HAL_DAC_ENABLE_IT

__HAL_DAC_DISABLE_IT

Description:

- Disable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt.

Return value:

- None

__HAL_DAC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: DAC handle
- __INTERRUPT__: DAC interrupt source to check This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt (STM32L072xx STM32L073xx STM32L082xx STM32L083xx only)

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 FLAG.

Return value:

- None

__HAL_DAC_CLEAR_FLAG

Description:

- Clear the DAC's flag.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the 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

IS_DAC_OUTPUT_BUFFER_STATE

DAC trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T21_TRGO	TIM21 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_EXT_IT9	EXTI Line9 event selected as external conversion trigger for DAC channel
DAC_TRIGGER_SOFTWARE	Conversion started by software trigger for DAC channel
DAC_TRIGGER_T3_TRGO	TIM3 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T3_CH3	TIM3 CH3 selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel

IS_DAC_TRIGGER

16 HAL DAC Extension Driver

16.1 DACEx Firmware driver API description

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 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 Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
  - DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
  - DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
  - DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
  - DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
  - DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
  - DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
  - DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
  - DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
  - DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
  - DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
  - DAC_TRIANGLEAMPLITUDE_2047: Select max triangle

amplitude of 2047
- DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values
- **HAL:** status

HAL_DACEx_NoiseWaveGenerate

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

Function description
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 Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:
  - DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation
  - DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values
- **HAL:** status

HAL_DACEx_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.

HAL_DACEx_DualSetValue

Function name	**HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)**
Function description	Set the specified data holding register value for dual DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected • **Data1:** Data for DAC 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	• **HAL:** status
Notes	• 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**

16.2 DACEx Firmware driver defines

16.2.1 DACEx

DACEx lfsrunmask triangleamplitude

DAC_LFSRUNMASK_BIT0	Unmask DAC channel LFSR bit0 for noise wave generation
DAC_LFSRUNMASK_BITS1_0	Unmask DAC channel LFSR bit[1:0] for noise wave generation
DAC_LFSRUNMASK_BITS2_0	Unmask DAC channel LFSR bit[2:0] for noise wave generation
DAC_LFSRUNMASK_BITS3_0	Unmask DAC channel LFSR bit[3:0] for noise wave generation
DAC_LFSRUNMASK_BITS4_0	Unmask DAC channel LFSR bit[4:0] for noise wave generation
DAC_LFSRUNMASK_BITS5_0	Unmask DAC channel LFSR bit[5:0] for noise wave generation
DAC_LFSRUNMASK_BITS6_0	Unmask DAC channel LFSR bit[6:0] for noise wave generation
DAC_LFSRUNMASK_BITS7_0	Unmask DAC channel LFSR bit[7:0] for noise wave generation
DAC_LFSRUNMASK_BITS8_0	Unmask DAC channel LFSR bit[8:0] for noise wave generation
DAC_LFSRUNMASK_BITS9_0	Unmask DAC channel LFSR bit[9:0] for noise wave generation
DAC_LFSRUNMASK_BITS10_0	Unmask DAC channel LFSR bit[10:0] for noise wave generation

DAC_LFSRUNMASK_BITS11_0	Unmask DAC channel LFSR bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_1	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 1023
DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 2047
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 4095

IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE

17 HAL DMA Generic Driver

17.1 DMA Firmware driver registers structures

17.1.1 DMA_InitTypeDef

Data Fields

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

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
  Specifies the request selected for the specified channel. This parameter can be a value of ***DMA_request***
- ***uint32_t DMA_InitTypeDef::Direction***
  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_Data_transfer_direction***
- ***uint32_t DMA_InitTypeDef::PeriphInc***
  Specifies whether the Peripheral address register should be incremented or not. When Memory to Memory transfer is used, this is the Source Increment mode 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. When Memory to Memory transfer is used, this is the Destination Increment mode This parameter can be a value of ***DMA_Memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***
  Specifies the Peripheral data width. When Memory to Memory transfer is used, this is the Source Alignment format This parameter can be a value of ***DMA_Peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***
  Specifies the Memory data width. When Memory to Memory transfer is used, this is the Destination Alignment format This parameter can be a value of ***DMA_Memory_data_size***
- ***uint32_t DMA_InitTypeDef::Mode***
  Specifies the operation mode of the DMAy Channelx (Normal or Circular). 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

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***

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

17.2 DMA Firmware driver API description

17.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DMA
- De-Initialize the DMA

This section contains the following APIs:

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

17.2.2 Peripheral State 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()***

17.2.3 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()***

17.2.4 Detailed description of functions

HAL_DMA_Init

Function name	**HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)**
Function description	Initializes the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.
Parameters	• **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_DeInit

Function name	**HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)**
Function description	DeInitializes 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	Starts the DMA Transfer.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **SrcAddress:** The source memory Buffer address • **DstAddress:** The destination memory Buffer address • **DataLength:** The length of data to be transferred from source to destination
Return values	• **HAL:** status

HAL_DMA_Start_IT

Function name	**HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)**
Function description	Start the DMA Transfer with interrupt enabled.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **SrcAddress:** The source memory Buffer address • **DstAddress:** The destination memory Buffer address • **DataLength:** The length of data to be transferred from source to destination
Return values	• **HAL:** status

HAL_DMA_Abort

Function name	**HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)**
Function description	Aborts the DMA Transfer.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_Abort_IT

Function name	**HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)**
Function description	Aborts the DMA Transfer in Interrupt mode.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values	• **HAL:** status

HAL_DMA_PollForTransfer

Function name	**HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)**
Function description	Polling for transfer complete.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CompleteLevel:** Specifies the DMA level complete. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

HAL_DMA_IRQHandler

Function name	**void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)**
Function description	Handles DMA interrupt request.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **None**

HAL_DMA_GetState

Function name	**HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)**
Function description	Returns the DMA state.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** state

HAL_DMA_GetError

Function name	**uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)**
Function description	Return the DMA error code.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **DMA:** Error Code

17.3 DMA Firmware driver defines

17.3.1 DMA

DMA Data Buffer Size Check

IS_DMA_BUFFER_SIZE

DMA Data Transfer directions

DMA_PERIPH_TO_MEMORY	Peripheral to memory direction
DMA_MEMORY_TO_PERIPH	Memory to peripheral direction
DMA_MEMORY_TO_MEMORY	Memory to memory direction

IS_DMA_DIRECTION

DMA Error Codes

HAL_DMA_ERROR_NONE	No error
HAL_DMA_ERROR_TE	Transfer error
HAL_DMA_ERROR_NO_XFER	no ongoing transfer
HAL_DMA_ERROR_TIMEOUT	Timeout error

IS_DMA_ALL_INSTANCE

IS_DMA_ALL_CONTROLLER

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None.

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None.

__HAL_DMA_GET_TC_FLAG_INDEX	**Description:** • Returns the current DMA Channel transfer complete flag. **Parameters:** • __HANDLE__: DMA handle **Return value:** • The: specified transfer complete flag index.
__HAL_DMA_GET_HT_FLAG_INDEX	**Description:** • Returns the current DMA Channel half transfer complete flag. **Parameters:** • __HANDLE__: DMA handle **Return value:** • The: specified half transfer complete flag index.
__HAL_DMA_GET_TE_FLAG_INDEX	**Description:** • Returns the current DMA Channel transfer error flag. **Parameters:** • __HANDLE__: DMA handle **Return value:** • The: specified transfer error flag index.
__HAL_DMA_GET_GI_FLAG_INDEX	**Description:** • Returns the current DMA Channel Global interrupt flag. **Parameters:** • __HANDLE__: DMA handle **Return value:** • The: specified transfer error flag index.
__HAL_DMA_GET_FLAG	**Description:** • Get the DMA Channel pending flags. **Parameters:** • __HANDLE__: DMA handle • __FLAG__: Get the specified flag. This parameter can be any combination of the following values: – DMA_FLAG_TCIFx: Transfer complete flag – DMA_FLAG_HTIFx: Half transfer

complete flag
  - DMA_FLAG_TEIFx: Transfer error flag
  - DMA_ISR_GIFx: Global interrupt 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

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_TCIFx: Transfer complete flag
  - DMA_FLAG_HTIFx: Half transfer complete flag
  - DMA_FLAG_TEIFx: Transfer error flag
  - DMA_ISR_GIFx: Global interrupt flag Where x can be 0_4, 1_5, 2_6 or 3_7 to select the DMA Channel flag.

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_SOURCE

Description:

- Checks whether the specified DMA Channel interrupt is enabled or not.

Parameters:

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

Return value:

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

__HAL_DMA_GET_COUNTER

Description:

- Returns the number of remaining data units in the current 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

DMA_FLAG_TC1

DMA_FLAG_HT1

DMA_FLAG_TE1

DMA_FLAG_GL2

DMA_FLAG_TC2

DMA_FLAG_HT2

DMA_FLAG_TE2

DMA_FLAG_GL3

DMA_FLAG_TC3

DMA_FLAG_HT3

DMA_FLAG_TE3

DMA_FLAG_GL4

DMA_FLAG_TC4

DMA_FLAG_HT4

DMA_FLAG_TE4

DMA_FLAG_GL5

DMA_FLAG_TC5

DMA_FLAG_HT5

DMA_FLAG_TE5

DMA_FLAG_GL6

DMA_FLAG_TC6

DMA_FLAG_HT6

DMA_FLAG_TE6

DMA_FLAG_GL7

DMA_FLAG_TC7

DMA_FLAG_HT7

DMA_FLAG_TE7

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_TRIGGER	Index of the DMA handle used for Trigger DMA requests

DMA Interrupt Definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory Data Size Alignment

DMA_MDATAALIGN_BYTE	Memory data alignment : Byte
DMA_MDATAALIGN_HALFWORD	Memory data alignment : HalfWord
DMA_MDATAALIGN_WORD	Memory data alignment : Word

IS_DMA_MEMORY_DATA_SIZE

DMA Memory Incremented Mode

DMA_MINC_ENABLE	Memory increment mode Enable
DMA_MINC_DISABLE	Memory increment mode Disable

IS_DMA_MEMORY_INC_STATE

DMA Mode

DMA_NORMAL	Normal Mode
DMA_CIRCULAR	Circular Mode

IS_DMA_MODE

DMA Peripheral Data Size Alignment

DMA_PDATAALIGN_BYTE	Peripheral data alignment : Byte
DMA_PDATAALIGN_HALFWORD	Peripheral data alignment : HalfWord
DMA_PDATAALIGN_WORD	Peripheral data alignment : Word

IS_DMA_PERIPHERAL_DATA_SIZE

DMA Peripheral Incremented Mode

DMA_PINC_ENABLE	Peripheral increment mode Enable
DMA_PINC_DISABLE	Peripheral increment mode Disable

IS_DMA_PERIPHERAL_INC_STATE

DMA Priority Level

DMA_PRIORITY_LOW	Priority level : Low
DMA_PRIORITY_MEDIUM	Priority level : Medium
DMA_PRIORITY_HIGH	Priority level : High
DMA_PRIORITY_VERY_HIGH	Priority level : Very_High

IS_DMA_PRIORITY

DMA request defintiions

DMA_REQUEST_0

DMA_REQUEST_1

DMA_REQUEST_2

DMA_REQUEST_3

DMA_REQUEST_4

DMA_REQUEST_5

DMA_REQUEST_6

DMA_REQUEST_7

DMA_REQUEST_8

DMA_REQUEST_9

DMA_REQUEST_10

DMA_REQUEST_11

DMA_REQUEST_12

DMA_REQUEST_13

DMA_REQUEST_14

DMA_REQUEST_15

IS_DMA_ALL_REQUEST

18 HAL FIREWALL Generic Driver

18.1 FIREWALL Firmware driver registers structures

18.1.1 FIREWALL_InitTypeDef

Data Fields

- ***uint32_t CodeSegmentStartAddress***
- ***uint32_t CodeSegmentLength***
- ***uint32_t NonVDataSegmentStartAddress***
- ***uint32_t NonVDataSegmentLength***
- ***uint32_t VDataSegmentStartAddress***
- ***uint32_t VDataSegmentLength***
- ***uint32_t VolatileDataExecution***
- ***uint32_t VolatileDataShared***

Field Documentation

- ***uint32_t FIREWALL_InitTypeDef::CodeSegmentStartAddress***
  Protected code segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- ***uint32_t FIREWALL_InitTypeDef::CodeSegmentLength***
  Protected code segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- ***uint32_t FIREWALL_InitTypeDef::NonVDataSegmentStartAddress***
  Protected non-volatile data segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- ***uint32_t FIREWALL_InitTypeDef::NonVDataSegmentLength***
  Protected non-volatile data segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- ***uint32_t FIREWALL_InitTypeDef::VDataSegmentStartAddress***
  Protected volatile data segment start address. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 in order to allow a 64-byte granularity.
- ***uint32_t FIREWALL_InitTypeDef::VDataSegmentLength***
  Protected volatile data segment length in bytes. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 for the length to be a multiple of 64 bytes.
- ***uint32_t FIREWALL_InitTypeDef::VolatileDataExecution***
  Set VDE bit specifying whether or not the volatile data segment can be executed. When VDS = 1 (set by parameter VolatileDataShared), VDE bit has no meaning. This parameter can be a value of ***FIREWALL_VolatileData_Executable***
- ***uint32_t FIREWALL_InitTypeDef::VolatileDataShared***
  Set VDS bit in specifying whether or not the volatile data segment can be shared with a non-protected application code. This parameter can be a value of ***FIREWALL_VolatileData_Shared***

18.2 FIREWALL Firmware driver API description

18.2.1 How to use this driver

The FIREWALL HAL driver can be used as follows:

1. Declare a FIREWALL_InitTypeDef initialization structure.
2. Resort to HAL_FIREWALL_Config() API to initialize the Firewall

3. Enable the FIREWALL in calling HAL_FIREWALL_EnableFirewall() API
4. To ensure that any code executed outside the protected segment closes the FIREWALL, the user must set the flag FIREWALL_PRE_ARM_SET in calling __HAL_FIREWALL_PREARM_ENABLE() macro if called within a protected code segment or HAL_FIREWALL_EnablePreArmFlag() API if called outside of protected code segment after HAL_FIREWALL_Config() call.

18.2.2 Initialization and Configuration functions

This subsection provides the functions allowing to initialize the Firewall. Initialization is done by HAL_FIREWALL_Config():

- Enable the Firewall clock thru __HAL_RCC_FIREWALL_CLK_ENABLE() macro.
- Set the protected code segment address start and length.
- Set the protected non-volatile and/or volatile data segments address starts and lengths if applicable.
- Set the volatile data segment execution and sharing status.
- Length must be set to 0 for an unprotected segment.

This section contains the following APIs:

- *HAL_FIREWALL_Config()*
- *HAL_FIREWALL_GetConfig()*
- *HAL_FIREWALL_EnableFirewall()*
- *HAL_FIREWALL_EnablePreArmFlag()*
- *HAL_FIREWALL_DisablePreArmFlag()*

18.2.3 Detailed description of functions

HAL_FIREWALL_Config

Function name	**HAL_StatusTypeDef HAL_FIREWALL_Config (FIREWALL_InitTypeDef * fw_init)**
Function description	Initialize the Firewall according to the FIREWALL_InitTypeDef structure parameters.
Parameters	• **fw_init:** Firewall initialization structure
Return values	• **HAL:** status
Notes	• The API returns HAL_ERROR if the Firewall is already enabled.

HAL_FIREWALL_GetConfig

Function name	**void HAL_FIREWALL_GetConfig (FIREWALL_InitTypeDef * fw_config)**
Function description	Retrieve the Firewall configuration.
Parameters	• **fw_config:** Firewall configuration, type is same as initialization structure
Return values	• **None**
Notes	• This API can't be executed inside a code area protected by the Firewall when the Firewall is enabled • If NVDSL register is different from 0, that is, if the non volatile data segment is defined, this API can't be executed when the

Firewall is enabled.
- User should resort to __HAL_FIREWALL_GET_PREARM() macro to retrieve FPA bit status

HAL_FIREWALL_EnableFirewall

Function name	**void HAL_FIREWALL_EnableFirewall (void)**
Function description	Enable FIREWALL.
Return values	• **None**
Notes	• Firewall is enabled in clearing FWDIS bit of SYSCFG CFGR1 register. Once enabled, the Firewall cannot be disabled by software. Only a system reset can set again FWDIS bit.

HAL_FIREWALL_EnablePreArmFlag

Function name	**void HAL_FIREWALL_EnablePreArmFlag (void)**
Function description	Enable FIREWALL pre arm.
Return values	• **None**
Notes	• When FPA bit is set, any code executed outside the protected segment will close the Firewall. • This API provides the same service as __HAL_FIREWALL_PREARM_ENABLE() macro but can't be executed inside a code area protected by the Firewall. • When the Firewall is disabled, user can resort to HAL_FIREWALL_EnablePreArmFlag() API any time. • When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined), ** this API can be executed when the Firewall is closed ** when the Firewall is opened, user should resort to __HAL_FIREWALL_PREARM_ENABLE() macro instead • When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined) ** FW_CR register can be accessed only when the Firewall is opened: user should resort to __HAL_FIREWALL_PREARM_ENABLE() macro instead.

HAL_FIREWALL_DisablePreArmFlag

Function name	**void HAL_FIREWALL_DisablePreArmFlag (void)**
Function description	Disable FIREWALL pre arm.
Return values	• **None**
Notes	• When FPA bit is reset, any code executed outside the protected segment when the Firewall is opened will generate a system reset. • This API provides the same service as __HAL_FIREWALL_PREARM_DISABLE() macro but can't be executed inside a code area protected by the Firewall. • When the Firewall is disabled, user can resort to HAL_FIREWALL_EnablePreArmFlag() API any time.

- When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined), ** this API can be executed when the Firewall is closed ** when the Firewall is opened, user should resort to __HAL_FIREWALL_PREARM_DISABLE() macro instead
- When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined) ** FW_CR register can be accessed only when the Firewall is opened: user should resort to __HAL_FIREWALL_PREARM_DISABLE() macro instead.

18.3 FIREWALL Firmware driver defines

18.3.1 FIREWALL

FIREWALL Exported Macros

__HAL_FIREWALL_IS_ENABLED

Description:

- Check whether the FIREWALL is enabled or not.

Return value:

- FIREWALL: enabling status (TRUE or FALSE).

__HAL_FIREWALL_PREARM_ENABLE

Notes:

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise it generates a system reset. This macro provides the same service as HAL_FIREWALL_EnablePreArmFlag() API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_PREARM_DISABLE

Notes:

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise, it

	generates a system reset. This macro provides the same service as HAL_FIREWALL_DisablePreArmFlag() API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.
__HAL_FIREWALL_VOLATILEDATA_SHARED_ENABLE	**Notes:** • When VDS bit is set, the volatile data segment is shared with non-protected application code. It can be accessed whatever the Firewall state (opened or closed). This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.
__HAL_FIREWALL_VOLATILEDATA_SHARED_DISABLE	**Notes:** • When VDS bit is reset, the volatile data segment is not shared and cannot be hit by a non protected executable code when the Firewall is closed. If it is accessed in such a condition, a system reset is generated by the Firewall. This macro can be executed inside a code area protected by the Firewall. This macro can be executed

whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_VOLATILEDATA_EXECUTION_ENABLE

Notes:

- VDE bit is ignored when VDS is set. IF VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is set (with VDS = 0), the volatile data segment is executable. When the Firewall call is closed, a "call gate" entry procedure is required to open first the Firewall. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_VOLATILEDATA_EXECUTION_DISABLE

Notes:

- VDE bit is ignored when VDS is set. IF VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is reset (with VDS = 0), the volatile data segment cannot be executed. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0.

Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_VOLATILEDATA_SHARED

Description:

- Check whether or not the volatile data segment is shared.

Return value:

- VDS: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_VOLATILEDATA_EXECUTION

Description:

- Check whether or not the volatile data segment is declared executable.

Return value:

- VDE: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_PREARM

Description:

- Check whether or not the Firewall pre arm bit is set.

Return value:

- FPA: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

FIREWALL pre arm status

FIREWALL_PRE_ARM_RESET

FIREWALL_PRE_ARM_SET

FIREWALL volatile data segment execution status

FIREWALL_VOLATILEDATA_NOT_EXECUTABLE

FIREWALL_VOLATILEDATA_EXECUTABLE

FIREWALL volatile data segment share status

FIREWALL_VOLATILEDATA_NOT_SHARED

FIREWALL_VOLATILEDATA_SHARED

19 HAL FLASH Generic Driver

19.1 FLASH Firmware driver registers structures

19.1.1 FLASH_ProcessTypeDef

Data Fields

- ***__IO FLASH_ProcedureTypeDef ProcedureOnGoing***
- ***__IO uint32_t NbPagesToErase***
- ***__IO uint32_t Address***
- ***__IO uint32_t Page***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing***
  Internal variable to indicate which procedure is ongoing or not in IT context
- ***__IO uint32_t FLASH_ProcessTypeDef::NbPagesToErase***
  Internal variable to save the remaining sectors to erase in IT context
- ***__IO uint32_t FLASH_ProcessTypeDef::Address***
  Internal variable to save address selected for program or erase
- ***__IO uint32_t FLASH_ProcessTypeDef::Page***
  Internal variable to define the current page which is erasing
- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
  FLASH locking object
- ***__IO uint32_t FLASH_ProcessTypeDef::ErrorCode***
  FLASH error code This parameter can be a value of ***FLASH_Error_Codes***

19.2 FLASH Firmware driver API description

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 STM32L0xx 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
   - Program functions: Fast Word and Half Page(should be executed from internal SRAM).
2. DATA EEPROM Programming functions: this group includes all needed functions to erase and program the DATA EEPROM memory:
   - Lock and Unlock the DATA EEPROM interface.
   - Erase function: Erase Byte, erase HalfWord, erase Word, erase Double Word (should be executed from internal SRAM).
   - Program functions: Fast Program Byte, Fast Program Half-Word, FastProgramWord, Program Byte, Program Half-Word, Program Word and Program Double-Word (should be executed from internal SRAM).
3. 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
   - Set/Get the Read protection Level.
   - Set/Get the BOR level.
   - Get the Write protection.
   - Get the user option bytes.
4. 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
5. FLASH Interface configuration functions: this group includes the management of following features:
   - Enable/Disable the RUN PowerDown mode.
   - Enable/Disable the SLEEP PowerDown mode.
6. FLASH Peripheral State methods: this group includes the management of following features:
   - Wait for the FLASH operation
   - Get the specific FLASH error flag

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 preread buffer
- Enable/Disable the Flash power-down
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

19.2.3 Programming operation functions

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

The FLASH Memory Programming functions, includes the following functions:

- HAL_FLASH_Unlock(void);
- HAL_FLASH_Lock(void);
- HAL_FLASH_Program(uint32_t TypeProgram, uint32_t Address, uint32_t Data)
- HAL_FLASH_Program_IT(uint32_t TypeProgram, uint32_t Address, uint32_t Data)

Any operation of erase or program should follow these steps:

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

19.2.4 Option Bytes Programming functions

The FLASH_Option Bytes Programming_functions, includes the following functions:

- HAL_FLASH_OB_Unlock(void);
- HAL_FLASH_OB_Lock(void);
- HAL_FLASH_OB_Launch(void);
- HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
- HAL_FLASHEx_OBGetConfig(FLASH_OBProgramInitTypeDef *pOBInit);

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASH_OB_Unlock() function to enable the Flash option control register access.
2. Call the following functions to program the desired option bytes.
   - HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
3. Once all needed option bytes to be programmed are correctly written, call the HAL_FLASH_OB_Launch(void) function to launch the Option Bytes programming process.
4. Call the HAL_FLASH_OB_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection. As a result, these 2 options are exclusive each other.
2. To activate PCROP mode for Flash sectors(s), you need to follow the sequence below:
   - Use this function HAL_FLASHEx_AdvOBProgram with PCROPState = OB_PCROP_STATE_ENABLE.

19.2.5 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()*

19.2.6 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()*

19.2.7 Detailed description of functions

HAL_FLASH_Program

Function name	**HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)**
Function description	Program word at a specified address.
Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • **Address:** Specifie the address to be programmed. • **Data:** Specifie the data to be programmed
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_FLASH_Program_IT

Function name	**HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint32_t Data)**
Function description	Program word at a specified address with interrupt enabled.
Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • **Address:** Specifie the address to be programmed. • **Data:** Specifie the data to be programmed
Return values	• **HAL_StatusTypeDef:** HAL Status

HAL_FLASH_IRQHandler

Function name	**void HAL_FLASH_IRQHandler (void)**
Function description	This function handles FLASH interrupt request.
Return values	• **None**

HAL_FLASH_EndOfOperationCallback

Function name	**void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)**
Function description	FLASH end of operation interrupt callback.
Parameters	• **ReturnValue:** The value saved in this parameter depends on the ongoing procedure – 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 – 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	• **HAL:** Status
Notes	• 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	• **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	• **Timeout:** maximum flash operation timeout
Return values	• **HAL:** Status

19.3 FLASH Firmware driver defines

19.3.1 FLASH

FLASH Error Codes

HAL_FLASH_ERROR_NONE	No error
HAL_FLASH_ERROR_PGA	Programming alignment error
HAL_FLASH_ERROR_WRP	Write protection error
HAL_FLASH_ERROR_OPTV	Option validity error
HAL_FLASH_ERROR_SIZE	
HAL_FLASH_ERROR_RD	Read protected error
HAL_FLASH_ERROR_FWWERR	FLASH Write or Erase operation aborted
HAL_FLASH_ERROR_NOTZERO	FLASH Write operation is done in a not-erased region

FLASH Flags

FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_EOP	FLASH End of Programming flag
FLASH_FLAG_ENDHV	FLASH End of High Voltage flag
FLASH_FLAG_READY	FLASH Ready flag after low power mode
FLASH_FLAG_WRPERR	FLASH Write protected error flag
FLASH_FLAG_PGAERR	FLASH Programming Alignment error flag
FLASH_FLAG_SIZERR	FLASH Size error flag
FLASH_FLAG_OPTVERR	FLASH Option Validity error flag
FLASH_FLAG_RDERR	FLASH Read protected error flag
FLASH_FLAG_FWWERR	FLASH Write or Errase operation aborted
FLASH_FLAG_NOTZEROERR	FLASH Read protected error flag

FLASH Interrupts

__HAL_FLASH_ENABLE_IT

Description:

- Enable the specified FLASH interrupt.

Parameters:

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

Return value:

- none

__HAL_FLASH_DISABLE_IT

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP End of FLASH Operation Interrupt
  - FLASH_IT_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_ENDHV FLASH End of High Voltage flag
- FLASH_FLAG_READY FLASH Ready flag after low power mode
- FLASH_FLAG_PGAERR FLASH Programming Alignment error flag
- FLASH_FLAG_SIZERR FLASH Size error flag
- FLASH_FLAG_OPTVERR FLASH Option validity error flag (not valid with STM32L031xx/STM32L041xx)
- FLASH_FLAG_RDERR FLASH Read protected error flag
- FLASH_FLAG_WRPERR FLASH Write protected error flag
- FLASH_FLAG_FWWERR FLASH Fetch While Write Error flag
- FLASH_FLAG_NOTZEROERR Not Zero area error flag

Return value:

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

__HAL_FLASH_CLEAR_FLAG

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_PGAERR FLASH Programming Alignment error flag
  - FLASH_FLAG_SIZERR FLASH Size error flag
  - FLASH_FLAG_OPTVERR FLASH Option validity error flag (not valid with STM32L031xx/STM32L041xx)
  - FLASH_FLAG_RDERR FLASH Read protected error flag
  - FLASH_FLAG_WRPERR FLASH Write protected error flag
  - FLASH_FLAG_FWWERR FLASH Fetch While Write Error flag
  - FLASH_FLAG_NOTZEROERR Not Zero area error flag

Return value:

- none

FLASH Interrupts

FLASH_IT_EOP	End of programming interrupt source
FLASH_IT_ERR	Error interrupt source

FLASH Keys

FLASH_PDKEY1	Flash power down key1
FLASH_PDKEY2	Flash power down key2: used with FLASH_PDKEY1 to unlock the RUN_PD bit in FLASH_ACR
FLASH_PEKEY1	Flash program erase key1
FLASH_PEKEY2	Flash program erase key: used with FLASH_PEKEY2 to unlock the write access to the FLASH_PECR register and data EEPROM
FLASH_PRGKEY1	Flash program memory key1
FLASH_PRGKEY2	Flash program memory key2: used with FLASH_PRGKEY2 to unlock the program memory
FLASH_OPTKEY1	Flash option key1
FLASH_OPTKEY2	Flash option key2: used with FLASH_OPTKEY1 to unlock the write access to the option byte block

FLASH Latency

FLASH_LATENCY_0	FLASH Zero Latency cycle
FLASH_LATENCY_1	FLASH One Latency cycle

FLASH size information

FLASH_SIZE	
FLASH_PAGE_SIZE	FLASH Page Size in bytes

FLASH Type Program

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

20 HAL FLASH Extension Driver

20.1 FLASHEx Firmware driver registers structures

20.1.1 FLASH_EraseInitTypeDef

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t PageAddress***
- ***uint32_t NbPages***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase***
  TypeErase: Page Erase only. This parameter can be a value of ***FLASHEx_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::PageAddress***
  PageAddress: Initial FLASH address to be erased This parameter must be a value belonging to FLASH Programm address (depending on the devices)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***
  NbPages: Number of pages to be erased. This parameter must be a value between 1 and (max number of pages - value of Initial page)

20.1.2 FLASH_OBProgramInitTypeDef

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPState***
- ***uint32_t WRPSector***
- ***uint32_t WRPSector2***
- ***uint8_t RDPLevel***
- ***uint8_t BORLevel***
- ***uint8_t USERConfig***
- ***uint8_t BOOTBit1Config***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType***
  OptionType: Option byte to be configured. This parameter can be a value of ***FLASHEx_Option_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPState***
  WRPState: Write protection activation or deactivation. This parameter can be a value of ***FLASHEx_WRP_State***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector***
  WRPSector: This bitfield specifies the sector (s) which are write protected. This parameter can be a combination of ***FLASHEx_Option_Bytes_Write_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector2***
  WRPSector2 : This bitfield specifies the sector(s) upper Sector31 which are write protected. This parameter can be a combination of ***FLASHEx_Option_Bytes_Write_Protection2***
- ***uint8_t FLASH_OBProgramInitTypeDef::RDPLevel***
  RDPLevel: Set the read protection level. This parameter can be a value of ***FLASHEx_Option_Bytes_Read_Protection***

- ***uint8_t FLASH_OBProgramInitTypeDef::BORLevel***
  BORLevel: Set the BOR Level. This parameter can be a value of ***FLASHEx_Option_Bytes_BOR_Level***
- ***uint8_t FLASH_OBProgramInitTypeDef::USERConfig***
  USERConfig: Program the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY. This parameter can be a combination of ***FLASHEx_Option_Bytes_IWatchdog***, ***FLASHEx_Option_Bytes_nRST_STOP*** and ***FLASHEx_Option_Bytes_nRST_STDBY***
- ***uint8_t FLASH_OBProgramInitTypeDef::BOOTBit1Config***
  BOOT1Config: Together with input pad Boot0, this bit selects the boot source, flash, ram or system memory This parameter can be a value of ***FLASHEx_Option_Bytes_BOOTBit1***

20.1.3 FLASH_AdvOBProgramInitTypeDef

Data Fields

- ***uint32_t OptionType***
- ***uint32_t PCROPState***
- ***uint32_t PCROPSector***
- ***uint32_t PCROPSector2***
- ***uint16_t BootConfig***

Field Documentation

- ***uint32_t FLASH_AdvOBProgramInitTypeDef::OptionType***
  OptionType: Option byte to be configured for extension . This parameter can be a value of ***FLASHEx_OptionAdv_Type***
- ***uint32_t FLASH_AdvOBProgramInitTypeDef::PCROPState***
  PCROPState: PCROP activation or deactivation. This parameter can be a value of ***FLASHEx_PCROP_State***
- ***uint32_t FLASH_AdvOBProgramInitTypeDef::PCROPSector***
  PCROPSector : This bitfield specifies the sector(s) which are read/write protected. This parameter can be a combination of ***FLASHEx_Option_Bytes_PC_ReadWrite_Protection***
- ***uint32_t FLASH_AdvOBProgramInitTypeDef::PCROPSector2***
  PCROPSector : This bitfield specifies the sector(s) upper Sector31 which are read/write protected. This parameter can be a combination of ***FLASHEx_Option_Bytes_PC_ReadWrite_Protection2***
- ***uint16_t FLASH_AdvOBProgramInitTypeDef::BootConfig***
  BootConfig: specifies Option bytes for boot config This parameter can be a value of ***FLASHEx_Option_Bytes_BOOT***

20.2 FLASHEx Firmware driver API description

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

Any operation of erase or program should follow these steps:

1. Call the @ref HAL_FLASH_OB_Unlock() function to enable the Flash option control register access.
2. Call following function to program the desired option bytes.
   – @ref HAL_FLASHEx_OBProgram: - To Enable/Disable the desired sector write protection. - To set the desired read Protection Level. - To configure the user option Bytes: IWDG, STOP and the Standby. - To Set the BOR level.
3. Once all needed option bytes to be programmed are correctly written, call the @ref HAL_FLASH_OB_Launch(void) function to launch the Option Bytes programming process.
4. Call the @ref HAL_FLASH_OB_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection (nWRPi bits). As a result, these 2 options are exclusive each other.
2. In order to activate the PcROP (change the function of the nWRPi option bits), the WPRMOD option bit must be activated.
3. The active value of nWRPi bits is inverted when PCROP mode is active, this means: if WPRMOD = 1 and nWRPi = 1 (default value), then the user sector "i" is read/write protected.
4. To activate PCROP mode for Flash sector(s), you need to call the following function:
   – @ref HAL_FLASHEx_AdvOBProgram in selecting sectors to be read/write protected
   – @ref HAL_FLASHEx_OB_SelectPCROP to enable the read/write protection

This section contains the following APIs:

- ***HAL_FLASHEx_OBProgram()***
- ***HAL_FLASHEx_OBGetConfig()***
- ***HAL_FLASHEx_AdvOBProgram()***
- ***HAL_FLASHEx_AdvOBGetConfig()***
- ***HAL_FLASHEx_OB_SelectPCROP()***
- ***HAL_FLASHEx_OB_DeSelectPCROP()***

20.2.3 DATA EEPROM Programming functions

Any operation of erase or program should follow these steps:

1. Call the @ref HAL_FLASHEx_DATAEEPROM_Unlock() function to enable the data EEPROM access and Flash program erase control register access.
2. Call the desired function to erase or program data.
3. Call the @ref HAL_FLASHEx_DATAEEPROM_Lock() to disable the data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).

This section contains the following APIs:

- *HAL_FLASHEx_DATAEEPROM_Unlock()*
- *HAL_FLASHEx_DATAEEPROM_Lock()*
- *HAL_FLASHEx_DATAEEPROM_Erase()*
- *HAL_FLASHEx_DATAEEPROM_Program()*
- *HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram()*
- *HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram()*

20.2.4 Detailed description of functions

HAL_FLASHEx_Erase

Function name	**HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)**
Function description	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 page erase of the specified FLASH memory pages with interrupt enabled.
Parameters	• **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
Return values	• **HAL_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) End of erase is done when HAL_FLASH_EndOfOperationCallback is called with parameter 0xFFFFFFFF

HAL_FLASHEx_OBProgram

Function name	**HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)**
Function description	Program option bytes.
Parameters	• **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	• **HAL_StatusTypeDef:** HAL Status

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_AdvOBProgram

Function name	**HAL_StatusTypeDef HAL_FLASHEx_AdvOBProgram (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)**
Function description	Program option bytes.
Parameters	• **pAdvOBInit:** pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.
Return values	• **HAL_StatusTypeDef:** HAL Status

HAL_FLASHEx_AdvOBGetConfig

Function name	**void HAL_FLASHEx_AdvOBGetConfig (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)**
Function description	Get the OBEX byte configuration.
Parameters	• **pAdvOBInit:** pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.
Return values	• **None**

HAL_FLASHEx_OB_SelectPCROP

Function name	**HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP (void)**
Function description	Select the Protection Mode (WPRMOD).
Return values	• **HAL:** status
Notes	• Once WPRMOD bit is active, unprotection of a protected sector is not possible

- Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag

HAL_FLASHEx_OB_DeSelectPCROP

Function name	**HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP (void)**
Function description	Deselect the Protection Mode (WPRMOD).
Return values	• **HAL:** status
Notes	• Once WPRMOD bit is active, unprotection of a protected sector is not possible • Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag

HAL_FLASHEx_DATAEEPROM_Unlock

Function name	**HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Unlock (void)**
Function description	Unlocks the data memory and FLASH_PECR register access.
Return values	• **HAL_StatusTypeDef:** HAL Status

HAL_FLASHEx_DATAEEPROM_Lock

Function name	**HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Lock (void)**
Function description	Locks the Data memory and FLASH_PECR register access.
Return values	• **HAL_StatusTypeDef:** HAL Status

HAL_FLASHEx_DATAEEPROM_Erase

Function name	**HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Erase (uint32_t Address)**
Function description	Erase a word in data memory.
Parameters	• **Address:** specifies the address to be erased.
Return values	• **HAL_StatusTypeDef:** HAL Status
Notes	• To correctly run this function, the HAL_FLASHEx_DATAEEPROM_Unlock() function must be called before. Call the HAL_FLASHEx_DATAEEPROM_Lock() to the data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).

HAL_FLASHEx_DATAEEPROM_Program

Function name	**HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)**
Function description	Program word at a specified address.
Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASHEx Type Program Data • **Address:** specifie the address to be programmed. • **Data:** specifie the data to be programmed
Return values	• **HAL_StatusTypeDef:** HAL Status
Notes	• To correctly run this function, the HAL_FLASHEx_DATAEEPROM_Unlock() function must be called before. Call the HAL_FLASHEx_DATAEEPROM_Unlock() to he data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation). • The function HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram() can be called before this function to configure the Fixed Time Programming.

HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram

Function name	**void HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram (void)**
Function description	Enable DATA EEPROM fixed Time programming (2*Tprog).
Return values	• **None**

HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram

Function name	**void HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram (void)**
Function description	Disables DATA EEPROM fixed Time programming (2*Tprog).
Return values	• **None**

20.3 FLASHEx Firmware driver defines

20.3.1 FLASHEx

FLASHEx Address

IS_FLASH_DATA_ADDRESS

IS_FLASH_DATA_BANK1_ADDRESS

IS_FLASH_DATA_BANK2_ADDRESS

IS_FLASH_PROGRAM_ADDRESS

IS_FLASH_PROGRAM_BANK1_ADDRESS

IS_FLASH_PROGRAM_BANK2_ADDRESS

IS_NBPAGES

FLASHEx Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- __LATENCY__: FLASH Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0 FLASH Zero Latency cycle
  - FLASH_LATENCY_1 FLASH One Latency cycle

Return value:

- none

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- FLASH: Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0 FLASH Zero Latency cycle
  - FLASH_LATENCY_1 FLASH One Latency cycle

__HAL_FLASH_PREFETCH_BUFFER_ENABLE

Description:

- Enable the FLASH prefetch buffer.

Return value:

- none

__HAL_FLASH_PREFETCH_BUFFER_DISABLE

Description:

- Disable the FLASH prefetch buffer.

Return value:

- none

__HAL_FLASH_BUFFER_CACHE_ENABLE

Description:

- Enable the FLASH Buffer cache.

	Return value: • none
__HAL_FLASH_BUFFER_CACHE_DISABLE	**Description:** • Disable the FLASH Buffer cache. **Return value:** • none
__HAL_FLASH_PREREAD_BUFFER_ENABLE	**Description:** • Enable the FLASH preread buffer. **Return value:** • none
__HAL_FLASH_PREREAD_BUFFER_DISABLE	**Description:** • Disable the FLASH preread buffer. **Return value:** • none
__HAL_FLASH_SLEEP_POWERDOWN_ENABLE	**Description:** • Enable the FLASH power down during Sleep mode. **Return value:** • none
__HAL_FLASH_SLEEP_POWERDOWN_DISABLE	**Description:** • Disable the FLASH power down during Sleep mode. **Return value:** • none
__HAL_FLASH_POWER_DOWN_ENABLE	**Notes:** • Writing this bit to 0 this bit, automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.
__HAL_FLASH_POWER_DOWN_DISABLE	**Notes:** • Writing this bit to 0 this bit, automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.

FLASHEx Option Advanced Type

OPTIONBYTE_PCROP	PCROP option byte configuration
OPTIONBYTE_BOOTCONFIG	BOOTConfig option byte configuration

FLASHEx Option Bytes BOOT

OB_BOOT_BANK1	At startup, if boot pin 0 and BOOT1 bit are set in boot from user Flash position and this parameter is selected the device will boot from Bank 1 (Default)
OB_BOOT_BANK2	At startup, if boot pin 0 and BOOT1 bit are set in boot from user Flash position and this parameter is selected the device will boot from Bank 2

FLASH Option Bytes BOOT Bit1 Setup

OB_BOOT_BIT1_RESET	BOOT Bit 1 Reset
OB_BOOT_BIT1_SET	BOOT Bit 1 Set

FLASHEx Option Bytes BOR Level

OB_BOR_OFF	BOR is disabled at power down, the reset is asserted when the VDD power supply reaches the PDR(Power Down Reset) threshold (1.5V)
OB_BOR_LEVEL1	BOR Reset threshold levels for 1.7V - 1.8V VDD power supply
OB_BOR_LEVEL2	BOR Reset threshold levels for 1.9V - 2.0V VDD power supply
OB_BOR_LEVEL3	BOR Reset threshold levels for 2.3V - 2.4V VDD power supply
OB_BOR_LEVEL4	BOR Reset threshold levels for 2.55V - 2.65V VDD power supply
OB_BOR_LEVEL5	BOR Reset threshold levels for 2.8V - 2.9V VDD power supply

FLASHEx Option Bytes IWatchdog

OB_IWDG_SW	Software WDG selected
OB_IWDG_HW	Hardware WDG selected

FLASHEx Option Bytes nRST_STDBY

OB_STDBY_NORST	No reset generated when entering in STANDBY
OB_STDBY_RST	Reset generated when entering in STANDBY

FLASHEx Option Bytes nRST_STOP

OB_STOP_NORST	No reset generated when entering in STOP
OB_STOP_RST	Reset generated when entering in STOP

FLASH Option Bytes PC Read/Write Protection

OB_PCROP_Pages0to31

OB_PCROP_Pages32to63

OB_PCROP_Pages64to95

OB_PCROP_Pages96to127

OB_PCROP_Pages128to159

OB_PCROP_Pages160to191
OB_PCROP_Pages192to223
OB_PCROP_Pages224to255
OB_PCROP_Pages256to287
OB_PCROP_Pages288to319
OB_PCROP_Pages320to351
OB_PCROP_Pages352to383
OB_PCROP_Pages384to415
OB_PCROP_Pages416to447
OB_PCROP_Pages448to479
OB_PCROP_Pages480to511
OB_PCROP_Pages512to543
OB_PCROP_Pages544to575
OB_PCROP_Pages576to607
OB_PCROP_Pages608to639
OB_PCROP_Pages640to671
OB_PCROP_Pages672to703
OB_PCROP_Pages704to735
OB_PCROP_Pages736to767
OB_PCROP_Pages768to799
OB_PCROP_Pages800to831
OB_PCROP_Pages832to863
OB_PCROP_Pages864to895
OB_PCROP_Pages896to927
OB_PCROP_Pages928to959
OB_PCROP_Pages960to991
OB_PCROP_Pages992to1023
OB_PCROP_AllPages PC Read/Write protection of all Sectors

FLASH Option Bytes PC Read/Write Protection (Sector 2)

OB_PCROP2_Pages1024to1055
OB_PCROP2_Pages1056to1087
OB_PCROP2_Pages1088to1119
OB_PCROP2_Pages1120to1151
OB_PCROP2_Pages1152to1183
OB_PCROP2_Pages1184to1215
OB_PCROP2_Pages1216to1247

OB_PCROP2_Pages1248to1279

OB_PCROP2_Pages1280to1311

OB_PCROP2_Pages1312to1343

OB_PCROP2_Pages1344to1375

OB_PCROP2_Pages1376to1407

OB_PCROP2_Pages1408to1439

OB_PCROP2_Pages1440to1471

OB_PCROP2_Pages1472to1503

OB_PCROP2_Pages1504to1535

OB_PCROP2_AllPages PC Read/Write protection of all Sectors PCROP2

FLASHEx Option Bytes Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

OB_RDP_LEVEL_2

FLASH Option Bytes Write ProtectionP

OB_WRP_Pages0to31

OB_WRP_Pages32to63

OB_WRP_Pages64to95

OB_WRP_Pages96to127

OB_WRP_Pages128to159

OB_WRP_Pages160to191

OB_WRP_Pages192to223

OB_WRP_Pages224to255

OB_WRP_Pages256to287

OB_WRP_Pages288to319

OB_WRP_Pages320to351

OB_WRP_Pages352to383

OB_WRP_Pages384to415

OB_WRP_Pages416to447

OB_WRP_Pages448to479

OB_WRP_Pages480to511

OB_WRP_Pages512to543

OB_WRP_Pages544to575

OB_WRP_Pages576to607

OB_WRP_Pages608to639

OB_WRP_Pages640to671

OB_WRP_Pages672to703

OB_WRP_Pages704to735

OB_WRP_Pages736to767

OB_WRP_Pages768to799

OB_WRP_Pages800to831

OB_WRP_Pages832to863

OB_WRP_Pages864to895

OB_WRP_Pages896to927

OB_WRP_Pages928to959

OB_WRP_Pages960to991

OB_WRP_Pages992to1023

OB_WRP_AllPages Write protection of all Sectors

FLASH Option Bytes Write Protection

OB_WRP2_Pages1024to1055

OB_WRP2_Pages1056to1087

OB_WRP2_Pages1088to1119

OB_WRP2_Pages1120to1151

OB_WRP2_Pages1152to1183

OB_WRP2_Pages1184to1215

OB_WRP2_Pages1216to1247

OB_WRP2_Pages1248to1279

OB_WRP2_Pages1280to1311

OB_WRP2_Pages1312to1343

OB_WRP2_Pages1344to1375

OB_WRP2_Pages1376to1407

OB_WRP2_Pages1408to1439

OB_WRP2_Pages1440to1471

OB_WRP2_Pages1472to1503

OB_WRP2_Pages1504to1535

OB_WRP2_AllPages Write protection of all Sectors WRP2

FLASHEx Option Type

OPTIONBYTE_WRP WRP option byte configuration

OPTIONBYTE_RDP RDP option byte configuration

OPTIONBYTE_USER USER option byte configuration

OPTIONBYTE_BOR BOR option byte configuration

OPTIONBYTE_BOOT_BIT1 BOOT PIN1 option byte configuration

FLASHEx PCROP State

OB_PCROP_STATE_DISABLE	Disable PCROP for selected sectors
OB_PCROP_STATE_ENABLE	Enable PCROP for selected sectors

FLASHEx Selection Protection Mode

OB_PCROP_DESELECTED	Disabled PCROP, nWPRi bits used for Write Protection on sector i
OB_PCROP_SELECTED	Enable PCROP, nWPRi bits used for PCRoP Protection on sector i

FLASHEx_Type_Erase

FLASH_TYPEERASE_PAGES	Page erase only

FLASHEx Type Program Data

FLASH_TYPEPROGRAMDATA_BYTE	Program byte (8-bit) at a specified address.
FLASH_TYPEPROGRAMDATA_HALFWORD	Program a half-word (16-bit) at a specified address.
FLASH_TYPEPROGRAMDATA_WORD	Program a word (32-bit) at a specified address.

FLASHEx WRP State

OB_WRPSTATE_DISABLE	Disable the write protection of the desired sectors
OB_WRPSTATE_ENABLE	Enable the write protection of the desired sectors

21 HAL FLASH__RAMFUNC Generic Driver

21.1 FLASH__RAMFUNC Firmware driver API description

21.1.1 Peripheral errors functions

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

This section contains the following APIs:

- *HAL_FLASHEx_GetError()*

21.1.2 Detailed description of functions

HAL_FLASHEx_EnableRunPowerDown

Function name	**__RAM_FUNC HAL_FLASHEx_EnableRunPowerDown (void)**
Function description	Enable the power down mode during RUN mode.
Return values	• **HAL:** status
Notes	• This function can be used only when the user code is running from Internal SRAM.

HAL_FLASHEx_DisableRunPowerDown

Function name	**__RAM_FUNC HAL_FLASHEx_DisableRunPowerDown (void)**
Function description	Disable the power down mode during RUN mode.
Return values	• **HAL:** status
Notes	• This function can be used only when the user code is running from Internal SRAM.

HAL_FLASHEx_EraseParallelPage

Function name	**__RAM_FUNC HAL_FLASHEx_EraseParallelPage (uint32_t Page_Address1, uint32_t Page_Address2)**
Function description	Erases a specified 2 pages in program memory in parallel.
Parameters	• **Page_Address1:** The page address in program memory to be erased in the first Bank (BANK1). This parameter should be between FLASH_BASE and FLASH_BANK1_END. • **Page_Address2:** The page address in program memory to be erased in the second Bank (BANK2). This parameter should be between FLASH_BANK2_BASE and FLASH_BANK2_END.
Return values	• **HAL:** status
Notes	• This function can be used only for STM32L07xxx/STM32L08xxx devices. 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).
- A Page is erased in the Program memory only if the address to load is the start address of a page (multiple of FLASH_PAGE_SIZE bytes).

HAL_FLASHEx_ProgramParallelHalfPage

Function name	**__RAM_FUNC HAL_FLASHEx_ProgramParallelHalfPage (uint32_t Address1, uint32_t * pBuffer1, uint32_t Address2, uint32_t * pBuffer2)**
Function description	Program 2 half pages in program memory in parallel (half page size is 16 Words).
Parameters	• **Address1:** specifies the first address to be written in the first bank (BANK1). This parameter should be between FLASH_BASE and (FLASH_BANK1_END - FLASH_PAGE_SIZE). • **pBuffer1:** pointer to the buffer containing the data to be written to the first half page in the first bank. • **Address2:** specifies the second address to be written in the second bank (BANK2). This parameter should be between FLASH_BANK2_BASE and (FLASH_BANK2_END - FLASH_PAGE_SIZE). • **pBuffer2:** pointer to the buffer containing the data to be written to the second half page in the second bank.
Return values	• **HAL:** status
Notes	• This function can be used only for STM32L07xxx/STM32L08xxx devices. • 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). • Half page write is possible only from SRAM. • A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 64 bytes) and the 15 remaining words to load are in the same half page. • During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.). • If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write operation from the beginning.

HAL_FLASHEx_HalfPageProgram

Function name	**__RAM_FUNC HAL_FLASHEx_HalfPageProgram (uint32_t Address, uint32_t * pBuffer)**
Function description	Program a half page in program memory.
Parameters	• **Address:** specifies the address to be written.

	• **pBuffer:** pointer to the buffer containing the data to be written to the half page.
Return values	• **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) • Half page write is possible only from SRAM. • A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 64 bytes) and the 15 remaining words to load are in the same half page. • During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.). • If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write operation from the beginning.

HAL_FLASHEx_GetError

Function name	**__RAM_FUNC HAL_FLASHEx_GetError (uint32_t * Error)**
Function description	Get the specific FLASH errors flag.
Parameters	• **Error:** pointer is the error value. It can be a mixed of: – HAL_FLASH_ERROR_RD FLASH Read Protection error flag (PCROP) – HAL_FLASH_ERROR_SIZE FLASH Programming Parallelism error flag – HAL_FLASH_ERROR_PGA FLASH Programming Alignment error flag – HAL_FLASH_ERROR_WRP FLASH Write protected error flag – HAL_FLASH_ERROR_OPTV FLASH Option valid error flag – HAL_FLASH_ERROR_FWWERR FLASH Write or Erase operation aborted – HAL_FLASH_ERROR_NOTZERO FLASH Write operation is done in a not-erased region
Return values	• **HAL:** Status

22 HAL GPIO Generic Driver

22.1 GPIO Firmware driver registers structures

22.1.1 GPIO_InitTypeDef

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be a combination of ***GPIO_pins_define***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode_define***
- ***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_define***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed_define***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins This parameter can be a value of ***GPIOEx_Alternate_function_selection***

22.2 GPIO Firmware driver API description

22.2.1 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.

22.2.2 How to use this driver

1. Enable the GPIO IOPORT clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   - Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. HAL_GPIO_DeInit allows to set register values to their reset value. This function is also to be used when unconfiguring pin which was used as an external interrupt or in event mode. That is the only way to reset the corresponding bit in EXTI & SYSCFG registers.
5. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
6. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
7. To lock pin configuration until next reset use HAL_GPIO_LockPin().
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
9. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
10. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

22.2.3 Initialization and de-initialization functions

This section contains the following APIs:

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

22.2.4 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()*

22.2.5 Detailed description of functions

HAL_GPIO_Init

Function name	**void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)**
Function description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	• **GPIOx:** where x can be (A..E and H) to select the GPIO peripheral for STM32L0XX family devices. Note that GPIOE is not available on all devices. • **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• **None**

HAL_GPIO_DeInit

Function name	**void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)**
Function description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	• **GPIOx:** where x can be (A..E and H) to select the GPIO peripheral for STM32L0XX family devices. Note that GPIOE is not available on all devices. • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.
Return values	• **None**

HAL_GPIO_ReadPin

Function name	**GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Reads the specified input port pin.
Parameters	• **GPIOx:** where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices. • **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.
Return values	• **The:** input port pin value.

HAL_GPIO_WritePin

Function name	**void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)**
Function description	Sets or clears the selected data port bit.
Parameters	• **GPIOx:** where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices. • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). All port bits are not necessarily available on all GPIOs. • **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pin GPIO_PIN_SET: to set the port pin
Return values	• **None**
Notes	• This function uses GPIOx_BSRR register 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	Toggles the specified GPIO pins.
Parameters	• **GPIOx:** Where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices. All port bits are not necessarily available on all GPIOs. • **GPIO_Pin:** Specifies the pins 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..E and H) to select the GPIO peripheral for STM32L0xx family. Note that GPIOE is not available on all devices. • **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.
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	This function handles EXTI interrupt request.
Parameters	• **GPIO_Pin:** Specifies the pins connected to the 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 callbacks.
Parameters	• **GPIO_Pin:** Specifies the pins connected to the EXTI line.
Return values	• **None**

22.3 GPIO Firmware driver defines

22.3.1 GPIO

GPIO Exported Constants

GPIO_PIN_MASK

IS_GPIO_PIN

IS_GPIO_MODE

IS_GPIO_SPEED

IS_GPIO_PULL

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG	**Description:** • Checks whether the specified EXTI line flag is set or not. **Parameters:** • __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15) **Return value:** • The: new state of __EXTI_LINE__ (SET or RESET).
__HAL_GPIO_EXTI_CLEAR_FLAG	**Description:** • Clears 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:** • Checks whether the specified EXTI line is asserted or not. **Parameters:** • __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15) **Return value:** • The: new state of __EXTI_LINE__ (SET or RESET).
__HAL_GPIO_EXTI_CLEAR_IT	**Description:** • Clears 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:** • Generates 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 Exported Types

IS_GPIO_PIN_ACTION

Mode definition

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

Pin definition

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

Pull definition

GPIO_NOPULL	No Pull-up or Pull-down activation
GPIO_PULLUP	Pull-up activation
GPIO_PULLDOWN	Pull-down activation

Speed definition

GPIO_SPEED_FREQ_LOW	range up to 0.4 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_MEDIUM	range 0.4 MHz to 2 MHz, please refer to the product

	datasheet
GPIO_SPEED_FREQ_HIGH	range 2 MHz to 10 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_VERY_HIGH	range 10 MHz to 35 MHz, please refer to the product datasheet

23 HAL GPIO Extension Driver

23.1 GPIOEx Firmware driver defines

23.1.1 GPIOEx

Alternate function selection

GPIO_AF0_EVENTOUT

GPIO_AF0_TIM21

GPIO_AF0_SPI1

GPIO_AF0_MCO

GPIO_AF0_SWDIO

GPIO_AF0_SWCLK

GPIO_AF0_USART1

GPIO_AF0_SPI2

GPIO_AF0_LPTIM1

GPIO_AF0_TIM22

GPIO_AF0_LPUART1

GPIO_AF0_USART2

GPIO_AF0_TIM2

GPIO_AF0_USB

GPIO_AF1_I2C1

GPIO_AF1_SPI2

GPIO_AF1_TIM21

GPIO_AF1_LCD

GPIO_AF2_TIM2

GPIO_AF2_TIM3

GPIO_AF2_EVENTOUT

GPIO_AF2_LPTIM1

GPIO_AF2_LPUART1

GPIO_AF2_MCO

GPIO_AF2_RTC

GPIO_AF2_SPI2

GPIO_AF2_USART5

GPIO_AF2_SPI1

GPIO_AF2_USB

GPIO_AF3_EVENTOUT

GPIO_AF3_I2C1

GPIO_AF3_TSC

GPIO_AF4_USART2

GPIO_AF4_LPUART1

GPIO_AF4_USART1

GPIO_AF4_EVENTOUT

GPIO_AF4_TIM22

GPIO_AF4_TIM3

GPIO_AF4_I2C1

GPIO_AF5_TIM2

GPIO_AF5_TIM21

GPIO_AF5_TIM22

GPIO_AF5_USART1

GPIO_AF5_SPI2

GPIO_AF5_I2C2

GPIO_AF6_USART4

GPIO_AF6_LPUART1

GPIO_AF6_EVENTOUT

GPIO_AF6_I2C1

GPIO_AF6_I2C2

GPIO_AF6_USART5

GPIO_AF6_TIM21

GPIO_AF7_COMP1

GPIO_AF7_COMP2

GPIO_AF7_I2C3

GPIO_AF7_LPUART1

Pin available

GPIOA_PIN_AVAILABLE

GPIOB_PIN_AVAILABLE

GPIOC_PIN_AVAILABLE

GPIOD_PIN_AVAILABLE

GPIOE_PIN_AVAILABLE

GPIOH_PIN_AVAILABLE

24 HAL I2C Generic Driver

24.1 I2C Firmware driver registers structures

24.1.1 I2C_InitTypeDef

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***

24.1.2 __I2C_HandleTypeDef

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_TypeDef* __I2C_HandleTypeDef::Instance***
  I2C registers base address
- ***I2C_InitTypeDef __I2C_HandleTypeDef::Init***
  I2C communication parameters
- ***uint8_t* __I2C_HandleTypeDef::pBuffPtr***
  Pointer to I2C transfer buffer
- ***uint16_t __I2C_HandleTypeDef::XferSize***
  I2C transfer size
- ***__IO uint16_t __I2C_HandleTypeDef::XferCount***
  I2C transfer counter
- ***__IO uint32_t __I2C_HandleTypeDef::XferOptions***
  I2C sequantial transfer options, this parameter can be a value of ***I2C_XFEROPTIONS***
- ***__IO uint32_t __I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state
- ***HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)***
  I2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef __I2C_HandleTypeDef::Lock***
  I2C locking object
- ***__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t __I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t __I2C_HandleTypeDef::AddrEventCount***
  I2C Address Event counter

24.2 I2C Firmware driver API description

24.2.1 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

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 sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
- 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

You can refer to the I2C HAL driver header file for more useful macros

24.2.2 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()***

24.2.3 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()***

24.2.4 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()*

24.2.5 Detailed description of functions

HAL_I2C_Init

Function name	**HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)**
Function description	Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and initialize the associated handle.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **HAL:** status

HAL_I2C_DeInit

Function name	**HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)**
Function description	DeInitialize the I2C peripheral.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **HAL:** status

HAL_I2C_MspInit

Function name	**void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)**
Function description	Initialize the I2C MSP.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None**

HAL_I2C_MspDeInit

Function name	**void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)**
Function description	DeInitialize the I2C MSP.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None**

HAL_I2C_Master_Transmit

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Transmits in master mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Master_Receive

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receives in master mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Slave_Transmit

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Transmits in slave mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Slave_Receive

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive in slave mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Mem_Write

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Write an amount of data in blocking mode to a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be 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 • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Mem_Read

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Read an amount of data in blocking mode from a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be 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

	• **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_IsDeviceReady

Function name	**HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)**
Function description	Checks if target device is ready for communication.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **Trials:** Number of trials • **Timeout:** Timeout duration
Return values	• **HAL:** status
Notes	• This function is used with Memory devices

HAL_I2C_Master_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**
Function description	Transmit in master mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Master_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**
Function description	Receive in master mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer

- **Size:** Amount of data to be sent

Return values	• **HAL:** status

HAL_I2C_Slave_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function description	Transmit in slave mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Slave_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function description	Receive in slave mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Mem_Write_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**
Function description	Write an amount of data in non-blocking mode with Interrupt to a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be 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	• **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_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	• **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 • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status
Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_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	• **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 • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status
Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_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	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status
Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_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	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status
Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EnableListen_IT

Function name	**HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)**

Function description	Enable the Address listen mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **HAL:** status

HAL_I2C_DisableListen_IT

Function name	**HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)**
Function description	Disable the Address listen mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
Return values	• **HAL:** status

HAL_I2C_Master_Abort_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)**
Function description	Abort a master I2C IT or DMA process communication with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
Return values	• **HAL:** status

HAL_I2C_Master_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**
Function description	Transmit in master mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Master_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**

Function description	Receive in master mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Slave_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function description	Transmit in slave mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Slave_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function description	Receive in slave mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**
Function description	Write an amount of data in non-blocking mode with DMA to a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be 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

24.3 I2C Firmware driver defines

24.3.1 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:** • Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_GET_IT_SOURCE

Description:

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

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

__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_TXIS Transmit interrupt status
  - I2C_FLAG_RXNE Receive data register not empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_TC Transfer complete (master mode)
  - I2C_FLAG_TCR Transfer complete reload
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception

- I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
- I2C_FLAG_ALERT SMBus alert
- I2C_FLAG_BUSY Bus busy
- I2C_FLAG_DIR Transfer direction (slave mode)

Return value:

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

__HAL_I2C_CLEAR_FLAG

Description:

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_I2C_ENABLE

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_GENERATE_NACK

Description:

- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C General Call Addressing Mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C No-Stretch Mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Own Address2 Masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

I2C Reload End Mode

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C Start or Stop Mode

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

25 HAL I2C Extension Driver

25.1 I2CEx Firmware driver API description

25.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32L0xx 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

25.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_ConfigAnalogFilter()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_ConfigDigitalFilter()
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
   – HAL_I2CEx_EnableWakeUp()
   – HAL_I2CEx_DisableWakeUp()
4. Configure the enable or disable of fast mode plus driving capability using the functions :
   – HAL_I2CEx_EnableFastModePlus()
   – HAL_I2CEx_DisableFastModePlus()

25.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters
- Configure Wake Up Feature

This section contains the following APIs:

- ***HAL_I2CEx_ConfigAnalogFilter()***
- ***HAL_I2CEx_ConfigDigitalFilter()***
- ***HAL_I2CEx_EnableWakeUp()***
- ***HAL_I2CEx_DisableWakeUp()***
- ***HAL_I2CEx_EnableFastModePlus()***
- ***HAL_I2CEx_DisableFastModePlus()***

25.1.4 Detailed description of functions

HAL_I2CEx_ConfigAnalogFilter

Function name	**HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)**
Function description	Configure I2C Analog noise filter.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. • **AnalogFilter:** New state of the Analog filter.

Return values	• **HAL:** status

HAL_I2CEx_ConfigDigitalFilter

Function name	**HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)**
Function description	Configure I2C Digital noise filter.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. • **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.
Return values	• **HAL:** status

HAL_I2CEx_EnableWakeUp

Function name	**HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)**
Function description	Enable I2C wakeup from stop mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	• **HAL:** status

HAL_I2CEx_DisableWakeUp

Function name	**HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)**
Function description	Disable I2C wakeup from stop mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	• **HAL:** status

HAL_I2CEx_EnableFastModePlus

Function name	**void HAL_I2CEx_EnableFastModePlus (uint32_t ConfigFastModePlus)**
Function description	Enable the I2C fast mode plus driving capability.
Parameters	• **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values
Return values	• **None**
Notes	• For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9. • For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C1 parameter. • For all I2C2 pins fast mode plus driving capability can be

enabled only by using I2C_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C3 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. • For all I2C3 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C3 parameter.

25.2 I2CEx Firmware driver defines

25.2.1 I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FMP_NOT_SUPPORTED	Fast Mode Plus not supported
I2C_FASTMODEPLUS_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
I2C_FASTMODEPLUS_I2C3	Enable Fast Mode Plus on I2C3 pins

26 HAL I2S Generic Driver

26.1 I2S Firmware driver registers structures

26.1.1 I2S_InitTypeDef

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***

26.1.2 I2S_HandleTypeDef

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_InitTypeDef I2S_HandleTypeDef::Init***
- ***uint16_t* I2S_HandleTypeDef::pTxBuffPtr***
- ***__IO uint16_t I2S_HandleTypeDef::TxXferSize***
- ***__IO uint16_t I2S_HandleTypeDef::TxXferCount***
- ***uint16_t* I2S_HandleTypeDef::pRxBuffPtr***
- ***__IO uint16_t I2S_HandleTypeDef::RxXferSize***
- ***__IO uint16_t I2S_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx***
- ***__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock***
- ***__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State***
- ***__IO uint32_t I2S_HandleTypeDef::ErrorCode***

26.2 I2S Firmware driver API description

26.2.1 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.
   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 initilalized 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. 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. Make sure that either: External clock source is configured after setting correctly the define constant HSE_VALUE in the stm32l0xx_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 USART 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

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

26.2.2 Initialization and de-initialization functions

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

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

This section contains the following APIs:

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

26.2.3 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()*

26.2.4 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()*

26.2.5 Detailed description of functions

HAL_I2S_Init

Function name	**HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)**
Function description	Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **HAL:** status

HAL_I2S_DeInit

Function name	**HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)**
Function description	DeInitializes the I2S peripheral.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **HAL:** status

HAL_I2S_MspInit

Function name	**void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)**
Function description	I2S MSP Init.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None**

HAL_I2S_MspDeInit

Function name	**void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)**
Function description	I2S MSP DeInit.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None**

HAL_I2S_Transmit

Function name	**HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Transmit an amount of data in blocking mode.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to data buffer. • **Size:** number of data sample to be sent: • **Timeout:** Timeout duration
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 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). • This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

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.
- This function can use an Audio Frequency up to 44KHz when I2S Clock Source is 32MHz

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). • This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

HAL_I2S_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)**
Function description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to the Receive data buffer. • **Size:** number of data sample to be sent:
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 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.

- This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

HAL_I2S_IRQHandler

Function name	**void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)**
Function description	This function handles I2S interrupt request.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None**

HAL_I2S_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to the Transmit data buffer. • **Size:** number of data sample to be sent:
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 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	Stops 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_TxHalfCpltCallback

Function name	**void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)**
Function description	Tx Transfer Half completed callbacks.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None**

HAL_I2S_TxCpltCallback

Function name	**void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)**
Function description	Tx Transfer completed callbacks.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values	• **None**

HAL_I2S_RxHalfCpltCallback

Function name	**void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)**
Function description	Rx Transfer half completed callbacks.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None**

HAL_I2S_RxCpltCallback

Function name	**void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)**
Function description	Rx Transfer completed callbacks.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None**

HAL_I2S_ErrorCallback

Function name	**void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)**
Function description	I2S error callbacks.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None**

HAL_I2S_GetState

Function name	**HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)**
Function description	Return the I2S state.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **HAL:** state

HAL_I2S_GetError

Function name	**uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)**
Function description	Return the I2S error code.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **I2S:** Error Code

26.3 I2S Firmware driver defines

26.3.1 I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error Code

HAL_I2S_ERROR_NONE	No error
HAL_I2S_ERROR_UDR	I2S Underrun error
HAL_I2S_ERROR_OVR	I2S Overrun error
HAL_I2S_ERROR_FRE	I2S Frame format error
HAL_I2S_ERROR_DMA	DMA transfer error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

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

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

Description:

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

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE_IT

Description:

- Enable the specified I2S interrupts.

Parameters:

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

Return value:

- None

__HAL_I2S_DISABLE_IT

Description:

- Disable the specified I2S interrupts.

Parameters:

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

Return value:

- None

__HAL_I2S_GET_IT_SOURCE

Description:

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

Parameters:

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

Return value:

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

__HAL_I2S_GET_FLAG

Description:

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

Parameters:

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

Return value:

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

__HAL_I2S_CLEAR_OVRFLAG

Description:

- Clears the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_UDRFLAG

Description:

- Clears the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

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 Legacy

I2S_STANDARD_PHILLIPS

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

27 HAL IRDA Generic Driver

27.1 IRDA Firmware driver registers structures

27.1.1 IRDA_InitTypeDef

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 ***IRDA_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***

27.1.2 IRDA_HandleTypeDef

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

27.2 IRDA Firmware driver API description

27.2.1 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. 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

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

27.2.2 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: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_IRDA_Init()***
- ***HAL_IRDA_DeInit()***
- ***HAL_IRDA_MspInit()***
- ***HAL_IRDA_MspDeInit()***

27.2.3 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.

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. (#) 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 (#) Blocking mode APIs are : (++) HAL_IRDA_Transmit() (++) HAL_IRDA_Receive() (#) Non Blocking mode APIs with Interrupt are : (++) HAL_IRDA_Transmit_IT() (++) HAL_IRDA_Receive_IT() (++) HAL_IRDA_IRQHandler() (#) Non Blocking mode functions with DMA are : (++) HAL_IRDA_Transmit_DMA() (++) HAL_IRDA_Receive_DMA() (++) HAL_IRDA_DMAPause() (++) HAL_IRDA_DMAResume() (++) HAL_IRDA_DMAStop() (#) 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() (#) 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() (#) 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() (#) 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()***

27.2.4 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()***

27.2.5 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_MspInit

Function name	**void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)**
Function description	Initialize the IRDA MSP.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_MspDeInit

Function name	**void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)**
Function description	DeInitialize the IRDA MSP.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_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	• **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
- **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_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
- **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
- **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_IT

Function name
HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description
Receive an amount of data in interrupt mode.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer.
- **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 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_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	• **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	• **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_DMA

Function name	**HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in DMA mode.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be received.
Return values	• **HAL:** status
Notes	• When the IRDA parity is enabled (PCE = 1), the received data contains the parity bit (MSB position). • 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_DMAPause

Function name	**HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)**
Function description	Pause the DMA Transfer.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that

contains the configuration information for the specified IRDA module.

Return values
- **HAL:** status

HAL_IRDA_DMAResume

Function name
HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)

Function description
Resume the DMA Transfer.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_IRDA_DMAStop

Function name
HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)

Function description
Stop the DMA Transfer.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_IRDA_Abort

Function name
HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)

Function description
Abort ongoing transfers (blocking mode).

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name
HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)

Function description
Abort ongoing Transmit transfer (blocking mode).

Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Receive transfer (blocking mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when

user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name	**void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)**
Function description	Handle IRDA interrupt request.

Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_TxCpltCallback

Function name	**void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Tx Transfer completed callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_RxCpltCallback

Function name	**void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Rx Transfer completed callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_TxHalfCpltCallback

Function name	**void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Tx Half Transfer completed callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• **None**

HAL_IRDA_RxHalfCpltCallback

Function name	**void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Rx Half Transfer complete callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_ErrorCallback

Function name	**void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA error callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_AbortCpltCallback

Function name	**void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Complete callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_AbortTransmitCpltCallback

Function name	**void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Complete callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_AbortReceiveCpltCallback

Function name	**void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Receive Complete callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None**

HAL_IRDA_GetState

Function name	**HAL_IRDA_StateTypeDef HAL_IRDA_GetState (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

27.3 IRDA Firmware driver defines

27.3.1 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 Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE	**Description:** • Reset IRDA handle state. **Parameters:** • __HANDLE__: IRDA handle. **Return value:** • None
__HAL_IRDA_FLUSH_DRREGISTER	**Description:** • Flush the IRDA DR register. **Parameters:** • __HANDLE__: specifies the IRDA Handle. **Return value:** • None
__HAL_IRDA_CLEAR_FLAG	**Description:** • Clear the specified IRDA pending flag. **Parameters:** • __HANDLE__: specifies the IRDA Handle. • __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:

- IRDA_CLEAR_PEF
- IRDA_CLEAR_FEF
- IRDA_CLEAR_NEF
- IRDA_CLEAR_OREF
- IRDA_CLEAR_TCF
- IRDA_CLEAR_IDLEF

Return value:

- None

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK Receive enable acknowledge flag
  - IRDA_FLAG_TEACK Transmit enable acknowledge flag
  - IRDA_FLAG_BUSY Busy flag
  - IRDA_FLAG_ABRF Auto Baud rate detection flag
  - IRDA_FLAG_ABRE Auto Baud rate detection error flag
  - IRDA_FLAG_TXE Transmit data register empty flag
  - IRDA_FLAG_TC Transmission Complete flag
  - IRDA_FLAG_RXNE Receive data register not empty flag
  - IRDA_FLAG_ORE OverRun Error flag
  - IRDA_FLAG_NE Noise Error flag
  - IRDA_FLAG_FE Framing Error flag
  - IRDA_FLAG_PE Parity Error flag

Return value:

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

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame

error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_GET_IT

Description:

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

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __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_SOURCE

Description:

- 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_SAMPLE_ENABLE

Description:
- Enable the IRDA one bit sample method.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:
- Disable the IRDA one bit sample method.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

__HAL_IRDA_ENABLE

Description:
- Enable UART/USART associated to IRDA Handle.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

__HAL_IRDA_DISABLE

Description:
- Disable UART/USART associated to IRDA Handle.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

IRDA Exported Types

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

IRDA Flags

IRDA_FLAG_REACK	IRDA Receive enable acknowledge flag
IRDA_FLAG_TEACK	IRDA Transmit enable acknowledge flag
IRDA_FLAG_BUSY	IRDA Busy flag
IRDA_FLAG_ABRF	IRDA Auto baud rate flag
IRDA_FLAG_ABRE	IRDA Auto baud rate error
IRDA_FLAG_TXE	IRDA Transmit data register empty
IRDA_FLAG_TC	IRDA Transmission complete
IRDA_FLAG_RXNE	IRDA Read data register not empty
IRDA_FLAG_ORE	IRDA Overrun error
IRDA_FLAG_NE	IRDA Noise error
IRDA_FLAG_FE	IRDA 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_IT_ORE	
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_NORMAL	IRDA normal power mode
IRDA_POWERMODE_LOWPOWER	IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE	Associated UART disabled in IRDA mode
IRDA_MODE_ENABLE	Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE	One-bit sampling disabled
IRDA_ONE_BIT_SAMPLE_ENABLE	One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE	No parity
IRDA_PARITY_EVEN	Even parity
IRDA_PARITY_ODD	Odd parity

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST	Auto-Baud Rate Request
IRDA_RXDATA_FLUSH_REQUEST	Receive Data flush Request
IRDA_TXDATA_FLUSH_REQUEST	Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE	IRDA disabled
IRDA_STATE_ENABLE	IRDA enabled

IRDA Transfer Mode

IRDA_MODE_RX	RX mode
IRDA_MODE_TX	TX mode
IRDA_MODE_TX_RX	RX and TX mode

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 IRDA Extension Driver

28.1 IRDAEx Firmware driver defines

28.1.1 IRDAEx

IRDAEx Exported Macros

IRDA_GETCLOCKSOURCE

Description:

- Reports the IRDA clock source.

Parameters:

- __HANDLE__: specifies the UART Handle
- __CLOCKSOURCE__: : output variable

Return value:

- IRDA: clocking source, written in __CLOCKSOURCE__.

IRDA_MASK_COMPUTATION

Description:

- Reports the mask to apply to retrieve the received data according to the word length and to the parity bits activation.

Parameters:

- __HANDLE__: specifies the IRDA Handle

Return value:

- mask: to apply to USART RDR register value.

IRDAEx Word length

IRDA_WORDLENGTH_7B

IRDA_WORDLENGTH_8B

IRDA_WORDLENGTH_9B

IS_IRDA_WORD_LENGTH

29 HAL IWDG Generic Driver

29.1 IWDG Firmware driver registers structures

29.1.1 IWDG_InitTypeDef

Data Fields

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

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

29.1.2 IWDG_HandleTypeDef

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

29.2 IWDG Firmware driver API description

29.2.1 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 (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded 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 @32KHz (LSI): ~0.512ms / ~32.0s The IWDG timeout may vary due to LSI frequency dispersion. STM32L0xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM5 CH4 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

29.2.2 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 withy exact timebase.
     - 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

29.2.3 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 timebase.

This section contains the following APIs:

- ***HAL_IWDG_Init()***

29.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- ***HAL_IWDG_Refresh()***

29.2.5 Detailed description of functions

HAL_IWDG_Init

Function name	**HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)**
Function description	Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL:** status

HAL_IWDG_Refresh

Function name	**HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)**
Function description	Refresh the IWDG.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL:** status

29.3 IWDG Firmware driver defines

29.3.1 IWDG

IWDG Exported Macros

__HAL_IWDG_START

Description:

- Enable the IWDG peripheral.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

__HAL_IWDG_RELOAD_COUNTER

Description:

- Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR & IWDG_WINR registers disabled).

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

IWDG Prescaler

IWDG_PRESCALER_4	IWDG prescaler set to 4
IWDG_PRESCALER_8	IWDG prescaler set to 8
IWDG_PRESCALER_16	IWDG prescaler set to 16
IWDG_PRESCALER_32	IWDG prescaler set to 32
IWDG_PRESCALER_64	IWDG prescaler set to 64
IWDG_PRESCALER_128	IWDG prescaler set to 128
IWDG_PRESCALER_256	IWDG prescaler set to 256

IWDG Window option

IWDG_WINDOW_DISABLE

30 HAL LCD Generic Driver

30.1 LCD Firmware driver registers structures

30.1.1 LCD_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Divider***
- ***uint32_t Duty***
- ***uint32_t Bias***
- ***uint32_t VoltageSource***
- ***uint32_t Contrast***
- ***uint32_t DeadTime***
- ***uint32_t PulseOnDuration***
- ***uint32_t HighDrive***
- ***uint32_t BlinkMode***
- ***uint32_t BlinkFrequency***
- ***uint32_t MuxSegment***

Field Documentation

- ***uint32_t LCD_InitTypeDef::Prescaler***
  Configures the LCD Prescaler. This parameter can be one value of ***LCD_Prescaler***
- ***uint32_t LCD_InitTypeDef::Divider***
  Configures the LCD Divider. This parameter can be one value of ***LCD_Divider***
- ***uint32_t LCD_InitTypeDef::Duty***
  Configures the LCD Duty. This parameter can be one value of ***LCD_Duty***
- ***uint32_t LCD_InitTypeDef::Bias***
  Configures the LCD Bias. This parameter can be one value of ***LCD_Bias***
- ***uint32_t LCD_InitTypeDef::VoltageSource***
  Selects the LCD Voltage source. This parameter can be one value of ***LCD_Voltage_Source***
- ***uint32_t LCD_InitTypeDef::Contrast***
  Configures the LCD Contrast. This parameter can be one value of ***LCD_Contrast***
- ***uint32_t LCD_InitTypeDef::DeadTime***
  Configures the LCD Dead Time. This parameter can be one value of ***LCD_DeadTime***
- ***uint32_t LCD_InitTypeDef::PulseOnDuration***
  Configures the LCD Pulse On Duration. This parameter can be one value of ***LCD_PulseOnDuration***
- ***uint32_t LCD_InitTypeDef::HighDrive***
  Configures the LCD High Drive. This parameter can be one value of ***LCD_HighDrive***
- ***uint32_t LCD_InitTypeDef::BlinkMode***
  Configures the LCD Blink Mode. This parameter can be one value of ***LCD_BlinkMode***
- ***uint32_t LCD_InitTypeDef::BlinkFrequency***
  Configures the LCD Blink frequency. This parameter can be one value of ***LCD_BlinkFrequency***
- ***uint32_t LCD_InitTypeDef::MuxSegment***
  Enable or disable mux segment. This parameter can be one value of ***LCD_MuxSegment***

30.1.2 LCD_HandleTypeDef

Data Fields

- ***LCD_TypeDef * Instance***
- ***LCD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***LCD_TypeDef* LCD_HandleTypeDef::Instance***
- ***LCD_InitTypeDef LCD_HandleTypeDef::Init***
- ***HAL_LockTypeDef LCD_HandleTypeDef::Lock***
- ***__IO HAL_LCD_StateTypeDef LCD_HandleTypeDef::State***
- ***__IO uint32_t LCD_HandleTypeDef::ErrorCode***

30.2 LCD Firmware driver API description

30.2.1 How to use this driver

The LCD HAL driver can be used as follow:

1. Declare a LCD_HandleTypeDef handle structure.
2. Prepare the initialization of the LCD low level resources by implementing your HAL_LCD_MspInit() API:
   a. Enable the LCDCLK (same as RTCCLK): to configure the RTCCLK/LCDCLK, use the RCC function HAL_RCCEx_PeriphCLKConfig, indicating here RCC_PERIPHCLK_LCD and the selected clock source (HSE, LSI or LSE)
   b. The frequency generator allows you to achieve various LCD frame rates starting from an LCD input clock frequency (LCDCLK) which can vary from 32 kHz up to 1 MHz.
   c. LCD pins configuration: - Enable the clock for the LCD GPIOs - Configure these LCD pins as alternate function no-pull.
   d. Enable the LCD interface clock.
3. Set the Prescaler, Divider, Blink mode, Blink Frequency Duty, Bias, Voltage Source, Dead Time, Pulse On Duration and Contrast in the hlcd Init structure.
4. Initialize the LCD registers by calling the HAL_LCD_Init() API.
   a. The HAL_LCD_Init() API configures the low level Hardware (GPIO, CLOCK, ...etc) by calling the user customized HAL_LCD_MspInit() API.
5. After calling the HAL_LCD_Init() the LCD RAM memory is cleared
6. Optionally you can update the LCD configuration using these macros:
   a. LCD High Drive using the __HAL_LCD_HIGHDRIVER_ENABLE() and __HAL_LCD_HIGHDRIVER_DISABLE() macros
   b. LCD Pulse ON Duration using the __HAL_LCD_PULSEONDURATION_CONFIG() macro
   c. LCD Dead Time using the __HAL_LCD_DEADTIME_CONFIG() macro
   d. The LCD Blink mode and frequency using the __HAL_LCD_BLINK_CONFIG() macro
   e. The LCD Contrast using the __HAL_LCD_CONTRAST_CONFIG() macro
7. Write to the LCD RAM memory using the HAL_LCD_Write() API, this API can be called several times to update the different LCD RAM registers before calling HAL_LCD_UpdateDisplayRequest() API.
8. The HAL_LCD_Clear() API can be used to clear the LCD RAM memory.
9. When the LCD RAM memory is updated, enable the update display request calling the HAL_LCD_UpdateDisplayRequest() API.

LCD and low power modes: The LCD remain active during STOP mode.

30.2.2 Initialization and Configuration functions

This section contains the following APIs:

- ***HAL_LCD_DeInit()***
- ***HAL_LCD_Init()***
- ***HAL_LCD_MspDeInit()***
- ***HAL_LCD_MspInit()***

30.2.3 IO operation functions

Using its double buffer memory the LCD controller ensures the coherency of the displayed information without having to use interrupts to control LCD_RAM modification. The application software can access the first buffer level (LCD_RAM) through the APB interface. Once it has modified the LCD_RAM using the HAL_LCD_Write() API, it sets the UDR flag in the LCD_SR register using the HAL_LCD_UpdateDisplayRequest() API. This UDR flag (update display request) requests the updated information to be moved into the second buffer level (LCD_DISPLAY). This operation is done synchronously with the frame (at the beginning of the next frame), until the update is completed, the LCD_RAM is write protected and the UDR flag stays high. Once the update is completed another flag (UDD - Update Display Done) is set and generates an interrupt if the UDDIE bit in the LCD_FCR register is set. The time it takes to update LCD_DISPLAY is, in the worst case, one odd and one even frame. The update will not occur (UDR = 1 and UDD = 0) until the display is enabled (LCDEN = 1).

This section contains the following APIs:

- ***HAL_LCD_Write()***
- ***HAL_LCD_Clear()***
- ***HAL_LCD_UpdateDisplayRequest()***

30.2.4 Peripheral State functions

This subsection provides a set of functions allowing to control the LCD:

- HAL_LCD_GetState() API can be helpful to check in run-time the state of the LCD peripheral State.
- HAL_LCD_GetError() API to return the LCD error code.

This section contains the following APIs:

- ***HAL_LCD_GetState()***
- ***HAL_LCD_GetError()***

30.2.5 Detailed description of functions

HAL_LCD_DeInit

Function name	**HAL_StatusTypeDef HAL_LCD_DeInit (LCD_HandleTypeDef * hlcd)**
Function description	DeInitializes the LCD peripheral.
Parameters	• **hlcd:** LCD handle
Return values	• **HAL:** status

HAL_LCD_Init

Function name	**HAL_StatusTypeDef HAL_LCD_Init (LCD_HandleTypeDef * hlcd)**
Function description	Initializes the LCD peripheral according to the specified parameters in the LCD_InitStruct.
Parameters	• **hlcd:** LCD handle
Return values	• **None**
Notes	• This function can be used only when the LCD is disabled. The LCD HighDrive can be enabled/disabled using related macros up to user.

HAL_LCD_MspInit

Function name	**void HAL_LCD_MspInit (LCD_HandleTypeDef * hlcd)**
Function description	LCD MSP Init.
Parameters	• **hlcd:** LCD handle
Return values	• **None**

HAL_LCD_MspDeInit

Function name	**void HAL_LCD_MspDeInit (LCD_HandleTypeDef * hlcd)**
Function description	LCD MSP DeInit.
Parameters	• **hlcd:** LCD handle
Return values	• **None**

HAL_LCD_Write

Function name	**HAL_StatusTypeDef HAL_LCD_Write (LCD_HandleTypeDef * hlcd, uint32_t RAMRegisterIndex, uint32_t RAMRegisterMask, uint32_t Data)**
Function description	Writes a word in the specific LCD RAM.
Parameters	• **hlcd:** LCD handle • **RAMRegisterIndex:** specifies the LCD RAM Register. This parameter can be one of the following values: – LCD_RAM_REGISTER0: LCD RAM Register 0 – LCD_RAM_REGISTER1: LCD RAM Register 1 – LCD_RAM_REGISTER2: LCD RAM Register 2 – LCD_RAM_REGISTER3: LCD RAM Register 3 – LCD_RAM_REGISTER4: LCD RAM Register 4 – LCD_RAM_REGISTER5: LCD RAM Register 5 – LCD_RAM_REGISTER6: LCD RAM Register 6 – LCD_RAM_REGISTER7: LCD RAM Register 7 – LCD_RAM_REGISTER8: LCD RAM Register 8 – LCD_RAM_REGISTER9: LCD RAM Register 9 – LCD_RAM_REGISTER10: LCD RAM Register 10 – LCD_RAM_REGISTER11: LCD RAM Register 11 – LCD_RAM_REGISTER12: LCD RAM Register 12

- – LCD_RAM_REGISTER13: LCD RAM Register 13
- – LCD_RAM_REGISTER14: LCD RAM Register 14
- – LCD_RAM_REGISTER15: LCD RAM Register 15
- **RAMRegisterMask:** specifies the LCD RAM Register Data Mask.
- **Data:** specifies LCD Data Value to be written.

Return values	• **None**
Notes	• For LCD glass COM*SEG as 8*40 for example, the LCD common terminals COM[0,7] are mapped on 32bits LCD_RAM_REGISTER[0,14] according to rules: COM(n) spread on LCD_RAM_REGISTER(2*n) and LCD_RAM_REGISTER(2*n+1).The segment terminals SEG[0,39] of COM(n) correspond to LSB bits of related LCD_RAM_REGISTER(2*n)[0,31] and LCD_RAM_REGISTER(2*n+1)[0,7]

HAL_LCD_Clear

Function name	**HAL_StatusTypeDef HAL_LCD_Clear (LCD_HandleTypeDef * hlcd)**
Function description	Clears the LCD RAM registers.
Parameters	• **hlcd:** LCD handle
Return values	• **None**

HAL_LCD_UpdateDisplayRequest

Function name	**HAL_StatusTypeDef HAL_LCD_UpdateDisplayRequest (LCD_HandleTypeDef * hlcd)**
Function description	Enables the Update Display Request.
Parameters	• **hlcd:** LCD handle
Return values	• **None**
Notes	• Each time software modifies the LCD_RAM it must set the UDR bit to transfer the updated data to the second level buffer. The UDR bit stays set until the end of the update and during this time the LCD_RAM is write protected. • When the display is disabled, the update is performed for all LCD_DISPLAY locations. When the display is enabled, the update is performed only for locations for which commons are active (depending on DUTY). For example if DUTY = 1/2, only the LCD_DISPLAY of COM0 and COM1 will be updated.

HAL_LCD_GetState

Function name	**HAL_LCD_StateTypeDef HAL_LCD_GetState (LCD_HandleTypeDef * hlcd)**
Function description	Returns the LCD state.
Parameters	• **hlcd:** LCD handle

Return values	• **HAL:** state

HAL_LCD_GetError

Function name	**uint32_t HAL_LCD_GetError (LCD_HandleTypeDef * hlcd)**
Function description	Return the LCD error code.
Parameters	• **hlcd:** LCD handle
Return values	• **LCD:** Error Code

LCD_WaitForSynchro

Function name	**HAL_StatusTypeDef LCD_WaitForSynchro (LCD_HandleTypeDef * hlcd)**
Function description	Waits until the LCD FCR register is synchronized in the LCDCLK domain.
Parameters	• **hlcd:** LCD handle
Return values	• **None**

30.3 LCD Firmware driver defines

30.3.1 LCD

LCD Bias

LCD_BIAS_1_4	1/4 Bias
LCD_BIAS_1_2	1/2 Bias
LCD_BIAS_1_3	1/3 Bias
IS_LCD_BIAS	

LCD Blink Frequency

LCD_BLINKFREQUENCY_DIV8	The Blink frequency = fLCD/8
LCD_BLINKFREQUENCY_DIV16	The Blink frequency = fLCD/16
LCD_BLINKFREQUENCY_DIV32	The Blink frequency = fLCD/32
LCD_BLINKFREQUENCY_DIV64	The Blink frequency = fLCD/64
LCD_BLINKFREQUENCY_DIV128	The Blink frequency = fLCD/128
LCD_BLINKFREQUENCY_DIV256	The Blink frequency = fLCD/256
LCD_BLINKFREQUENCY_DIV512	The Blink frequency = fLCD/512
LCD_BLINKFREQUENCY_DIV1024	The Blink frequency = fLCD/1024
IS_LCD_BLINK_FREQUENCY	

LCD Blink Mode

LCD_BLINKMODE_OFF	Blink disabled
LCD_BLINKMODE_SEG0_COM0	Blink enabled on SEG[0], COM[0] (1 pixel)
LCD_BLINKMODE_SEG0_ALLCOM	Blink enabled on SEG[0], all COM (up to 8 pixels

	according to the programmed duty)
LCD_BLINKMODE_ALLSEG_ALLCOM	Blink enabled on all SEG and all COM (all pixels)

IS_LCD_BLINK_MODE

LCD Voltage output buffer enable

LCD_VOLTBUFOUT_DISABLE	Voltage output buffer disabled
LCD_VOLTBUFOUT_ENABLE	BUFEN[1] Voltage output buffer enabled

IS_LCD_VOLTBUFOUT

LCD Contrast

LCD_CONTRASTLEVEL_0	Maximum Voltage = 2.60V
LCD_CONTRASTLEVEL_1	Maximum Voltage = 2.73V
LCD_CONTRASTLEVEL_2	Maximum Voltage = 2.86V
LCD_CONTRASTLEVEL_3	Maximum Voltage = 2.99V
LCD_CONTRASTLEVEL_4	Maximum Voltage = 3.12V
LCD_CONTRASTLEVEL_5	Maximum Voltage = 3.25V
LCD_CONTRASTLEVEL_6	Maximum Voltage = 3.38V
LCD_CONTRASTLEVEL_7	Maximum Voltage = 3.51V

IS_LCD_CONTRAST

LCD Dead Time

LCD_DEADTIME_0	No dead Time
LCD_DEADTIME_1	One Phase between different couple of Frame
LCD_DEADTIME_2	Two Phase between different couple of Frame
LCD_DEADTIME_3	Three Phase between different couple of Frame
LCD_DEADTIME_4	Four Phase between different couple of Frame
LCD_DEADTIME_5	Five Phase between different couple of Frame
LCD_DEADTIME_6	Six Phase between different couple of Frame
LCD_DEADTIME_7	Seven Phase between different couple of Frame

IS_LCD_DEAD_TIME

LCD Divider

LCD_DIVIDER_16	LCD frequency = CLKPS/16
LCD_DIVIDER_17	LCD frequency = CLKPS/17
LCD_DIVIDER_18	LCD frequency = CLKPS/18
LCD_DIVIDER_19	LCD frequency = CLKPS/19
LCD_DIVIDER_20	LCD frequency = CLKPS/20
LCD_DIVIDER_21	LCD frequency = CLKPS/21
LCD_DIVIDER_22	LCD frequency = CLKPS/22
LCD_DIVIDER_23	LCD frequency = CLKPS/23

LCD_DIVIDER_24	LCD frequency = CLKPS/24
LCD_DIVIDER_25	LCD frequency = CLKPS/25
LCD_DIVIDER_26	LCD frequency = CLKPS/26
LCD_DIVIDER_27	LCD frequency = CLKPS/27
LCD_DIVIDER_28	LCD frequency = CLKPS/28
LCD_DIVIDER_29	LCD frequency = CLKPS/29
LCD_DIVIDER_30	LCD frequency = CLKPS/30
LCD_DIVIDER_31	LCD frequency = CLKPS/31
IS_LCD_DIVIDER	

LCD Duty

LCD_DUTY_STATIC	Static duty
LCD_DUTY_1_2	1/2 duty
LCD_DUTY_1_3	1/3 duty
LCD_DUTY_1_4	1/4 duty
LCD_DUTY_1_8	1/8 duty
IS_LCD_DUTY	

LCD Error Code

HAL_LCD_ERROR_NONE	No error
HAL_LCD_ERROR_FCRSF	Synchro flag timeout error
HAL_LCD_ERROR_UDR	Update display request flag timeout error
HAL_LCD_ERROR_UDD	Update display done flag timeout error
HAL_LCD_ERROR_ENS	LCD enabled status flag timeout error
HAL_LCD_ERROR_RDY	LCD Booster ready timeout error

LCD Exported Macros

__HAL_LCD_RESET_HANDLE_STATE

Description:

- Reset LCD handle state.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_ENABLE

Description:

- macros to enables or disables the LCD

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_DISABLE

__HAL_LCD_VOLTOUTBUFFER_ENABLE

Description:

- macros to enables or disables the Voltage output buffer

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_VOLTOUTBUFFER_DISABLE

__HAL_LCD_HIGHDRIVER_ENABLE

Description:

- Macros to enable or disable the low resistance divider.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

Notes:

- When this mode is enabled, the PulseOn Duration (PON) have to be programmed to 1/CK_PS (LCD_PULSEONDURATION_1).

__HAL_LCD_HIGHDRIVER_DISABLE

__HAL_LCD_PULSEONDURATION_CONFIG

Description:

- Macro to configure the LCD pulses on duration.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DURATION__: specifies the LCD pulse on duration in terms of CK_PS (prescaled LCD clock period) pulses. This parameter can be one of the following values:
  - LCD_PULSEONDURATION_0: 0 pulse
  - LCD_PULSEONDURATION_1: Pulse ON duration = 1/CK_PS
  - LCD_PULSEONDURATION_2: Pulse ON duration = 2/CK_PS
  - LCD_PULSEONDURATION_3: Pulse ON duration = 3/CK_PS
  - LCD_PULSEONDURATION_4: Pulse ON duration = 4/CK_PS
  - LCD_PULSEONDURATION_5: Pulse ON duration = 5/CK_PS
  - LCD_PULSEONDURATION_6: Pulse

ON duration = 6/CK_PS
 – LCD_PULSEONDURATION_7: Pulse ON duration = 7/CK_PS

Return value:

- None

__HAL_LCD_DEADTIME_CONFIG

Description:

- Macro to configure the LCD dead time.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DEADTIME__: specifies the LCD dead time. This parameter can be one of the following values:
  - LCD_DEADTIME_0: No dead Time
  - LCD_DEADTIME_1: One Phase between different couple of Frame
  - LCD_DEADTIME_2: Two Phase between different couple of Frame
  - LCD_DEADTIME_3: Three Phase between different couple of Frame
  - LCD_DEADTIME_4: Four Phase between different couple of Frame
  - LCD_DEADTIME_5: Five Phase between different couple of Frame
  - LCD_DEADTIME_6: Six Phase between different couple of Frame
  - LCD_DEADTIME_7: Seven Phase between different couple of Frame

Return value:

- None

__HAL_LCD_CONTRAST_CONFIG

Description:

- Macro to configure the LCD Contrast.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __CONTRAST__: specifies the LCD Contrast. This parameter can be one of the following values:
  - LCD_CONTRASTLEVEL_0: Maximum Voltage = 2.60V
  - LCD_CONTRASTLEVEL_1: Maximum Voltage = 2.73V
  - LCD_CONTRASTLEVEL_2: Maximum Voltage = 2.86V
  - LCD_CONTRASTLEVEL_3: Maximum Voltage = 2.99V
  - LCD_CONTRASTLEVEL_4: Maximum Voltage = 3.12V
  - LCD_CONTRASTLEVEL_5: Maximum Voltage = 3.25V

- LCD_CONTRASTLEVEL_6: Maximum Voltage = 3.38V
- LCD_CONTRASTLEVEL_7: Maximum Voltage = 3.51V

Return value:

- None

__HAL_LCD_BLINK_CONFIG

Description:

- Macro to configure the LCD Blink mode and Blink frequency.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __BLINKMODE__: specifies the LCD blink mode. This parameter can be one of the following values:
  - LCD_BLINKMODE_OFF: Blink disabled
  - LCD_BLINKMODE_SEG0_COM0: Blink enabled on SEG[0], COM[0] (1 pixel)
  - LCD_BLINKMODE_SEG0_ALLCOM: Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)
  - LCD_BLINKMODE_ALLSEG_ALLCOM: Blink enabled on all SEG and all COM (all pixels)
- __BLINKFREQUENCY__: specifies the LCD blink frequency.
  - LCD_BLINKFREQUENCY_DIV8: The Blink frequency = fLcd/8
  - LCD_BLINKFREQUENCY_DIV16: The Blink frequency = fLcd/16
  - LCD_BLINKFREQUENCY_DIV32: The Blink frequency = fLcd/32
  - LCD_BLINKFREQUENCY_DIV64: The Blink frequency = fLcd/64
  - LCD_BLINKFREQUENCY_DIV128: The Blink frequency = fLcd/128
  - LCD_BLINKFREQUENCY_DIV256: The Blink frequency = fLcd/256
  - LCD_BLINKFREQUENCY_DIV512: The Blink frequency = fLcd/512
  - LCD_BLINKFREQUENCY_DIV1024: The Blink frequency = fLcd/1024

Return value:

- None

__HAL_LCD_ENABLE_IT

Description:

- Enables or disables the specified LCD interrupt.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __INTERRUPT__: specifies the LCD interrupt source to be enabled or disabled. This parameter can be one of the following values:
  - LCD_IT_SOF: Start of Frame Interrupt
  - LCD_IT_UDD: Update Display Done Interrupt

Return value:

- None

__HAL_LCD_DISABLE_IT

__HAL_LCD_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __IT__: specifies the LCD interrupt source to check. This parameter can be one of the following values:
  - LCD_IT_SOF: Start of Frame Interrupt
  - LCD_IT_UDD: Update Display Done Interrupt.

Return value:

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

Notes:

- If the device is in STOP mode (PCLK not provided) UDD will not generate an interrupt even if UDDIE = 1. If the display is not enabled the UDD interrupt will never occur.

__HAL_LCD_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - LCD_FLAG_ENS: LCD Enabled flag. It indicates the LCD controller status.

Return value:

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

Notes:

- The ENS bit is set immediately when the LCDEN bit in the LCD_CR goes from 0 to 1. On deactivation it reflects the real status of LCD so it becomes 0 at the end of the last displayed frame. LCD_FLAG_SOF: Start of Frame flag. This flag is set by hardware at the beginning of a new frame, at the same time as the display data is updated. LCD_FLAG_UDR: Update Display Request flag. LCD_FLAG_UDD: Update Display Done flag. LCD_FLAG_RDY: Step_up converter Ready flag. It indicates the status of the step-up converter. LCD_FLAG_FCRSF: LCD Frame Control Register Synchronization Flag. This flag is set by hardware each time the LCD_FCR register is updated in the LCDCLK domain.

__HAL_LCD_CLEAR_FLAG

Description:

- Clears the specified LCD pending flag.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - LCD_FLAG_SOF: Start of Frame Interrupt
  - LCD_FLAG_UDD: Update Display Done Interrupt

Return value:

- None

LCD Flag

LCD_FLAG_ENS

LCD_FLAG_SOF

LCD_FLAG_UDR

LCD_FLAG_UDD

LCD_FLAG_RDY

LCD_FLAG_FCRSF

LCD HighDrive

LCD_HIGHDRIVE_0	Low resistance Drive
LCD_HIGHDRIVE_1	High resistance Drive
IS_LCD_HIGHDRIVE	

LCD Interrupts

LCD_IT_SOF

LCD_IT_UDD

LCD Mux Segment

LCD_MUXSEGMENT_DISABLE	SEG pin multiplexing disabled
LCD_MUXSEGMENT_ENABLE	SEG[31:28] are multiplexed with SEG[43:40]

IS_LCD_MUXSEGMENT

LCD Prescaler

LCD_PRESCALER_1	CLKPS = LCDCLK
LCD_PRESCALER_2	CLKPS = LCDCLK/2
LCD_PRESCALER_4	CLKPS = LCDCLK/4
LCD_PRESCALER_8	CLKPS = LCDCLK/8
LCD_PRESCALER_16	CLKPS = LCDCLK/16
LCD_PRESCALER_32	CLKPS = LCDCLK/32
LCD_PRESCALER_64	CLKPS = LCDCLK/64
LCD_PRESCALER_128	CLKPS = LCDCLK/128
LCD_PRESCALER_256	CLKPS = LCDCLK/256
LCD_PRESCALER_512	CLKPS = LCDCLK/512
LCD_PRESCALER_1024	CLKPS = LCDCLK/1024
LCD_PRESCALER_2048	CLKPS = LCDCLK/2048
LCD_PRESCALER_4096	CLKPS = LCDCLK/4096
LCD_PRESCALER_8192	CLKPS = LCDCLK/8192
LCD_PRESCALER_16384	CLKPS = LCDCLK/16384
LCD_PRESCALER_32768	CLKPS = LCDCLK/32768

IS_LCD_PRESCALER

LCD Pulse On Duration

LCD_PULSEONDURATION_0	Pulse ON duration = 0 pulse
LCD_PULSEONDURATION_1	Pulse ON duration = 1/CK_PS
LCD_PULSEONDURATION_2	Pulse ON duration = 2/CK_PS
LCD_PULSEONDURATION_3	Pulse ON duration = 3/CK_PS
LCD_PULSEONDURATION_4	Pulse ON duration = 4/CK_PS
LCD_PULSEONDURATION_5	Pulse ON duration = 5/CK_PS
LCD_PULSEONDURATION_6	Pulse ON duration = 6/CK_PS
LCD_PULSEONDURATION_7	Pulse ON duration = 7/CK_PS

IS_LCD_PULSE_ON_DURATION

LCD RAMRegister

LCD_RAM_REGISTER0	LCD RAM Register 0
LCD_RAM_REGISTER1	LCD RAM Register 1
LCD_RAM_REGISTER2	LCD RAM Register 2
LCD_RAM_REGISTER3	LCD RAM Register 3
LCD_RAM_REGISTER4	LCD RAM Register 4
LCD_RAM_REGISTER5	LCD RAM Register 5

LCD_RAM_REGISTER6	LCD RAM Register 6
LCD_RAM_REGISTER7	LCD RAM Register 7
LCD_RAM_REGISTER8	LCD RAM Register 8
LCD_RAM_REGISTER9	LCD RAM Register 9
LCD_RAM_REGISTER10	LCD RAM Register 10
LCD_RAM_REGISTER11	LCD RAM Register 11
LCD_RAM_REGISTER12	LCD RAM Register 12
LCD_RAM_REGISTER13	LCD RAM Register 13
LCD_RAM_REGISTER14	LCD RAM Register 14
LCD_RAM_REGISTER15	LCD RAM Register 15
IS_LCD_RAM_REGISTER	

LCD Voltage Source

LCD_VOLTAGESOURCE_INTERNAL	Internal voltage source for the LCD
LCD_VOLTAGESOURCE_EXTERNAL	External voltage source for the LCD
IS_LCD_VOLTAGE_SOURCE	

31 HAL LPTIM Generic Driver

31.1 LPTIM Firmware driver registers structures

31.1.1 LPTIM_ClockConfigTypeDef

Data Fields

- ***uint32_t Source***
- ***uint32_t Prescaler***

Field Documentation

- ***uint32_t LPTIM_ClockConfigTypeDef::Source***
  Selects the clock source. This parameter can be a value of ***LPTIM_Clock_Source***
- ***uint32_t LPTIM_ClockConfigTypeDef::Prescaler***
  Specifies the counter clock Prescaler. This parameter can be a value of ***LPTIM_Clock_Prescaler***

31.1.2 LPTIM_ULPClockConfigTypeDef

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_ULPClockConfigTypeDef::Polarity***
  Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of ***LPTIM_Clock_Polarity***
- ***uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime***
  Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of ***LPTIM_Clock_Sample_Time***

31.1.3 LPTIM_TriggerConfigTypeDef

Data Fields

- ***uint32_t Source***
- ***uint32_t ActiveEdge***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
  Selects the Trigger source. This parameter can be a value of ***LPTIM_Trigger_Source***
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
  Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_External_Trigger_Polarity***
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
  Selects the trigger sampling time to configure the clock glitch filter. Note: This

parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_Trigger_Sample_Time***

31.1.4 LPTIM_InitTypeDef

Data Fields

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***
- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t OutputPolarity***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
  Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
  Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
  Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::OutputPolarity***
  Specifies the Output polarity. This parameter can be a value of ***LPTIM_Output_Polarity***
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
  Specifies whether the update of the autorelaod and the compare values is done immediately or after the end of current period. This parameter can be a value of ***LPTIM_Updating_Mode***
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
  Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of ***LPTIM_Counter_Source***

31.1.5 LPTIM_HandleTypeDef

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***

Field Documentation

- ***LPTIM_TypeDef* LPTIM_HandleTypeDef::Instance***
  Register base address
- ***LPTIM_InitTypeDef LPTIM_HandleTypeDef::Init***
  LPTIM required parameters
- ***HAL_StatusTypeDef LPTIM_HandleTypeDef::Status***
  LPTIM peripheral status
- ***HAL_LockTypeDef LPTIM_HandleTypeDef::Lock***
  LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef LPTIM_HandleTypeDef::State***
  LPTIM peripheral state

31.2 LPTIM Firmware driver API description

31.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize LPTIM MSP.

This section contains the following APIs:

- ***HAL_LPTIM_Init()***
- ***HAL_LPTIM_DeInit()***
- ***HAL_LPTIM_MspInit()***
- ***HAL_LPTIM_MspDeInit()***

31.2.2 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- ***HAL_LPTIM_PWM_Start()***
- ***HAL_LPTIM_PWM_Stop()***
- ***HAL_LPTIM_PWM_Start_IT()***
- ***HAL_LPTIM_PWM_Stop_IT()***
- ***HAL_LPTIM_OnePulse_Start()***
- ***HAL_LPTIM_OnePulse_Stop()***
- ***HAL_LPTIM_OnePulse_Start_IT()***
- ***HAL_LPTIM_OnePulse_Stop_IT()***
- ***HAL_LPTIM_SetOnce_Start()***
- ***HAL_LPTIM_SetOnce_Stop()***
- ***HAL_LPTIM_SetOnce_Start_IT()***
- ***HAL_LPTIM_SetOnce_Stop_IT()***
- ***HAL_LPTIM_Encoder_Start()***
- ***HAL_LPTIM_Encoder_Stop()***
- ***HAL_LPTIM_Encoder_Start_IT()***
- ***HAL_LPTIM_Encoder_Stop_IT()***
- ***HAL_LPTIM_TimeOut_Start()***
- ***HAL_LPTIM_TimeOut_Stop()***

- *HAL_LPTIM_TimeOut_Start_IT()*
- *HAL_LPTIM_TimeOut_Stop_IT()*
- *HAL_LPTIM_Counter_Start()*
- *HAL_LPTIM_Counter_Stop()*
- *HAL_LPTIM_Counter_Start_IT()*
- *HAL_LPTIM_Counter_Stop_IT()*

31.2.3 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- *HAL_LPTIM_ReadCounter()*
- *HAL_LPTIM_ReadAutoReload()*
- *HAL_LPTIM_ReadCompare()*

31.2.4 LPTIM IRQ handler

This section provides LPTIM IRQ handler function.

This section contains the following APIs:

- *HAL_LPTIM_IRQHandler()*
- *HAL_LPTIM_CompareMatchCallback()*
- *HAL_LPTIM_AutoReloadMatchCallback()*
- *HAL_LPTIM_TriggerCallback()*
- *HAL_LPTIM_CompareWriteCallback()*
- *HAL_LPTIM_AutoReloadWriteCallback()*
- *HAL_LPTIM_DirectionUpCallback()*
- *HAL_LPTIM_DirectionDownCallback()*

31.2.5 Peripheral State functions

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

This section contains the following APIs:

- *HAL_LPTIM_GetState()*

31.2.6 Detailed description of functions

HAL_LPTIM_Init

Function name	**HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)**
Function description	Initializes the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_DeInit

Function name	**HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)**
Function description	DeInitializes the LPTIM peripheral.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_MspInit

Function name	**void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)**
Function description	Initializes the LPTIM MSP.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_MspDeInit

Function name	**void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)**
Function description	DeInitializes LPTIM MSP.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_PWM_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Starts the LPTIM PWM generation.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_PWM_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the LPTIM PWM generation.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Starts the LPTIM PWM generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF
Return values	• **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the LPTIM PWM generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Starts the LPTIM One pulse generation.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the LPTIM One pulse generation.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t**

Pulse)

Function description	Starts the LPTIM One pulse generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the LPTIM One pulse generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Starts the LPTIM in Set once mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the LPTIM Set once mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Starts the LPTIM Set once mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

	• **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the LPTIM Set once mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Encoder_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Starts the Encoder interface.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the Encoder interface.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Starts the Encoder interface in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the Encoder interface in nterrupt mode.

Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)**
Function description	Starts the Timeout function.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_TimeOut_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the Timeout function.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)**
Function description	Starts the Timeout function in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_TimeOut_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the Timeout function in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Counter_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Starts the Counter mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_Counter_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the Counter mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Starts the Counter mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stops the Counter mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_ReadCounter

Function name	**uint32_t HAL_LPTIM_ReadCounter (LPTIM_HandleTypeDef * hlptim)**
Function description	This function returns the current counter value.
Parameters	• **hlptim:** LPTIM handle
Return values	• **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name	**uint32_t HAL_LPTIM_ReadAutoReload (LPTIM_HandleTypeDef * hlptim)**
Function description	This function return the current Autoreload (Period) value.
Parameters	• **hlptim:** LPTIM handle
Return values	• **Autoreload:** value.

HAL_LPTIM_ReadCompare

Function name	**uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hlptim)**
Function description	This function return the current Compare (Pulse) value.
Parameters	• **hlptim:** LPTIM handle
Return values	• **Compare:** value.

HAL_LPTIM_IRQHandler

Function name	**void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)**
Function description	This function handles LPTIM interrupt request.
Parameters	• **hlptim:** LPTIM handle
Return values	• **None**

HAL_LPTIM_CompareMatchCallback

Function name	**void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Compare match callback in non blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_AutoReloadMatchCallback

Function name	**void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Autoreload match callback in non blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_TriggerCallback

Function name	**void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Trigger detected callback in non blocking mode.

Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_CompareWriteCallback

Function name	**void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Compare write callback in non blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_AutoReloadWriteCallback

Function name	**void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Autoreload write callback in non blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_DirectionUpCallback

Function name	**void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Direction counter changed from Down to Up callback in non blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_DirectionDownCallback

Function name	**void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Direction counter changed from Up to Down callback in non blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None**

HAL_LPTIM_GetState

Function name	**HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hlptim)**
Function description	Returns the LPTIM state.
Parameters	• **hlptim:** LPTIM handle
Return values	• **HAL:** state

31.3 LPTIM Firmware driver defines

31.3.1 LPTIM

LPTIM Clock configuration structure

LPTIM_EXTI_LINE_WAKEUPTIMER_EVENT	External interrupt line 29 Connected to the LPTIM EXTI Line

Clock polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

Clock sample time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

Clock source

LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

Counter source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM Exported constants

IS_LPTIM_AUTORELOAD

IS_LPTIM_COMPARE

IS_LPTIM_CLOCK_SOURCE

IS_LPTIM_CLOCK_PRESCALER

IS_LPTIM_CLOCK_PRESCALERDIV1

IS_LPTIM_OUTPUT_POLARITY

IS_LPTIM_CLOCK_SAMPLE_TIME

IS_LPTIM_CLOCK_POLARITY

IS_LPTIM_EXT_TRG_POLARITY

IS_LPTIM_TRIG_SAMPLE_TIME

IS_LPTIM_UPDATE_MODE

IS_LPTIM_COUNTER_SOURCE

IS_LPTIM_PERIOD

IS_LPTIM_PULSE

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable/Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

__HAL_LPTIM_START_CONTINUOUS

Description:

- Starts the LPTIM peripheral in Continuous or in single mode.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

__HAL_LPTIM_AUTORELOAD_SET

Description:

- Writes the passed parameter in the Autoreload register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: : Autoreload value

Return value:

- None

__HAL_LPTIM_COMPARE_SET

Description:

- Writes the passed parameter in the Compare register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: : Compare value

Return value:

- None

__HAL_LPTIM_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: : LPTIM flag to check This parameter can be a value of:
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clears the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.

- __FLAG__: : LPTIM flag to clear. This parameter can be a value of:
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- None.

__HAL_LPTIM_ENABLE_IT

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: : LPTIM handle.
- __INTERRUPT__: : LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.

- LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

__HAL_LPTIM_DISABLE_IT

Description:

- Disable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: : LPTIM handle.
- __INTERRUPT__: : LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

__HAL_LPTIM_GET_IT_SOURCE

Description:

- Checks whether the specified LPTIM interrupt is set or not.

Parameters:

- __HANDLE__: : LPTIM handle.
- __INTERRUPT__: : LPTIM interrupt to check. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter

direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- Interrupt: status.

__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_IT

Description:

- Enable interrupt on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None

__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_IT

Description:

- Disable interrupt on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None

__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_EVENT

Description:

- Enable event on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None.

__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_EVENT

Description:

- Disable event on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None.

__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on

	the LPTIM Wake-up Timer associated Exti line. **Return value:** • None.
__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable falling edge trigger on the LPTIM Wake-up Timer associated Exti line. **Return value:** • None.
__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE	**Description:** • Enable rising edge trigger on the LPTIM Wake-up Timer associated Exti line. **Return value:** • None.
__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE	**Description:** • Disable rising edge trigger on the LPTIM Wake-up Timer associated Exti line. **Return value:** • None.
__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • Enable rising & falling edge trigger on the LPTIM Wake-up Timer associated Exti line. **Return value:** • None.
__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable rising & falling edge trigger on the LPTIM Wake-up Timer associated Exti line. **Return value:** • None.
__HAL_LPTIM_WAKEUPTIMER_EXTI_GET_FLAG	**Description:** • Check whether the LPTIM Wake-up Timer associated Exti line interrupt flag is set or not. **Return value:** • Line: Status.

__HAL_LPTIM_WAKEUPTIMER_EXTI_CLEAR_FLAG	**Description:** • Clear the LPTIM Wake-up Timer associated Exti line flag. **Return value:** • None.
__HAL_LPTIM_WAKEUPTIMER_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on the LPTIM Wake-up Timer associated Exti line. **Return value:** • None.

Trigger polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

Flag definition

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_CMPOK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM_FLAG_CMPM

Interrupts definition

LPTIM_IT_DOWN

LPTIM_IT_UP

LPTIM_IT_ARROK

LPTIM_IT_CMPOK

LPTIM_IT_EXTTRIG

LPTIM_IT_ARRM

LPTIM_IT_CMPM

Output polarity

LPTIM_OUTPUTPOLARITY_HIGH

LPTIM_OUTPUTPOLARITY_LOW

Trigger sample time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

Trigger source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

Updating mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

32 HAL PCD Generic Driver

32.1 PCD Firmware driver registers structures

32.1.1 PCD_InitTypeDef

Data Fields

- ***uint32_t dev_endpoints***
- ***uint32_t speed***
- ***uint32_t ep0_mps***
- ***uint32_t phy_itface***
- ***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_Speed***
- ***uint32_t PCD_InitTypeDef::ep0_mps***
  Set the Endpoint 0 Max Packet size. This parameter can be any value of ***PCD_USB_EP0_MPS***
- ***uint32_t PCD_InitTypeDef::phy_itface***
  Select the used PHY interface. This parameter can be any value of ***PCD_USB_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 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

32.1.2 PCD_EPTypeDef

Data Fields

- ***uint8_t num***
- ***uint8_t is_in***
- ***uint8_t is_stall***
- ***uint8_t type***
- ***uint16_t pmaadress***
- ***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_USB_EP_Type***
- ***uint16_t PCD_EPTypeDef::pmaadress***
  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

32.1.3 PCD_HandleTypeDef

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***
- ***PCD_LPM_StateTypeDef LPM_State***
- ***uint32_t BESL***
- ***uint32_t lpm_active***
- ***uint32_t battery_charging_active***

- ***void * pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t PCD_HandleTypeDef::USB_Address***
  USB Address
- ***PCD_EPTypeDef PCD_HandleTypeDef::IN_ep[8]***
  IN endpoint parameters
- ***PCD_EPTypeDef 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
- ***PCD_LPM_StateTypeDef PCD_HandleTypeDef::LPM_State***
  LPM State
- ***uint32_t PCD_HandleTypeDef::BESL***
- ***uint32_t PCD_HandleTypeDef::lpm_active***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_HandleTypeDef::battery_charging_active***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE
- ***void* PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler

32.2 PCD Firmware driver API description

32.2.1 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();

32.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- *HAL_PCD_Init()*
- *HAL_PCD_DeInit()*
- *HAL_PCD_MspInit()*
- *HAL_PCD_MspDeInit()*

32.2.3 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()*

32.2.4 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()*
- *PCD_WritePMA()*
- *PCD_ReadPMA()*

32.2.5 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_PCD_GetState()***

32.2.6 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 OTG 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 OTG 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	• **None**

HAL_PCD_SuspendCallback

Function name	**void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)**
Function description	Suspend event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None**

HAL_PCD_ResumeCallback

Function name	**void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)**
Function description	Resume event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None**

HAL_PCD_ISOOUTIncompleteCallback

Function name	**void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Incomplete ISO OUT callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• **None**

HAL_PCD_ISOINIncompleteCallback

Function name	**void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Incomplete ISO IN callbacks.
Parameters	• **hpcd:** PCD handle

	•	**epnum:** endpoint number
Return values	•	**None**

HAL_PCD_ConnectCallback

Function name	**void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)**
Function description	Connection event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None**

HAL_PCD_DisconnectCallback

Function name	**void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)**
Function description	Disconnection event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None**

HAL_PCD_DevConnect

Function name	**HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)**
Function description	Connect the USB device.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DevDisconnect

Function name	**HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)**
Function description	Disconnect the USB device.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_SetAddress

Function name	**HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)**
Function description	Set the USB Device address.
Parameters	• **hpcd:** PCD handle • **address:** new device address
Return values	• **HAL:** status

HAL_PCD_EP_Open

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)**
Function description	Open and configure an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address • **ep_mps:** endpoint max packert size • **ep_type:** endpoint type
Return values	• **HAL:** status

HAL_PCD_EP_Close

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Deactivate an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_Receive

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)**
Function description	Receive an amount of data.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address • **pBuf:** pointer to the reception buffer • **len:** amount of data to be received
Return values	• **HAL:** status

HAL_PCD_EP_Transmit

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)**
Function description	Send an amount of data.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address • **pBuf:** pointer to the transmission buffer • **len:** amount of data to be sent
Return values	• **HAL:** status

HAL_PCD_EP_GetRxCount

Function name	**uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Get Received Data Size.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **Data:** Size

HAL_PCD_EP_SetStall

Function name	**HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Set a STALL condition over an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_ClrStall

Function name	**HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Clear a STALL condition over in an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_Flush

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Flush an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_ActivateRemoteWakeup

Function name	**HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)**
Function description	HAL_PCD_ActivateRemoteWakeup : active remote wakeup signalling.
Parameters	• **hpcd:** PCD handle
Return values	• **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	• **hpcd:** PCD handle
Return values	• **status:**

PCD_WritePMA

Function name	**void PCD_WritePMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)**
Function description	Copy a buffer from user memory area to packet memory area (PMA)
Parameters	• **USBx:** USB device • **pbUsrBuf:** pointer to user memory area. • **wPMABufAddr:** address into PMA. • **wNBytes:** no. of bytes to be copied.
Return values	• **None**

PCD_ReadPMA

Function name	**void PCD_ReadPMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)**
Function description	Copy a buffer from user memory area to packet memory area (PMA)
Parameters	• **USBx:** USB device • **pbUsrBuf:** pointer to user memory area. • **wPMABufAddr:** address into PMA. • **wNBytes:** no. of bytes to be copied.
Return values	• **None**

HAL_PCD_GetState

Function name	**PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)**
Function description	Return the PCD state.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** state

32.3 PCD Firmware driver defines

32.3.1 PCD

PCD End Point

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD_SNG_BUF

PCD_DBL_BUF

IS_PCD_ALL_INSTANCE

PCD Interrupt

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

USB_WAKEUP_EXTI_LINE

External interrupt line 18 Connected to the USB FS EXTI Line

__HAL_USB_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_WAKEUP_EXTI_DISABLE_IT

__HAL_USB_EXTI_GENERATE_SWIT

PCD_SET_ENDPOINT

PCD_GET_ENDPOINT

PCD_SET_EPTYPE

Description:

- sets the type in the endpoint register(bits EP_TYPE[1:0])

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wType: Endpoint Type.

Return value:

- None

PCD_GET_EPTYPE

Description:

- gets the type in the endpoint register(bits EP_TYPE[1:0])

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- Endpoint: Type

PCD_FreeUserBuffer

Description:

- free buffer used from the application realizing it to the line toggles bit SW_BUF in the double buffered endpoint register

Parameters:

- USBx: USB device.
- bEpNum

Return value:

- None

PCD_GET_DB_DIR

Description:

- gets direction of the double buffered endpoint

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- EP_DBUF_OUT: if the endpoint counter not yet programmed.

PCD_SET_EP_TX_STATUS

Description:

- sets the status for tx transfer (bits STAT_TX[1:0]).

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

- None

PCD_SET_EP_RX_STATUS

Description:

- sets the status for rx transfer (bits STAT_TX[1:0])

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

- None

PCD_SET_EP_TXRX_STATUS

Description:
- sets the status for rx & tx (bits STAT_TX[1:0] & STAT_RX[1:0])

Parameters:
- USBx: USB device.
- bEpNum: Endpoint Number.
- wStaterx: new state.
- wStatetx: new state.

Return value:
- None

PCD_GET_EP_TX_STATUS

Description:
- gets the status for tx/rx transfer (bits STAT_TX[1:0] /STAT_RX[1:0])

Parameters:
- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:
- status

PCD_GET_EP_RX_STATUS

PCD_SET_EP_TX_VALID

Description:
- sets directly the VALID tx/rx-status into the endpoint register

Parameters:
- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:
- None

PCD_SET_EP_RX_VALID

PCD_GET_EP_TX_STALL_STATUS

Description:
- checks stall condition in an endpoint.

Parameters:
- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:
- TRUE: = endpoint in stall condition.

PCD_GET_EP_RX_STALL_STATUS

PCD_SET_EP_KIND

Description:
- set & clear EP_KIND bit.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_EP_KIND

PCD_SET_OUT_STATUS

Description:

- Sets/clears directly STATUS_OUT bit in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_OUT_STATUS

PCD_SET_EP_DBUF

Description:

- Sets/clears directly EP_KIND bit in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_EP_DBUF

PCD_CLEAR_RX_EP_CTR

Description:

- Clears bit CTR_RX / CTR_TX in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_TX_EP_CTR

PCD_RX_DTOG

Description:

- Toggles DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_TX_DTOG

PCD_CLEAR_RX_DTOG

Description:

- Clears DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_TX_DTOG

PCD_SET_EP_ADDRESS

Description:

- Sets address in an endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- bAddr: Address.

Return value:

- None

PCD_GET_EP_ADDRESS

Description:

- Gets address in an endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_EP_TX_ADDRESS

PCD_EP_TX_CNT

PCD_EP_RX_ADDRESS

PCD_EP_RX_CNT

PCD_SET_EP_TX_ADDRESS

Description:

- sets address of the tx/rx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wAddr: address to be set (must be word aligned).

Return value:

- None

PCD_SET_EP_RX_ADDRESS

PCD_GET_EP_TX_ADDRESS

Description:

- Gets address of the tx/rx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- address: of the buffer.

PCD_GET_EP_RX_ADDRESS

PCD_CALC_BLK32

Description:

- Sets counter of rx buffer with no.

Parameters:

- dwReg: Register.
- wCount: Counter.
- wNBlocks: Nb of block

Return value:

- None

PCD_CALC_BLK2

PCD_SET_EP_CNT_RX_REG

PCD_SET_EP_RX_DBUF0_CNT

PCD_SET_EP_TX_CNT

Description:

- sets counter for the tx/rx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wCount: Counter value.

Return value:

- None

PCD_SET_EP_RX_CNT

PCD_GET_EP_TX_CNT

Description:

- gets counter of the tx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- Counter: value

PCD_GET_EP_RX_CNT

PCD_SET_EP_DBUF0_ADDR

Description:

- Sets buffer 0/1 address in a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

Return value:

- Counter: value

PCD_SET_EP_DBUF1_ADDR

PCD_SET_EP_DBUF_ADDR

Description:

- Sets addresses in a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.
- wBuf1Addr: = buffer 1 address.

Return value:

- None

PCD_GET_EP_DBUF0_ADDR

Description:

- Gets buffer 0/1 address of a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_GET_EP_DBUF1_ADDR

PCD_SET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 address of a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- bDir: endpoint dir EP_DBUF_OUT = OUT and EP_DBUF_IN = IN
- wCount: Counter value

Return value:

- None

PCD_SET_EP_DBUF1_CNT

PCD_SET_EP_DBUF_CNT

PCD_GET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 rx/tx counter for double buffering.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_GET_EP_DBUF1_CNT

PCD Speed

PCD_SPEED_HIGH

PCD_SPEED_FULL

PCD USB Core PHY

PCD_PHY_EMBEDDED

PCD USB 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 USB EP Type

PCD_EP_TYPE_CTRL

PCD_EP_TYPE_ISOC

PCD_EP_TYPE_BULK

PCD_EP_TYPE_INTR

33 HAL PCD Extension Driver

33.1 PCDEx Firmware driver API description

33.1.1 Peripheral extended features functions

This section contains the following APIs:

- *HAL_PCDEx_PMAConfig()*
- *HAL_PCDEx_ActivateBCD()*
- *HAL_PCDEx_DeActivateBCD()*
- *HAL_PCDEx_BCD_VBUSDetect()*
- *HAL_PCDEx_ActivateLPM()*
- *HAL_PCDEx_DeActivateLPM()*
- *HAL_PCDEx_LPM_Callback()*
- *HAL_PCDEx_BCD_Callback()*

33.1.2 Detailed description of functions

HAL_PCDEx_PMAConfig

Function name	**HAL_StatusTypeDef HAL_PCDEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaadress)**
Function description	Configure PMA for EP.
Parameters	• **hpcd:** : Device instance • **ep_addr:** endpoint address • **ep_kind:** endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used • **pmaadress:** 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

HAL_PCDEx_ActivateLPM

Function name	**HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)**
Function description	Activate LPM feature.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_DeActivateLPM

Function name	**HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)**
Function description	Deactivate LPM feature.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_ActivateBCD

Function name	**HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)**
Function description	Activate BatteryCharging feature.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_DeActivateBCD

Function name	**HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)**
Function description	Deactivate BatteryCharging feature.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_BCD_VBUSDetect

Function name	**void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)**
Function description	Handle BatteryCharging Process.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_LPM_Callback

Function name	**void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)**
Function description	Send LPM message to user layer callback.
Parameters	• **hpcd:** PCD handle • **msg:** LPM message
Return values	• **HAL:** status

HAL_PCDEx_BCD_Callback

Function name	**void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)**
Function description	Send BatteryCharging message to user layer callback.
Parameters	• **hpcd:** PCD handle • **msg:** LPM message
Return values	• **HAL:** status

34 HAL PWR Generic Driver

34.1 PWR Firmware driver registers structures

34.1.1 PWR_PVDTypeDef

Data Fields

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

Field Documentation

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

34.2 PWR Firmware driver API description

34.2.1 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_PWR_DeInit()***
- ***HAL_PWR_EnableBkUpAccess()***
- ***HAL_PWR_DisableBkUpAccess()***

34.2.2 Peripheral Control functions

Backup domain

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.

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).
- The PVD can use an external input analog voltage (PVD_IN) which is compared internally to VREFINT. The PVD_IN (PB7) has to be configured in Analog mode when PWR_PVDLevel_7 is selected (PLS[2:0] = 111).
- 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_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp pin is used to wake up 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: WakeUp Pin 1 on PA.00. WakeUp Pin 2 on PC.13. WakeUp Pin 3 on PE.06 .

Main and Backup Regulators configuration

Low Power modes configuration

The device features 5 low-power modes:

- Low power run mode: regulator in low power mode, limited clock frequency, limited number of peripherals running.
- Sleep mode: Cortex-M0+ core stopped, peripherals kept running.
- Low power sleep mode: Cortex-M0+ core stopped, limited clock frequency, limited number of peripherals running, regulator in low power mode.
- Stop mode: All clocks are stopped, regulator running, regulator in low power mode.
- Standby mode: VCORE domain powered off

Low power run mode

To further reduce the consumption when the system is in Run mode, the regulator can be configured in low power mode. In this mode, the system frequency should not exceed MSI frequency range1. In Low power run mode, all I/O pins keep the same state as in Run mode.

- Entry:
  - VCORE in range2
  - Decrease the system frequency not to exceed the frequency of MSI frequency range1.
  - The regulator is forced in low power mode using the HAL_PWREx_EnableLowPowerRunMode() function.
- Exit:
  - The regulator is forced in Main regulator mode using the HAL_PWREx_DisableLowPowerRunMode() function.
  - Increase the system frequency if needed.

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. If the WFE instruction was used to enter sleep mode, the MCU exits Sleep mode as soon as an event occurs.

Low power sleep mode

- Entry: The Low power sleep mode is entered by using the HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFx) functions with
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- The Flash memory can be switched off by using the control bits (SLEEP_PD in the FLASH_ACR register. This reduces power consumption but increases the wake-up time.
- Exit:
  - If the WFI instruction was used to enter Low power sleep mode, any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Low power sleep mode. If the WFE instruction was used to enter Low power sleep mode, the MCU exits Sleep mode as soon as an event occurs.

Stop mode

The Stop mode is based on the Cortex-M0+ deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the VCORE domain are stopped, the PLL, the MSI, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. To get the lowest consumption in Stop mode, the internal Flash memory also enters low power mode. When the Flash memory is in power-down mode, an additional startup delay is incurred when waking up from Stop mode. To minimize the consumption In Stop mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering Stop mode. They can be switched on again by software after exiting Stop mode using the ULP bit in the PWR_CR register. In Stop mode, all I/O pins keep the same state as in Run mode.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode function with:
  - Main regulator ON.
  - Low Power regulator ON.
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- Exit:
  - By issuing an interrupt or a wakeup event, the MSI or HSI16 RC oscillator is selected as system clock depending the bit STOPWUCK in the RCC_CFGR register

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M0+ deepsleep mode, with the voltage regulator disabled. The VCORE domain is consequently powered off. The PLL, the MSI, 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. To minimize the consumption In Standby mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering the Standby mode. They can be switched on again by software after exiting the Standby mode. function.

- Entry:
  - The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
- Exit:

- WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), 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 mode
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to:
    - Configure the EXTI Line 17 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
    - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function
    - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to:
    - Configure the EXTI Line 19 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
    - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.
    - Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to:
    - Configure the EXTI Line 20 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
    - Enable the RTC WakeUp Interrupt using the RTC_ITConfig() function.
    - Configure the RTC to generate the RTC WakeUp event using the RTC_WakeUpClockConfig(), RTC_SetWakeUpCounter() and RTC_WakeUpCmd() functions.
- RTC auto-wakeup (AWU) from the Standby mode
  - To wake up from the Standby mode with an RTC alarm event, it is necessary to:
    - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function.
    - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
  - To wake up from the Standby mode with an RTC Tamper or time stamp event, it is necessary to:
    - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.
    - Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
  - To wake up from the Standby mode with an RTC WakeUp event, it is necessary to:
    - Enable the RTC WakeUp Interrupt using the RTC_ITConfig() function
    - Configure the RTC to generate the RTC WakeUp event using the RTC_WakeUpClockConfig(), RTC_SetWakeUpCounter() and RTC_WakeUpCmd() functions.
- Comparator auto-wakeup (AWU) from the Stop mode
  - To wake up from the Stop mode with an comparator 1 or comparator 2 wakeup event, it is necessary to:

   – Configure the EXTI Line 21 for comparator 1 or EXTI Line 22 for comparator 2 to be sensitive to 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_EnableBkUpAccess()*
- *HAL_PWR_DisableBkUpAccess()*
- *HAL_PWR_ConfigPVD()*
- *HAL_PWR_EnablePVD()*
- *HAL_PWR_DisablePVD()*
- *HAL_PWR_EnableWakeUpPin()*
- *HAL_PWR_DisableWakeUpPin()*
- *HAL_PWR_EnterSLEEPMode()*
- *HAL_PWR_EnterSTOPMode()*
- *HAL_PWR_EnterSTANDBYMode()*
- *HAL_PWR_EnableSleepOnExit()*
- *HAL_PWR_DisableSleepOnExit()*
- *HAL_PWR_EnableSEVOnPend()*
- *HAL_PWR_DisableSEVOnPend()*
- *HAL_PWR_PVD_IRQHandler()*
- *HAL_PWR_PVDCallback()*

34.2.3 Detailed description of functions

HAL_PWR_DeInit

Function name	**void HAL_PWR_DeInit (void)**
Function description	Deinitializes the HAL 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).
Return values	• **None**
Notes	• If the HSE divided by 2, 4, 8 or 16 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.
Return values	• **None**
Notes	• Applies to RTC registers, RTC backup data registers. • If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_ConfigPVD

Function name	**void HAL_PWR_ConfigPVD (PWR_PVDTypeDef * sConfigPVD)**
Function description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	• **sConfigPVD:** pointer to an PWR_PVDTypeDef 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_PWR_PVD_IRQHandler

Function name	**void HAL_PWR_PVD_IRQHandler (void)**
Function description	This function handles the PWR PVD interrupt request.
Return values	• **None**
Notes	• This API should be called under the PVD_IRQHandler().

HAL_PWR_PVDCallback

Function name	**void HAL_PWR_PVDCallback (void)**
Function description	PWR PVD interrupt callback.
Return values	• **None**

HAL_PWR_EnableWakeUpPin

Function name	**void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)**
Function description	Enables the WakeUp PINx functionality.
Parameters	• **WakeUpPinx:** Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: – PWR_WAKEUP_PIN1 – PWR_WAKEUP_PIN2 – PWR_WAKEUP_PIN3 for stm32l07xxx and stm32l08xxx

devices only.

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 one of the following values: – PWR_WAKEUP_PIN1 – PWR_WAKEUP_PIN2 – PWR_WAKEUP_PIN3 for stm32l07xxx and stm32l08xxx devices only.
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 in 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, MSI or HSI16 RCoscillator is selected as system clock depending the bit STOPWUCK in the RCC_CFGR register. • 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. • Before entering in this function, it is important to ensure that the WUF wakeup flag is cleared. To perform this action, it is possible to call the following macro : __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU)

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. This parameter can be one of the following values: – PWR_MAINREGULATOR_ON: SLEEP mode with regulator ON – PWR_LOWPOWERREGULATOR_ON: SLEEP mode with low power regulator ON • **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	• **None**
Notes	• In Standby mode, all I/O pins are high impedance except for: Reset pad (still available)RTC_AF1 pin (PC13) if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out.RTC_AF2 pin (PC13) if configured for tamper.WKUP pin 1 (PA00) if enabled.WKUP pin 2 (PC13) if enabled.WKUP pin 3 (PE06) if enabled, for stm32l07xxx and stm32l08xxx devices only.WKUP pin 3 (PA02) if enabled, for stm32l031xx devices only.

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	• **None**
Notes	• Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name	**void HAL_PWR_DisableSleepOnExit (void)**
Function description	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.
Return values	• **None**
Notes	• Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name	**void HAL_PWR_EnableSEVOnPend (void)**
Function description	Enables CORTEX M0+ SEVONPEND bit.
Return values	• **None**
Notes	• 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 M0+ SEVONPEND bit.
Return values	• **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.

34.3 PWR Firmware driver defines

34.3.1 PWR

PWR Exported Macros

__HAL_PWR_VOLTAGESCALING_CONFIG

Description:

- macros configure the main internal regulator output voltage.

Parameters:

- __REGULATOR__: specifies the regulator output voltage to achieve a tradeoff between performance and power consumption when the device does not operate at the maximum frequency (refer to the datasheets for more details). This parameter can be one of the following values:
  – PWR_REGULATOR_VOLTAGE_SCALE1: Regulator voltage output Scale 1 mode, System

frequency up to 32 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE2: Regulator voltage output Scale 2 mode, System frequency up to 16 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE3: Regulator voltage output Scale 3 mode, System frequency up to 4.2 MHz

Return value:

- None

__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 B), 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.
  - PWR_FLAG_VREFINTRDY: Internal voltage reference (VREFINT) ready flag. This bit indicates the state of the internal voltage reference, VREFINT.
  - PWR_FLAG_VOS: Voltage Scaling select flag. A delay is required for the internal regulator to be ready after the

voltage range is changed. The VOSF bit indicates that the regulator has reached the voltage level defined with bits VOS of PWR_CR register.
  - PWR_FLAG_REGLP: Regulator LP flag. When the MCU exits from Low power run mode, this bit stays at 1 until the regulator is ready in main mode. A polling on this bit is recommended to wait for the regulator main mode. This bit is reset by hardware when the regulator is ready.

Return value:

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

__HAL_PWR_CLEAR_FLAG

Description:

- Clear the PWR 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

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable interrupt on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable interrupt on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

Description:

- Enable event on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_EVENT

Description:

- Disable event on PVD Exti Line 16.

Return value:

	• None.
__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE	**Description:** • PVD EXTI line configuration: set falling edge trigger. **Return value:** • None.
__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable the PVD Extended Interrupt Falling Trigger. **Return value:** • None.
__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE	**Description:** • PVD EXTI line configuration: set rising edge trigger. **Return value:** • None.
__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE	**Description:** • Disable the PVD Extended Interrupt Rising Trigger. **Return value:** • None.
__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • PVD EXTI line configuration: set rising & falling edge trigger. **Return value:** • None.
__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable the PVD Extended Interrupt Rising & Falling Trigger. **Return value:** • None.
__HAL_PWR_PVD_EXTI_GET_FLAG	**Description:** • Check whether the specified PVD EXTI interrupt flag is set or not. **Return value:** • EXTI: PVD Line Status.
__HAL_PWR_PVD_EXTI_CLEAR_FLAG	**Description:** • Clear the PVD EXTI flag.

	Return value: • None.
__HAL_PWR_PVD_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None.
__HAL_PWR_PVD_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None.

PWREx Flag Setting Time Out Value

PWR_FLAG_SETTING_DELAY_US

PWR Flag

PWR_FLAG_WU

PWR_FLAG_SB

PWR_FLAG_PVDO

PWR_FLAG_VREFINTRDY

PWR_FLAG_VOS

PWR_FLAG_REGLP

PVD detection level

PWR_PVDLEVEL_0

PWR_PVDLEVEL_1

PWR_PVDLEVEL_2

PWR_PVDLEVEL_3

PWR_PVDLEVEL_4

PWR_PVDLEVEL_5

PWR_PVDLEVEL_6

PWR_PVDLEVEL_7

PWR PVD Mode

PWR_PVD_MODE_NORMAL	basic mode is used
PWR_PVD_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
PWR_PVD_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVD_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_MODE_IT

PVD_MODE_EVT

PVD_RISING_EDGE

PVD_FALLING_EDGE

PWR Register alias address

PWR_WAKEUP_PIN1

PWR_WAKEUP_PIN2

PWR_WAKEUP_PIN3

PWR Regulator state in SLEEP/STOP mode

PWR_MAINREGULATOR_ON

PWR_LOWPOWERREGULATOR_ON

PWR Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE1

PWR_REGULATOR_VOLTAGE_SCALE2

PWR_REGULATOR_VOLTAGE_SCALE3

IS_PWR_VOLTAGE_SCALING_RANGE

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

PWR_SLEEPENTRY_WFE

PWR STOP mode entry

PWR_STOPENTRY_WFI

PWR_STOPENTRY_WFE

35 HAL PWR Extension Driver

35.1 PWREx Firmware driver defines

35.1.1 PWREx

PWREx Exported Macros

__HAL_PWR_FLASHWAKEUP_ENABLE	**Notes:** • When entering low power mode (stop or standby only), if DS_EE_KOFF and RUN_PD of FLASH_ACR register are both set , the Flash memory will not be woken up when exiting from deep-sleep mode.
__HAL_PWR_FLASHWAKEUP_DISABLE	**Notes:** • When entering low power mode (stop or standby only), if DS_EE_KOFF and RUN_PD of FLASH_ACR register are both set , the Flash memory will not be woken up when exiting from deep-sleep mode.

36 HAL RCC Generic Driver

36.1 RCC Firmware driver registers structures

36.1.1 RCC_PLLInitTypeDef

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLMUL***
- ***uint32_t PLLDIV***

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::PLLDIV***
  PLLDIV: Division factor for PLL VCO input clock This parameter must be a value of ***RCC_PLL_Division_Factor***

36.1.2 RCC_OscInitTypeDef

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***uint32_t HSI48State***
- ***uint32_t MSIState***
- ***uint32_t MSICalibrationValue***
- ***uint32_t MSIClockRange***
- ***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 = 0x1F
- ***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***
- ***uint32_t RCC_OscInitTypeDef::MSIState***
  The new state of the MSI. This parameter can be a value of ***RCC_MSI_Config***
- ***uint32_t RCC_OscInitTypeDef::MSICalibrationValue***
  The MSI calibration trimming value. (default is RCC_MSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t RCC_OscInitTypeDef::MSIClockRange***
  The MSI frequency range. This parameter can be a value of ***RCC_MSI_Clock_Range***
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL***
  PLL structure parameters

36.1.3 RCC_ClkInitTypeDef

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

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::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_AHB_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***

36.2 RCC Firmware driver API description

36.2.1 RCC specific features

After reset the device is running from multispeed internal oscillator clock (MSI 2.097MHz) with Flash 0 wait state and Flash prefetch buffer is disabled, 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 MSI 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 (I2S, RTC, ADC, USB OTG FS/SDIO/RNG) (*) SDIO only for STM32L0xxxD devices

36.2.2 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.

36.2.3 Initialization and de-initialization functions

This section provides functions allowing to configure the internal/external oscillators (MSI, HSE, HSI, LSE, LSI, PLL, CSS and MCO) and the System buses clocks (SYSCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

1. MSI (Multispeed internal), Seven frequency ranges are available: 65.536 kHz, 131.072 kHz, 262.144 kHz, 524.288 kHz, 1.048 MHz, 2.097 MHz (default value) and 4.194 MHz.
2. HSI (high-speed internal), 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
3. LSI (low-speed internal), ~37 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 1 to 24 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 or HSE), featuring different output clocks:
   - The first output is used to generate the high speed system clock (up to 32 MHz)
   - The second output is used to generate the clock for the USB OTG FS (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 MSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M0+ NMI (Non-Maskable Interrupt) exception vector.
8. MCO1/MCO2/MCO3 (microcontroller clock output), used to output SYSCLK, HSI, LSI, MSI, LSE, HSE, HSI48 or PLL clock (through a configurable prescaler) on PA8/PA9/PB13 pins.

System, AHB and APB buses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): MSI, 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) and APB2 (PCLK2) clocks are 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. All the peripheral clocks are derived from the System clock (SYSCLK) except: RTC: RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 16. You have to use @ref __HAL_RCC_RTC_CONFIG() and @ref __HAL_RCC_RTC_ENABLE() macros to configure this clock. LCD: LCD clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 16. You have to use @ref __HAL_RCC_LCD_CONFIG() macros to configure this clock. USB FS and RNG: USB FS require a frequency equal to 48 MHz to work correctly. This clock is derived of the main PLL through PLL Multiplier or HSI48 RC oscillator. IWDG clock which is always the LSI clock.
2. The maximum frequency of the SYSCLK and HCLK is 32 MHz, PCLK2 32 MHz and PCLK1 32 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly.

This section contains the following APIs:

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

36.2.4 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_GetSysClockFreq()***
- ***HAL_RCC_GetHCLKFreq()***
- ***HAL_RCC_GetPCLK1Freq()***
- ***HAL_RCC_GetPCLK2Freq()***
- ***HAL_RCC_GetOscConfig()***
- ***HAL_RCC_GetClockConfig()***
- ***HAL_RCC_NMI_IRQHandler()***
- ***HAL_RCC_CSSCallback()***

36.2.5 Detailed description of functions

HAL_RCC_DeInit

Function name	**void HAL_RCC_DeInit (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: MSI ON and used as system clock sourceHSI, HSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS

and MCO1/MCO2/MCO3 OFFAll interrupts disabled
- This function does not modify the configuration of the Peripheral clocksLSI, LSE and RTC clocksHSI48 clock

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	**HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)**
Function description	Initializes the CPU, AHB and APB buses clocks according to the specified parameters in the RCC_ClkInitStruct.
Parameters	• **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef 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	• **HAL:** status
Notes	• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function • The MSI is used (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 HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source. • Depending on the device voltage range, the software has to set correctly HPRE[3:0] bits to ensure that HCLK not exceed

the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

HAL_RCC_MCOConfig

Function name	**void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)**
Function description	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_MCO2 Clock source to output on MCO2 pin(PA9). – RCC_MCO3 Clock source to output on MCO3 pin(PB13) • **RCC_MCOSource:** specifies the clock source to output. 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 selected as MCO clock – RCC_MCO1SOURCE_HSE HSE selected as MCO clock – RCC_MCO1SOURCE_MSI MSI oscillator clock selected as MCO clock – RCC_MCO1SOURCE_PLLCLK PLL clock selected as MCO clock – RCC_MCO1SOURCE_LSI LSI clock selected as MCO clock – RCC_MCO1SOURCE_LSE LSE clock selected as MCO clock – RCC_MCO1SOURCE_HSI48 HSI48 clock 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
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_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 MSI, function returns a value based on MSI Value as defined by the MSI range. • If SYSCLK source is HSI, function returns values based on HSI_VALUE(*) • If SYSCLK source is HSE, function returns a value based on HSE_VALUE(**) • If SYSCLK source is PLL, function returns a value based on HSE_VALUE(**) or HSI_VALUE(*) multiplied/divided by the PLL factors. • (*) HSI_VALUE is a constant defined in stm32l0xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature. • (**) HSE_VALUE is a constant defined in stm32l0xx_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_GetPCLK2Freq

Function name	**uint32_t HAL_RCC_GetPCLK2Freq (void)**
Function description	Returns the PCLK2 frequency.
Return values	• **PCLK2:** frequency
Notes	• Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_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**

36.3 RCC Firmware driver defines

36.3.1 RCC

AHB Peripheral Clock Sleep Enable Disable

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_MIF_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM_CLK_SLEEP_ENABLE

__HAL_RCC_DMA1_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_MIF_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM_CLK_SLEEP_DISABLE

__HAL_RCC_DMA1_CLK_SLEEP_DISABLE

AHB Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MIF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SRAM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_MIF_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SRAM_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMA1_IS_CLK_SLEEP_DISABLED

AHB Clock Source

RCC_SYSCLK_DIV1	SYSCLK not divided
RCC_SYSCLK_DIV2	SYSCLK divided by 2
RCC_SYSCLK_DIV4	SYSCLK divided by 4
RCC_SYSCLK_DIV8	SYSCLK divided by 8
RCC_SYSCLK_DIV16	SYSCLK divided by 16
RCC_SYSCLK_DIV64	SYSCLK divided by 64
RCC_SYSCLK_DIV128	SYSCLK divided by 128
RCC_SYSCLK_DIV256	SYSCLK divided by 256
RCC_SYSCLK_DIV512	SYSCLK divided by 512

AHB Peripheral Force Release Reset

__HAL_RCC_AHB_FORCE_RESET

__HAL_RCC_DMA1_FORCE_RESET

__HAL_RCC_MIF_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_AHB_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

__HAL_RCC_DMA1_RELEASE_RESET

__HAL_RCC_MIF_RELEASE_RESET

AHB Peripheral Clock Enable Disable

__HAL_RCC_DMA1_CLK_ENABLE

__HAL_RCC_MIF_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_DMA1_CLK_DISABLE

__HAL_RCC_MIF_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

AHB Peripheral Clock Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_ENABLED

__HAL_RCC_MIF_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_DMA1_IS_CLK_DISABLED

__HAL_RCC_MIF_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

APB1 APB2 Clock Source

RCC_HCLK_DIV1	HCLK not divided
RCC_HCLK_DIV2	HCLK divided by 2
RCC_HCLK_DIV4	HCLK divided by 4
RCC_HCLK_DIV8	HCLK divided by 8
RCC_HCLK_DIV16	HCLK divided by 16

APB1 Peripheral Clock Enable Disable

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_WWDG_CLK_DISABLE

__HAL_RCC_PWR_CLK_DISABLE

APB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_DISABLED

__HAL_RCC_PWR_IS_CLK_DISABLED

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE

__HAL_RCC_PWR_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

__HAL_RCC_PWR_CLK_SLEEP_DISABLE

APB1 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_WWDG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_ENABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_DISABLED

APB1 Peripheral Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_WWDG_FORCE_RESET

__HAL_RCC_PWR_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_WWDG_RELEASE_RESET

__HAL_RCC_PWR_RELEASE_RESET

APB2 Peripheral Clock Enable Disable

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_DBGMCU_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_DBGMCU_CLK_DISABLE

APB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

__HAL_RCC_DBGMCU_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_DISABLED

__HAL_RCC_DBGMCU_IS_CLK_DISABLED

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

__HAL_RCC_DBGMCU_CLK_SLEEP_ENABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_DBGMCU_CLK_SLEEP_DISABLE

APB2 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DBGMCU_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DBGMCU_IS_CLK_SLEEP_DISABLED

APB2 Peripheral Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_DBGMCU_FORCE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_DBGMCU_RELEASE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

BitAddress AliasRegion

RCC_OFFSET

RCC_CR_OFFSET

RCC_CFGR_OFFSET

RCC_CSR_OFFSET

RCC_CR_BYTE2_ADDRESS

CIER_BYTE0_ADDRESS

Flags

RCC_FLAG_HSIRDY	Internal High Speed clock ready flag
RCC_FLAG_HSIDIV	HSI16 divider flag
RCC_FLAG_MSIRDY	MSI clock ready flag
RCC_FLAG_HSERDY	External High Speed clock ready flag
RCC_FLAG_PLLRDY	PLL clock ready flag
RCC_FLAG_LSIRDY	Internal Low Speed oscillator Ready
RCC_FLAG_LSERDY	External Low Speed oscillator Ready
RCC_FLAG_LSECSS	CSS on LSE failure Detection
RCC_FLAG_OBLRST	Options bytes loading reset flag
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_FWRST	RCC flag FW reset
RCC_FLAG_HSI48RDY	HSI48 clock ready flag

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be enabled. This

parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY main PLL ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_LSECSS LSE CSS interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt (not available on all devices)

Notes:

- The CSS interrupt doesn't have an enable bit; once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI will be executed indefinitely, and since NMI has higher priority than any other IRQ (and main program) the application will be stacked in the NMI ISR unless the CSS interrupt pending bit is cleared.

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY main PLL ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_LSECSS LSE CSS interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt (not available on all devices)

Notes:

- The CSS interrupt doesn't have an enable bit; once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI will be executed indefinitely, and since NMI has higher priority than any other IRQ (and main program) the application will be stacked in the NMI ISR unless the CSS interrupt pending bit is cleared.

__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_MSIRDY MSI ready interrupt – RCC_IT_LSECSS LSE CSS interrupt – RCC_IT_HSI48RDY HSI48 ready interrupt (not available on all devices) – RCC_IT_CSS Clock Security System 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 PLL ready interrupt – RCC_IT_MSIRDY MSI ready interrupt – RCC_IT_LSECSS LSE CSS interrupt – RCC_IT_CSS Clock Security System 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_HSI48RDY HSI48 oscillator clock ready (not available on all devices)
- RCC_FLAG_HSIDIV HSI16 divider flag
- RCC_FLAG_MSIRDY MSI oscillator clock ready
- RCC_FLAG_HSERDY HSE oscillator clock ready
- RCC_FLAG_PLLRDY PLL clock ready
- RCC_FLAG_LSECSS LSE oscillator clock CSS detected
- RCC_FLAG_LSERDY LSE oscillator clock ready
- RCC_FLAG_FWRST Firewall reset
- RCC_FLAG_LSIRDY LSI oscillator clock ready
- RCC_FLAG_OBLRST Option Byte Loader (OBL) 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_LPWRRST Low Power reset

Return value:

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

Get Clock source

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
  - RCC_SYSCLKSOURCE_MSI MSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSI HSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSE HSE oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_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_MSI MSI used as system clock – RCC_SYSCLKSOURCE_STATUS_HSI HSI used as system clock – RCC_SYSCLKSOURCE_STATUS_HSE HSE used as system clock – RCC_SYSCLKSOURCE_STATUS_PLLCLK PLL used as system clock

RTC HSE Prescaler

RCC_RTC_HSE_DIV_2	HSE is divided by 2 for RTC clock
RCC_RTC_HSE_DIV_4	HSE is divided by 4 for RTC clock
RCC_RTC_HSE_DIV_8	HSE is divided by 8 for RTC clock
RCC_RTC_HSE_DIV_16	HSE is divided by 16 for RTC 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.

HSI48 Config

RCC_HSI48_OFF

RCC_HSI48_ON

HSI Config

RCC_HSI_OFF	HSI clock deactivation
RCC_HSI_ON	HSI clock activation
RCC_HSI_DIV4	HSI_DIV4 clock activation
RCC_HSI_OUTEN	HSI_OUTEN clock activation
RCC_HSICALIBRATION_DEFAULT	

HSI Configuration

__HAL_RCC_HSI_CONFIG

Description:

- Macro to enable or disable the Internal High Speed oscillator (HSI).

Parameters:

- __STATE__: specifies the new state of the HSI. This parameter can be one of the following values:
  - RCC_HSI_OFF turn OFF the HSI oscillator
  - RCC_HSI_ON turn ON the HSI oscillator
  - RCC_HSI_DIV4 turn ON the HSI oscillator and divide it by 4

Notes:

- 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 to clock the PLL and/or system clock. HSI can not be stopped 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 stop the HSI. The HSI is stopped by hardware when entering STOP and STANDBY modes.
- When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock

	cycles.
__HAL_RCC_HSI_ENABLE	**Notes:** • The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. 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.

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_MSIRDY	MSI Ready Interrupt flag
RCC_IT_LSECSS	LSE Clock Security System Interrupt flag
RCC_IT_CSS	Clock Security System Interrupt flag
RCC_IT_HSI48RDY	HSI48 Ready Interrupt flag

IOPORT Peripheral Clock Enable Disable

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOH_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOH_CLK_DISABLE

IOPORT Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

IOPORT Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOA_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_DISABLED

IOPORT Peripheral Force Release Reset

__HAL_RCC_IOP_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOH_FORCE_RESET

__HAL_RCC_IOP_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOH_RELEASE_RESET

IOPORT Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOH_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOH_IS_CLK_DISABLED

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.

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

RCC_MCO1SOURCE_SYSCLK

RCC_MCO1SOURCE_MSI

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_LSE

RCC_MCO1SOURCE_LSI

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLLCLK

RCC_MCO1SOURCE_HSI48

MCO Clock Prescaler

RCC_MCODIV_1

RCC_MCODIV_2

RCC_MCODIV_4

RCC_MCODIV_8

RCC_MCODIV_16

MCO Index

RCC_MCO1

RCC_MCO2

RCC_MCO3

MSI Clock Range

RCC_MSIRANGE_0	MSI = 65.536 KHz

RCC_MSIRANGE_1	MSI = 131.072 KHz
RCC_MSIRANGE_2	MSI = 262.144 KHz
RCC_MSIRANGE_3	MSI = 524.288 KHz
RCC_MSIRANGE_4	MSI = 1.048 MHz
RCC_MSIRANGE_5	MSI = 2.097 MHz
RCC_MSIRANGE_6	MSI = 4.194 MHz

MSI Config

RCC_MSI_OFF

RCC_MSI_ON

RCC_MSICALIBRATION_DEFAULT

MSI Configuration

__HAL_RCC_MSI_ENABLE

Notes:

- After enabling the MSI, the application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source.

__HAL_RCC_MSI_DISABLE

Notes:

- The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the MSI. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

__HAL_RCC_MSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro adjusts Internal Multi Speed oscillator (MSI) calibration value.

Parameters:

- _MSICALIBRATIONVALUE_: specifies the calibration trimming value. (default is RCC_MSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 0xFF.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal MSI RC. Refer to the Application Note AN3300 for more details on how to calibrate the MSI.

__HAL_RCC_MSI_RANGE_CONFIG

__HAL_RCC_GET_MSI_RANGE

Description:

- Macro to get the Internal Multi Speed oscillator (MSI) clock range in run mode.

Return value:

- MSI: clock range. This parameter must be one of the following values:
  - RCC_MSIRANGE_0 MSI clock is around 65.536 KHz
  - RCC_MSIRANGE_1 MSI clock is around 131.072 KHz
  - RCC_MSIRANGE_2 MSI clock is around 262.144 KHz
  - RCC_MSIRANGE_3 MSI clock is around 524.288 KHz
  - RCC_MSIRANGE_4 MSI clock is around 1.048 MHz
  - RCC_MSIRANGE_5 MSI clock is around 2.097 MHz (default after Reset or wake-up from STANDBY)
  - RCC_MSIRANGE_6 MSI clock is around 4.194 MHz

Oscillator Type

RCC_OSCILLATORTYPE_NONE

RCC_OSCILLATORTYPE_HSE

RCC_OSCILLATORTYPE_HSI

RCC_OSCILLATORTYPE_LSE

RCC_OSCILLATORTYPE_LSI

RCC_OSCILLATORTYPE_MSI

RCC_OSCILLATORTYPE_HSI48

PLL Clock Source

RCC_PLLSOURCE_HSI	HSI clock selected as PLL entry clock source
RCC_PLLSOURCE_HSE	HSE clock selected as PLL entry clock source

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:

- The main PLL can not be disabled if it is used as system clock source

__HAL_RCC_PLL_CONFIG

Description:

- Macro to configure the main 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:
  - RCC_PLL_MUL3 PLLVCO = PLL clock entry x 3
  - RCC_PLL_MUL4 PLLVCO = PLL clock entry x 4
  - RCC_PLL_MUL6 PLLVCO = PLL clock entry x 6
  - RCC_PLL_MUL8 PLLVCO = PLL clock entry x 8
  - RCC_PLL_MUL12 PLLVCO = PLL clock entry x 12
  - RCC_PLL_MUL16 PLLVCO = PLL clock entry x 16
  - RCC_PLL_MUL24 PLLVCO = PLL clock entry x 24
  - RCC_PLL_MUL32 PLLVCO = PLL clock entry x 32
  - RCC_PLL_MUL48 PLLVCO = PLL clock entry x 48
- __PLLDIV__: specifies the division factor

for PLL VCO input clock This parameter can be one of the following values:
- RCC_PLL_DIV2 PLL clock output = PLLVCO / 2
- RCC_PLL_DIV3 PLL clock output = PLLVCO / 3
- RCC_PLL_DIV4 PLL clock output = PLLVCO / 4

Notes:

- This function must be used only when the main PLL is disabled.
- The PLL VCO clock frequency must not exceed 96 MHz when the product is in Range 1, 48 MHz when the product is in Range 2 and 24 MHz when the product is in Range 3.

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- 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_HSI HSI oscillator clock selected as PLL input clock
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL input clock

PLL Division Factor

RCC_PLL_DIV2

RCC_PLL_DIV3

RCC_PLL_DIV4

PLL Multiplication Factor

RCC_PLL_MUL3

RCC_PLL_MUL4

RCC_PLL_MUL6

RCC_PLL_MUL8

RCC_PLL_MUL12

RCC_PLL_MUL16

RCC_PLL_MUL24

RCC_PLL_MUL32

RCC_PLL_MUL48

RCC RTC Clock Configuration

__HAL_RCC_RTC_CLKPRESCALER

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_DIV2 HSE divided by 2 selected as RTC clock
  - RCC_RTCCLKSOURCE_HSE_DIV4 HSE divided by 4 selected as RTC clock
  - RCC_RTCCLKSOURCE_HSE_DIV8 HSE divided by 8 selected as RTC clock
  - RCC_RTCCLKSOURCE_HSE_DIV16 HSE divided by 16 selected as RTC clock

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). RTC prescaler cannot be modified if HSE is enabled (HSEON = 1).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_RTC_CONFIG

__HAL_RCC_GET_RTC_SOURCE

Description:

- 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_DIVX HSE divided by X selected as RTC clock (X can be retrieved thanks to __HAL_RCC_GET_RTC_HSE_PRESCALER()

__HAL_RCC_GET_RTC_HSE_PRESCALER

Description:

- Get the RTC and LCD HSE clock divider (RTCCLK / LCDCLK).

Return value:

- Returned: value can be one of the following values:
  - RCC_RTC_HSE_DIV_2 HSE divided by 2 selected as RTC clock
  - RCC_RTC_HSE_DIV_4 HSE divided by 4 selected as RTC clock
  - RCC_RTC_HSE_DIV_8 HSE divided by 8 selected as RTC clock
  - RCC_RTC_HSE_DIV_16 HSE divided by 16 selected as RTC clock

__HAL_RCC_RTC_ENABLE

Notes:

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

__HAL_RCC_RTC_DISABLE

Notes:

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

__HAL_RCC_BACKUPRESET_FORCE

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

__HAL_RCC_BACKUPRESET_RELEASE

RTC LCD Clock Source

RCC_RTCCLKSOURCE_NO_CLK	No clock
RCC_RTCCLKSOURCE_LSE	LSE oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_LSI	LSI oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIVX	HSE oscillator clock divided by X used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIV2	HSE oscillator clock divided by 2 used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIV4	HSE oscillator clock divided by 4 used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIV8	HSE oscillator clock divided by 8 used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIV16	HSE oscillator clock divided by 16 used as RTC clock

System Clock Source

RCC_SYSCLKSOURCE_MSI	MSI selected as system clock
RCC_SYSCLKSOURCE_HSI	HSI selected as system clock
RCC_SYSCLKSOURCE_HSE	HSE selected as system clock
RCC_SYSCLKSOURCE_PLLCLK	PLL selected as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_MSI	MSI used as system clock
RCC_SYSCLKSOURCE_STATUS_HSI	HSI used as system clock
RCC_SYSCLKSOURCE_STATUS_HSE	HSE used as system clock
RCC_SYSCLKSOURCE_STATUS_PLLCLK	PLL used as system clock

System Clock Type

RCC_CLOCKTYPE_SYSCLK	SYSCLK to configure
RCC_CLOCKTYPE_HCLK	HCLK to configure
RCC_CLOCKTYPE_PCLK1	PCLK1 to configure
RCC_CLOCKTYPE_PCLK2	PCLK2 to configure

RCC Timeout

RCC_DBP_TIMEOUT_VALUE

RCC_LSE_TIMEOUT_VALUE

CLOCKSWITCH_TIMEOUT_VALUE

HSE_TIMEOUT_VALUE

MSI_TIMEOUT_VALUE

HSI_TIMEOUT_VALUE

HSI48_TIMEOUT_VALUE

HSI48_TIMEOUT_VALUE

LSI_TIMEOUT_VALUE

PLL_TIMEOUT_VALUE

37 HAL RCC Extension Driver

37.1 RCCEx Firmware driver registers structures

37.1.1 RCC_PeriphCLKInitTypeDef

Data Fields

- ***uint32_t PeriphClockSelection***
- ***uint32_t RTCClockSelection***
- ***uint32_t LCDClockSelection***
- ***uint32_t Usart1ClockSelection***
- ***uint32_t Usart2ClockSelection***
- ***uint32_t Lpuart1ClockSelection***
- ***uint32_t I2c1ClockSelection***
- ***uint32_t I2c3ClockSelection***
- ***uint32_t LptimClockSelection***
- ***uint32_t UsbClockSelection***

Field Documentation

- ***uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  specifies the RTC clock source. This parameter can be a value of ***RCC_RTC_LCD_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::LCDClockSelection***
  specifies the LCD clock source. This parameter can be a value of ***RCC_RTC_LCD_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
  USART1 clock source This parameter can be a value of ***RCCEx_USART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection***
  USART2 clock source This parameter can be a value of ***RCCEx_USART2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection***
  LPUART1 clock source This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection***
  I2C1 clock source This parameter can be a value of ***RCCEx_I2C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection***
  I2C3 clock source This parameter can be a value of ***RCCEx_I2C3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::LptimClockSelection***
  LPTIM1 clock source This parameter can be a value of ***RCCEx_LPTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection***
  Specifies USB and RNG Clock Selection This parameter can be a value of ***RCCEx_USB_Clock_Source***

37.1.2 RCC_CRSInitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t ReloadValue***
- ***uint32_t ErrorLimitValue***
- ***uint32_t HSI48CalibrationValue***

Field Documentation

- ***uint32_t RCC_CRSInitTypeDef::Prescaler***
  Specifies the division factor of the SYNC signal. This parameter can be a value of ***RCCEx_CRS_SynchroDivider***
- ***uint32_t RCC_CRSInitTypeDef::Source***
  Specifies the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroSource***
- ***uint32_t RCC_CRSInitTypeDef::Polarity***
  Specifies the input polarity for the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroPolarity***
- ***uint32_t RCC_CRSInitTypeDef::ReloadValue***
  Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro **__HAL_RCC_CRS_RELOADVALUE_CALCULATE(__FTARGET__, __FSYNC__)** This parameter must be a number between 0 and 0xFFFF or a value of ***RCCEx_CRS_ReloadValueDefault*** .
- ***uint32_t RCC_CRSInitTypeDef::ErrorLimitValue***
  Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of ***RCCEx_CRS_ErrorLimitDefault***
- ***uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue***
  Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of ***RCCEx_CRS_HSI48CalibrationDefault***

37.1.3 RCC_CRSSynchroInfoTypeDef

Data Fields

- ***uint32_t ReloadValue***
- ***uint32_t HSI48CalibrationValue***
- ***uint32_t FreqErrorCapture***
- ***uint32_t FreqErrorDirection***

Field Documentation

- ***uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue***
  Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue***
  Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture***
  Specifies the value loaded in the .FECAP, the frequency error counter value latched in

the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF

- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection***
  Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of ***RCCEx_CRS_FreqErrorDirection***

37.2 RCCEx Firmware driver API description

37.2.1 Extended Peripheral Control functions

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

Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKFreq()***
- ***HAL_RCCEx_EnableLSECSS()***
- ***HAL_RCCEx_DisableLSECSS()***
- ***HAL_RCCEx_EnableLSECSS_IT()***
- ***HAL_RCCEx_LSECSS_IRQHandler()***
- ***HAL_RCCEx_LSECSS_Callback()***
- ***HAL_RCCEx_EnableHSI48_VREFINT()***
- ***HAL_RCCEx_DisableHSI48_VREFINT()***

37.2.2 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_RCCEx_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_RCCEx_CRSWaitSynchronization()
   - According to CRS status, user can decide to adjust again the calibration or continue application if synchronization is OK

4. User can retrieve information related to synchronization in calling function @ref HAL_RCCEx_CRSGetSynchronizationInfo()
5. Regarding synchronization status and synchronization information, user can try a new calibration in changing synchronization configuration and call again HAL_RCCEx_CRSConfig. Note: When the SYNC event is detected during the downcounting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value should be incremented), while when it is detected during the upcounting phase it means that the actual frequency is higher (and that the TRIM value should be decremented).
6. In interrupt mode, user can resort to the available macros (__HAL_RCC_CRS_XXX_IT). Interrupts will go through CRS Handler (RCC_IRQn/RCC_IRQHandler)
   - Call function @ref HAL_RCCEx_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_RCCEx_CRS_IRQHandler():
     - @ref HAL_RCCEx_CRS_SyncOkCallback()
     - @ref HAL_RCCEx_CRS_SyncWarnCallback()
     - @ref HAL_RCCEx_CRS_ExpectedSyncCallback()
     - @ref HAL_RCCEx_CRS_ErrorCallback()
7. To force a SYNC EVENT, user can use the function @ref HAL_RCCEx_CRSSoftwareSynchronizationGenerate(). This function can be called before calling @ref HAL_RCCEx_CRSConfig (for instance in Systick handler)

This section contains the following APIs:

- ***HAL_RCCEx_CRSConfig()***
- ***HAL_RCCEx_CRSSoftwareSynchronizationGenerate()***
- ***HAL_RCCEx_CRSGetSynchronizationInfo()***
- ***HAL_RCCEx_CRSWaitSynchronization()***
- ***HAL_RCCEx_CRS_IRQHandler()***
- ***HAL_RCCEx_CRS_SyncOkCallback()***
- ***HAL_RCCEx_CRS_SyncWarnCallback()***
- ***HAL_RCCEx_CRS_ExpectedSyncCallback()***
- ***HAL_RCCEx_CRS_ErrorCallback()***

37.2.3 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name	**HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function description	Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.
Parameters	• **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1,

I2C1, I2C3, RTC, USB/RNG and LPTIM1 clocks).

Return values	• **HAL:** status
Notes	• If HAL_ERROR returned, first switch-OFF HSE clock oscillator with HAL_RCC_OscConfig() to possibly update HSE divider.

HAL_RCCEx_GetPeriphCLKConfig

Function name	**void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function description	Get the PeriphClkInit according to the internal RCC configuration registers.
Parameters	• **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1, I2C1, I2C3, RTC, USB/RNG and LPTIM1 clocks).
Return values	• **None**

HAL_RCCEx_GetPeriphCLKFreq

Function name	**uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)**
Function description	Return the peripheral clock frequency.
Parameters	• **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values: – RCC_PERIPHCLK_RTC RTC peripheral clock – RCC_PERIPHCLK_LCD LCD peripheral clock (*) – RCC_PERIPHCLK_USB USB or RNG peripheral clock (*) – RCC_PERIPHCLK_USART1 USART1 peripheral clock (*) – RCC_PERIPHCLK_USART2 USART2 peripheral clock – RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock – RCC_PERIPHCLK_I2C1 I2C1 peripheral clock – RCC_PERIPHCLK_I2C2 I2C2 peripheral clock (*) – RCC_PERIPHCLK_I2C3 I2C3 peripheral clock (*)
Return values	• **Frequency:** in Hz (0: means that no available frequency for the peripheral)
Notes	• Return 0 if peripheral clock is unknown • (*) means that this peripheral is not present on all the devices

HAL_RCCEx_EnableLSECSS

Function name	**void HAL_RCCEx_EnableLSECSS (void)**
Function description	Enables the LSE Clock Security System.
Return values	• **None**

HAL_RCCEx_DisableLSECSS

Function name	**void HAL_RCCEx_DisableLSECSS (void)**
Function description	Disables the LSE Clock Security System.
Return values	• **None**
Notes	• Once enabled this bit cannot be disabled, except after an LSE failure detection (LSECSSD=1). In that case the software MUST disable the LSECSSON bit. Reset by power on reset and RTC software reset (RTCRST bit).

HAL_RCCEx_EnableLSECSS_IT

Function name	**void HAL_RCCEx_EnableLSECSS_IT (void)**
Function description	Enable the LSE Clock Security System IT & corresponding EXTI line.
Return values	• **None**
Notes	• LSE Clock Security System IT is mapped on RTC EXTI line 19

HAL_RCCEx_LSECSS_IRQHandler

Function name	**void HAL_RCCEx_LSECSS_IRQHandler (void)**
Function description	Handle the RCC LSE Clock Security System interrupt request.
Return values	• **None**

HAL_RCCEx_LSECSS_Callback

Function name	**void HAL_RCCEx_LSECSS_Callback (void)**
Function description	RCCEx LSE Clock Security System interrupt callback.
Return values	• **None**

HAL_RCCEx_EnableHSI48_VREFINT

Function name	**void HAL_RCCEx_EnableHSI48_VREFINT (void)**
Function description	Enables Vrefint for the HSI48.
Return values	• **None**
Notes	• This is functional only if the LOCK is not set

HAL_RCCEx_DisableHSI48_VREFINT

Function name	**void HAL_RCCEx_DisableHSI48_VREFINT (void)**
Function description	Disables the Vrefint for the HSI48.
Return values	• **None**
Notes	• This is functional only if the LOCK is not set

HAL_RCCEx_CRSConfig

Function name	**void HAL_RCCEx_CRSConfig (RCC_CRSInitTypeDef * pInit)**
Function description	Start automatic synchronization for polling mode.
Parameters	• **pInit:** Pointer on RCC_CRSInitTypeDef structure
Return values	• **None**

HAL_RCCEx_CRSSoftwareSynchronizationGenerate

Function name	**void HAL_RCCEx_CRSSoftwareSynchronizationGenerate (void)**
Function description	Generate the software synchronization event.
Return values	• **None**

HAL_RCCEx_CRSGetSynchronizationInfo

Function name	**void HAL_RCCEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)**
Function description	Return synchronization info.
Parameters	• **pSynchroInfo:** Pointer on RCC_CRSSynchroInfoTypeDef structure
Return values	• **None**

HAL_RCCEx_CRSWaitSynchronization

Function name	**uint32_t HAL_RCCEx_CRSWaitSynchronization (uint32_t Timeout)**
Function description	Wait for CRS Synchronization status.
Parameters	• **Timeout:** Duration of the timeout
Return values	• **Combination:** of Synchronization status This parameter can be a combination of the following values: – RCC_CRS_TIMEOUT – RCC_CRS_SYNCOK – RCC_CRS_SYNCWARN – RCC_CRS_SYNCERR – RCC_CRS_SYNCMISS – RCC_CRS_TRIMOVF
Notes	• Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency. • If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

HAL_RCCEx_CRS_IRQHandler

Function name	**void HAL_RCCEx_CRS_IRQHandler (void)**
Function description	Handle the Clock Recovery System interrupt request.

Return values	• **None**

HAL_RCCEx_CRS_SyncOkCallback

Function name	**void HAL_RCCEx_CRS_SyncOkCallback (void)**
Function description	RCCEx Clock Recovery System SYNCOK interrupt callback.
Return values	• **None**

HAL_RCCEx_CRS_SyncWarnCallback

Function name	**void HAL_RCCEx_CRS_SyncWarnCallback (void)**
Function description	RCCEx Clock Recovery System SYNCWARN interrupt callback.
Return values	• **None**

HAL_RCCEx_CRS_ExpectedSyncCallback

Function name	**void HAL_RCCEx_CRS_ExpectedSyncCallback (void)**
Function description	RCCEx Clock Recovery System Expected SYNC interrupt callback.
Return values	• **None**

HAL_RCCEx_CRS_ErrorCallback

Function name	**void HAL_RCCEx_CRS_ErrorCallback (uint32_t Error)**
Function description	RCCEx Clock Recovery System Error interrupt callback.
Parameters	• **Error:** Combination of Error status. This parameter can be a combination of the following values: – RCC_CRS_SYNCERR – RCC_CRS_SYNCMISS – RCC_CRS_TRIMOVF
Return values	• **None**

37.3 RCCEx Firmware driver defines

37.3.1 RCCEx

AHB Peripheral Clock Sleep Enable Disable

__HAL_RCC_TSC_CLK_SLEEP_ENABLE

__HAL_RCC_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_TSC_CLK_SLEEP_DISABLE

__HAL_RCC_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_TSC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TSC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_DISABLED

AHB Peripheral Force Release Reset

__HAL_RCC_TSC_FORCE_RESET

__HAL_RCC_TSC_RELEASE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_RNG_RELEASE_RESET

APB1 Peripheral Clock Enable Disable

__HAL_RCC_USB_CLK_ENABLE

__HAL_RCC_USB_CLK_DISABLE

__HAL_RCC_USB_IS_CLK_ENABLED

__HAL_RCC_USB_IS_CLK_DISABLED

__HAL_RCC_CRS_CLK_ENABLE

__HAL_RCC_CRS_CLK_DISABLE

__HAL_RCC_CRS_IS_CLK_ENABLED

__HAL_RCC_CRS_IS_CLK_DISABLED

__HAL_RCC_LCD_CLK_ENABLE

__HAL_RCC_LCD_CLK_DISABLE

__HAL_RCC_LCD_IS_CLK_ENABLED

__HAL_RCC_LCD_IS_CLK_DISABLED

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART4_CLK_ENABLE

__HAL_RCC_USART5_CLK_ENABLE

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_I2C3_CLK_ENABLE

__HAL_RCC_DAC_CLK_ENABLE

__HAL_RCC_LPTIM1_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART4_CLK_DISABLE

__HAL_RCC_USART5_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_I2C3_CLK_DISABLE

__HAL_RCC_DAC_CLK_DISABLE

__HAL_RCC_LPTIM1_CLK_DISABLE

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART4_IS_CLK_ENABLED

__HAL_RCC_USART5_IS_CLK_ENABLED

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_I2C3_IS_CLK_ENABLED

__HAL_RCC_DAC_IS_CLK_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_USART4_IS_CLK_DISABLED

__HAL_RCC_USART5_IS_CLK_DISABLED

__HAL_RCC_LPUART1_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_I2C3_IS_CLK_DISABLED

__HAL_RCC_DAC_IS_CLK_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_DISABLED

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART4_CLK_SLEEP_ENABLE

__HAL_RCC_USART5_CLK_SLEEP_ENABLE

__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_DAC_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART4_CLK_SLEEP_DISABLE

__HAL_RCC_USART5_CLK_SLEEP_DISABLE

__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_DAC_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DAC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DAC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USB_CLK_SLEEP_ENABLE

__HAL_RCC_USB_CLK_SLEEP_DISABLE

__HAL_RCC_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_USB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LCD_CLK_SLEEP_ENABLE

__HAL_RCC_LCD_CLK_SLEEP_DISABLE

__HAL_RCC_LCD_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LCD_IS_CLK_SLEEP_DISABLED

APB1 Peripheral Force Release Reset

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_LPTIM1_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_I2C3_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART4_FORCE_RESET

__HAL_RCC_USART5_FORCE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_DAC_FORCE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_LPTIM1_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_I2C3_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART4_RELEASE_RESET

__HAL_RCC_USART5_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_DAC_RELEASE_RESET

__HAL_RCC_USB_FORCE_RESET

__HAL_RCC_USB_RELEASE_RESET

__HAL_RCC_CRS_FORCE_RESET

__HAL_RCC_CRS_RELEASE_RESET

__HAL_RCC_LCD_FORCE_RESET

__HAL_RCC_LCD_RELEASE_RESET

APB2 Peripheral Clock Enable Disable

__HAL_RCC_TIM21_CLK_ENABLE

__HAL_RCC_TIM22_CLK_ENABLE

__HAL_RCC_ADC1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_TIM21_CLK_DISABLE

__HAL_RCC_TIM22_CLK_DISABLE

__HAL_RCC_ADC1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_FIREWALL_CLK_ENABLE

__HAL_RCC_FIREWALL_CLK_DISABLE

__HAL_RCC_TIM21_IS_CLK_ENABLED

__HAL_RCC_TIM22_IS_CLK_ENABLED

__HAL_RCC_ADC1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_TIM21_IS_CLK_DISABLED

__HAL_RCC_TIM22_IS_CLK_DISABLED

__HAL_RCC_ADC1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_FIREWALL_IS_CLK_ENABLED

__HAL_RCC_FIREWALL_IS_CLK_DISABLED

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM21_CLK_SLEEP_ENABLE

__HAL_RCC_TIM22_CLK_SLEEP_ENABLE

__HAL_RCC_ADC1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM21_CLK_SLEEP_DISABLE

__HAL_RCC_TIM22_CLK_SLEEP_DISABLE

__HAL_RCC_ADC1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM21_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM22_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ADC1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM21_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM22_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ADC1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_DISABLED

APB2 Peripheral Force Release Reset

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_ADC1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_TIM21_FORCE_RESET

__HAL_RCC_TIM22_FORCE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_ADC1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_TIM21_RELEASE_RESET

__HAL_RCC_TIM22_RELEASE_RESET

RCCEx CRS Default Error Limit Value

RCC_CRS_ERRORLIMIT_DEFAULT	Default Frequency error limit

RCCEx CRS Flags

RCC_CRS_FLAG_SYNCOK	SYNC event OK flag
RCC_CRS_FLAG_SYNCWARN	SYNC warning flag
RCC_CRS_FLAG_ERR	Error flag
RCC_CRS_FLAG_ESYNC	Expected SYNC flag
RCC_CRS_FLAG_SYNCERR	SYNC error
RCC_CRS_FLAG_SYNCMISS	SYNC missed
RCC_CRS_FLAG_TRIMOVF	Trimming overflow or underflow

RCCEx CRS Frequency Error Direction

RCC_CRS_FREQERRORDIR_UP	Upcounting direction, the actual frequency is above the target
RCC_CRS_FREQERRORDIR_DOWN	Downcounting direction, the actual frequency is below the target

RCCEx CRS Default HSI48 Calibration vakye

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

RCCEx CRS Interrupt Sources

RCC_CRS_IT_SYNCOK	SYNC event OK
RCC_CRS_IT_SYNCWARN	SYNC warning
RCC_CRS_IT_ERR	Error
RCC_CRS_IT_ESYNC	Expected SYNC
RCC_CRS_IT_SYNCERR	SYNC error
RCC_CRS_IT_SYNCMISS	SYNC missed
RCC_CRS_IT_TRIMOVF	Trimming overflow or underflow

RCCEx CRS 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).

RCCEx CRS Status

RCC_CRS_NONE

RCC_CRS_TIMEOUT

RCC_CRS_SYNCOK

RCC_CRS_SYNCWARN

RCC_CRS_SYNCERR

RCC_CRS_SYNCMISS

RCC_CRS_TRIMOVF

RCCEx CRS 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

RCCEx CRS Synchronization Polarity

RCC_CRS_SYNC_POLARITY_RISING	Synchro Active on rising edge (default)
RCC_CRS_SYNC_POLARITY_FALLING	Synchro Active on falling edge

RCCEx CRS Synchronization Source

RCC_CRS_SYNC_SOURCE_GPIO	Synchro Signal source GPIO

RCC_CRS_SYNC_SOURCE_LSE	Synchro Signal source LSE
RCC_CRS_SYNC_SOURCE_USB	Synchro Signal source USB SOF (default)

RCCEx Exported Macros

__HAL_RCC_LSECSS_EXTI_ENABLE_IT	**Description:** • Enable interrupt on RCC LSE CSS EXTI Line 19. **Return value:** • None
__HAL_RCC_LSECSS_EXTI_DISABLE_IT	**Description:** • Disable interrupt on RCC LSE CSS EXTI Line 19. **Return value:** • None
__HAL_RCC_LSECSS_EXTI_ENABLE_EVENT	**Description:** • Enable event on RCC LSE CSS EXTI Line 19. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_DISABLE_EVENT	**Description:** • Disable event on RCC LSE CSS EXTI Line 19. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_ENABLE_FALLING_EDGE	**Description:** • RCC LSE CSS EXTI line configuration: set falling edge trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable the RCC LSE CSS Extended Interrupt Falling Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_EDGE	**Description:** • RCC LSE CSS EXTI line configuration: set rising edge trigger. **Return value:**

	• None.
__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_EDGE	**Description:** • Disable the RCC LSE CSS Extended Interrupt Rising Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • RCC LSE CSS EXTI line configuration: set rising & falling edge trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_GET_FLAG	**Description:** • Check whether the specified RCC LSE CSS EXTI interrupt flag is set or not. **Return value:** • EXTI: RCC LSE CSS Line Status.
__HAL_RCC_LSECSS_EXTI_CLEAR_FLAG	**Description:** • Clear the RCC LSE CSS EXTI flag. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None.
__HAL_RCC_I2C1_CONFIG	**Description:** • Macro to configure the I2C1 clock (I2C1CLK). **Parameters:** • __I2C1_CLKSOURCE__: specifies the I2C1 clock source. This

parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_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_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock

__HAL_RCC_I2C3_CONFIG

Description:

- Macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3_CLKSOURCE__: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_SYSCLK System Clock selected as I2C3 clock

__HAL_RCC_GET_I2C3_SOURCE

Description:

- Macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3

clock
- RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock
- RCC_I2C3CLKSOURCE_SYSCLK System Clock selected as I2C3 clock

__HAL_RCC_USART1_CONFIG

Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1_CLKSOURCE__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 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_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2_CLKSOURCE__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

__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_LPUART1_CONFIG

Description:

- Macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- __LPUART1_CLKSOURCE__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as

LPUART1 clock
  - RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_GET_LPUART1_SOURCE

Description:

- Macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

Parameters:

- __LPTIM1_CLKSOURCE__: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK PCLK selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI HSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_HSI LSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPTIM1 clock

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- Macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK PCLK selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_LSI HSI selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_HSI System Clock selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock (USBCLK).

Parameters:

- __USB_CLKSOURCE__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_USBCLKSOURCE_HSI48 HSI48 selected as USB clock
  - RCC_USBCLKSOURCE_PLL PLL Clock selected as USB clock

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBCLKSOURCE_HSI48 HSI48 selected as USB clock
  - RCC_USBCLKSOURCE_PLL PLL Clock selected as USB clock

__HAL_RCC_RNG_CONFIG

Description:

- Macro to configure the RNG clock (RNGCLK).

Parameters:

- __RNG_CLKSOURCE__: specifies the USB clock source. This parameter can be one of the

following values:
– RCC_RNGCLKSOURCE_HSI48 HSI48 selected as RNG clock
– RCC_RNGCLKSOURCE_PLLCLK PLL Clock selected as RNG clock

__HAL_RCC_GET_RNG_SOURCE

Description:

- Macro to get the RNG clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_RNGCLKSOURCE_HSI48 HSI48 selected as RNG clock
  – RCC_RNGCLKSOURCE_PLLCLK PLL Clock selected as RNG clock

__HAL_RCC_HSI48M_CONFIG

Description:

- Macro to select the HSI48M clock source.

Parameters:

- __HSI48M_CLKSOURCE__: specifies the HSI48M clock source dedicated for USB an RNG peripherals. This parameter can be one of the following values:
  – RCC_HSI48M_PLL A dedicated 48MHZ PLL output.
  – RCC_HSI48M_HSI48 48MHZ issued from internal HSI48 oscillator.

Notes:

- This macro can be replaced by either __HAL_RCC_RNG_CONFIG or __HAL_RCC_USB_CONFIG to configure respectively RNG or UBS clock sources.

__HAL_RCC_GET_HSI48M_SOURCE

Description:

- Macro to get the HSI48M clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_HSI48M_PLL A

dedicated 48MHZ PLL output.
 – RCC_HSI48M_HSI48 48MHZ issued from internal HSI48 oscillator.

Notes:

- This macro can be replaced by either __HAL_RCC_GET_RNG_SOURCE or __HAL_RCC_GET_USB_SOURCE to get respectively RNG or UBS clock sources.

__HAL_RCC_HSISTOP_ENABLE

Notes:

- The Enable of this function has not effect on the HSION bit.

__HAL_RCC_HSISTOP_DISABLE

Description:

- Macro to disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for USART and I2C.

Return value:

- None

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configures 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

__HAL_RCC_WAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configures the wake up from stop clock.

Parameters:

- __RCC_STOPWUCLK__: specifies the clock source used after wake up from stop This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_MSI MSI selected as system clock source
  - RCC_STOP_WAKEUPCLOCK_HSI HSI selected as system clock source

Return value:

- None

__HAL_RCC_CRS_ENABLE_IT

Description:

- Enables 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
  - RCC_CRS_IT_SYNCWARN
  - RCC_CRS_IT_ERR
  - RCC_CRS_IT_ESYNC

Return value:

- None

__HAL_RCC_CRS_DISABLE_IT

Description:

- Disables 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
  - RCC_CRS_IT_SYNCWARN
  - RCC_CRS_IT_ERR
  - RCC_CRS_IT_ESYNC

Return value:

- None

__HAL_RCC_CRS_GET_IT_SOURCE

Description:

- Check 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
  - RCC_CRS_IT_SYNCWARN
  - RCC_CRS_IT_ERR
  - RCC_CRS_IT_ESYNC

Return value:

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

__HAL_RCC_CRS_CLEAR_IT

Description:

- Clear the CRS interrupt pending bits bits to clear the selected interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK
  - RCC_CRS_IT_SYNCWARN
  - RCC_CRS_IT_ERR
  - RCC_CRS_IT_ESYNC
  - RCC_CRS_IT_TRIMOVF
  - RCC_CRS_IT_SYNCERR
  - RCC_CRS_IT_SYNCMISS

__HAL_RCC_CRS_GET_FLAG

Description:

- Checks 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
  - RCC_CRS_FLAG_SYNCWA RN
  - RCC_CRS_FLAG_ERR

- RCC_CRS_FLAG_ESYNC
- RCC_CRS_FLAG_TRIMOVF
- RCC_CRS_FLAG_SYNCERR
- RCC_CRS_FLAG_SYNCMISS

Return value:

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

__HAL_RCC_CRS_CLEAR_FLAG

Description:

- Clears the CRS specified FLAG.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK
  - RCC_CRS_FLAG_SYNCWARN
  - RCC_CRS_FLAG_ERR
  - RCC_CRS_FLAG_ESYNC
  - RCC_CRS_FLAG_TRIMOVF
  - RCC_CRS_FLAG_SYNCERR
  - RCC_CRS_FLAG_SYNCMISS

Return value:

- None

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_ENABLE

Description:

- Enables the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_DISABLE

Description:

- Disables the oscillator clock for frequency error counter.

Return value:

- None

__HAL_RCC_CRS_AUTOMATIC_CALIB_ENABLE

Description:

- Enables the automatic hardware

adjustment of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_AUTOMATIC_CALIB_DISABLE

Description:

- Enables or disables the automatic hardware adjustment of TRIM bits.

Return value:

- None

__HAL_RCC_CRS_RELOADVALUE_CALCULATE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

__HAL_RCC_HSI_OUT_ENABLE

Notes:

- After reset, the HSI output is not available

__HAL_RCC_HSI_OUT_DISABLE

Notes:

- After reset, the HSI output is not available

__HAL_RCC_HSI48_ENABLE	**Notes:** • After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable and can be used to clock the USB. The HSI48 is stopped by hardware when entering STOP and STANDBY modes.
__HAL_RCC_HSI48_DISABLE	
__HAL_RCC_GET_HSI48_STATE	**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
__HAL_RCC_HSI48M_DIV6_OUT_ENABLE	**Notes:** • After reset, the HSI48Mhz (divided by 6) output is not available
__HAL_RCC_HSI48M_DIV6_OUT_DISABLE	

RCC LSE CSS external interrupt line

RCC_EXTI_LINE_LSECSS	External interrupt line 19 connected to the LSE CSS EXTI Line

RCCEx HSI48M Clock Source

RCC_FLAG_HSI48

RCC_HSI48M_PLL

RCC_HSI48M_HSI48

RCCEx I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1

RCC_I2C1CLKSOURCE_SYSCLK

RCC_I2C1CLKSOURCE_HSI

RCCEx I2C3 Clock Source

RCC_I2C3CLKSOURCE_PCLK1

RCC_I2C3CLKSOURCE_SYSCLK

RCC_I2C3CLKSOURCE_HSI

IOPORT Peripheral Clock Enable Disable

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

IOPORT Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOE_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_DISABLED

IOPORT Peripheral Force Release Reset

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

LCD Configuration

__HAL_RCC_LCD_CONFIG

Description:

- Macro to configures LCD clock (LCDCLK).

Parameters:

- __LCD_CLKSOURCE__: specifies the LCD clock source. This parameter can be one of the following values:
  - RCC_RTCCLKSOURCE_LSE LSE selected as LCD clock
  - RCC_RTCCLKSOURCE_LSI LSI selected as LCD clock
  - RCC_RTCCLKSOURCE_HSE_DIV2 HSE divided by 2 selected as LCD clock
  - RCC_RTCCLKSOURCE_HSE_DIV4 HSE divided by 4 selected as LCD clock
  - RCC_RTCCLKSOURCE_HSE_DIV8

HSE divided by 8 selected as LCD clock
- RCC_RTCCLKSOURCE_HSE_DIV16 HSE divided by 16 selected as LCD clock

Notes:

- LCD and RTC use the same configuration LCD can however be used in the Stop low power mode if the LSE or LSI is used as the LCD clock source.

__HAL_RCC_GET_LCD_SOURCE

__HAL_RCC_GET_LCD_HSE_PRESCALER

RCCEx LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_PCLK

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_HSI

RCC_LPTIM1CLKSOURCE_LSE

RCCEx LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_PCLK1

RCC_LPUART1CLKSOURCE_SYSCLK

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_LSE

RCCEx LSE Drive Configuration

RCC_LSEDRIVE_LOW

RCC_LSEDRIVE_MEDIUMLOW

RCC_LSEDRIVE_MEDIUMHIGH

RCC_LSEDRIVE_HIGH

RCC Extended MCOx Clock Config

__HAL_RCC_MCO1_CONFIG

Description:

- Macro to configure the MCO clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK 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_MSI MSI oscillator clock selected as MCO clock

  - RCC_MCO1SOURCE_HSE HSE oscillator clock selected as MCO clock
  - RCC_MCO1SOURCE_PLLCLK PLL clock selected as MCO clock
  - RCC_MCO1SOURCE_LSI LSI clock selected as MCO clock
  - RCC_MCO1SOURCE_LSE LSE clock selected as MCO clock
  - RCC_MCO1SOURCE_HSI48 HSI48 clock 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

AHB Peripheral Clock Enable Disable

__HAL_RCC_TSC_CLK_ENABLE

__HAL_RCC_TSC_CLK_DISABLE

__HAL_RCC_TSC_IS_CLK_ENABLED

__HAL_RCC_TSC_IS_CLK_DISABLED

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_RNG_CLK_DISABLE

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_DISABLED

RCCEx Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_USB

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_LCD

RCC_PERIPHCLK_I2C3

RCCEx RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_PLLCLK

RCCEx StopWakeUp Clock

RCC_STOP_WAKEUPCLOCK_MSI

RCC_STOP_WAKEUPCLOCK_HSI

RCCEx TIM Prescaler Selection

RCC_TIMPRES_DESACTIVATED

RCC_TIMPRES_ACTIVATED

RCCEx USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_SYSCLK

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_LSE

RCCEx USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_SYSCLK

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_LSE

RCCEx USB Clock Source

RCC_USBCLKSOURCE_HSI48

RCC_USBCLKSOURCE_PLL

38 HAL RNG Generic Driver

38.1 RNG Firmware driver registers structures

38.1.1 RNG_HandleTypeDef

Data Fields

- ***RNG_TypeDef * Instance***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***uint32_t RandomNumber***

Field Documentation

- ***RNG_TypeDef* RNG_HandleTypeDef::Instance***
  Register base address
- ***HAL_LockTypeDef RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef RNG_HandleTypeDef::State***
  RNG communication state
- ***uint32_t RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data

38.2 RNG Firmware driver API description

38.2.1 How to use this driver

The RNG HAL driver can be used as follows:

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

38.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

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

38.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- *HAL_RNG_GenerateRandomNumber()*
- *HAL_RNG_GenerateRandomNumber_IT()*
- *HAL_RNG_IRQHandler()*
- *HAL_RNG_GetRandomNumber()*
- *HAL_RNG_GetRandomNumber_IT()*
- *HAL_RNG_ReadLastRandomNumber()*
- *HAL_RNG_ReadyDataCallback()*
- *HAL_RNG_ErrorCallback()*

38.2.4 Peripheral State functions

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

This section contains the following APIs:

- *HAL_RNG_GetState()*

38.2.5 Detailed description of functions

HAL_RNG_Init

Function name	**HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)**
Function description	Initializes the RNG peripheral and creates the associated handle.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** status

HAL_RNG_DeInit

Function name	**HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)**
Function description	DeInitializes the RNG peripheral.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** status

HAL_RNG_MspInit

Function name	**void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)**
Function description	Initializes the RNG MSP.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None**

HAL_RNG_MspDeInit

Function name	**void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)**
Function description	DeInitializes the RNG MSP.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None**

HAL_RNG_GetRandomNumber

Function name	**uint32_t HAL_RNG_GetRandomNumber (RNG_HandleTypeDef * hrng)**
Function description	return generated random number in polling mode (Obsolete).
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	• **random:** value

HAL_RNG_GetRandomNumber_IT

Function name	**uint32_t HAL_RNG_GetRandomNumber_IT (RNG_HandleTypeDef * hrng)**
Function description	Returns a 32-bit random number with interrupt enabled (Obsolete), Use HAL_RNG_GenerateRandomNumber_IT() API instead.
Parameters	• **hrng:** RNG handle
Return values	• **32-bit:** random number

HAL_RNG_GenerateRandomNumber

Function name	**HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)**
Function description	Generates a 32-bit random number.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure. • **random32bit:** pointer to generated random number variable if successful.
Return values	• **HAL:** status
Notes	• Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_GenerateRandomNumber_IT

Function name	**HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)**
Function description	Generates a 32-bit random number in interrupt mode.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** status

HAL_RNG_ReadLastRandomNumber

Function name	**uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)**
Function description	Read latest generated random number.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **random:** value

HAL_RNG_IRQHandler

Function name	**void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)**
Function description	Handles RNG interrupt request.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None**
Notes	• In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used. • In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using __HAL_RNG_CLEAR_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG. • User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

HAL_RNG_ErrorCallback

Function name	**void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)**
Function description	RNG error callbacks.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None**

HAL_RNG_ReadyDataCallback

Function name	**void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)**
Function description	Data Ready callback in non-blocking mode.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.. • **random32bit:** generated random value
Return values	• **None**

HAL_RNG_GetState

Function name	**HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)**
Function description	Returns the RNG state.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** state

38.3 RNG Firmware driver defines

38.3.1 RNG

RNG Interrupt definition

RNG_IT_DRDY	Data ready interrupt
RNG_IT_CEI	Clock error interrupt
RNG_IT_SEI	Seed error interrupt

RNG Flag definition

RNG_FLAG_DRDY	Data ready
RNG_FLAG_CECS	Clock error current status
RNG_FLAG_SECS	Seed error current status

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:

- Reset RNG handle state.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_ENABLE

Description:

- Enables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE

Description:

- Disables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_FLAG	**Description:** • Check the selected RNG flag status. **Parameters:** • __HANDLE__: RNG Handle • __FLAG__: RNG flag This parameter can be one of the following values: – RNG_FLAG_DRDY: Data ready – RNG_FLAG_CECS: Clock error current status – RNG_FLAG_SECS: Seed error current status **Return value:** • The: new state of __FLAG__ (SET or RESET).
__HAL_RNG_CLEAR_FLAG	**Description:** • Clears the selected RNG flag status. **Parameters:** • __HANDLE__: RNG handle • __FLAG__: RNG flag to clear **Return value:** • None **Notes:** • WARNING: This is a dummy macro for HAL code alignment, flags RNG_FLAG_DRDY, RNG_FLAG_CECS and RNG_FLAG_SECS are read-only.
__HAL_RNG_ENABLE_IT	**Description:** • Enables the RNG interrupts. **Parameters:** • __HANDLE__: RNG Handle **Return value:** • None
__HAL_RNG_DISABLE_IT	**Description:** • Disables the RNG interrupts. **Parameters:** • __HANDLE__: RNG Handle **Return value:** • None
__HAL_RNG_GET_IT	**Description:** • Checks whether the specified RNG

interrupt has occurred or not.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG_IT_DRDY: Data ready interrupt
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

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

__HAL_RNG_CLEAR_IT

Description:

- Clears the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

39 HAL RTC Generic Driver

39.1 RTC Firmware driver registers structures

39.1.1 RTC_InitTypeDef

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutRemap***
- ***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::OutPutRemap***
  Specifies the remap for RTC output. This parameter can be a value of ***RTC_Output_ALARM_OUT_Remap***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***
  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***

39.1.2 RTC_TimeTypeDef

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***

39.1.3 RTC_DateTypeDef

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

39.1.4 RTC_AlarmTypeDef

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***

39.1.5 RTC_HandleTypeDef

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

39.2 RTC Firmware driver API description

39.2.1 Backup Domain Operating Condition

As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low power modes or under reset).

39.2.2 Backup Domain Reset

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

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

- Software reset, triggered by setting the RTCRST bit in the RCC Control Status register (RCC_CSR).
- Power reset (BOR/POR/PDR).

39.2.3 Backup Domain Access

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

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

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

39.2.4 How to use RTC Driver

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

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

39.2.5 RTC and low power modes

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

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

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

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

39.2.6 Initialization and de-initialization functions

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

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

This section contains the following APIs:

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

39.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

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

39.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- ***HAL_RTC_SetAlarm()***
- ***HAL_RTC_SetAlarm_IT()***
- ***HAL_RTC_DeactivateAlarm()***
- ***HAL_RTC_GetAlarm()***

- *HAL_RTC_AlarmIRQHandler()*
- *HAL_RTC_AlarmAEventCallback()*
- *HAL_RTC_PollForAlarmAEvent()*

39.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- *HAL_RTC_WaitForSynchro()*

39.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- *HAL_RTC_GetState()*

39.2.11 Detailed description of functions

HAL_RTC_Init

Function name	**HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)**
Function description	Initialize the RTC peripheral.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTC_DeInit

Function name	**HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)**
Function description	DeInitialize the RTC peripheral.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• This function 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	• **hrtc:** RTC handle • **sDate:** Pointer to date structure • **Format:** specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTC_GetDate

Function name	**HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)**
Function description	Get RTC current date.
Parameters	• **hrtc:** RTC handle • **sDate:** Pointer to Date structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status
Notes	• You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_SetAlarm

Function name	**HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**
Function description	Set the specified RTC Alarm.
Parameters	• **hrtc:** RTC handle • **sAlarm:** Pointer to Alarm structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTC_SetAlarm_IT

Function name	**HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**
Function description	Set the specified RTC Alarm with Interrupt.
Parameters	• **hrtc:** RTC handle • **sAlarm:** Pointer to Alarm structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status
Notes	• The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()). • The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

HAL_RTC_DeactivateAlarm

Function name	**HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)**
Function description	Deactivate the specified RTC Alarm.
Parameters	• **hrtc:** RTC handle • **Alarm:** Specifies the Alarm. This parameter can be one of the following values: – RTC_ALARM_A: AlarmA – RTC_ALARM_B: AlarmB
Return values	• **HAL:** status

HAL_RTC_GetAlarm

Function name	**HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)**
Function description	Get the RTC Alarm value and masks.
Parameters	• **hrtc:** RTC handle • **sAlarm:** Pointer to Date structure • **Alarm:** Specifies the Alarm. This parameter can be one of the following values: – RTC_ALARM_A: AlarmA – RTC_ALARM_B: AlarmB • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format

Return values	• **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name	**void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)**
Function description	Handle Alarm interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTC_PollForAlarmAEvent

Function name	**HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle AlarmA Polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTC_AlarmAEventCallback

Function name	**void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Alarm A callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTC_WaitForSynchro

Function name	**HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)**
Function description	Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• The RTC Resynchronization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function. • To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState

Function name	**HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)**
Function description	Return the RTC handle state.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** state

RTC_EnterInitMode

Function name	**HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)**
Function description	Enter the RTC Initialization mode.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• The RTC Initialization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.

RTC_ByteToBcd2

Function name	**uint8_t RTC_ByteToBcd2 (uint8_t Value)**
Function description	Convert a 2 digit decimal to BCD format.
Parameters	• **Value:** Byte to be converted
Return values	• **Converted:** byte

RTC_Bcd2ToByte

Function name	**uint8_t RTC_Bcd2ToByte (uint8_t Value)**
Function description	Convert from 2 digit BCD to Binary.
Parameters	• **Value:** BCD value to be converted
Return values	• **Converted:** word

39.3 RTC Firmware driver defines

39.3.1 RTC

RTC AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC AlarmMask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
RTC_ALARMSUBSECONDMASK_SS14_1	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
RTC_ALARMSUBSECONDMASK_SS14_2	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_3	SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_4	SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_5	SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_6	SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_7	SS[14:7] are don't care in Alarm ` comparison. Only SS[6:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_8	SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_9	SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_10	SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_11	SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_12	SS[14:12] are don't care in Alarm comparison.Only SS[11:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_13	SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
RTC_ALARMSUBSECONDMASK_SS14	SS[14] is don't care in Alarm comparison.Only SS[13:0] are compared
RTC_ALARMSUBSECONDMASK_NONE	SS[14:0] are compared and must match to activate alarm.

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC DayLightSaving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- __HANDLE__: RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:

- Disable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:

- Enable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMA_ENABLE

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMA_DISABLE

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_ENABLE

Description:

- Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_DISABLE

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARM_ENABLE_IT

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA: Alarm A interrupt
  - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA: Alarm A interrupt
  - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

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

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA: Alarm A interrupt
  - RTC_IT_ALRB: Alarm B interrupt

Return value:

- 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
  - RTC_IT_ALRB: Alarm B 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_ALRBF
  - RTC_FLAG_ALRAWF
  - RTC_FLAG_ALRBWF

Return value:

- None

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarm's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC

Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_GET_FLAG

Description:

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_ALARM_EXTI_CLEAR_FLAG

Description:

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None.

RTC Flags Definitions

RTC_FLAG_RECALPF

RTC_FLAG_TAMP2F

RTC_FLAG_TAMP1F

RTC_FLAG_TSOVF

RTC_FLAG_TSF

RTC_FLAG_WUTF

RTC_FLAG_ALRBF

RTC_FLAG_ALRAF

RTC_FLAG_INITF

RTC_FLAG_RSF

RTC_FLAG_INITS

RTC_FLAG_SHPF

RTC_FLAG_WUTWF

RTC_FLAG_ALRBWF

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	Enable Timestamp Interrupt
RTC_IT_WUT	Enable Wakeup timer Interrupt
RTC_IT_ALRA	Enable Alarm A Interrupt
RTC_IT_ALRB	Enable Alarm B Interrupt
RTC_IT_TAMP	Enable all Tamper Interrupt
RTC_IT_TAMP1	Enable Tamper 1 Interrupt
RTC_IT_TAMP2	Enable Tamper 2 Interrupt

RTC Private macros to check input parameters

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_OPENDRAIN

RTC_OUTPUT_TYPE_PUSHPULL

RTC StoreOperation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

40 HAL RTC Extension Driver

40.1 RTCEx Firmware driver registers structures

40.1.1 RTC_TamperTypeDef

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Interrupt***
- ***uint32_t Trigger***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of
  RTCEx_Tamper_Pins_Definitions
- ***uint32_t RTC_TamperTypeDef::Interrupt***
  Specifies the Tamper Interrupt. This parameter can be a value of
  RTCEx_Tamper_Interrupt_Definitions
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of
  RTCEx_Tamper_Trigger_Definitions
- ***uint32_t RTC_TamperTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of
  RTCEx_Tamper_EraseBackUp_Definitions
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of
  RTCEx_Tamper_MaskFlag_Definitions
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the RTC Filter Tamper. This parameter can be a value of
  RTCEx_Tamper_Filter_Definitions
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of
  RTCEx_Tamper_Sampling_Frequencies_Definitions
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration . This parameter can be a value of
  RTCEx_Tamper_Pin_Precharge_Duration_Definitions
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp . This parameter can be a value of
  RTCEx_Tamper_Pull_UP_Definitions
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of
  RTCEx_Tamper_TimeStampOnTamperDetection_Definitions

40.2 RTCEx Firmware driver API description

40.2.1 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- *HAL_RTCEx_SetTimeStamp()*
- *HAL_RTCEx_SetTimeStamp_IT()*
- *HAL_RTCEx_DeactivateTimeStamp()*
- *HAL_RTCEx_GetTimeStamp()*
- *HAL_RTCEx_SetTamper()*
- *HAL_RTCEx_SetTamper_IT()*
- *HAL_RTCEx_DeactivateTamper()*
- *HAL_RTCEx_TamperTimeStampIRQHandler()*
- *HAL_RTCEx_TimeStampEventCallback()*
- *HAL_RTCEx_Tamper1EventCallback()*
- *HAL_RTCEx_Tamper2EventCallback()*
- *HAL_RTCEx_Tamper3EventCallback()*
- *HAL_RTCEx_PollForTimeStampEvent()*
- *HAL_RTCEx_PollForTamper1Event()*
- *HAL_RTCEx_PollForTamper2Event()*
- *HAL_RTCEx_PollForTamper3Event()*

40.2.2 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

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

40.2.3 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- *HAL_RTCEx_BKUPWrite()*
- *HAL_RTCEx_BKUPRead()*
- *HAL_RTCEx_SetSmoothCalib()*
- *HAL_RTCEx_SetSynchroShift()*
- *HAL_RTCEx_SetCalibrationOutPut()*
- *HAL_RTCEx_DeactivateCalibrationOutPut()*
- *HAL_RTCEx_SetRefClock()*
- *HAL_RTCEx_DeactivateRefClock()*
- *HAL_RTCEx_EnableBypassShadow()*
- *HAL_RTCEx_DisableBypassShadow()*

40.2.4 Extended features functions

This section provides functions allowing to:

- RTC Alram B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- *HAL_RTCEx_AlarmBEventCallback()*
- *HAL_RTCEx_PollForAlarmBEvent()*

40.2.5 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)**
Function description	Set TimeStamp.
Parameters	• **hrtc:** RTC handle • **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: – RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin. – RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. • **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values: – RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin on STM32L05x/6x/7x/8x and PA2 on STM32L03x/4x/2x/1x.
Return values	• **HAL:** status
Notes	• This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_SetTimeStamp_IT

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)**
Function description	Set TimeStamp with Interrupt.
Parameters	• **hrtc:** RTC handle • **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: – RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin. – RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. • **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: – RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin on STM32L05x/6x/7x/8x and PA2 on STM32L03x/4x/2x/1x.
Return values	• **HAL:** status
Notes	• This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_DeactivateTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)**
Function description	Deactivate TimeStamp.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_GetTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)**
Function description	Get the RTC TimeStamp value.
Parameters	• **hrtc:** RTC handle • **sTimeStamp:** Pointer to Time structure • **sTimeStampDate:** Pointer to Date structure • **Format:** specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTCEx_SetTamper

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function description	Set Tamper.
Parameters	• **hrtc:** RTC handle • **sTamper:** Pointer to Tamper Structure.
Return values	• **HAL:** status
Notes	• By calling this API we disable the tamper interrupt for all tampers.

HAL_RTCEx_SetTamper_IT

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function description	Set Tamper with interrupt.
Parameters	• **hrtc:** RTC handle • **sTamper:** Pointer to RTC Tamper.
Return values	• **HAL:** status
Notes	• By calling this API we force the tamper interrupt for all tampers.

HAL_RTCEx_DeactivateTamper

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)**
Function description	Deactivate Tamper.
Parameters	• **hrtc:** RTC handle • **Tamper:** Selected tamper pin. This parameter can be RTC_Tamper_1 and/or RTC_TAMPER_2 for STM32L05x/6x. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3 for STM32L01x/2x/3x/7x/8x.
Return values	• **HAL:** status

HAL_RTCEx_TamperTimeStampIRQHandler

Function name	**void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)**
Function description	Handle TimeStamp interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_Tamper1EventCallback

Function name	**void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)**

Function description	Tamper 1 callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_Tamper2EventCallback

Function name	**void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Tamper 2 callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_Tamper3EventCallback

Function name	**void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Tamper 3 callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_TimeStampEventCallback

Function name	**void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	TimeStamp callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_PollForTimeStampEvent

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle TimeStamp polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTCEx_PollForTamper1Event

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Tamper 1 Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration

Return values	• **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	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **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	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **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	• **hrtc:** RTC handle • **WakeUpCounter:** Wake up counter • **WakeUpClock:** Wake up clock
Return values	• **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)**
Function description	Set wake up timer with interrupt.
Parameters	• **hrtc:** RTC handle • **WakeUpCounter:** Wake up counter • **WakeUpClock:** Wake up clock
Return values	• **HAL:** status

HAL_RTCEx_DeactivateWakeUpTimer

Function name	**uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)**

Function description	Deactivate wake up timer counter.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name	**uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)**
Function description	Get wake up timer counter.
Parameters	• **hrtc:** RTC handle
Return values	• **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name	**void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)**
Function description	Handle Wake Up Timer interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_WakeUpTimerEventCallback

Function name	**void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Wake Up Timer callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Wake Up Timer Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTCEx_BKUPWrite

Function name	**void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)**
Function description	Write a data in a specified RTC Backup data register.
Parameters	• **hrtc:** RTC handle • **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to

19 to specify the register.

- **Data:** Data to be written in the specified RTC Backup data register.

Return values	• **None**

HAL_RTCEx_BKUPRead

Function name	**uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)**
Function description	Reads data from the specified RTC Backup data Register.
Parameters	• **hrtc:** RTC handle • **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.
Return values	• **Read:** value

HAL_RTCEx_SetSmoothCalib

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)**
Function description	Set the Smooth calibration parameters.
Parameters	• **hrtc:** RTC handle • **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be can be one of the following values : – RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s. – RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s. – RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s. • **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values: – RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses. – RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added. • **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
Return values	• **HAL:** status
Notes	• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue mut be equal to 0.

HAL_RTCEx_SetSynchroShift

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)**
Function description	Configure the Synchronization Shift Control Settings.
Parameters	• **hrtc:** RTC handle • **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values : – RTC_SHIFTADD1S_SET: Add one second to the clock calendar. – RTC_SHIFTADD1S_RESET: No effect. • **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	• **HAL:** status
Notes	• When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)**
Function description	Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	• **hrtc:** RTC handle • **CalibOutput:** : Select the Calibration output Selection . This parameter can be one of the following values: – RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz. – RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	• **HAL:** status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)**
Function description	Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_SetRefClock

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)**

Function description	Enable the RTC reference clock detection.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_DeactivateRefClock

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)**
Function description	Disable the RTC reference clock detection.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_EnableBypassShadow

Function name	**HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)**
Function description	Enable the Bypass Shadow feature.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_DisableBypassShadow

Function name	**HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)**
Function description	Disable the Bypass Shadow feature.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_AlarmBEventCallback

Function name	**void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Alarm B callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None**

HAL_RTCEx_PollForAlarmBEvent

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Alarm B Polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

40.3 RTCEx Firmware driver defines

40.3.1 RTCEx

RTCEx Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTCEx 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_FLAG

Description:

- Get the selected RTC shift operation's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - RTC_FLAG_SHPF

Return value:

- None

RTCEx Calib Output selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTCEx Flags Definitions

RTC_FLAG_TAMP3F

RTCEx Interrupts Definitions

RTC_IT_TAMP3

Private macros to check input parameters

IS_RTC_OUTPUT

IS_RTC_BKP

IS_TIMESTAMP_EDGE

IS_RTC_TAMPER

IS_RTC_TAMPER_INTERRUPT

IS_RTC_TIMESTAMP_PIN

IS_RTC_TAMPER_TRIGGER

IS_RTC_TAMPER_ERASE_MODE

IS_RTC_TAMPER_MASKFLAG_STATE

IS_RTC_TAMPER_FILTER

IS_RTC_TAMPER_SAMPLING_FREQ

IS_RTC_TAMPER_PRECHARGE_DURATION

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

IS_RTC_TAMPER_PULLUP_STATE

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SHIFT_ADD1S

IS_RTC_CALIB_OUTPUT

RTCEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

RTCEx Smooth calib Minus pulses Definitions

IS_RTC_SMOOTH_CALIB_MINUS

RTCEx Smooth calib period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC	If RTCCLK = 32768 Hz, Smooth calibation period is 32s, else 2exp20 RTCCLK pulses
RTC_SMOOTHCALIB_PERIOD_16SEC	If RTCCLK = 32768 Hz, Smooth calibation period is 16s, else 2exp19 RTCCLK pulses
RTC_SMOOTHCALIB_PERIOD_8SEC	If RTCCLK = 32768 Hz, Smooth calibation period is 8s, else 2exp18 RTCCLK pulses

RTCEx Smooth calib Plus pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET	The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8
RTC_SMOOTHCALIB_PLUSPULSES_RESET	The number of RTCCLK pulses subbstited during a 32-second window =

CALM[8:0]

RTCEx Substract Fraction Of Second Value

IS_RTC_SHIFT_SUBFS

RTC Tamper

__HAL_RTC_TAMPER1_ENABLE

Description:

- Enable the RTC Tamper1 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER1_DISABLE

Description:

- Disable the RTC Tamper1 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_ENABLE

Description:

- Enable the RTC Tamper2 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_DISABLE

Description:

- Disable the RTC Tamper2 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_ENABLE

Description:

- Enable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_DISABLE

Description:

- Disable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- 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: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:

- 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: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt

- RTC_IT_TAMP2: Tamper2 interrupt
- RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT

Description:

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

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC_IT_TAMP: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_FLAG

Description:

- 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: Tamper1 flag
  - RTC_FLAG_TAMP2F: Tamper2 flag
  - RTC_FLAG_TAMP3F: Tamper3 flag

Return value:

- None

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC_FLAG_TAMP1F: Tamper1 flag
  - RTC_FLAG_TAMP2F: Tamper2 flag
  - RTC_FLAG_TAMP3F: Tamper3 flag

Return value:

- None

RTCEx Tamper EraseBackUp Definitions

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTCEx Tamper Filter Definitions

RTC_TAMPERFILTER_DISABLE	Tamper filter is disabled
RTC_TAMPERFILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at the active level
RTC_TAMPERFILTER_4SAMPLE	Tamper is activated after 4 consecutive samples at the active level
RTC_TAMPERFILTER_8SAMPLE	Tamper is activated after 8 consecutive samples at the active leve.

RTCEx Tamper Interrupt Definitions

RTC_TAMPER1_INTERRUPT

RTC_TAMPER2_INTERRUPT

RTC_TAMPER3_INTERRUPT

RTC_ALL_TAMPER_INTERRUPT

RTCEx Tamper MaskFlag Definitions

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTCEx Tamper Pins Definition

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTCEx Tamper Pin Precharge Duration Definitions

RTC_TAMPERPRECHARGEDURATION_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
RTC_TAMPERPRECHARGEDURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTCEx Tamper Pull UP Definitions

RTC_TAMPER_PULLUP_ENABLE	Tamper pins are pre-charged before sampling
RTC_TAMPER_PULLUP_DISABLE	Tamper pins pre-charge is disabled

RTCEx Tamper Sampling Frequencies Definitions

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024	Each of the tamper inputs are

	sampled with a frequency = RTCCLK / 1024
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

EXTI RTC Tamper Timestamp EXTI

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_IT	**Description:** • Enable interrupt on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT	**Description:** • Disable interrupt on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_EVENT	**Description:** • Enable event on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_EVENT	**Description:** • Disable event on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE	**Description:** • Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE	**Description:** • Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE	**Description:** • Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG	**Description:** • Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not. **Return value:** • Line: Status.

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG	**Description:** • Clear the RTC Tamper and Timestamp associated Exti line flag. **Return value:** • None.
__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line. **Return value:** • None.

RTCEx Tamper TimeStampOnTamperDetection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TIMESTAMPONTAMPERDETECTION_DISABLE	TimeStamp on Tamper Detection event is not saved

RTCEx Tamper Trigger Definitions

RTC_TAMPERTRIGGER_RISINGEDGE

RTC_TAMPERTRIGGER_FALLINGEDGE

RTC_TAMPERTRIGGER_LOWLEVEL

RTC_TAMPERTRIGGER_HIGHLEVEL

RTC Timestamp

__HAL_RTC_TIMESTAMP_ENABLE	**Description:** • Enable the RTC TimeStamp peripheral. **Parameters:** • __HANDLE__: specifies the RTC handle. **Return value:** • None
__HAL_RTC_TIMESTAMP_DISABLE	**Description:** • Disable the RTC TimeStamp peripheral. **Parameters:** • __HANDLE__: specifies the RTC handle. **Return value:** • None

__HAL_RTC_TIMESTAMP_ENABLE_IT	**Description:** • Enable the RTC TimeStamp interrupt. **Parameters:** • __HANDLE__: specifies the RTC handle. • __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be: – RTC_IT_TS: TimeStamp interrupt **Return value:** • None
__HAL_RTC_TIMESTAMP_DISABLE_IT	**Description:** • Disable the RTC TimeStamp interrupt. **Parameters:** • __HANDLE__: specifies the RTC handle. • __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be: – RTC_IT_TS: TimeStamp interrupt **Return value:** • None
__HAL_RTC_TIMESTAMP_GET_IT	**Description:** • Check whether the specified RTC TimeStamp interrupt has occurred or not. **Parameters:** • __HANDLE__: specifies the RTC handle. • __INTERRUPT__: specifies the RTC TimeStamp interrupt to check. This parameter can be: – RTC_IT_TS: TimeStamp interrupt **Return value:** • None
__HAL_RTC_TIMESTAMP_GET_IT_SOURCE	**Description:** • Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC 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_CLEAR_FLAG

Description:

- 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

RTCEx TimeStamp Pin Selection

RTC_TIMESTAMPPIN_DEFAULT

RTCEx Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC WakeUp Timer

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp

Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

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

Parameters:

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

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

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

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- 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_WAKEUPTIMER_CLEAR_FLAG	**Description:** • 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_WAKEUPTIMER_EXTI_ENABLE_IT	**Description:** • Enable interrupt on the RTC WakeUp Timer associated Exti line. **Return value:** • None
__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT	**Description:** • Disable interrupt on the RTC WakeUp Timer associated Exti line. **Return value:** • None
__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT	**Description:** • Enable event on the RTC WakeUp Timer associated Exti line. **Return value:** • None.
__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT	**Description:** • Disable event on the RTC WakeUp Timer associated Exti line. **Return value:** • None.
__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE	**Description:** • Enable falling edge trigger on the RTC WakeUp Timer associated Exti line. **Return value:**

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_GET_FLAG

Description:

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG

Description:

- Clear the RTC WakeUp Timer associated Exti line flag.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

RTCEx Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

41 HAL SMARTCARD Generic Driver

41.1 SMARTCARD Firmware driver registers structures

41.1.1 SMARTCARD_InitTypeDef

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)

41.1.2 SMARTCARD_AdvFeatureInitTypeDef

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***

41.1.3 SMARTCARD_HandleTypeDef

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

41.2 SMARTCARD Firmware driver API description

41.2.1 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.

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

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

41.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

The HAL_SMARTCARD_Init() API follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***

41.2.3 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.

(+) 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. (+) Blocking mode APIs are : (++) HAL_SMARTCARD_Transmit() (++) HAL_SMARTCARD_Receive() (+) Non Blocking mode APIs with Interrupt are : (++) HAL_SMARTCARD_Transmit_IT() (++) HAL_SMARTCARD_Receive_IT() (++) HAL_SMARTCARD_IRQHandler() (+) Non Blocking mode functions with DMA are : (++) HAL_SMARTCARD_Transmit_DMA() (++) HAL_SMARTCARD_Receive_DMA() (+) A set of Transfer Complete Callbacks are provided in non Blocking mode: (++) HAL_SMARTCARD_TxCpltCallback() (++) HAL_SMARTCARD_RxCpltCallback() (++) HAL_SMARTCARD_ErrorCallback()

1. 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()
2. 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()
3. 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 tranmission, 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.

- 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.
- Blocking mode APIs are :
  - HAL_SMARTCARD_Transmit()
  - HAL_SMARTCARD_Receive()
- Non Blocking mode APIs with Interrupt are :
  - HAL_SMARTCARD_Transmit_IT()
  - HAL_SMARTCARD_Receive_IT()
  - HAL_SMARTCARD_IRQHandler()
- Non Blocking mode functions with DMA are :

  - HAL_SMARTCARD_Transmit_DMA()
  - HAL_SMARTCARD_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SMARTCARD_TxCpltCallback()
  - HAL_SMARTCARD_RxCpltCallback()
  - HAL_SMARTCARD_ErrorCallback() (#) 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() (#) 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() (#) 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 tranmission, 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()***

41.2.4 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()***

41.2.5 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_DeInit

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	DeInitialize the SMARTCARD peripheral.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status

HAL_SMARTCARD_MspInit

Function name	**void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Initialize the SMARTCARD MSP.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_MspDeInit

Function name	**void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	DeInitialize the SMARTCARD MSP.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_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	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be sent. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

HAL_SMARTCARD_Receive

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be received. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

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	• **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_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	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing Receive transfer (blocking mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set

handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name	**void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Handle SMARTCARD interrupt requests.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_TxCpltCallback

Function name	**void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Tx Transfer completed callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_RxCpltCallback

Function name	**void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Rx Transfer completed callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the

specified SMARTCARD module.

Return values	• **None**

HAL_SMARTCARD_ErrorCallback

Function name	**void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD error callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_AbortCpltCallback

Function name	**void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD Abort Complete callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name	**void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD Abort Complete callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name	**void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD Abort Receive Complete callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None**

HAL_SMARTCARD_GetState

Function name	**HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (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

41.3 SMARTCARD Firmware driver defines

41.3.1 SMARTCARD

SMARTCARD Advanced Features Initialization

SMARTCARD_ADVFEATURE_NO_INIT

SMARTCARD_ADVFEATURE_TXINVERT_INIT

SMARTCARD_ADVFEATURE_RXINVERT_INIT

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

SMARTCARD_ADVFEATURE_SWAP_INIT

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE	SMARTCARD frame phase on first clock transition
SMARTCARD_PHASE_2EDGE	SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW	SMARTCARD frame low polarity
SMARTCARD_POLARITY_HIGH	SMARTCARD frame high polarity

SMARTCARD auto retry counter LSB position in CR3 register

SMARTCARD_CR3_SCARCNT_LSB_POS	SMARTCARD auto retry counter LSB position in CR3 register

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE	Binary data inversion disable
SMARTCARD_ADVFEATURE_DATAINV_ENABLE	Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR	DMA enable on Reception Error
SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR	DMA disable on Reception Error

SMARTCARD DMA Requests

SMARTCARD_DMAREQ_TX

SMARTCARD_DMAREQ_RX

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle states.

Parameters:

- __HANDLE__: SMARTCARD handle.

Return value:

- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard Data registers.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FLAG

Description:

- Clear the specified SMARTCARD pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag

- SMARTCARD_CLEAR_FEF Framing error clear flag
- SMARTCARD_CLEAR_NEF Noise detected clear flag
- SMARTCARD_CLEAR_OREF OverRun error clear flag
- SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
- SMARTCARD_CLEAR_TCF Transmission complete clear flag
- SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
- SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the

SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD_FLAG_REACK Receive enable acknowledge flag
  - SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
  - SMARTCARD_FLAG_BUSY Busy flag
  - SMARTCARD_FLAG_EOBF End of block flag
  - SMARTCARD_FLAG_RTOF Receiver timeout flag
  - SMARTCARD_FLAG_TXE Transmit data register empty flag
  - SMARTCARD_FLAG_TC Transmission complete flag
  - SMARTCARD_FLAG_RXNE Receive data register not empty flag
  - SMARTCARD_FLAG_IDLE Idle line detection flag
  - SMARTCARD_FLAG_ORE Overrun error flag
  - SMARTCARD_FLAG_NE Noise error flag
  - SMARTCARD_FLAG_FE Framing error flag
  - SMARTCARD_FLAG_PE Parity error flag

Return value:

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

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_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_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_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_CLEAR_IT

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - – SMARTCARD_CLEAR_PEF Parity error clear flag
  - – SMARTCARD_CLEAR_FEF Framing error clear flag
  - – SMARTCARD_CLEAR_NEF Noise detected clear flag
  - – SMARTCARD_CLEAR_OREF OverRun error clear flag
  - – SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
  - – SMARTCARD_CLEAR_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_SEND_REQ

Description:

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  – SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
  – SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ENABLE

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_DISABLE

Description:

- Disable the USART associated to the SMARTCARD Handle.

	Parameters: • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_DMA_REQUEST_ENABLE	**Description:** • Macros to enable or disable the SmartCard DMA request. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2. • __REQUEST__: specifies the SmartCard DMA request. This parameter can be one of the following values: – SMARTCARD_DMAREQ_TX: SmartCard DMA transmit request – SMARTCARD_DMAREQ_RX: SmartCard DMA receive request
__HAL_SMARTCARD_DMA_REQUEST_DISABLE	

SMARTCARD Flags

SMARTCARD_FLAG_REACK	SMARTCARD receive enable acknowledge flag
SMARTCARD_FLAG_TEACK	SMARTCARD transmit enable acknowledge flag
SMARTCARD_FLAG_BUSY	SMARTCARD busy flag
SMARTCARD_FLAG_EOBF	SMARTCARD end of block flag
SMARTCARD_FLAG_RTOF	SMARTCARD receiver timeout flag
SMARTCARD_FLAG_TXE	SMARTCARD transmit data register empty
SMARTCARD_FLAG_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_LSB_POS	SMARTCARD guard time value LSB position in GTPR register

SMARTCARD interruptions flags mask

SMARTCARD_IT_MASK	SMARTCARD interruptions flags mask

SMARTCARD Interrupt 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 IT CLEAR Flags

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 Time Out Clear Flag
SMARTCARD_CLEAR_EOBF	End Of Block Clear Flag

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE	SMARTCARD frame last data bit clock pulse not output to SCLK pin
SMARTCARD_LASTBIT_ENABLE	SMARTCARD frame last data bit clock pulse output to SCLK pin

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX	SMARTCARD RX mode
SMARTCARD_MODE_TX	SMARTCARD TX mode
SMARTCARD_MODE_TX_RX	SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE	Most significant bit sent/received first disable
SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE	Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_ENABLE	SMARTCARD NACK transmission disabled
SMARTCARD_NACK_DISABLE	SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE	SMARTCARD frame one-bit sample disabled
SMARTCARD_ONE_BIT_SAMPLE_ENABLE	SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE	RX overrun enable
SMARTCARD_ADVFEATURE_OVERRUN_DISABLE	RX overrun disable

SMARTCARD Parity

SMARTCARD_PARITY_EVEN	SMARTCARD frame even parity
SMARTCARD_PARITY_ODD	SMARTCARD frame odd parity

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST	Receive data flush request
SMARTCARD_TXDATA_FLUSH_REQUEST	Transmit data flush request

SMARTCARD block length LSB position in RTOR register

SMARTCARD_RTOR_BLEN_LSB_POS	SMARTCARD block length LSB position in RTOR register

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE	RX pin active level inversion disable
SMARTCARD_ADVFEATURE_RXINV_ENABLE	RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE	TX/RX pins swap disable
SMARTCARD_ADVFEATURE_SWAP_ENABLE	TX/RX pins swap enable

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5	SMARTCARD frame with 0.5 stop bit
SMARTCARD_STOPBITS_1_5	SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE	SMARTCARD receiver timeout disabled
SMARTCARD_TIMEOUT_ENABLE	SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE	TX pin active level inversion disable
SMARTCARD_ADVFEATURE_TXINV_ENABLE	TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B	SMARTCARD frame length

42 HAL SMARTCARD Extension Driver

42.1 SMARTCARDEx Firmware driver API description

42.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.

42.1.2 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()***

42.1.3 Detailed description of functions

HAL_SMARTCARDEx_BlockLength_Config

Function name	**void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)**
Function description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **BlockLength:** SMARTCARD block length (8-bit long at most)
Return values	• **None**

HAL_SMARTCARDEx_TimeOut_Config

Function name	**void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t**

	TimeOutValue)
Function description	Update on the fly the receiver timeout value in RTOR register.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFFF.
Return values	• **None**

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Enable the SMARTCARD receiver timeout feature.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Disable the SMARTCARD receiver timeout feature.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status

43 HAL SMBUS Generic Driver

43.1 SMBUS Firmware driver registers structures

43.1.1 SMBUS_InitTypeDef

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 acknoledge 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

43.1.2 SMBUS_HandleTypeDef

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

43.2 SMBUS Firmware driver API description

43.2.1 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

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

43.2.2 Initialization and de-initialization functions

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

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filer mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode

- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.

This section contains the following APIs:

- ***HAL_SMBUS_Init()***
- ***HAL_SMBUS_DeInit()***
- ***HAL_SMBUS_MspInit()***
- ***HAL_SMBUS_MspDeInit()***

43.2.3 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()***

43.2.4 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()*

43.2.5 Detailed description of functions

HAL_SMBUS_Init

Function name	**HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)**
Function description	Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and initialize the associated handle.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_DeInit

Function name	**HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)**
Function description	DeInitialize the SMBUS peripheral.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_MspInit

Function name	**void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)**
Function description	Initialize the SMBUS MSP.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_MspDeInit

Function name	**void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)**
Function description	DeInitialize the SMBUS MSP.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_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	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent

	• **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• **HAL:** status

HAL_SMBUS_EnableAlert_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Enable the SMBUS alert mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	• **HAL:** status

HAL_SMBUS_DisableAlert_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Disable the SMBUS alert mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	• **HAL:** status

HAL_SMBUS_EnableListen_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Enable the Address listen mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_DisableListen_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Disable the Address listen mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_EV_IRQHandler

Function name	**void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)**
Function description	Handle SMBUS event interrupt request.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_ER_IRQHandler

Function name	**void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)**
Function description	Handle SMBUS error interrupt request.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_MasterTxCpltCallback

Function name	**void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Master Tx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_MasterRxCpltCallback

Function name	**void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Master Rx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_SlaveTxCpltCallback

Function name	**void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Slave Tx Transfer completed callback.

Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_SlaveRxCpltCallback

Function name	**void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Slave Rx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_AddrCallback

Function name	**void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)**
Function description	Slave Address Match callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **TransferDirection:** Master request Transfer Direction (Write/Read) • **AddrMatchCode:** Address Match Code
Return values	• **None**

HAL_SMBUS_ListenCpltCallback

Function name	**void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Listen Complete callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_ErrorCallback

Function name	**void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	SMBUS error callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None**

HAL_SMBUS_GetState

Function name	**uint32_t HAL_SMBUS_GetState (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

43.3 SMBUS Firmware driver defines

43.3.1 SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS_ADDRESSINGMODE_10BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE	No error
HAL_SMBUS_ERROR_BERR	BERR error
HAL_SMBUS_ERROR_ARLO	ARLO error
HAL_SMBUS_ERROR_ACKF	ACKF error
HAL_SMBUS_ERROR_OVR	OVR error
HAL_SMBUS_ERROR_HALTIMEOUT	Timeout error
HAL_SMBUS_ERROR_BUSTIMEOUT	Bus Timeout error
HAL_SMBUS_ERROR_ALERT	Alert error

HAL_SMBUS_ERROR_PECERR	PEC error

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable

- SMBUS_IT_TCI Transfer complete interrupt enable
- SMBUS_IT_STOPI STOP detection interrupt enable
- SMBUS_IT_NACKI NACK received interrupt enable
- SMBUS_IT_ADDRI Address match interrupt enable
- SMBUS_IT_RXI RX interrupt enable
- SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- The: new state of __IT__ (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_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP

detection flag
- – SMBUS_FLAG_BERR Bus error
- – SMBUS_FLAG_ARLO Arbitration lost
- – SMBUS_FLAG_OVR Overrun/Underrun
- – SMBUS_FLAG_PECERR PEC error in reception
- – SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
- – SMBUS_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_DISABLE

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_GENERATE_NACK

Description:

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS ownaddress2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

44 HAL SPI Generic Driver

44.1 SPI Firmware driver registers structures

44.1.1 SPI_InitTypeDef

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***

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 Directional mode state. This parameter can be a value of ***SPI_Direction_mode***
- ***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 a number between Min_Data = 0 and Max_Data = 65535

44.1.2 __SPI_HandleTypeDef

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***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***void(* RxISR***
- ***void(* TxISR***
- ***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
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA handle parameters
- ***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
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  SPI locking object
- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state

- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code

44.2 SPI Firmware driver API description

44.2.1 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 Channel
      – Enable the DMAx clock
      – Configure the DMA handle parameters
      – Configure the DMA Tx or Rx Channel
      – Associate the initilalized hdma_tx(or _rx) handle to the hspi DMA Tx (or Rx) handle
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Channel
3. Program the Mode, Direction , 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 customed 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 @note
4. TX/RX processes are HAL_SPI_TransmitReceive(), HAL_SPI_TransmitReceive_IT() and HAL_SPI_TransmitReceive_DMA()
5. RX processes are HAL_SPI_Receive(), HAL_SPI_Receive_IT() and HAL_SPI_Receive_DMA()
6. TX processes are HAL_SPI_Transmit(), HAL_SPI_Transmit_IT() and HAL_SPI_Transmit_DMA()

44.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze 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
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx periperal.

This section contains the following APIs:

- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***
- ***HAL_SPI_MspDeInit()***

44.2.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectivelly 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. Blocking mode APIs are :
   - HAL_SPI_Transmit()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceive() in full duplex mode
3. Non Blocking mode API's with Interrupt are :
   - HAL_SPI_Transmit_IT()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive_IT() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceive_IT()in full duplex mode
   - HAL_SPI_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_SPI_Transmit_DMA()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceive_DMA() in full duplex mode
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_SPI_TxCpltCallback()
   - HAL_SPI_RxCpltCallback()
   - HAL_SPI_TxRxCpltCallback()

- HAL_SPI_TxHalfCpltCallback()
- HAL_SPI_RxHalfCpltCallback()
- HAL_SPI_TxRxHalfCpltCallback()
- HAL_SPI_ErrorCallback()

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_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()***

44.2.4 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()***

44.2.5 Detailed description of functions

HAL_SPI_Init

Function name	**HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)**
Function description	Initializes the SPI according to the specified parameters in the SPI_InitTypeDef and create the associated handle.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL:** status

HAL_SPI_DeInit

Function name	**HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)**
Function description	DeInitializes 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	SPI MSP Init.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None**

HAL_SPI_MspDeInit

Function name	**void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)**
Function description	SPI MSP DeInit.
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	**HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent

	• **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_SPI_TransmitReceive

Function name	**HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**
Function description	Transmit and Receive an amount of data in blocking mode.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pTxData:** pointer to transmission data buffer • **pRxData:** pointer to reception data buffer to be • **Size:** amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_SPI_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in no-blocking mode with Interrupt.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• **HAL:** status

HAL_SPI_Receive_IT

Function name	**HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in no-blocking mode with Interrupt.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name	**HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function description	Transmit and Receive an amount of data in no-blocking mode with Interrupt.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains

the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer to be
- **Size:** amount of data to be sent

Return values
- **HAL:** status

HAL_SPI_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in no-blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• **HAL:** status

HAL_SPI_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in no-blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• **HAL:** status
Notes	• When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name	**HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function description	Transmit and Receive an amount of data in no-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	Pauses 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	Resumes 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	Stops 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_IRQHandler

Function name	**void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)**
Function description	This function handles SPI interrupt request.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL:** status

HAL_SPI_TxCpltCallback

Function name	**void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None**

HAL_SPI_RxCpltCallback

Function name	**void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Rx Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None**

HAL_SPI_TxRxCpltCallback

Function name	**void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx and Rx Transfer completed callbacks.
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 callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None**

HAL_SPI_TxHalfCpltCallback

Function name	**void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx Half Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None**

HAL_SPI_RxHalfCpltCallback

Function name	**void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Rx Half Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None**

HAL_SPI_TxRxHalfCpltCallback

Function name	**void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx and Rx Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None**

HAL_SPI_GetState

Function name	**HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)**
Function description	Return the SPI 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

44.3 SPI Firmware driver defines

44.3.1 SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI data size

SPI_DATASIZE_8BIT

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	Flag: RXNE,TXE, BSY

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:

- SPI_IT_TXE: Tx buffer empty interrupt enable
- SPI_IT_RXNE: RX buffer not empty interrupt enable
- SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

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

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 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 SPIx where x: 1 or 2 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

Return value:

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

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enables the SPI.

Parameters:

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

Return value:

- None

__HAL_SPI_DISABLE

Description:

- Disables the SPI.

Parameters:

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

Return value:

- None

SPI Flag definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI_FLAG_BSY

SPI_FLAG_FRE

SPI Interrupt configuration 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 Slave Select management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

45 HAL TIM Generic Driver

45.1 TIM Firmware driver registers structures

45.1.1 TIM_Base_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***

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***

45.1.2 TIM_OC_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCFastMode***

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::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.

45.1.3 TIM_OnePulse_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::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

45.1.4 TIM_IC_InitTypeDef

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

45.1.5 TIM_Encoder_InitTypeDef

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

45.1.6 TIM_ClockConfigTypeDef

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

45.1.7 TIM_ClearInputConfigTypeDef

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state. This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources. This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity. This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler. This parameter 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

45.1.8 TIM_SlaveConfigTypeDef

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

45.1.9 TIM_HandleTypeDef

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 ***DMA_Handle_index***
- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State***
  TIM operation state

45.2 TIM Firmware driver API description

45.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   – Input Capture
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

45.2.2 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 timebase
   - 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: HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT(), HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT(), HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT(), HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT(), HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT(), HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA() or HAL_TIM_Encoder_Start_IT()
6. The DMA Burst is managed with the two following functions: HAL_TIM_DMABurst_WriteStart HAL_TIM_DMABurst_ReadStart

45.2.3 Timer Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Timer Base.
- Stop the Timer Base.
- Start the Timer Base and enable interrupt.
- Stop the Timer Base and disable interrupt.
- Start the Timer Base and enable DMA transfer.
- Stop the Timer Base and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_Base_Init()***
- ***HAL_TIM_Base_DeInit()***
- ***HAL_TIM_Base_MspInit()***
- ***HAL_TIM_Base_MspDeInit()***
- ***HAL_TIM_Base_Start()***
- ***HAL_TIM_Base_Stop()***
- ***HAL_TIM_Base_Start_IT()***
- ***HAL_TIM_Base_Stop_IT()***
- ***HAL_TIM_Base_Start_DMA()***
- ***HAL_TIM_Base_Stop_DMA()***

45.2.4 Peripheral State functions

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

This section contains the following APIs:

- ***HAL_TIM_Base_GetState()***
- ***HAL_TIM_OC_GetState()***

- *HAL_TIM_PWM_GetState()*
- *HAL_TIM_IC_GetState()*
- *HAL_TIM_OnePulse_GetState()*
- *HAL_TIM_Encoder_GetState()*
- *TIM_DMAError()*
- *TIM_DMADelayPulseCplt()*
- *TIM_DMACaptureCplt()*

45.2.5 Detailed description of functions

HAL_TIM_Base_Init

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIM_Base_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes the TIM Base peripheral.
Parameters	• **htim:** : TIM 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	DeInitializes 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_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes the TIM peripheral.
Parameters	• **htim:** TIM Output Compare handle
Return values	• **HAL:** status

HAL_TIM_OC_MspInit

Function name	**void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Output Compare MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_OC_MspDeInit

Function name	**void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes TIM Output Compare MSP.
Parameters	• **htim:** : TIM 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 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	• **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_TIM_OC_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channel to be enabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_OC_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Output Compare signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channel to be disabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_OC_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**

Function description	Starts the TIM Output Compare signal generation in DMA mode.
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_TIM_OC_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Output Compare signal generation in DMA mode.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channel to be disabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Init

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM handle
Return values	• **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	• **HAL:** status

HAL_TIM_PWM_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the PWM signal generation in interrupt mode.

Parameters	• **htim:** : TIM handle • **Channel:** TIM Channel to be enabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the PWM signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channels to be disabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**
Function description	Starts the TIM PWM signal generation in DMA mode.
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 • **pData:** The source Buffer address. This buffer contains the values which will be loaded inside the capture/compare registers. • **Length:** The length of data to be transferred from memory to TIM peripheral
Return values	• **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM PWM signal generation in DMA mode.

Parameters	• **htim:** : TIM 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_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes the TIM peripheral.
Parameters	• **htim:** TIM Input Capture handle
Return values	• **HAL:** status

HAL_TIM_IC_MspInit

Function name	**void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM INput Capture MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_IC_MspDeInit

Function name	**void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes TIM Input Capture MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_IC_Start

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Input Capture measurement.

Parameters	• **htim:** : TIM handle • **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Input Capture measurement.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channels to be disabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected

– TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_IC_Start_DMA

Function name
: **HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**

Function description
: Starts the TIM Input Capture measurement on in DMA mode.

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
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values
- **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name
: **HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description
: Stops the TIM Input Capture measurement on 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_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 will 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	• **htim:** : TIM handle • **OutputChannel:** : TIM Channels to be enabled. This parameter is not used since both channels TIM_CHANNEL_1 and TIM_CHANNEL_2 are automatically selected.
Return values	• **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	• **htim:** : TIM handle • **OutputChannel:** : TIM Channels to be disable. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• **HAL:** status

HAL_TIM_OnePulse_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **OutputChannel:** 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
Return values	• **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	• **htim:** : TIM handle • **OutputChannel:** 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
Return values	• **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	• **htim:** TIM Encoder Interface handle • **sConfig:** TIM Encoder Interface configuration structure

Return values	• **HAL:** status

HAL_TIM_Encoder_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes the TIM Encoder interface.
Parameters	• **htim:** TIM Encoder handle
Return values	• **HAL:** status

HAL_TIM_Encoder_MspInit

Function name	**void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Encoder Interface MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_Encoder_MspDeInit

Function name	**void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes TIM Encoder Interface MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_Encoder_Start

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Encoder Interface.
Parameters	• **htim:** : TIM 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 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 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 handle
- **Channel:** TIM Channels to be disabled. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values
- **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name
HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function description
Starts the TIM Encoder Interface in DMA mode.

Parameters
- **htim:** : TIM 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 2 selected
- **pData1:** The destination Buffer address for IC1.
- **pData2:** The destination Buffer address for IC2.

	• **Length:** The length of data to be transferred from TIM peripheral to memory.
Return values	• **HAL:** status

HAL_TIM_Encoder_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Encoder Interface in DMA mode.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• **HAL:** status

HAL_TIM_IRQHandler

Function name	**void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)**
Function description	This function handles TIM interrupts requests.
Parameters	• **htim:** TIM handle
Return values	• **None**

HAL_TIM_OC_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)**
Function description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	• **htim:** : TIM handle • **sConfig:** TIM Output Compare configuration structure • **Channel:** TIM Channel to be configure. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_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	• **htim:** : TIM handle • **sConfig:** TIM PWM configuration structure • **Channel:** TIM Channel 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 values	• **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	• **htim:** : TIM handle • **sConfig:** TIM Input Capture configuration structure • **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_OnePulse_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)**
Function description	Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.
Parameters	• **htim:** : TIM handle • **sConfig:** TIM One Pulse configuration structure • **OutputChannel:** 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 • **InputChannel:** 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
Return values	• **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	• **htim:** : TIM handle • **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral. • **Channel:** specifies the TIM Channel. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 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_ConfigClockSource

Function name	**HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)**
Function description	Configures the clock source to be used.
Parameters	• **htim:** : TIM handle • **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	• **HAL:** status

HAL_TIM_ConfigTI1Input

Function name	**HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)**
Function description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	• **htim:** : TIM handle • **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: – TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input – TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
Return values	• **HAL:** status

HAL_TIM_SlaveConfigSynchronization

Function name	**HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**
Function description	Configures the TIM in Slave mode.
Parameters	• **htim:** : TIM handle • **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	• **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	• **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	• **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name	**HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (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 memory to the TIM peripheral.
Parameters	• **htim:** : TIM handle • **BurstBaseAddress:** TIM Base address from when the DMA will starts the Data write. This parameters can be on 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_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_DCR
- **BurstRequestSrc:** TIM DMA Request sources. This parameters can be on 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_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_WriteStop

Function name	**HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**
Function description	Stops the TIM DMA Burst mode.
Parameters	• **htim:** : TIM handle • **BurstRequestSrc:** TIM DMA Request sources to disable
Return values	• **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name	**HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**
Function description	Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters
- **htim:** : TIM handle
- **BurstBaseAddress:** TIM Base address from when the DMA will starts the Data read. This parameters can be on 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_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_DCR
- **BurstRequestSrc:** TIM DMA Request sources. This parameters can be on 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_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_ReadStop

Function name	**HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**
Function description	Stop the DMA burst reading.
Parameters	• **htim:** : TIM handle • **BurstRequestSrc:** TIM DMA Request sources to disable.
Return values	• **HAL:** status

HAL_TIM_GenerateEvent

Function name	**HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)**
Function description	Generate a software event.
Parameters	• **htim:** : TIM handle • **EventSource:** specifies the event source. This parameter can be one of the following values: – TIM_EventSource_Update: Timer update Event source – TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source – TIM_EventSource_CC2: Timer Capture Compare 2 Event source – TIM_EventSource_CC3: Timer Capture Compare 3 Event source – TIM_EventSource_CC4: Timer Capture Compare 4 Event source

- – TIM_EVENTSOURCE_TRIGGER : Timer Trigger Event source

Return values	• **HAL:** status
Notes	• TIM6 can only generate an update event.

HAL_TIM_ReadCapturedValue

Function name	**uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Read the captured value from Capture Compare unit.
Parameters	• **htim:** : TIM handle • **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name	**void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)**
Function description	Period elapsed callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_OC_DelayElapsedCallback

Function name	**void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)**
Function description	Output Compare callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_IC_CaptureCallback

Function name	**void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)**
Function description	Input Capture callback in non blocking mode.
Parameters	• **htim:** TIM IC handle
Return values	• **None**

HAL_TIM_PWM_PulseFinishedCallback

Function name	**void HAL_TIM_PWM_PulseFinishedCallback**

	(TIM_HandleTypeDef * htim)
Function description	PWM Pulse finished callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_TriggerCallback

Function name	**void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)**
Function description	Hall Trigger detection callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_ErrorCallback

Function name	**void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)**
Function description	Timer error callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None**

HAL_TIM_Base_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM Base state.
Parameters	• **htim:** : TIM handle
Return values	• **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	• **htim:** TIM Ouput Compare handle
Return values	• **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	• **htim:** : TIM handle
Return values	• **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	• **htim:** : TIM handle
Return values	• **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	• **htim:** TIM OPM handle
Return values	• **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	• **htim:** : TIM handle
Return values	• **HAL:** state

TIM_DMADelayPulseCplt

Function name	**void TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)**
Function description	TIM DMA Delay Pulse complete callback.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• **None**

TIM_DMAError

Function name	**void TIM_DMAError (DMA_HandleTypeDef * hdma)**
Function description	TIM DMA error callback.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• **None**

TIM_DMACaptureCplt

Function name	**void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)**

Function description	TIM DMA Capture complete callback.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• **None**

45.3 TIM Firmware driver defines

45.3.1 TIM

TIM channels

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_ALL

Clear input polarity

TIM_CLEARINPUTPOLARITY_INVERTED	Polarity for ETRx pin
TIM_CLEARINPUTPOLARITY_NONINVERTED	Polarity for ETRx pin

Clear input prescaler

TIM_CLEARINPUTPRESCALER_DIV1	No prescaler is used
TIM_CLEARINPUTPRESCALER_DIV2	Prescaler for External ETR pin: Capture performed once every 2 events.
TIM_CLEARINPUTPRESCALER_DIV4	Prescaler for External ETR pin: Capture performed once every 4 events.
TIM_CLEARINPUTPRESCALER_DIV8	Prescaler for External ETR pin: Capture performed once every 8 events.

Clear input source

TIM_CLEARINPUTSOURCE_ETR

TIM_CLEARINPUTSOURCE_NONE

Clock division

TIM_CLOCKDIVISION_DIV1

TIM_CLOCKDIVISION_DIV2

TIM_CLOCKDIVISION_DIV4

Clock polarity

TIM_CLOCKPOLARITY_INVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_NONINVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_RISING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_FALLING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_BOTHEDGE	Polarity for TIx clock sources

Clock prescaler

TIM_CLOCKPRESCALER_DIV1	No prescaler is used
TIM_CLOCKPRESCALER_DIV2	Prescaler for External ETR Clock: Capture performed once every 2 events.
TIM_CLOCKPRESCALER_DIV4	Prescaler for External ETR Clock: Capture performed once every 4 events.
TIM_CLOCKPRESCALER_DIV8	Prescaler for External ETR Clock: Capture performed once every 8 events.

Clock source

TIM_CLOCKSOURCE_ETRMODE2

TIM_CLOCKSOURCE_INTERNAL

TIM_CLOCKSOURCE_ITR0

TIM_CLOCKSOURCE_ITR1

TIM_CLOCKSOURCE_ITR2

TIM_CLOCKSOURCE_ITR3

TIM_CLOCKSOURCE_TI1ED

TIM_CLOCKSOURCE_TI1

TIM_CLOCKSOURCE_TI2

TIM_CLOCKSOURCE_ETRMODE1

Counter mode

TIM_COUNTERMODE_UP

TIM_COUNTERMODE_DOWN

TIM_COUNTERMODE_CENTERALIGNED1

TIM_COUNTERMODE_CENTERALIGNED2

TIM_COUNTERMODE_CENTERALIGNED3

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_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_DCR

TIM_DMABASE_OR

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

DMA sources

TIM_DMA_UPDATE

TIM_DMA_CC1

TIM_DMA_CC2

TIM_DMA_CC3

TIM_DMA_CC4

TIM_DMA_TRIGGER

Encoder_Mode

TIM_ENCODERMODE_TI1

TIM_ENCODERMODE_TI2

TIM_ENCODERMODE_TI12

ETR polarity

TIM_ETRPOLARITY_INVERTED	Polarity for ETR source
TIM_ETRPOLARITY_NONINVERTED	Polarity for ETR source

ETR prescaler

TIM_ETRPRESCALER_DIV1	No prescaler is used
TIM_ETRPRESCALER_DIV2	ETR input source is divided by 2
TIM_ETRPRESCALER_DIV4	ETR input source is divided by 4
TIM_ETRPRESCALER_DIV8	ETR input source is divided by 8

Event sources

TIM_EVENTSOURCE_UPDATE

TIM_EVENTSOURCE_CC1

TIM_EVENTSOURCE_CC2

TIM_EVENTSOURCE_CC3

TIM_EVENTSOURCE_CC4

TIM_EVENTSOURCE_TRIGGER

TIM Exported Constants

IS_TIM_PERIOD

IS_TIM_PRESCALER

IS_TIM_COUNTER_MODE

IS_TIM_CLOCKDIVISION_DIV

IS_TIM_PWM_MODE

IS_TIM_OC_MODE

IS_TIM_FAST_STATE

IS_TIM_OC_POLARITY

IS_TIM_CHANNELS

IS_TIM_OPM_CHANNELS

IS_TIM_IC_POLARITY

IS_TIM_IC_PRESCALER

IS_TIM_OPM_MODE

IS_TIM_ENCODER_MODE

IS_TIM_DMA_SOURCE

IS_TIM_EVENT_SOURCE

IS_TIM_CLOCKSOURCE

IS_TIM_CLOCKPOLARITY

IS_TIM_CLOCKPRESCALER

IS_TIM_CLOCKFILTER

IS_TIM_CLEARINPUT_SOURCE

IS_TIM_CLEARINPUT_POLARITY

IS_TIM_CLEARINPUT_PRESCALER

IS_TIM_CLEARINPUT_FILTER

IS_TIM_TRGO_SOURCE

IS_TIM_SLAVE_MODE

IS_TIM_MSM_STATE

IS_TIM_TRIGGER_SELECTION

IS_TIM_INTERNAL_TRIGGEREVENT_SELECTION

IS_TIM_TRIGGERPOLARITY

IS_TIM_TRIGGERPRESCALER

IS_TIM_TRIGGERFILTER

IS_TIM_TI1SELECTION

IS_TIM_DMA_BASE

IS_TIM_DMA_LENGTH

IS_TIM_IC_FILTER

TIM Exported Macro

__HAL_TIM_RESET_HANDLE_STATE	**Description:** • Reset UART handle state. **Parameters:** • __HANDLE__: : TIM handle **Return value:** • None
__HAL_TIM_ENABLE	**Description:** • Enable the TIM peripheral. **Parameters:** • __HANDLE__: : TIM handle **Return value:** • None
TIM_CCER_CCxE_MASK	
__HAL_TIM_DISABLE	**Description:** • Disable the TIM peripheral. **Parameters:** • __HANDLE__: : TIM handle **Return value:**

- None

__HAL_TIM_ENABLE_IT

__HAL_TIM_ENABLE_DMA

__HAL_TIM_DISABLE_IT

__HAL_TIM_DISABLE_DMA

__HAL_TIM_GET_FLAG

__HAL_TIM_CLEAR_FLAG

__HAL_TIM_GET_IT_SOURCE

__HAL_TIM_CLEAR_IT

__HAL_TIM_IS_TIM_COUNTING_DOWN

__HAL_TIM_SET_PRESCALER

TIM_SET_ICPRESCALERVALUE

TIM_RESET_ICPRESCALERVALUE

TIM_SET_CAPTUREPOLARITY

TIM_RESET_CAPTUREPOLARITY

__HAL_TIM_SET_COMPARE

Description:

- 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:

- 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:

- None

__HAL_TIM_SET_COUNTER

Description:

- 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:

- Gets the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: : TIM handle.

Return value:

- None

__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:

- None

__HAL_TIM_SET_CLOCK DIVISION

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
  - TIM_CLOCKDIVISION_DIV2
  - TIM_CLOCKDIVISION_DIV4

Return value:

- None

__HAL_TIM_GET_CLOCK DIVISION

Description:

- Gets the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: : TIM handle.

Return value:

- None

__HAL_TIM_SET_ICPRESCALER

Description:

- 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_ICPRESCALER

Description:

- 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:

- None

__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_CAPTURE POLARITY

Description:

- 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_FALLIN G: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHE DGE: Rising and Falling Edge

Return value:

- None

Notes:

- The polarity TIM_INPUTCHANNELPOLARITY_BOTHEDGE is not authorized for TIM Channel 4.

Flag definition

TIM_FLAG_UPDATE

TIM_FLAG_CC1

TIM_FLAG_CC2

TIM_FLAG_CC3

TIM_FLAG_CC4

TIM_FLAG_TRIGGER

TIM_FLAG_CC1OF

TIM_FLAG_CC2OF

TIM_FLAG_CC3OF

TIM_FLAG_CC4OF

Input capture polarity

TIM_ICPOLARITY_RISING

TIM_ICPOLARITY_FALLING

TIM_ICPOLARITY_BOTHEDGE

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

Input capture selection

TIM_ICSELECTION_DIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively
TIM_ICSELECTION_INDIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively
TIM_ICSELECTION_TRC	TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

IS_TIM_IC_SELECTION

Input channel polarity

TIM_INPUTCHANNELPOLARITY_RISING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for TIx source

Interrupt definition

TIM_IT_UPDATE

TIM_IT_CC1

TIM_IT_CC2

TIM_IT_CC3

TIM_IT_CC4

TIM_IT_TRIGGER

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

Master slave mode

TIM_MASTERSLAVEMODE_ENABLE

TIM_MASTERSLAVEMODE_DISABLE

One pulse mode

TIM_OPMODE_SINGLE

TIM_OPMODE_REPETITIVE

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_ACTIVE

TIM_OCMODE_FORCED_INACTIVE

Output compare N state

TIM_OUTPUTNSTATE_DISABLE

TIM_OUTPUTNSTATE_ENABLE

Output compare polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

Output compare state

TIM_OUTPUTSTATE_DISABLE

TIM_OUTPUTSTATE_ENABLE

Output fast state

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

Slave mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TI1 selection

TIM_TI1SELECTION_CH1

TIM_TI1SELECTION_XORCOMBINATION

Trigger polarity

TIM_TRIGGERPOLARITY_INVERTED	Polarity for ETRx trigger sources
TIM_TRIGGERPOLARITY_NONINVERTED	Polarity for ETRx trigger sources
TIM_TRIGGERPOLARITY_RISING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_FALLING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_BOTHEDGE	Polarity for TIxFPx or TI1_ED trigger sources

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.

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

46 HAL TIM Extension Driver

46.1 TIMEx Firmware driver registers structures

46.1.1 TIM_MasterConfigTypeDef

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***

46.2 TIMEx Firmware driver API description

46.2.1 TIM specific features integration

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for: Input Capture Output Compare PWM generation (Edge and Center-aligned Mode) One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

46.2.2 How to use this driver

1. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
2. 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 operating mode using one of the configuration function of this driver:
   - HAL_TIMEx_MasterConfigSynchronization() to configure the peripheral in master mode.
5. Remap the Timer I/O using HAL_TIMEx_RemapConfig() API.

46.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure Master and the Slave synchronization.

This section contains the following APIs:

- *HAL_TIMEx_MasterConfigSynchronization()*
- *HAL_TIMEx_RemapConfig()*

46.2.4 Detailed description of functions

HAL_TIMEx_RemapConfig

Function name	**HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)**
Function description	Configures the remapping of the TIM2, TIM3, TIM21 and TIM22 inputs.
Parameters	• **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Remap:** specifies the TIM input remapping source. This parameter is a combination of the following values depending on TIM instance:
Return values	• **HAL:** status
Notes	• It is not possible to connect TIM2 and TIM21 on GPIOB5_AF4 at the same time. When selecting TIM3_TI2_GPIOB5_AF4, Channel2 of TIM3 will be connected to GPIOB5_AF4 and Channel2 of TIM22 will be connected to some other GPIOs. (refer to alternate functions for more details) When selecting TIM3_TI2_GPIO_DEF, Channel2 of Timer 3 will be connected an GPIO (other than GPIOB5_AF4) and Channel2 of TIM22 will be connected to GPIOB5_AF4.

HAL_TIMEx_MasterConfigSynchronization

Function name	**HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)**
Function description	Configures the TIM in master mode.
Parameters	• **htim:** TIM handle. • **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.
Return values	• **HAL:** status

46.3 TIMEx Firmware driver defines

46.3.1 TIMEx

Remaping

TIM2_ETR_GPIO

TIM2_ETR_HSI48

TIM2_ETR_HSI16

TIM2_ETR_LSE

TIM2_ETR_COMP2_OUT

TIM2_ETR_COMP1_OUT

TIM2_TI4_GPIO

TIM2_TI4_COMP2

TIM2_TI4_COMP1

TIM21_ETR_GPIO

TIM21_ETR_COMP2_OUT

TIM21_ETR_COMP1_OUT

TIM21_ETR_LSE

TIM21_TI1_GPIO

TIM21_TI1_MCO

TIM21_TI1_RTC_WKUT_IT

TIM21_TI1_HSE_RTC

TIM21_TI1_MSI

TIM21_TI1_LSE

TIM21_TI1_LSI

TIM21_TI1_COMP1_OUT

TIM21_TI2_GPIO

TIM21_TI2_COMP2_OUT

TIM22_ETR_GPIO

TIM22_ETR_COMP2_OUT

TIM22_ETR_COMP1_OUT

TIM22_ETR_LSE

TIM22_TI1_GPIO1

TIM22_TI1_COMP2_OUT

TIM22_TI1_COMP1_OUT

TIM22_TI1_GPIO2

TIM3_TI4_GPIO_DEF

TIM3_TI4_GPIOC9_AF2

TIM3_TI2_GPIO_DEF

TIM3_TI2_GPIOB5_AF4

TIM3_TI1_USB_SOF

TIM3_TI1_GPIO

TIM3_ETR_GPIO

TIM3_ETR_HSI

IS_TIM_REMAP

IS_CHANNEL_AVAILABLE

Trigger 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

IS_TIM_TRGO_SOURCE

47 HAL TSC Generic Driver

47.1 TSC Firmware driver registers structures

47.1.1 TSC_InitTypeDef

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

47.1.2 TSC_IOConfigTypeDef

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

47.1.3 TSC_HandleTypeDef

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

47.2 TSC Firmware driver API description

47.2.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group
3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty
10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

47.2.2 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.

47.2.3 IO Operation functions

This section provides functions allowing to:

- Start acquisition in polling mode.
- Start acquisition in interrupt mode.
- Stop conversion in polling mode.
- Stop conversion in interrupt mode.
- Get group acquisition status.
- Get group acquisition value.

This section contains the following APIs:

- ***HAL_TSC_Start()***
- ***HAL_TSC_Start_IT()***
- ***HAL_TSC_Stop()***
- ***HAL_TSC_Stop_IT()***
- ***HAL_TSC_GroupGetStatus()***
- ***HAL_TSC_GroupGetValue()***
- ***HAL_TSC_PollForAcquisition()***

47.2.4 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()***

47.2.5 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()***
- ***HAL_TSC_ConvCpltCallback()***
- ***HAL_TSC_ErrorCallback()***

47.2.6 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- ***HAL_TSC_Init()***
- ***HAL_TSC_DeInit()***
- ***HAL_TSC_MspInit()***
- ***HAL_TSC_MspDeInit()***

47.2.7 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_DeInit

Function name	**HAL_StatusTypeDef HAL_TSC_DeInit (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_MspDeInit

Function name	**void HAL_TSC_MspDeInit (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	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status

HAL_TSC_Start_IT

Function name	**HAL_StatusTypeDef HAL_TSC_Start_IT (TSC_HandleTypeDef * htsc)**
Function description	Enables the interrupt and starts the acquisition.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status.

HAL_TSC_Stop

Function name	**HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)**
Function description	Stops the acquisition previously launched in polling mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status

HAL_TSC_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TSC_Stop_IT (TSC_HandleTypeDef * htsc)**
Function description	Stops the acquisition previously launched in interrupt mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status

HAL_TSC_PollForAcquisition

Function name	**HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)**
Function description	Start acquisition and wait until completion.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** state
Notes	• There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

HAL_TSC_GroupGetStatus

Function name	**TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (TSC_HandleTypeDef * htsc, uint32_t gx_index)**
Function description	Gets the acquisition status for a group.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **gx_index:** Index of the group
Return values	• **Group:** status

HAL_TSC_GroupGetValue

Function name	**uint32_t HAL_TSC_GroupGetValue (TSC_HandleTypeDef * htsc, uint32_t gx_index)**
Function description	Gets the acquisition measure for a group.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

- **gx_index:** Index of the group

Return values
- **Acquisition:** measure

HAL_TSC_IOConfig

Function name
: **HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)**

Function description
: Configures TSC IOs.

Parameters
- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **config:** pointer to the configuration structure.

Return values
- **HAL:** status

HAL_TSC_IODischarge

Function name
: **HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, uint32_t choice)**

Function description
: Discharge TSC IOs.

Parameters
- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **choice:** enable or disable

Return values
- **HAL:** status

HAL_TSC_GetState

Function name
: **HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)**

Function description
: Return the TSC state.

Parameters
- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values
- **HAL:** state

HAL_TSC_IRQHandler

Function name
: **void HAL_TSC_IRQHandler (TSC_HandleTypeDef * htsc)**

Function description
: Handles TSC interrupt request.

Parameters
- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values
- **None**

HAL_TSC_ConvCpltCallback

Function name
: **void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef * htsc)**

Function description
: Acquisition completed callback in non blocking mode.

Parameters
- **htsc:** pointer to a TSC_HandleTypeDef structure that

contains the configuration information for the specified TSC.

Return values • **None**

HAL_TSC_ErrorCallback

Function name	**void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)**
Function description	Error callback in non blocking mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **None**

47.3 TSC Firmware driver defines

47.3.1 TSC

TSC Exported Constants

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

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

TSC_SS_PRESC_DIV1

TSC_SS_PRESC_DIV2

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

TSC_MCV_255

TSC_MCV_511

TSC_MCV_1023

TSC_MCV_2047

TSC_MCV_4095

TSC_MCV_8191

TSC_MCV_16383

TSC_IODEF_OUT_PP_LOW

TSC_IODEF_IN_FLOAT

TSC_SYNC_POLARITY_FALLING

TSC_SYNC_POLARITY_RISING

TSC_ACQ_MODE_NORMAL

TSC_ACQ_MODE_SYNCHRO

TSC_IOMODE_UNUSED

TSC_IOMODE_CHANNEL

TSC_IOMODE_SHIELD

TSC_IOMODE_SAMPLING

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

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_IOS

TSC Exported Macros

__HAL_TSC_RESET_HANDLE_STATE

Description:

- Reset TSC handle state.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_ENABLE

Description:

- Enable the TSC peripheral.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_DISABLE	**Description:** • Disable the TSC peripheral. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_START_ACQ	**Description:** • Start acquisition. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_STOP_ACQ	**Description:** • Stop acquisition. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_SET_IODEF_OUTPPLOW	**Description:** • Set IO default mode to output push-pull low. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_SET_IODEF_INFLOAT	**Description:** • Set IO default mode to input floating. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_SET_SYNC_POL_FALL	**Description:** • Set synchronization polarity to falling edge. **Parameters:** • __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_SET_SYNC_POL_RISE_HIGH

Description:

- Set synchronization polarity to rising edge and high level.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_ENABLE_IT

Description:

- Enable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_DISABLE_IT

Description:

- Disable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_GET_IT_SOURCE

Description:

- Check 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_HYSTERESIS

Description:

- Enable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_HYSTERESIS

Description:

- Disable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_OPEN_ANALOG_SWITCH

Description:

- Open analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_CLOSE_ANALOG_SWITCH

Description:

- Close analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_CHANNEL

Description:

- Enable a group of IOs in channel mode.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_CHANNEL

Description:

- Disable a group of channel IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_SAMPLING

Description:

- Enable a group of IOs in sampling mode.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_SAMPLING

Description:

- Disable a group of sampling IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_GROUP

Description:

- Enable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None

__HAL_TSC_DISABLE_GROUP

Description:

- Disable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None

__HAL_TSC_GET_GROUP_STATUS

Description:

- Gets acquisition group status.

Parameters:

- __HANDLE__: TSC Handle
- __GX_INDEX__: Group index

Return value:

- SET: or RESET

TSC Flags Definition

TSC_FLAG_EOA

TSC_FLAG_MCE

TSC Interrupts Definition

TSC_IT_EOA

TSC_IT_MCE

48 HAL UART Generic Driver

48.1 UART Firmware driver registers structures

48.1.1 UART_InitTypeDef

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, Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))If oversampling is 8, Baud Rate Register[15:4] = ((2 * PCLKx) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * PCLKx) / ((huart->Init.BaudRate)))[3:0]) >> 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 ***UARTEx_Word_Length***.
- ***uint32_t UART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.
- ***uint32_t UART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***UART_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.
- ***uint32_t UART_InitTypeDef::OverSampling***
  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.
- ***uint32_t UART_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.

48.1.2 UART_AdvFeatureInitTypeDef

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t AutoBaudRateEnable***
- ***uint32_t AutoBaudRateMode***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of ***UART_Advanced_Features_Initialization_Type***.
- ***uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***UART_Tx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***UART_Rx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***UART_Data_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***UART_Rx_Tx_Swap***.
- ***uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***UART_Overrun_Disable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***UART_DMA_Disable_on_Rx_Error***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable***
  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of ***UART_AutoBaudRate_Enable***
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode***
  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of ***UART_AutoBaud_Rate_Mode***.
- ***uint32_t UART_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***UART_MSB_First***.

48.1.3 UART_HandleTypeDef

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

48.2 UART Firmware driver API description

48.2.1 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: 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 LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.
8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

48.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init()and HAL_MultiProcessor_Init()API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_UART_Init()***
- ***HAL_HalfDuplex_Init()***
- ***HAL_LIN_Init()***
- ***HAL_MultiProcessor_Init()***
- ***HAL_UART_DeInit()***
- ***HAL_UART_MspInit()***
- ***HAL_UART_MspDeInit()***

48.2.3 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()*

48.2.4 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
- UART_SetConfig() API configures the UART peripheral
- UART_AdvFeatureConfig() API optionally configures the UART advanced features
- UART_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- HAL_HalfDuplex_EnableTransmitter() API disables receiver and enables transmitter
- HAL_HalfDuplex_EnableReceiver() API disables transmitter and enables receiver
- HAL_LIN_SendBreak() API transmits the break characters

This section contains the following APIs:

- *HAL_MultiProcessor_EnableMuteMode()*
- *HAL_MultiProcessor_DisableMuteMode()*
- *HAL_MultiProcessor_EnterMuteMode()*
- *HAL_HalfDuplex_EnableTransmitter()*
- *HAL_HalfDuplex_EnableReceiver()*
- *HAL_LIN_SendBreak()*

48.2.5 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()*

48.2.6 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_LIN_Init

Function name	**HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)**
Function description	Initialize the LIN mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	• **huart:** UART handle. • **BreakDetectLength:** specifies the LIN break detection length. This parameter can be one of the following values: – UART_LINBREAKDETECTLENGTH_10B 10-bit break detection – UART_LINBREAKDETECTLENGTH_11B 11-bit break detection
Return values	• **HAL:** status

HAL_MultiProcessor_Init

Function name	**HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)**
Function description	Initialize the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.
Parameters	• **huart:** UART handle. • **Address:** UART node address (4-, 6-, 7- or 8-bit long). • **WakeUpMethod:** specifies the UART wakeup method. This parameter can be one of the following values: – UART_WAKEUPMETHOD_IDLELINE WakeUp by an idle line detection – UART_WAKEUPMETHOD_ADDRESSMARK WakeUp by an address mark
Return values	• **HAL:** status
Notes	• If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function. • If the user resorts to address mark wake up, the address

length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

HAL_UART_DeInit

Function name	**HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)**
Function description	DeInitialize the UART peripheral.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UART_MspInit

Function name	**void HAL_UART_MspInit (UART_HandleTypeDef * huart)**
Function description	Initialize the UART MSP.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_MspDeInit

Function name	**void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)**
Function description	DeInitialize the UART MSP.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_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	• **huart:** UART handle. • **pData:** 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 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_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	• **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_DMA

Function name	**HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in DMA mode.
Parameters	• **huart:** UART handle. • **pData:** pointer to data buffer. • **Size:** amount of data to be received.
Return values	• **HAL:** status
Notes	• When the UART parity is enabled (PCE = 1) the data received contain the parity bit. • 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_DMAPause

Function name	**HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)**
Function description	Pause the DMA Transfer.
Parameters	• **huart:** UART handle.

Return values	• **HAL:** status

HAL_UART_DMAResume

Function name	**HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)**
Function description	Resume the DMA Transfer.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UART_DMAStop

Function name	**HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)**
Function description	Stop the DMA Transfer.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UART_Abort

Function name	**HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)**
Function description	Abort ongoing transfers (blocking mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit

Function name	**HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Transmit transfer (blocking mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY

- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive

Function name	**HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Receive transfer (blocking mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name	**HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts

(Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name	**void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)**
Function description	Handle UART interrupt request.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_TxHalfCpltCallback

Function name	**void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function description	Tx Half Transfer completed callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_TxCpltCallback

Function name	**void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)**
Function description	Tx Transfer completed callback.

Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_RxHalfCpltCallback

Function name	**void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function description	Rx Half Transfer completed callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_RxCpltCallback

Function name	**void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)**
Function description	Rx Transfer completed callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_ErrorCallback

Function name	**void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)**
Function description	UART error callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_AbortCpltCallback

Function name	**void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_AbortTransmitCpltCallback

Function name	**void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_UART_AbortReceiveCpltCallback

Function name	**void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Receive Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

HAL_LIN_SendBreak

Function name	**HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)**
Function description	Transmit break characters.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name	**HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)**
Function description	Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name	**HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)**
Function description	Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name	**void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)**
Function description	Enter UART mute mode (means UART actually enters mute mode).
Parameters	• **huart:** UART handle.
Return values	• **None**
Notes	• To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name	**HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)**
Function description	Enable the UART transmitter and disable the UART receiver.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name	**HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)**
Function description	Enable the UART receiver and disable the UART transmitter.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status.

HAL_UART_GetState

Function name	**HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)**
Function description	Return the UART handle state.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	• **HAL:** state

HAL_UART_GetError

Function name	**uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)**
Function description	Return the UART handle error code.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	• **UART:** Error Code

UART_SetConfig

Function name	**HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)**
Function description	Configure the UART peripheral.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

UART_CheckIdleState

Function name	**HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)**

Function description	Check the UART Idle State.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

UART_WaitOnFlagUntilTimeout

Function name	**HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)**
Function description	Handle UART Communication Timeout.
Parameters	• **huart:** UART handle. • **Flag:** Specifies the UART flag to check • **Status:** Flag status (SET or RESET) • **Tickstart:** Tick start value • **Timeout:** Timeout duration
Return values	• **HAL:** status

UART_AdvFeatureConfig

Function name	**void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)**
Function description	Configure the UART peripheral advanced features.
Parameters	• **huart:** UART handle.
Return values	• **None**

48.3 UART Firmware driver defines

48.3.1 UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT	No advanced feature initialization
UART_ADVFEATURE_TXINVERT_INIT	TX pin active level inversion
UART_ADVFEATURE_RXINVERT_INIT	RX pin active level inversion
UART_ADVFEATURE_DATAINVERT_INIT	Binary data inversion
UART_ADVFEATURE_SWAP_INIT	TX/RX pins swap
UART_ADVFEATURE_RXOVERRUNDISABLE_INIT	RX overrun disable
UART_ADVFEATURE_DMADISABLEONERROR_INIT	DMA disable on Reception Error
UART_ADVFEATURE_AUTOBAUDRATE_INIT	Auto Baud rate detection initialization
UART_ADVFEATURE_MSBFIRST_INIT	Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE	RX Auto Baud rate detection enable

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE	RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT	Auto Baud rate detection on start bit
UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE	Auto Baud rate detection on falling edge
UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME	Auto Baud rate detection on 0x7F frame detection
UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME	Auto Baud rate detection on 0x55 frame detection

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS	UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS	UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS	UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE	Binary data inversion disable
UART_ADVFEATURE_DATAINV_ENABLE	Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR	DMA enable on Reception Error
UART_ADVFEATURE_DMA_DISABLEONRXERROR	DMA disable on Reception Error

UART DMA Rx

UART_DMA_RX_DISABLE	UART DMA RX disabled
UART_DMA_RX_ENABLE	UART DMA RX enabled

UART DMA Tx

UART_DMA_TX_DISABLE	UART DMA TX disabled
UART_DMA_TX_ENABLE	UART DMA TX enabled

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH	Driver enable signal is active high
UART_DE_POLARITY_LOW	Driver enable signal is active low

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE	**Description:** • Reset UART handle states. **Parameters:**

- __HANDLE__: UART handle.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART Data registers.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - 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

Return value:

- None

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OREFLAG

Description:

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clear the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_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_REACK Receive enable acknowledge flag
  - UART_FLAG_TEACK Transmit enable acknowledge flag

- UART_FLAG_WUF Wake up from stop mode flag
- UART_FLAG_RWU Receiver wake up flag (if the UART in mute mode)
- UART_FLAG_SBKF Send Break flag
- UART_FLAG_CMF Character match flag
- UART_FLAG_BUSY Busy flag
- UART_FLAG_ABRF Auto Baud rate detection flag
- UART_FLAG_ABRE Auto Baud rate detection error flag
- UART_FLAG_EOBF End of block flag
- UART_FLAG_RTOF Receiver timeout flag
- UART_FLAG_CTS CTS Change flag
- UART_FLAG_LBDF LIN Break detection flag
- 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
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_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
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_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
  - UART_IT_CM Character match interrupt

- UART_IT_CTS CTS change interrupt
- UART_IT_LBD LIN Break detection interrupt
- UART_IT_TXE Transmit Data Register empty interrupt
- UART_IT_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_SOURCE

Description:

- 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
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_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
  - 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

Return value:

- None

__HAL_UART_SEND_REQ

Description:

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - UART_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - UART_SENDBREAK_REQUEST Send Break Request
  - UART_MUTE_MODE_REQUEST Mute Mode Request
  - UART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - UART_TXDATA_FLUSH_REQUEST

	Transmit data flush Request **Return value:** • None
__HAL_UART_ONE_BIT_SAMPLE_ENABLE	**Description:** • Enable the UART one bit sample method. **Parameters:** • __HANDLE__: specifies the UART Handle. **Return value:** • None
__HAL_UART_ONE_BIT_SAMPLE_DISABLE	**Description:** • Disable the UART one bit sample method. **Parameters:** • __HANDLE__: specifies the UART Handle. **Return value:** • None
__HAL_UART_ENABLE	**Description:** • Enable UART. **Parameters:** • __HANDLE__: specifies the UART Handle. **Return value:** • None
__HAL_UART_DISABLE	**Description:** • Disable UART. **Parameters:** • __HANDLE__: specifies the UART Handle. **Return value:** • None
__HAL_UART_HWCONTROL_CTS_ENABLE	**Description:** • Enable CTS flow control. **Parameters:** • __HANDLE__: specifies the UART Handle. **Return value:** • None **Notes:** • This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init()

function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_ONE_BIT_ENABLE

Description:

- macros to enable the UART's one bit

	sampling method **Parameters:** • __HANDLE__: specifies the UART Handle. **Return value:** • None
__HAL_UART_ONE_BIT_DISABLE	**Description:** • macros to disable the UART's one bit sampling method **Parameters:** • __HANDLE__: specifies the UART Handle. **Return value:** • None

UART Status Flags

UART_FLAG_REACK	UART receive enable acknowledge flag
UART_FLAG_TEACK	UART transmit enable acknowledge flag
UART_FLAG_WUF	UART wake-up from stop mode flag
UART_FLAG_RWU	UART receiver wake-up from mute mode flag
UART_FLAG_SBKF	UART send break flag
UART_FLAG_CMF	UART character match flag
UART_FLAG_BUSY	UART busy flag
UART_FLAG_ABRF	UART auto Baud rate flag
UART_FLAG_ABRE	UART auto Baud rate error
UART_FLAG_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
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

UART 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_ERR	
UART_IT_ORE	
UART_IT_NE	UART noise error interruption
UART_IT_FE	UART frame error interruption

UART Interruption Clear Flags

UART_CLEAR_PEF	Parity Error Clear Flag
UART_CLEAR_FEF	Framing Error Clear Flag
UART_CLEAR_NEF	Noise 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
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 Local Interconnection Network mode

UART_LIN_DISABLE	Local Interconnect Network disable
UART_LIN_ENABLE	Local Interconnect Network enable

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B	LIN 10-bit break detection length
UART_LINBREAKDETECTLENGTH_11B	LIN 11-bit break detection length

UART Transfer Mode

UART_MODE_RX	RX mode
UART_MODE_TX	TX mode
UART_MODE_TX_RX	RX and TX mode

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE	Most significant bit sent/received first disable
UART_ADVFEATURE_MSBFIRST_ENABLE	Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE	UART mute mode disable
UART_ADVFEATURE_MUTEMODE_ENABLE	UART mute mode enable

UART One Bit Sampling Method

UART_ONE_BIT_SAMPLE_DISABLE	One-bit sampling disable
UART_ONE_BIT_SAMPLE_ENABLE	One-bit sampling enable

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE	RX overrun enable
UART_ADVFEATURE_OVERRUN_DISABLE	RX overrun disable

UART Over Sampling

UART_OVERSAMPLING_16	Oversampling by 16
UART_OVERSAMPLING_8	Oversampling by 8

UART Parity

UART_PARITY_NONE	No parity
UART_PARITY_EVEN	Even parity
UART_PARITY_ODD	Odd parity

UART Receiver TimeOut

UART_RECEIVER_TIMEOUT_DISABLE	UART receiver timeout disable
UART_RECEIVER_TIMEOUT_ENABLE	UART receiver timeout enable

UART Request Parameters

UART_AUTOBAUD_REQUEST	Auto-Baud Rate Request
UART_SENDBREAK_REQUEST	Send Break Request
UART_MUTE_MODE_REQUEST	Mute Mode Request
UART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
UART_TXDATA_FLUSH_REQUEST	Transmit data flush Request

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE	RX pin active level inversion disable
UART_ADVFEATURE_RXINV_ENABLE	RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE	TX/RX pins swap disable
UART_ADVFEATURE_SWAP_ENABLE	TX/RX pins swap enable

UART State

UART_STATE_DISABLE	UART disabled
UART_STATE_ENABLE	UART enabled

UART Number of Stop Bits

UART_STOPBITS_1	UART frame with 1 stop bit
UART_STOPBITS_1_5	UART frame with 1.5 stop bits
UART_STOPBITS_2	UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE	UART stop mode disable
UART_ADVFEATURE_STOPMODE_ENABLE	UART stop mode enable

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE	UART polling-based communications time-out value

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE	TX pin active level inversion disable
UART_ADVFEATURE_TXINV_ENABLE	TX pin active level inversion enable

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS	UART wake-up on address
UART_WAKEUP_ON_STARTBIT	UART wake-up on start bit
UART_WAKEUP_ON_READDATA_NONEMPTY	UART wake-up on receive data register not empty

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE	UART wake-up on idle line
UART_WAKEUPMETHOD_ADDRESSMARK	UART wake-up on address mark

49 HAL UART Extension Driver

49.1 UARTEx Firmware driver registers structures

49.1.1 UART_WakeUpTypeDef

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***
  Specifies which event will activat the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***
  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UARTEx_WakeUp_Address_Length***.
- ***uint8_t UART_WakeUpTypeDef::Address***
  UART/USART node address (7-bit long max).

49.2 UARTEx Firmware driver API description

49.2.1 UART peripheral extended features

49.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_RS485Ex_Init() API follows the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_RS485Ex_Init()***

49.2.3 Peripheral Control functions

This section provides the following functions:

- HAL_UARTEx_EnableClockStopMode() API enables the UART clock (HSI or LSE only) during stop mode
- HAL_UARTEx_DisableClockStopMode() API disables the above functionality
- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() API defines the wake-up from stop mode trigger: address match, Start Bit detection or RXNE bit status.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_EnableClockStopMode() API enables the UART HSI clock during stop mode
- HAL_UARTEx_DisableClockStopMode() API disables the above functionality
- HAL_UARTEx_WakeupCallback() called upon UART wakeup interrupt

This section contains the following APIs:

- ***HAL_MultiProcessorEx_AddressLength_Set()***
- ***HAL_UARTEx_StopModeWakeUpSourceConfig()***
- ***HAL_UARTEx_EnableStopMode()***
- ***HAL_UARTEx_DisableStopMode()***
- ***HAL_UARTEx_EnableClockStopMode()***
- ***HAL_UARTEx_DisableClockStopMode()***
- ***HAL_UARTEx_WakeupCallback()***

49.2.4 Detailed description of functions

HAL_RS485Ex_Init

Function name	**HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)**
Function description	Initialize the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle.
Parameters	• **huart:** UART handle. • **Polarity:** select the driver enable polarity. This parameter can be one of the following values: – UART_DE_POLARITY_HIGH DE signal is active high – UART_DE_POLARITY_LOW DE signal is active low • **AssertionTime:** Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time,

	depending on the oversampling rate) • **DeassertionTime:** Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).
Return values	• **HAL:** status

HAL_UARTEx_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_READDATA_NONEMPTY
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_MultiProcessorEx_AddressLength_Set

Function name	**HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)**

Function description	By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).
Parameters	• **huart:** UART handle. • **AddressLength:** this parameter can be one of the following values: – UART_ADDRESS_DETECT_4B 4-bit long address – UART_ADDRESS_DETECT_7B 6-, 7- or 8-bit long address
Return values	• **HAL:** status
Notes	• Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

HAL_UARTEx_EnableClockStopMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_EnableClockStopMode (UART_HandleTypeDef * huart)**
Function description	Enable UART Clock in Stop Mode The UART keeps the Clock ON during Stop mode.
Parameters	• **huart:** uart handle
Return values	• **HAL:** status

HAL_UARTEx_DisableClockStopMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_DisableClockStopMode (UART_HandleTypeDef * huart)**
Function description	Disable UART Clock in Stop Mode.
Parameters	• **huart:** uart handle
Return values	• **HAL:** status

HAL_UARTEx_WakeupCallback

Function name	**void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)**
Function description	UART wakeup from Stop mode callback.
Parameters	• **huart:** UART handle.
Return values	• **None**

49.3 UARTEx Firmware driver defines

49.3.1 UARTEx

UART Extended WakeUp Address Length

UART_ADDRESS_DETECT_4B	4-bit long wake-up address

UART_ADDRESS_DETECT_7B	7-bit long wake-up address

UART 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

50 HAL USART Generic Driver

50.1 USART Firmware driver registers structures

50.1.1 USART_InitTypeDef

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) / ((huart->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***.

50.1.2 USART_HandleTypeDef

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

50.2 USART Firmware driver API description

50.2.1 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: 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 husart handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API.

50.2.2 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: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***

50.2.3 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 API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback()user callback will be executed when a communication error is detected
2. Blocking mode API's are :
   – HAL_USART_Transmit()in simplex mode
   – HAL_USART_Receive() in full duplex receive only
   – HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
   – HAL_USART_Transmit_IT()in simplex mode
   – HAL_USART_Receive_IT() in full duplex receive only
   – HAL_USART_TransmitReceive_IT()in full duplex mode
   – HAL_USART_IRQHandler()
4. No-Blocking mode API's with DMA are :
   – HAL_USART_Transmit_DMA()in simplex mode
   – HAL_USART_Receive_DMA() in full duplex receive only
   – HAL_USART_TransmitReceive_DMA() in full duplex mode
   – HAL_USART_DMAPause()
   – HAL_USART_DMAResume()
   – HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
   – HAL_USART_TxCpltCallback()
   – HAL_USART_RxCpltCallback()
   – HAL_USART_TxHalfCpltCallback()
   – HAL_USART_RxHalfCpltCallback()
   – HAL_USART_ErrorCallback()
   – HAL_USART_TxRxCpltCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's : (+) HAL_USART_Abort() (+) HAL_USART_Abort_IT()
7. For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided: (+) HAL_USART_AbortCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows : (+) Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user. (+) Error is considered as Blocking : Transfer could not be

completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output). (#) There are two modes of transfer: (++) Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer. (++) No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback()user callback will be executed when a communication error is detected (#) Blocking mode API's are : (++) HAL_USART_Transmit()in simplex mode (++) HAL_USART_Receive() in full duplex receive only (++) HAL_USART_TransmitReceive() in full duplex mode (#) Non-Blocking mode API's with Interrupt are : (++) HAL_USART_Transmit_IT()in simplex mode (++) HAL_USART_Receive_IT() in full duplex receive only (++) HAL_USART_TransmitReceive_IT()in full duplex mode (++) HAL_USART_IRQHandler() (#) No-Blocking mode API's with DMA are : (++) HAL_USART_Transmit_DMA()in simplex mode (++) HAL_USART_Receive_DMA() in full duplex receive only (++) HAL_USART_TransmitReceive_DMA() in full duplex mode (++) HAL_USART_DMAPause() (++) HAL_USART_DMAResume() (++) HAL_USART_DMAStop() (#) 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() (#) Non-Blocking mode transfers could be aborted using Abort API's :

- HAL_USART_Abort()
- HAL_USART_Abort_IT() (#) For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided:
- HAL_USART_AbortCpltCallback() (#) In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
- Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user.
- Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_USART_Transmit()***
- ***HAL_USART_Receive()***
- ***HAL_USART_TransmitReceive()***
- ***HAL_USART_Transmit_IT()***
- ***HAL_USART_Receive_IT()***
- ***HAL_USART_TransmitReceive_IT()***
- ***HAL_USART_Transmit_DMA()***
- ***HAL_USART_Receive_DMA()***

- *HAL_USART_TransmitReceive_DMA()*
- *HAL_USART_DMAPause()*
- *HAL_USART_DMAResume()*
- *HAL_USART_DMAStop()*
- *HAL_USART_Abort()*
- *HAL_USART_Abort_IT()*
- *HAL_USART_IRQHandler()*
- *HAL_USART_TxCpltCallback()*
- *HAL_USART_TxHalfCpltCallback()*
- *HAL_USART_RxCpltCallback()*
- *HAL_USART_RxHalfCpltCallback()*
- *HAL_USART_TxRxCpltCallback()*
- *HAL_USART_ErrorCallback()*
- *HAL_USART_AbortCpltCallback()*

50.2.4 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()*

50.2.5 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 * husart)**
Function description	Initialize the USART MSP.
Parameters	• **husart:** USART handle.

Return values	• **None**

HAL_USART_MspDeInit

Function name	**void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)**
Function description	DeInitialize the USART MSP.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_Transmit

Function name	**HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, 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 * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **husart:** USART handle. • **pRxData:** Pointer to data buffer. • **Size:** Amount of data to be received. • **Timeout:** Timeout duration.
Return values	• **HAL:** status
Notes	• 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 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_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	• **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	• **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_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	• **husart:** USART handle. • **pTxData:** 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 pTxData.

HAL_USART_Receive_IT

Function name	**HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**
Function description	Receive an amount of data in interrupt mode.
Parameters	• **husart:** USART handle.

	• **pRxData:** pointer to data buffer. • **Size:** amount of data to be received.
Return values	• **HAL:** status
Notes	• 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 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_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	• **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	• **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_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	• **husart:** USART handle. • **pTxData:** 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 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 the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position). • The USART DMA transmit channel must be configured in order to generate the clock for the slave. • When 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_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 the USART parity is enabled (PCE = 1) the data received contain the parity bit. • 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_DMAPause

Function name	**HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)**
Function description	Pause the DMA Transfer.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_DMAResume

Function name	**HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)**
Function description	Resume the DMA Transfer.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_DMAStop

Function name	**HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)**
Function description	Stop the DMA Transfer.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_Abort

Function name	**HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)**
Function description	Abort ongoing transfers (blocking mode).
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name	**HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **husart:** USART handle.

Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name	**void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)**
Function description	Handle USART interrupt request.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_TxCpltCallback

Function name	**void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)**
Function description	Tx Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_TxHalfCpltCallback

Function name	**void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function description	Tx Half Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_RxCpltCallback

Function name	**void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)**
Function description	Rx Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_RxHalfCpltCallback

Function name	**void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function description	Rx Half Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_TxRxCpltCallback

Function name	**void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)**
Function description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_ErrorCallback

Function name	**void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)**
Function description	USART error callback.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_AbortCpltCallback

Function name	**void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)**
Function description	USART Abort Complete callback.
Parameters	• **husart:** USART handle.
Return values	• **None**

HAL_USART_GetState

Function name	**HAL_USART_StateTypeDef HAL_USART_GetState (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

50.3 USART Firmware driver defines

50.3.1 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 Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None

__HAL_USART_FLUSH_DRREGISTER

Description:

- Flush the USART Data registers.

Parameters:

- __HANDLE__: specifies the USART Handle.

__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_REACK 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_FLAG

Description:

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - USART_CLEAR_PEF
  - USART_CLEAR_FEF
  - USART_CLEAR_NEF
  - USART_CLEAR_OREF
  - USART_CLEAR_IDLEF
  - USART_CLEAR_TCF
  - USART_CLEAR_CTSF

Return value:

- None

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_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:

- 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:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
  - USART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

USART Exported Types

HAL_USART_ERROR_NONE	No error
HAL_USART_ERROR_PE	Parity error

HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

USART Flags

USART_FLAG_REACK	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_LBDF	USART LIN break detection 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
USART_LASTBIT_ENABLE	USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX	RX mode
USART_MODE_TX	TX mode
USART_MODE_TX_RX	RX and TX mode

USART Parity

USART_PARITY_NONE	No parity
USART_PARITY_EVEN	Even parity
USART_PARITY_ODD	Odd parity

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
USART_TXDATA_FLUSH_REQUEST	Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_0_5	USART frame with 0.5 stop bit
USART_STOPBITS_1	USART frame with 1 stop bit
USART_STOPBITS_1_5	USART frame with 1.5 stop bits
USART_STOPBITS_2	USART frame with 2 stop bits

51 HAL USART Extension Driver

51.1 USARTEx Firmware driver defines

51.1.1 USARTEx

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

52 HAL WWDG Generic Driver

52.1 WWDG Firmware driver registers structures

52.1.1 WWDG_InitTypeDef

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 Interupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

52.1.2 WWDG_HandleTypeDef

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***

Field Documentation

- ***WWDG_TypeDef* WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef WWDG_HandleTypeDef::Init***
  WWDG required parameters

52.2 WWDG Firmware driver API description

52.2.1 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) = PCLK1 / (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 @32 MHz(PCLK1): ~128.0 us / ~65.54 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

52.2.2 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.

52.2.3 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()***

52.2.4 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()***

52.2.5 Detailed description of functions

HAL_WWDG_Init

Function name	**HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)**
Function description	Initialize the WWDG according to the specified.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL:** status

HAL_WWDG_MspInit

Function name	**void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)**
Function description	Initialize the WWDG MSP.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **None**
Notes	• When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_Refresh

Function name	**HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)**
Function description	Refresh the WWDG.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL:** status

HAL_WWDG_IRQHandler

Function name	**void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)**
Function description	Handle WWDG interrupt request.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **None**
Notes	• The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name	**void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)**
Function description	WWDG Early Wakeup callback.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **None**

52.3 WWDG Firmware driver defines

52.3.1 WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE	EWI Disable
WWDG_EWI_ENABLE	EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
  – WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
  – WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

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

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  – WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or

RESET).

__HAL_WWDG_CLEAR_FLAG	**Description:** • Clear the WWDG's pending flags. **Parameters:** • __HANDLE__: WWDG handle • __FLAG__: specifies the flag to clear. This parameter can be one of the following values: – WWDG_FLAG_EWIF: Early wakeup interrupt flag **Return value:** • None
__HAL_WWDG_GET_IT_SOURCE	**Description:** • Check whether the specified WWDG interrupt source is enabled or not. **Parameters:** • __HANDLE__: WWDG Handle. • __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values: – WWDG_IT_EWI: Early Wakeup Interrupt **Return value:** • state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF	Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI	Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1	WWDG counter clock = (PCLK1/4096)/1
WWDG_PRESCALER_2	WWDG counter clock = (PCLK1/4096)/2
WWDG_PRESCALER_4	WWDG counter clock = (PCLK1/4096)/4
WWDG_PRESCALER_8	WWDG counter clock = (PCLK1/4096)/8

53 LL ADC Generic Driver

53.1 ADC Firmware driver registers structures

53.1.1 LL_ADC_CommonInitTypeDef

Data Fields

- ***uint32_t CommonClock***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::CommonClock***
  Set parameter common to several ADC: Clock source and prescaler. This parameter can be a value of ***ADC_LL_EC_COMMON_CLOCK_SOURCE***This feature can be modified afterwards using unitary function **LL_ADC_SetCommonClock()**.

53.1.2 LL_ADC_InitTypeDef

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()**.

53.1.3 LL_ADC_REG_InitTypeDef

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()**.

53.2 ADC Firmware driver API description

53.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (ADC_TypeDef * ADCx, uint32_t Register)**
Function description	Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.
Parameters	• **ADCx:** ADC instance • **Register:** This parameter can be one of the following values:

- LL_ADC_DMA_REG_REGULAR_DATA

Return values
- **ADC:** register address

Notes
- These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
- For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter.

Reference Manual to LL API cross reference:
- DR DATA LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonClock

Function name
__STATIC_INLINE void LL_ADC_SetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t CommonClock)

Function description
Set parameter common to several ADC: Clock source and prescaler.

Parameters
- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **CommonClock:** This parameter can be one of the following values: (1) ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous. (refer to function LL_ADC_SetClock()).
  - LL_ADC_CLOCK_ASYNC_DIV1 (1)
  - LL_ADC_CLOCK_ASYNC_DIV2 (1)
  - LL_ADC_CLOCK_ASYNC_DIV4 (1)
  - LL_ADC_CLOCK_ASYNC_DIV6 (1)
  - LL_ADC_CLOCK_ASYNC_DIV8 (1)
  - LL_ADC_CLOCK_ASYNC_DIV10 (1)
  - LL_ADC_CLOCK_ASYNC_DIV12 (1)
  - LL_ADC_CLOCK_ASYNC_DIV16 (1)
  - LL_ADC_CLOCK_ASYNC_DIV32 (1)
  - LL_ADC_CLOCK_ASYNC_DIV64 (1)
  - LL_ADC_CLOCK_ASYNC_DIV128 (1)
  - LL_ADC_CLOCK_ASYNC_DIV256 (1)

Return values
- **None**

Notes
- 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 LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:	• CCR PRESC LL_ADC_SetCommonClock

LL_ADC_GetCommonClock

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get parameter common to several ADC: Clock source and prescaler.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **Returned:** value can be one of the following values: (1) ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous. (refer to function LL_ADC_SetClock()). – LL_ADC_CLOCK_ASYNC_DIV1 (1) – LL_ADC_CLOCK_ASYNC_DIV2 (1) – LL_ADC_CLOCK_ASYNC_DIV4 (1) – LL_ADC_CLOCK_ASYNC_DIV6 (1) – LL_ADC_CLOCK_ASYNC_DIV8 (1) – LL_ADC_CLOCK_ASYNC_DIV10 (1) – LL_ADC_CLOCK_ASYNC_DIV12 (1) – LL_ADC_CLOCK_ASYNC_DIV16 (1) – LL_ADC_CLOCK_ASYNC_DIV32 (1) – LL_ADC_CLOCK_ASYNC_DIV64 (1) – LL_ADC_CLOCK_ASYNC_DIV128 (1) – LL_ADC_CLOCK_ASYNC_DIV256 (1)
Reference Manual to LL API cross reference:	• CCR PRESC LL_ADC_GetCommonClock

LL_ADC_SetCommonFrequencyMode

Function name	**__STATIC_INLINE void LL_ADC_SetCommonFrequencyMode (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Resolution)**
Function description	Set parameter common to several ADC: Clock low frequency mode.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE()) • **Resolution:** This parameter can be one of the following values:

	– LL_ADC_CLOCK_FREQ_MODE_HIGH – LL_ADC_CLOCK_FREQ_MODE_LOW
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:	• CCR LFMEN LL_ADC_SetCommonFrequencyMode

LL_ADC_GetCommonFrequencyMode

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCommonFrequencyMode (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get parameter common to several ADC: Clock low frequency mode.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **Returned:** value can be one of the following values: – LL_ADC_CLOCK_FREQ_MODE_HIGH – LL_ADC_CLOCK_FREQ_MODE_LOW
Reference Manual to LL API cross reference:	• CCR LFMEN LL_ADC_GetCommonFrequencyMode

LL_ADC_SetCommonPathInternalCh

Function name	**__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)**
Function description	Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE()) • **PathInternal:** This parameter can be a combination of the following values: (*) value not defined in all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. – LL_ADC_PATH_INTERNAL_NONE – LL_ADC_PATH_INTERNAL_VREFINT – LL_ADC_PATH_INTERNAL_TEMPSENSOR – LL_ADC_PATH_INTERNAL_VLCD (*)
Return values	• **None**
Notes	• One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT |

	LL_ADC_PATH_INTERNAL_TEMPSENSOR) • Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US. • ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet. • On this STM32 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 LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().
Reference Manual to LL API cross reference:	• CCR VREFEN LL_ADC_SetCommonPathInternalCh • CCR TSEN LL_ADC_SetCommonPathInternalCh • CCR VLCDEN LL_ADC_SetCommonPathInternalCh

LL_ADC_GetCommonPathInternalCh

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **Returned:** value can be a combination of the following values: (*) value not defined in all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. – LL_ADC_PATH_INTERNAL_NONE – LL_ADC_PATH_INTERNAL_VREFINT – LL_ADC_PATH_INTERNAL_TEMPSENSOR – LL_ADC_PATH_INTERNAL_VLCD (*)
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 VLCDEN 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: (1) Asynchronous clock prescaler can be configured using function LL_ADC_SetCommonClock(). – LL_ADC_CLOCK_SYNC_PCLK_DIV4 – LL_ADC_CLOCK_SYNC_PCLK_DIV2 – LL_ADC_CLOCK_SYNC_PCLK_DIV1 (2) – LL_ADC_CLOCK_ASYNC (1) • (2) Caution: This parameter has some clock ratio constraints: This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). Refer to reference manual.
Return values	• **None**
Notes	• On this STM32 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: (1) Asynchronous clock prescaler can be retrieved using function LL_ADC_GetCommonClock(). – LL_ADC_CLOCK_SYNC_PCLK_DIV4 – LL_ADC_CLOCK_SYNC_PCLK_DIV2 – LL_ADC_CLOCK_SYNC_PCLK_DIV1 (2) – LL_ADC_CLOCK_ASYNC (1) • (2) Caution: This parameter has some clock ratio constraints: This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). Refer to reference manual.
Reference Manual to LL API cross reference:	• CFGR2 CKMODE LL_ADC_GetClock

LL_ADC_SetCalibrationFactor

Function name	**__STATIC_INLINE void LL_ADC_SetCalibrationFactor (ADC_TypeDef * ADCx, uint32_t CalibrationFactor)**
Function description	Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters	• **ADCx:** ADC instance • **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x7F
Return values	• **None**
Notes	• This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration(). • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.
Reference Manual to LL API cross reference:	• CALFACT CALFACT LL_ADC_SetCalibrationFactor

LL_ADC_GetCalibrationFactor

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCalibrationFactor (ADC_TypeDef * ADCx)**
Function description	Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x7F
Notes	• Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().
Reference Manual to LL API cross reference:	• CALFACT CALFACT LL_ADC_GetCalibrationFactor

LL_ADC_SetResolution

Function name	**__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)**
Function description	Set ADC resolution.
Parameters	• **ADCx:** ADC instance • **Resolution:** This parameter can be one of the following values: – LL_ADC_RESOLUTION_12B – LL_ADC_RESOLUTION_10B – LL_ADC_RESOLUTION_8B – LL_ADC_RESOLUTION_6B
Return values	• **None**
Notes	• On this STM32 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	• **ADCx:** ADC instance • **DataAlignment:** This parameter can be one of the following values: – LL_ADC_DATA_ALIGN_RIGHT – LL_ADC_DATA_ALIGN_LEFT
Return values	• **None**
Notes	• Refer to reference manual for alignments formats dependencies to ADC resolutions. • On this STM32 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 ALIGN LL_ADC_SetDataAlignment

LL_ADC_GetDataAlignment

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetDataAlignment (ADC_TypeDef * ADCx)**
Function description	Get ADC conversion data alignment.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_DATA_ALIGN_RIGHT – LL_ADC_DATA_ALIGN_LEFT
Notes	• Refer to reference manual for alignments formats dependencies to ADC resolutions.
Reference Manual to	• CFGR1 ALIGN LL_ADC_GetDataAlignment

LL API cross reference:

LL_ADC_SetLowPowerMode

Function name	**__STATIC_INLINE void LL_ADC_SetLowPowerMode (ADC_TypeDef * ADCx, uint32_t LowPowerMode)**
Function description	Set ADC low power mode.
Parameters	• **ADCx:** ADC instance • **LowPowerMode:** This parameter can be one of the following values: – LL_ADC_LP_MODE_NONE – LL_ADC_LP_AUTOWAIT – LL_ADC_LP_AUTOPOWEROFF – LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF
Return values	• **None**
Notes	• Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: 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_3CYCLES_5 – LL_ADC_SAMPLINGTIME_7CYCLES_5 – LL_ADC_SAMPLINGTIME_12CYCLES_5 – LL_ADC_SAMPLINGTIME_19CYCLES_5 – LL_ADC_SAMPLINGTIME_39CYCLES_5 – LL_ADC_SAMPLINGTIME_79CYCLES_5 – LL_ADC_SAMPLINGTIME_160CYCLES_5
Return values	• **None**
Notes	• Unit: ADC clock cycles. • On this STM32 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_3CYCLES_5 – LL_ADC_SAMPLINGTIME_7CYCLES_5 – LL_ADC_SAMPLINGTIME_12CYCLES_5 – LL_ADC_SAMPLINGTIME_19CYCLES_5 – LL_ADC_SAMPLINGTIME_39CYCLES_5 – LL_ADC_SAMPLINGTIME_79CYCLES_5 – LL_ADC_SAMPLINGTIME_160CYCLES_5
Notes	• Unit: ADC clock cycles. • On this STM32 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_GetSamplingTimeCommonChannels

LL_ADC_REG_SetTriggerSource

Function name	**__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)**
Function description	Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	• **ADCx:** ADC instance • **TriggerSource:** This parameter can be one of the following values: (*) value not defined in all devices – LL_ADC_REG_TRIG_SOFTWARE – LL_ADC_REG_TRIG_EXT_TIM6_TRGO – LL_ADC_REG_TRIG_EXT_TIM21_CH2 – LL_ADC_REG_TRIG_EXT_TIM2_TRGO – LL_ADC_REG_TRIG_EXT_TIM2_CH4 – LL_ADC_REG_TRIG_EXT_TIM22_TRGO – LL_ADC_REG_TRIG_EXT_TIM2_CH3 (*) – LL_ADC_REG_TRIG_EXT_TIM3_TRGO – LL_ADC_REG_TRIG_EXT_EXTI_LINE11
Return values	• **None**
Notes	• 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:	• CFGR1 EXTSEL LL_ADC_REG_SetTriggerSource • CFGR1 EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices – LL_ADC_REG_TRIG_SOFTWARE – LL_ADC_REG_TRIG_EXT_TIM6_TRGO – LL_ADC_REG_TRIG_EXT_TIM21_CH2 – LL_ADC_REG_TRIG_EXT_TIM2_TRGO – LL_ADC_REG_TRIG_EXT_TIM2_CH4 – LL_ADC_REG_TRIG_EXT_TIM22_TRGO – LL_ADC_REG_TRIG_EXT_TIM2_CH3 (*) – LL_ADC_REG_TRIG_EXT_TIM3_TRGO – LL_ADC_REG_TRIG_EXT_EXTI_LINE11
Notes	• To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart. • Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	• CFGR1 EXTSEL LL_ADC_REG_GetTriggerSource • CFGR1 EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (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	• **Returned:** value can be one of the following values: – LL_ADC_REG_TRIG_EXT_RISING – LL_ADC_REG_TRIG_EXT_FALLING – LL_ADC_REG_TRIG_EXT_RISINGFALLING
Notes	• Applicable only for trigger source set to external trigger.
Reference Manual to LL API cross reference:	• CFGR1 EXTEN LL_ADC_REG_GetTriggerEdge

LL_ADC_REG_SetSequencerScanDirection

Function name	**__STATIC_INLINE void LL_ADC_REG_SetSequencerScanDirection (ADC_TypeDef * ADCx, uint32_t ScanDirection)**
Function description	Set ADC group regular sequencer scan direction.
Parameters	• **ADCx:** ADC instance • **ScanDirection:** This parameter can be one of the following values: – LL_ADC_REG_SEQ_SCAN_DIR_FORWARD – LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD
Return values	• **None**
Notes	• On 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:	• CFGR1 SCANDIR LL_ADC_REG_SetSequencerScanDirection

LL_ADC_REG_GetSequencerScanDirection

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerScanDirection (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular sequencer scan direction.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_SEQ_SCAN_DIR_FORWARD – LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD
Notes	• On some other STM32 families, this setting is not available and the default scan direction is forward.
Reference Manual to LL API cross reference:	• CFGR1 SCANDIR LL_ADC_REG_GetSequencerScanDirection

LL_ADC_REG_SetSequencerDiscont

Function name	**__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)**
Function description	Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	• **ADCx:** ADC instance • **SeqDiscont:** This parameter can be one of the following values: – LL_ADC_REG_SEQ_DISCONT_DISABLE

- LL_ADC_REG_SEQ_DISCONT_1RANK

Return values	• **None**
Notes	• It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode. • On this STM32 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 DISCEN LL_ADC_REG_SetSequencerDiscont •

LL_ADC_REG_GetSequencerDiscont

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_SEQ_DISCONT_DISABLE – LL_ADC_REG_SEQ_DISCONT_1RANK
Reference Manual to LL API cross reference:	• CFGR1 DISCEN LL_ADC_REG_GetSequencerDiscont •

LL_ADC_REG_SetSequencerChannels

Function name	**__STATIC_INLINE void LL_ADC_REG_SetSequencerChannels (ADC_TypeDef * ADCx, uint32_t Channel)**
Function description	Set ADC group regular sequence: channel on rank corresponding to channel number.
Parameters	• **ADCx:** ADC instance • **Channel:** This parameter can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. – 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 (1)
- LL_ADC_CHANNEL_17
- LL_ADC_CHANNEL_18
- LL_ADC_CHANNEL_VREFINT
- LL_ADC_CHANNEL_TEMPSENSOR
- LL_ADC_CHANNEL_VLCD (1)

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:
- CHSELR CHSEL0 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL1 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL2 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL3 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL4 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL5 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL6 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL7 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL8 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL9 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL10 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL11 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL12 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL13 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL14 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL15 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL16 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL17 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL18 LL_ADC_REG_SetSequencerChannels

LL_ADC_REG_SetSequencerChAdd

Function name	**__STATIC_INLINE void LL_ADC_REG_SetSequencerChAdd (ADC_TypeDef * ADCx, uint32_t Channel)**
Function description	Add channel to ADC group regular sequence: channel on rank corresponding to channel number.
Parameters	• **ADCx:** ADC instance • **Channel:** This parameter can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. – 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 (1) – LL_ADC_CHANNEL_17 – LL_ADC_CHANNEL_18 – LL_ADC_CHANNEL_VREFINT – LL_ADC_CHANNEL_TEMPSENSOR – LL_ADC_CHANNEL_VLCD (1)
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:

- CHSELR CHSEL0 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL1 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL2 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL3 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL4 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL5 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL6 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL7 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL8 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL9 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL10 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL11 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL12 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL13 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL14 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL15 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL16 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL17 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL18 LL_ADC_REG_SetSequencerChAdd

LL_ADC_REG_SetSequencerChRem

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerChRem (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Remove channel to ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx.
  - 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 (1)
  - LL_ADC_CHANNEL_17

- – LL_ADC_CHANNEL_18
- – LL_ADC_CHANNEL_VREFINT
- – LL_ADC_CHANNEL_TEMPSENSOR
- – LL_ADC_CHANNEL_VLCD (1)

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:
- CHSELR CHSEL0 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL1 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL2 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL3 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL4 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL5 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL6 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL7 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL8 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL9 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL10 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL11 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL12 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL13 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL14 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL15 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL16 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL17 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL18 LL_ADC_REG_SetSequencerChRem

LL_ADC_REG_GetSequencerChannels

Function name
__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerChannels (ADC_TypeDef * ADCx)

Function description
Get ADC group regular sequence: channel on rank corresponding

to channel number.

Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. – 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 (1) – LL_ADC_CHANNEL_17 – LL_ADC_CHANNEL_18 – LL_ADC_CHANNEL_VREFINT – LL_ADC_CHANNEL_TEMPSENSOR – LL_ADC_CHANNEL_VLCD (1)
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	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_CONV_SINGLE – LL_ADC_REG_CONV_CONTINUOUS

Notes	• Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
Reference Manual to LL API cross reference:	• CFGR1 CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDMATransfer

Function name	**__STATIC_INLINE void LL_ADC_REG_SetDMATransfer (ADC_TypeDef * ADCx, uint32_t DMATransfer)**
Function description	Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.
Parameters	• **ADCx:** ADC instance • **DMATransfer:** This parameter can be one of the following values: – LL_ADC_REG_DMA_TRANSFER_NONE – LL_ADC_REG_DMA_TRANSFER_LIMITED – LL_ADC_REG_DMA_TRANSFER_UNLIMITED
Return values	• **None**
Notes	• If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. • If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled). • To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr(). • On this STM32 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 DMAEN LL_ADC_REG_SetDMATransfer • CFGR1 DMACFG LL_ADC_REG_SetDMATransfer

LL_ADC_REG_GetDMATransfer

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.
Parameters	• **ADCx:** ADC instance

Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_DMA_TRANSFER_NONE – LL_ADC_REG_DMA_TRANSFER_LIMITED – LL_ADC_REG_DMA_TRANSFER_UNLIMITED
Notes	• If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. • If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled). • To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().
Reference Manual to LL API cross reference:	• CFGR1 DMAEN LL_ADC_REG_GetDMATransfer • CFGR1 DMACFG LL_ADC_REG_GetDMATransfer

LL_ADC_REG_SetOverrun

Function name	**__STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)**
Function description	Set ADC group regular behavior in case of overrun: data preserved or overwritten.
Parameters	• **ADCx:** ADC instance • **Overrun:** This parameter can be one of the following values: – LL_ADC_REG_OVR_DATA_PRESERVED – LL_ADC_REG_OVR_DATA_OVERWRITTEN
Return values	• **None**
Notes	• Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten. • On this STM32 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 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	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_OVR_DATA_PRESERVED – LL_ADC_REG_OVR_DATA_OVERWRITTEN
Reference Manual to LL API cross reference:	• CFGR1 OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_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	• **ADCx:** ADC instance • **AWDChannelGroup:** This parameter can be one of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. – 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 (1) – LL_ADC_AWD_CHANNEL_17_REG – LL_ADC_AWD_CHANNEL_18_REG – LL_ADC_AWD_CH_VREFINT_REG – LL_ADC_AWD_CH_TEMPSENSOR_REG

	– LL_ADC_AWD_CH_VLCD_REG (1)
Return values	• **None**
Notes	• Once monitored channels are selected, analog watchdog is enabled. • In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP(). • On this STM32 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_SetAnalogWDMonitChannels • CFGR1 AWDSGL LL_ADC_SetAnalogWDMonitChannels • CFGR1 AWDEN LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx)**
Function description	Get ADC analog watchdog monitored channel.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_AWD_DISABLE – LL_ADC_AWD_ALL_CHANNELS_REG – LL_ADC_AWD_CHANNEL_0_REG – LL_ADC_AWD_CHANNEL_1_REG – LL_ADC_AWD_CHANNEL_2_REG – LL_ADC_AWD_CHANNEL_3_REG – LL_ADC_AWD_CHANNEL_4_REG – LL_ADC_AWD_CHANNEL_5_REG – LL_ADC_AWD_CHANNEL_6_REG – LL_ADC_AWD_CHANNEL_7_REG – LL_ADC_AWD_CHANNEL_8_REG – LL_ADC_AWD_CHANNEL_9_REG – LL_ADC_AWD_CHANNEL_10_REG – LL_ADC_AWD_CHANNEL_11_REG – LL_ADC_AWD_CHANNEL_12_REG – LL_ADC_AWD_CHANNEL_13_REG – LL_ADC_AWD_CHANNEL_14_REG – LL_ADC_AWD_CHANNEL_15_REG – LL_ADC_AWD_CHANNEL_16_REG – LL_ADC_AWD_CHANNEL_17_REG – LL_ADC_AWD_CHANNEL_18_REG

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=0xFFF • **AWDThresholdLowValue:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None**
Notes	• If value of only one threshold high or low must be set, use function LL_ADC_SetAnalogWDThresholds(). • In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(). • On this STM32 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	• **ADCx:** ADC instance • **AWDThresholdsHighLow:** This parameter can be one of the following values: – LL_ADC_AWD_THRESHOLD_HIGH – LL_ADC_AWD_THRESHOLD_LOW • **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None**
Notes	• If values of both thresholds high or low must be set, use function LL_ADC_ConfigAnalogWDThresholds(). • In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(). • On this STM32 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_SetAnalogWDThresholds • TR LT LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow)**
Function description	Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.
Parameters	• **ADCx:** ADC instance • **AWDThresholdsHighLow:** This parameter can be one of the following values: – LL_ADC_AWD_THRESHOLD_HIGH

	– LL_ADC_AWD_THRESHOLD_LOW – LL_ADC_AWD_THRESHOLDS_HIGH_LOW
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• If raw data with ADC thresholds high and low is retrieved, the data of each threshold high or low can be isolated using helper macro: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW(). • In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().
Reference Manual to LL API cross reference:	• TR HT LL_ADC_GetAnalogWDThresholds • TR LT LL_ADC_GetAnalogWDThresholds

LL_ADC_SetOverSamplingScope

Function name	**__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)**
Function description	Set ADC oversampling scope.
Parameters	• **ADCx:** ADC instance • **OvsScope:** This parameter can be one of the following values: – LL_ADC_OVS_DISABLE – LL_ADC_OVS_GRP_REGULAR_CONTINUED
Return values	• **None**
Notes	• On this STM32 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:	• CFGR2 OVSE LL_ADC_SetOverSamplingScope

LL_ADC_GetOverSamplingScope

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling scope.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_OVS_DISABLE – LL_ADC_OVS_GRP_REGULAR_CONTINUED
Reference Manual to LL API cross reference:	• CFGR2 OVSE LL_ADC_GetOverSamplingScope

LL_ADC_SetOverSamplingDiscont

Function name	**__STATIC_INLINE void LL_ADC_SetOverSamplingDiscont (ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)**
Function description	Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.
Parameters	• **ADCx:** ADC instance • **OverSamplingDiscont:** This parameter can be one of the following values: – LL_ADC_OVS_REG_CONT – LL_ADC_OVS_REG_DISCONT
Return values	• **None**
Notes	• Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger) • On this STM32 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:	• CFGR2 TOVS LL_ADC_SetOverSamplingDiscont

LL_ADC_GetOverSamplingDiscont

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_OVS_REG_CONT – LL_ADC_OVS_REG_DISCONT
Notes	• Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)
Reference Manual to LL API cross reference:	• CFGR2 TOVS LL_ADC_GetOverSamplingDiscont

LL_ADC_ConfigOverSamplingRatioShift

Function name	**__STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)**
Function description	Set ADC oversampling.
Parameters	• **ADCx:** ADC instance

	• **Ratio:** This parameter can be one of the following values: – LL_ADC_OVS_RATIO_2 – LL_ADC_OVS_RATIO_4 – LL_ADC_OVS_RATIO_8 – LL_ADC_OVS_RATIO_16 – LL_ADC_OVS_RATIO_32 – LL_ADC_OVS_RATIO_64 – LL_ADC_OVS_RATIO_128 – LL_ADC_OVS_RATIO_256 • **Shift:** This parameter can be one of the following values: – LL_ADC_OVS_SHIFT_NONE – LL_ADC_OVS_SHIFT_RIGHT_1 – LL_ADC_OVS_SHIFT_RIGHT_2 – LL_ADC_OVS_SHIFT_RIGHT_3 – LL_ADC_OVS_SHIFT_RIGHT_4 – LL_ADC_OVS_SHIFT_RIGHT_5 – LL_ADC_OVS_SHIFT_RIGHT_6 – LL_ADC_OVS_SHIFT_RIGHT_7 – LL_ADC_OVS_SHIFT_RIGHT_8
Return values	• **None**
Notes	• This function set the 2 items of oversampling configuration: ratioshift • On this STM32 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:	• CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift • CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift

LL_ADC_GetOverSamplingRatio

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling ratio.
Parameters	• **ADCx:** ADC instance
Return values	• **Ratio:** This parameter can be one of the following values: – LL_ADC_OVS_RATIO_2 – LL_ADC_OVS_RATIO_4 – LL_ADC_OVS_RATIO_8 – LL_ADC_OVS_RATIO_16 – LL_ADC_OVS_RATIO_32 – LL_ADC_OVS_RATIO_64 – LL_ADC_OVS_RATIO_128 – LL_ADC_OVS_RATIO_256
Reference Manual to LL API cross reference:	• CFGR2 OVSR LL_ADC_GetOverSamplingRatio

LL_ADC_GetOverSamplingShift

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling shift.
Parameters	• **ADCx:** ADC instance
Return values	• **Shift:** This parameter can be one of the following values: – LL_ADC_OVS_SHIFT_NONE – LL_ADC_OVS_SHIFT_RIGHT_1 – LL_ADC_OVS_SHIFT_RIGHT_2 – LL_ADC_OVS_SHIFT_RIGHT_3 – LL_ADC_OVS_SHIFT_RIGHT_4 – LL_ADC_OVS_SHIFT_RIGHT_5 – LL_ADC_OVS_SHIFT_RIGHT_6 – LL_ADC_OVS_SHIFT_RIGHT_7 – LL_ADC_OVS_SHIFT_RIGHT_8
Reference Manual to LL API cross reference:	• CFGR2 OVSS LL_ADC_GetOverSamplingShift

LL_ADC_EnableInternalRegulator

Function name	**__STATIC_INLINE void LL_ADC_EnableInternalRegulator (ADC_TypeDef * ADCx)**
Function description	Enable ADC instance internal voltage regulator.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Notes	• On this STM32 serie, there are three possibilities to enable the voltage regulator: by enabling it manually using function LL_ADC_EnableInternalRegulator().by launching a calibration using function LL_ADC_StartCalibration().by enabling the ADC using function LL_ADC_Enable(). • On this STM32 serie, after ADC internal voltage regulator enable, a delay for ADC internal voltage regulator stabilization is required before performing a ADC calibration or ADC enable. Refer to device datasheet, parameter "tUP_LDO". Refer to literal LL_ADC_DELAY_INTERNAL_REGUL_STAB_US. • 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 ADVREGEN LL_ADC_EnableInternalRegulator

LL_ADC_DisableInternalRegulator

Function name	**__STATIC_INLINE void LL_ADC_DisableInternalRegulator (ADC_TypeDef * ADCx)**
Function description	Disable ADC internal voltage regulator.

Parameters	• **ADCx:** ADC instance
Return values	• **None**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
Reference Manual to LL API cross reference:	• CR ADVREGEN LL_ADC_DisableInternalRegulator

LL_ADC_IsInternalRegulatorEnabled

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (ADC_TypeDef * ADCx)**
Function description	Get the selected ADC instance internal voltage regulator state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** internal regulator is disabled, 1: internal regulator is enabled.
Reference Manual to LL API cross reference:	• CR ADVREGEN LL_ADC_IsInternalRegulatorEnabled

LL_ADC_Enable

Function name	**__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)**
Function description	Enable the selected ADC instance.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Notes	• On this STM32 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:	• CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name	**__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)**
Function description	Disable the selected ADC instance.

Parameters	• **ADCx:** ADC instance
Return values	• **None**
Notes	• On this STM32 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:	• CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabled (ADC_TypeDef * ADCx)**
Function description	Get the selected ADC instance enable state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** ADC is disabled, 1: ADC is enabled.
Notes	• On this STM32 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 transfered 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	• CR ADSTART LL_ADC_REG_StartConversion

reference:

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=0xFFF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	• DR DATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadConversionData10

Function name	**__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for ADC resolution 10 bits.
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0x3FF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	• DR DATA LL_ADC_REG_ReadConversionData10

LL_ADC_REG_ReadConversionData8

Function name	**__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for ADC resolution 8 bits.

Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	• DR DATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadConversionData6

Function name	**__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for ADC resolution 6 bits.
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x3F
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	• DR DATA LL_ADC_REG_ReadConversionData6

LL_ADC_IsActiveFlag_ADRDY

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (ADC_TypeDef * ADCx)**
Function description	Get flag ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Notes	• On this STM32 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_IsActiveFlag_EOCAL

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOCAL (ADC_TypeDef * ADCx)**
Function description	Get flag ADC end of calibration.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR EOCAL LL_ADC_IsActiveFlag_EOCAL

LL_ADC_ClearFlag_ADRDY

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Notes	• On this STM32 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	**__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group regular end of sequence conversions.

Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR EOSEQ LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group regular overrun.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group regular end of sampling phase.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_AWD1

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR AWD LL_ADC_ClearFlag_AWD1

LL_ADC_ClearFlag_EOCAL

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_EOCAL (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC end of calibration.
Parameters	• **ADCx:** ADC instance

Return values	• **None**
Reference Manual to LL API cross reference:	• ISR EOCAL LL_ADC_ClearFlag_EOCAL

LL_ADC_EnableIT_ADRDY

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)**
Function description	Enable ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group regular end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EOSEQIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)**
Function description	Enable ADC group regular interruption overrun.
Parameters	• **ADCx:** ADC instance

Return values	• **None**
Reference Manual to LL API cross reference:	• IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group regular end of sampling.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_AWD1

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER AWDIE LL_ADC_EnableIT_AWD1

LL_ADC_EnableIT_EOCAL

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOCAL (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC end of calibration.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EOCALIE LL_ADC_EnableIT_EOCAL

LL_ADC_DisableIT_ADRDY

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **None**

Reference Manual to LL API cross reference:	• IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EOSEQIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular overrun.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of sampling.
Parameters	• **ADCx:** ADC instance
Return values	• **None**

Reference Manual to LL API cross reference:	• IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_AWD1

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER AWDIE LL_ADC_DisableIT_AWD1

LL_ADC_DisableIT_EOCAL

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOCAL (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC end of calibration.
Parameters	• **ADCx:** ADC instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EOCALIE LL_ADC_DisableIT_EOCAL

LL_ADC_IsEnabledIT_ADRDY

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER EOSEQIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (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_IsEnabledIT_EOCAL

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOCAL (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC end of calibration (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER EOCALIE LL_ADC_IsEnabledIT_EOCAL

LL_ADC_CommonDeInit

Function name	**ErrorStatus LL_ADC_CommonDeInit (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC common registers are de-initialized – ERROR: not applicable
Notes	• This function is performing a hard reset, using high level clock source RCC ADC reset.

LL_ADC_CommonInit

Function name	**ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)**
Function description	Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE()) • **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC common registers are initialized

	– ERROR: ADC common registers are not initialized
Notes	• The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name	**void LL_ADC_CommonStructInit (LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)**
Function description	Set each LL_ADC_CommonInitTypeDef field to default value.
Parameters	• **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.
Return values	• **None**

LL_ADC_DeInit

Function name	**ErrorStatus LL_ADC_DeInit (ADC_TypeDef * ADCx)**
Function description	De-initialize registers of the selected ADC instance to their default reset values.
Parameters	• **ADCx:** ADC instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC registers are de-initialized – ERROR: ADC registers are not de-initialized
Notes	• To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit(). • If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Refer to function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name	**ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_InitStruct)**
Function description	Initialize some features of ADC instance.
Parameters	• **ADCx:** ADC instance • **ADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC registers are initialized – ERROR: ADC registers are not initialized
Notes	• These parameters have an impact on ADC scope: ADC instance. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance . • The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and

compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.

- After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular sequencer: 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	• **ADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.
Return values	• **None**

LL_ADC_REG_Init

Function name	**ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)**
Function description	Initialize some features of ADC group regular.
Parameters	• **ADCx:** ADC instance • **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC registers are initialized – ERROR: ADC registers are not initialized
Notes	• These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG"). • The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC

state.

- 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 LL_ADC_REG_SetSequencerChannels();Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name	**void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)**
Function description	Set each LL_ADC_REG_InitTypeDef field to default value.
Parameters	• **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.
Return values	• **None**

53.3 ADC Firmware driver defines

53.3.1 ADC

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE	ADC analog watchdog monitoring disabled
LL_ADC_AWD_ALL_CHANNELS_REG	ADC analog watchdog monitoring of all channels, converted by group regular only
LL_ADC_AWD_CHANNEL_0_REG	ADC analog watchdog monitoring of ADC 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_17_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group regular only
LL_ADC_AWD_CHANNEL_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_VREFINT_REG	ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only
LL_ADC_AWD_CH_TEMPSENSOR_REG	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only
LL_ADC_AWD_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_CH_VLCD_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_THRESHOLD_HIGH	ADC analog watchdog threshold high
LL_ADC_AWD_THRESHOLD_LOW	ADC analog watchdog threshold low
LL_ADC_AWD_THRESHOLDS_HIGH_LOW	ADC analog watchdog both thresholds high and low concatenated into the same data

ADC instance - Channel number

LL_ADC_CHANNEL_0	ADC external channel (channel connected to GPIO pin) ADCx_IN0
LL_ADC_CHANNEL_1	ADC external channel (channel connected to GPIO pin) ADCx_IN1
LL_ADC_CHANNEL_2	ADC external channel (channel connected to GPIO pin) ADCx_IN2
LL_ADC_CHANNEL_3	ADC external channel (channel connected to GPIO pin) ADCx_IN3
LL_ADC_CHANNEL_4	ADC external channel (channel connected to GPIO pin) ADCx_IN4
LL_ADC_CHANNEL_5	ADC external channel (channel connected to GPIO pin) ADCx_IN5
LL_ADC_CHANNEL_6	ADC external channel (channel connected to GPIO pin) ADCx_IN6
LL_ADC_CHANNEL_7	ADC external channel (channel connected to GPIO pin) ADCx_IN7
LL_ADC_CHANNEL_8	ADC external channel (channel connected to GPIO pin) ADCx_IN8

LL_ADC_CHANNEL_9	ADC external channel (channel connected to GPIO pin) ADCx_IN9
LL_ADC_CHANNEL_10	ADC external channel (channel connected to GPIO pin) ADCx_IN10
LL_ADC_CHANNEL_11	ADC external channel (channel connected to GPIO pin) ADCx_IN11
LL_ADC_CHANNEL_12	ADC external channel (channel connected to GPIO pin) ADCx_IN12
LL_ADC_CHANNEL_13	ADC external channel (channel connected to GPIO pin) ADCx_IN13
LL_ADC_CHANNEL_14	ADC external channel (channel connected to GPIO pin) ADCx_IN14
LL_ADC_CHANNEL_15	ADC external channel (channel connected to GPIO pin) ADCx_IN15
LL_ADC_CHANNEL_17	ADC external channel (channel connected to GPIO pin) ADCx_IN17
LL_ADC_CHANNEL_18	ADC external channel (channel connected to GPIO pin) ADCx_IN18
LL_ADC_CHANNEL_VREFINT	ADC internal channel connected to VrefInt: Internal voltage reference.
LL_ADC_CHANNEL_TEMPSENSOR	ADC internal channel connected to Temperature sensor.
LL_ADC_CHANNEL_16	ADC external channel (channel connected to GPIO pin) ADCx_IN16
LL_ADC_CHANNEL_VLCD	ADC internal channel connected to Vlcd: Vlcd voltage through a divider ladder of factor 1/4, 1/3 or 1/2 (set by LCD voltage generator biasing), to have Vlcd always below Vdda.

Channel - Sampling time

LL_ADC_SAMPLINGTIME_1CYCLE_5	Sampling time 1.5 ADC clock cycle
LL_ADC_SAMPLINGTIME_3CYCLES_5	Sampling time 3.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_7CYCLES_5	Sampling time 7.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_12CYCLES_5	Sampling time 12.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_19CYCLES_5	Sampling time 19.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_39CYCLES_5	Sampling time 39.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_79CYCLES_5	Sampling time 79.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_160CYCLES_5	Sampling time 160.5 ADC clock cycles

ADC instance - Clock source

LL_ADC_CLOCK_SYNC_PCLK_DIV4	ADC synchronous clock derived from AHB clock divided by 4
LL_ADC_CLOCK_SYNC_PCLK_DIV2	ADC synchronous clock derived from AHB clock divided by 2

LL_ADC_CLOCK_SYNC_PCLK_DIV1	ADC synchronous clock derived from AHB clock not divided
LL_ADC_CLOCK_ASYNC	ADC asynchronous clock. Asynchronous clock prescaler can be configured using function

ADC common - Clock frequency mode

LL_ADC_CLOCK_FREQ_MODE_HIGH	ADC clock mode to high frequency. On STM32L0, ADC clock frequency above 2.8MHz.
LL_ADC_CLOCK_FREQ_MODE_LOW	ADC clock mode to low frequency. On STM32L0, ADC clock frequency below 2.8MHz.

ADC common - Clock source

LL_ADC_CLOCK_ASYNC_DIV1	ADC asynchronous clock without prescaler
LL_ADC_CLOCK_ASYNC_DIV2	ADC asynchronous clock with prescaler division by 2. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV4	ADC asynchronous clock with prescaler division by 4. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV6	ADC asynchronous clock with prescaler division by 6. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV8	ADC asynchronous clock with prescaler division by 8. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV10	ADC asynchronous clock with prescaler division by 10. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV12	ADC asynchronous clock with prescaler division by 12. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV16	ADC asynchronous clock with prescaler division by 16. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV32	ADC asynchronous clock with prescaler division by 32. ADC common clock asynchonous prescaler is

	applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV64	ADC asynchronous clock with prescaler division by 64. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV128	ADC asynchronous clock with prescaler division by 128. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function
LL_ADC_CLOCK_ASYNC_DIV256	ADC asynchronous clock with prescaler division by 256. ADC common clock asynchonous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE	ADC measurement pathes all disabled
LL_ADC_PATH_INTERNAL_VREFINT	ADC measurement path to internal channel VrefInt
LL_ADC_PATH_INTERNAL_TEMPSENSOR	ADC measurement path to internal channel temperature sensor
LL_ADC_PATH_INTERNAL_VLCD	ADC measurement path to internal channel Vlcd

ADC instance - Data alignment

LL_ADC_DATA_ALIGN_RIGHT	ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)
LL_ADC_DATA_ALIGN_LEFT	ADC conversion data alignment: left aligned (aligment on data register MSB bit 15)

ADC flags

LL_ADC_FLAG_ADRDY	ADC flag ADC instance ready
LL_ADC_FLAG_EOC	ADC flag ADC group regular end of unitary conversion
LL_ADC_FLAG_EOS	ADC flag ADC group regular end of sequence conversions
LL_ADC_FLAG_OVR	ADC flag ADC group regular overrun
LL_ADC_FLAG_EOSMP	ADC flag ADC group regular end of sampling phase
LL_ADC_FLAG_AWD1	ADC flag ADC analog watchdog 1
LL_ADC_FLAG_EOCAL	ADC flag end of calibration

ADC instance - Groups

LL_ADC_GROUP_REGULAR	ADC group regular (available on all STM32 devices)

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US	Delay for ADC stabilization time

	(ADC voltage regulator start-up time)
LL_ADC_DELAY_VREFINT_STAB_US	Delay for internal voltage reference stabilization time
LL_ADC_DELAY_TEMPSENSOR_STAB_US	Delay for temperature sensor stabilization time
LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES	Delay required between ADC end of calibration and ADC enable

ADC 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
LL_ADC_IT_EOCAL	ADC interruption ADC end of calibration

ADC instance - Low power mode

LL_ADC_LP_MODE_NONE	No ADC low power mode activated
LL_ADC_LP_AUTOWAIT	ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function
LL_ADC_LP_AUTOPOWEROFF	ADC low power mode auto power-off: the ADC automatically powers-off after a ADC conversion and automatically wakes up when a new ADC conversion is triggered (with startup time between trigger and start of sampling). See description with function
LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF	ADC low power modes auto wait and auto power-off combined. See description with function

Oversampling - Discontinuous mode

LL_ADC_OVS_REG_CONT	ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)
LL_ADC_OVS_REG_DISCONT	ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Ratio

LL_ADC_OVS_RATIO_2	ADC oversampling ratio of 2 (2 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before

	potential shift)
LL_ADC_OVS_RATIO_4	ADC oversampling ratio of 4 (4 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_8	ADC oversampling ratio of 8 (8 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_16	ADC oversampling ratio of 16 (16 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_32	ADC oversampling ratio of 32 (32 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_64	ADC oversampling ratio of 64 (64 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_128	ADC oversampling ratio of 128 (128 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_256	ADC oversampling ratio of 256 (256 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

Oversampling - Oversampling scope

LL_ADC_OVS_DISABLE	ADC oversampling disabled.
LL_ADC_OVS_GRP_REGULAR_CONTINUED	ADC oversampling on conversions of ADC group regular. Literal suffix "continued" is kept for compatibility with other STM32 devices featuring ADC group injected, in this case other oversampling scope parameters are available.

Oversampling - Data shift

LL_ADC_OVS_SHIFT_NONE	ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_1	ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_2	ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_3	ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_4	ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_5	ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_6	ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_7	ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_8	ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE	ADC conversions are performed in single mode: one conversion per trigger
LL_ADC_REG_CONV_CONTINUOUS	ADC conversions are performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRANSFER_NONE	ADC conversions are not transferred by DMA
LL_ADC_REG_DMA_TRANSFER_LIMITED	ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.
LL_ADC_REG_DMA_TRANSFER_UNLIMITED	ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED	ADC group regular behavior in case of overrun: data preserved

LL_ADC_REG_OVR_DATA_OVERWRITTEN	ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE	ADC group regular sequencer discontinuous mode disable
LL_ADC_REG_SEQ_DISCONT_1RANK	ADC group regular sequencer discontinuous mode enable with sequence interruption every rank

ADC group regular - Sequencer 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_SOFTWARE	ADC group regular conversion trigger internal: SW start.
LL_ADC_REG_TRIG_EXT_TIM6_TRGO	ADC group regular conversion trigger from external IP: TIM6 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM21_CH2	ADC group regular conversion trigger from external IP: TIM21 channel 2 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_TIM2_CH4	ADC group regular conversion trigger from external IP: TIM2 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default

	setting).
LL_ADC_REG_TRIG_EXT_TIM22_TRGO	ADC group regular conversion trigger from external IP: TIM22 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM2_CH3	ADC group regular conversion trigger from external IP: TIM2 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM3_TRGO	ADC group regular conversion trigger from external IP: TIM3 TRG0. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_EXTI_LINE11	ADC group regular conversion trigger from external IP: external interrupt line 11. Trigger edge set to rising edge (default setting).

ADC instance - Resolution

LL_ADC_RESOLUTION_12B	ADC resolution 12 bits
LL_ADC_RESOLUTION_10B	ADC resolution 10 bits
LL_ADC_RESOLUTION_8B	ADC resolution 8 bits
LL_ADC_RESOLUTION_6B	ADC resolution 6 bits

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

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

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - 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 (1)
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18

- LL_ADC_CHANNEL_VREFINT
- LL_ADC_CHANNEL_TEMPSENSOR
- LL_ADC_CHANNEL_VLCD (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 (1)
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (2)
  - LL_ADC_CHANNEL_TEMPSENSOR (2)
  - LL_ADC_CHANNEL_VLCD (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_INTERNAL

Description:

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

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - 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 (1)
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR
  - LL_ADC_CHANNEL_VLCD (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_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - 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 (1)
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR
  - LL_ADC_CHANNEL_VLCD (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_INTERNAL_AVAILABLE

Description:

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

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR
  - LL_ADC_CHANNEL_VLCD (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_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - 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 (1)
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (2)
  - LL_ADC_CHANNEL_TEMPSENSOR (2)
  - LL_ADC_CHANNEL_VLCD (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 (1)
- LL_ADC_AWD_CHANNEL_17_REG
- LL_ADC_AWD_CHANNEL_18_REG
- LL_ADC_AWD_CH_VREFINT_REG
- LL_ADC_AWD_CH_TEMPSENSOR_REG
- LL_ADC_AWD_CH_VLCD_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_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

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

Return value:

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

Notes:

- To be used with function LL_ADC_ConfigAnalogWDThresholds() or LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param>, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_value_8_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

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

Return value:

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

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): < threshold_value_6_bits> = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW

Description:

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

Parameters:

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

Return value:

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

Notes:

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

<raw_value_with_both_thresholds>);

__LL_ADC_COMMON_INSTANCE

Description:

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

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

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

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

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

Parameters:

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

Return value:

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

Notes:

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

__LL_ADC_DIGITAL_SCALE

Description:

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

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:

- LL_ADC_RESOLUTION_12B
- LL_ADC_RESOLUTION_10B
- LL_ADC_RESOLUTION_8B
- LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data 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_DATA_RESOLUTION

Description:

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

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

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

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

- LL_ADC_RESOLUTION_6B

Return value:

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

Notes:

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

__LL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

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

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 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_TEMPERATURE

Description:

- 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_TEMPERATURE_TYP_PARAMS

Description:

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

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32L0, 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 STM32L0, refer to device datasheet parameter "V130" (corresponding to TS_CAL2).
- __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 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

54 LL BUS Generic Driver

54.1 BUS Firmware driver API description

54.1.1 Detailed description of functions

LL_AHB1_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable AHB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_MIF – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC (*) – LL_AHB1_GRP1_PERIPH_RNG (*) – LL_AHB1_GRP1_PERIPH_CRYP (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• AHBENR DMAEN LL_AHB1_GRP1_EnableClock • AHBENR MIFEN LL_AHB1_GRP1_EnableClock • AHBENR CRCEN LL_AHB1_GRP1_EnableClock • AHBENR TSCEN LL_AHB1_GRP1_EnableClock • AHBENR RNGEN LL_AHB1_GRP1_EnableClock • AHBENR CRYPEN LL_AHB1_GRP1_EnableClock

LL_AHB1_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if AHB1 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_MIF – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC (*) – LL_AHB1_GRP1_PERIPH_RNG (*) – LL_AHB1_GRP1_PERIPH_CRYP (*)
Return values	• **State:** of Periphs (1 or 0).
Reference Manual to LL API cross reference:	• AHBENR DMAEN LL_AHB1_GRP1_IsEnabledClock • AHBENR MIFEN LL_AHB1_GRP1_IsEnabledClock • AHBENR CRCEN LL_AHB1_GRP1_IsEnabledClock • AHBENR TSCEN LL_AHB1_GRP1_IsEnabledClock • AHBENR RNGEN LL_AHB1_GRP1_IsEnabledClock

- AHBENR CRYPEN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)**
Function description	Disable AHB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_MIF – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC (*) – LL_AHB1_GRP1_PERIPH_RNG (*) – LL_AHB1_GRP1_PERIPH_CRYP (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• AHBENR DMAEN LL_AHB1_GRP1_DisableClock • AHBENR MIFEN LL_AHB1_GRP1_DisableClock • AHBENR CRCEN LL_AHB1_GRP1_DisableClock • AHBENR TSCEN LL_AHB1_GRP1_DisableClock • AHBENR RNGEN LL_AHB1_GRP1_DisableClock • AHBENR CRYPEN LL_AHB1_GRP1_DisableClock

LL_AHB1_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)**
Function description	Force AHB1 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_ALL – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_MIF – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC (*) – LL_AHB1_GRP1_PERIPH_RNG (*) – LL_AHB1_GRP1_PERIPH_CRYP (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• AHBRSTR DMARST LL_AHB1_GRP1_ForceReset • AHBRSTR MIFRST LL_AHB1_GRP1_ForceReset • AHBRSTR CRCRST LL_AHB1_GRP1_ForceReset • AHBRSTR TSCRST LL_AHB1_GRP1_ForceReset • AHBRSTR RNGRST LL_AHB1_GRP1_ForceReset • AHBRSTR CRYPRST LL_AHB1_GRP1_ForceReset

LL_AHB1_GRP1_ReleaseReset

Function name	**__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset**

	(uint32_t Periphs)
Function description	Release AHB1 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_ALL – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_MIF – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC (*) – LL_AHB1_GRP1_PERIPH_RNG (*) – LL_AHB1_GRP1_PERIPH_CRYP (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• AHBRSTR DMARST LL_AHB1_GRP1_ReleaseReset • AHBRSTR MIFRST LL_AHB1_GRP1_ReleaseReset • AHBRSTR CRCRST LL_AHB1_GRP1_ReleaseReset • AHBRSTR TSCRST LL_AHB1_GRP1_ReleaseReset • AHBRSTR RNGRST LL_AHB1_GRP1_ReleaseReset • AHBRSTR CRYPRST LL_AHB1_GRP1_ReleaseReset

LL_AHB1_GRP1_EnableClockSleep

Function name	**__STATIC_INLINE void LL_AHB1_GRP1_EnableClockSleep (uint32_t Periphs)**
Function description	Enable AHB1 peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_MIF – LL_AHB1_GRP1_PERIPH_SRAM – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC (*) – LL_AHB1_GRP1_PERIPH_RNG (*) – LL_AHB1_GRP1_PERIPH_CRYP (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• AHBSMENR DMASMEN LL_AHB1_GRP1_EnableClockSleep • AHBSMENR MIFSMEN LL_AHB1_GRP1_EnableClockSleep • AHBSMENR SRAMSMEN LL_AHB1_GRP1_EnableClockSleep • AHBSMENR CRCSMEN LL_AHB1_GRP1_EnableClockSleep • AHBSMENR TSCSMEN LL_AHB1_GRP1_EnableClockSleep • AHBSMENR RNGSMEN LL_AHB1_GRP1_EnableClockSleep • AHBSMENR CRYPSMEN LL_AHB1_GRP1_EnableClockSleep

LL_AHB1_GRP1_DisableClockSleep

Function name	**__STATIC_INLINE void LL_AHB1_GRP1_DisableClockSleep (uint32_t Periphs)**
Function description	Disable AHB1 peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_MIF – LL_AHB1_GRP1_PERIPH_SRAM – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC (*) – LL_AHB1_GRP1_PERIPH_RNG (*) – LL_AHB1_GRP1_PERIPH_CRYP (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• AHBSMENR DMASMEN LL_AHB1_GRP1_DisableClockSleep • AHBSMENR MIFSMEN LL_AHB1_GRP1_DisableClockSleep • AHBSMENR SRAMSMEN LL_AHB1_GRP1_DisableClockSleep • AHBSMENR CRCSMEN LL_AHB1_GRP1_DisableClockSleep • AHBSMENR TSCSMEN LL_AHB1_GRP1_DisableClockSleep • AHBSMENR RNGSMEN LL_AHB1_GRP1_DisableClockSleep • AHBSMENR CRYPSMEN LL_AHB1_GRP1_DisableClockSleep

LL_APB1_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable APB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM6 (*) – LL_APB1_GRP1_PERIPH_TIM7 (*) – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_LPUART1 – 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_CRS (*)
- – LL_APB1_GRP1_PERIPH_PWR
- – LL_APB1_GRP1_PERIPH_DAC1 (*)
- – LL_APB1_GRP1_PERIPH_I2C3 (*)
- – LL_APB1_GRP1_PERIPH_LPTIM1

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 LCDEN LL_APB1_GRP1_EnableClock • APB1ENR WWDGEN LL_APB1_GRP1_EnableClock • APB1ENR SPI2EN LL_APB1_GRP1_EnableClock • APB1ENR USART2EN LL_APB1_GRP1_EnableClock • APB1ENR LPUART1EN 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 CRSEN LL_APB1_GRP1_EnableClock • APB1ENR PWREN LL_APB1_GRP1_EnableClock • APB1ENR DACEN LL_APB1_GRP1_EnableClock • APB1ENR I2C3EN LL_APB1_GRP1_EnableClock • APB1ENR LPTIM1EN LL_APB1_GRP1_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if APB1 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM6 (*) – LL_APB1_GRP1_PERIPH_TIM7 (*) – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_LPUART1 – 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_CRS (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 (*) – LL_APB1_GRP1_PERIPH_I2C3 (*)

	– LL_APB1_GRP1_PERIPH_LPTIM1
Return values	• **State:** of Periphs (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 LCDEN LL_APB1_GRP1_IsEnabledClock • APB1ENR WWDGEN LL_APB1_GRP1_IsEnabledClock • APB1ENR SPI2EN LL_APB1_GRP1_IsEnabledClock • APB1ENR USART2EN LL_APB1_GRP1_IsEnabledClock • APB1ENR LPUART1EN 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 CRSEN LL_APB1_GRP1_IsEnabledClock • APB1ENR PWREN LL_APB1_GRP1_IsEnabledClock • APB1ENR DACEN LL_APB1_GRP1_IsEnabledClock • APB1ENR I2C3EN LL_APB1_GRP1_IsEnabledClock • APB1ENR LPTIM1EN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)**
Function description	Disable APB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM6 (*) – LL_APB1_GRP1_PERIPH_TIM7 (*) – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_LPUART1 – 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_CRS (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 (*) – LL_APB1_GRP1_PERIPH_I2C3 (*) – LL_APB1_GRP1_PERIPH_LPTIM1
Return values	• **None**
Reference Manual to	• APB1ENR TIM2EN LL_APB1_GRP1_DisableClock

LL API cross reference:	• APB1ENR TIM3EN LL_APB1_GRP1_DisableClock • APB1ENR TIM6EN LL_APB1_GRP1_DisableClock • APB1ENR TIM7EN LL_APB1_GRP1_DisableClock • APB1ENR LCDEN LL_APB1_GRP1_DisableClock • APB1ENR WWDGEN LL_APB1_GRP1_DisableClock • APB1ENR SPI2EN LL_APB1_GRP1_DisableClock • APB1ENR USART2EN LL_APB1_GRP1_DisableClock • APB1ENR LPUART1EN 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 CRSEN LL_APB1_GRP1_DisableClock • APB1ENR PWREN LL_APB1_GRP1_DisableClock • APB1ENR DACEN LL_APB1_GRP1_DisableClock • APB1ENR I2C3EN LL_APB1_GRP1_DisableClock • APB1ENR LPTIM1EN LL_APB1_GRP1_DisableClock

LL_APB1_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)**
Function description	Force APB1 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – 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_LCD (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_LPUART1 – 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_CRS (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 (*) – LL_APB1_GRP1_PERIPH_I2C3 (*) – LL_APB1_GRP1_PERIPH_LPTIM1
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 LCDRST LL_APB1_GRP1_ForceReset
- APB1RSTR WWDGRST LL_APB1_GRP1_ForceReset
- APB1RSTR SPI2RST LL_APB1_GRP1_ForceReset
- APB1RSTR USART2RST LL_APB1_GRP1_ForceReset
- APB1RSTR LPUART1RST 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 CRSRST LL_APB1_GRP1_ForceReset
- APB1RSTR PWRRST LL_APB1_GRP1_ForceReset
- APB1RSTR DACRST LL_APB1_GRP1_ForceReset
- APB1RSTR I2C3RST LL_APB1_GRP1_ForceReset
- APB1RSTR LPTIM1RST LL_APB1_GRP1_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name	**__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)**
Function description	Release APB1 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – 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_LCD (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_LPUART1 – 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_CRS (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 (*) – LL_APB1_GRP1_PERIPH_I2C3 (*) – LL_APB1_GRP1_PERIPH_LPTIM1
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 LCDRST LL_APB1_GRP1_ReleaseReset • APB1RSTR WWDGRST LL_APB1_GRP1_ReleaseReset • APB1RSTR SPI2RST LL_APB1_GRP1_ReleaseReset

- APB1RSTR USART2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR LPUART1RST 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 CRSRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR PWRRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR DACRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR I2C3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR LPTIM1RST LL_APB1_GRP1_ReleaseReset

LL_APB1_GRP1_EnableClockSleep

Function name	**__STATIC_INLINE void LL_APB1_GRP1_EnableClockSleep (uint32_t Periphs)**
Function description	Enable APB1 peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM6 (*) – LL_APB1_GRP1_PERIPH_TIM7 (*) – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_LPUART1 – 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_CRS (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 (*) – LL_APB1_GRP1_PERIPH_I2C3 (*) – LL_APB1_GRP1_PERIPH_LPTIM1
Return values	• **None**
Reference Manual to LL API cross reference:	• APB1SMENR TIM2SMEN LL_APB1_GRP1_EnableClockSleep • APB1SMENR TIM3SMEN LL_APB1_GRP1_EnableClockSleep • APB1SMENR TIM6SMEN LL_APB1_GRP1_EnableClockSleep • APB1SMENR TIM7SMEN LL_APB1_GRP1_EnableClockSleep • APB1SMENR LCDSMEN LL_APB1_GRP1_EnableClockSleep • APB1SMENR WWDGSMEN

LL_APB1_GRP1_EnableClockSleep
- APB1SMENR SPI2SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR USART2SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR LPUART1SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR USART4SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR USART5SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR I2C1SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR I2C2SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR USBSMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR CRSSMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR PWRSMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR DACSMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR I2C3SMEN LL_APB1_GRP1_EnableClockSleep
- APB1SMENR LPTIM1SMEN LL_APB1_GRP1_EnableClockSleep

LL_APB1_GRP1_DisableClockSleep

Function name	**__STATIC_INLINE void LL_APB1_GRP1_DisableClockSleep (uint32_t Periphs)**
Function description	Disable APB1 peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM6 (*) – LL_APB1_GRP1_PERIPH_TIM7 (*) – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_LPUART1 – 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_CRS (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 (*)

- LL_APB1_GRP1_PERIPH_I2C3 (*)
- LL_APB1_GRP1_PERIPH_LPTIM1

Return values	• **None**
Reference Manual to LL API cross reference:	• APB1SMENR TIM2SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR TIM3SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR TIM6SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR TIM7SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR LCDSMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR WWDGSMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR SPI2SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR USART2SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR LPUART1SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR USART4SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR USART5SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR I2C1SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR I2C2SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR USBSMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR CRSSMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR PWRSMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR DACSMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR I2C3SMEN LL_APB1_GRP1_DisableClockSleep • APB1SMENR LPTIM1SMEN LL_APB1_GRP1_DisableClockSleep

LL_APB2_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable APB2 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_TIM21 – LL_APB2_GRP1_PERIPH_TIM22 (*)

	– LL_APB2_GRP1_PERIPH_FW – LL_APB2_GRP1_PERIPH_ADC1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_USART1 (*) – LL_APB2_GRP1_PERIPH_DBGMCU
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2ENR SYSCFGEN LL_APB2_GRP1_EnableClock • APB2ENR TIM21EN LL_APB2_GRP1_EnableClock • APB2ENR TIM22EN LL_APB2_GRP1_EnableClock • APB2ENR FWEN LL_APB2_GRP1_EnableClock • APB2ENR ADCEN LL_APB2_GRP1_EnableClock • APB2ENR SPI1EN LL_APB2_GRP1_EnableClock • APB2ENR USART1EN LL_APB2_GRP1_EnableClock • APB2ENR DBGEN LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if APB2 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_TIM21 – LL_APB2_GRP1_PERIPH_TIM22 (*) – LL_APB2_GRP1_PERIPH_FW – LL_APB2_GRP1_PERIPH_ADC1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_USART1 (*) – LL_APB2_GRP1_PERIPH_DBGMCU
Return values	• **State:** of Periphs (1 or 0).
Reference Manual to LL API cross reference:	• APB2ENR SYSCFGEN LL_APB2_GRP1_IsEnabledClock • APB2ENR TIM21EN LL_APB2_GRP1_IsEnabledClock • APB2ENR TIM22EN LL_APB2_GRP1_IsEnabledClock • APB2ENR FWEN LL_APB2_GRP1_IsEnabledClock • APB2ENR ADCEN LL_APB2_GRP1_IsEnabledClock • APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR DBGEN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)**
Function description	Disable APB2 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_TIM21

	– LL_APB2_GRP1_PERIPH_TIM22 (*) – LL_APB2_GRP1_PERIPH_FW – LL_APB2_GRP1_PERIPH_ADC1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_USART1 (*) – LL_APB2_GRP1_PERIPH_DBGMCU
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2ENR SYSCFGEN LL_APB2_GRP1_DisableClock • APB2ENR TIM21EN LL_APB2_GRP1_DisableClock • APB2ENR TIM22EN LL_APB2_GRP1_DisableClock • APB2ENR FWEN LL_APB2_GRP1_DisableClock • APB2ENR ADCEN LL_APB2_GRP1_DisableClock • APB2ENR SPI1EN LL_APB2_GRP1_DisableClock • APB2ENR USART1EN LL_APB2_GRP1_DisableClock • APB2ENR DBGEN LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)**
Function description	Force APB2 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_ALL – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_TIM21 – LL_APB2_GRP1_PERIPH_TIM22 (*) – LL_APB2_GRP1_PERIPH_ADC1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_USART1 (*) – LL_APB2_GRP1_PERIPH_DBGMCU
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2RSTR SYSCFGRST LL_APB2_GRP1_ForceReset • APB2RSTR TIM21RST LL_APB2_GRP1_ForceReset • APB2RSTR TIM22RST LL_APB2_GRP1_ForceReset • APB2RSTR ADCRST LL_APB2_GRP1_ForceReset • APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset • APB2RSTR USART1RST LL_APB2_GRP1_ForceReset • APB2RSTR DBGRST LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

Function name	**__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)**
Function description	Release APB2 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_ALL – LL_APB2_GRP1_PERIPH_SYSCFG

	– LL_APB2_GRP1_PERIPH_TIM21 – LL_APB2_GRP1_PERIPH_TIM22 (*) – LL_APB2_GRP1_PERIPH_ADC1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_USART1 (*) – LL_APB2_GRP1_PERIPH_DBGMCU
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2RSTR SYSCFGRST LL_APB2_GRP1_ReleaseReset • APB2RSTR TIM21RST LL_APB2_GRP1_ReleaseReset • APB2RSTR TIM22RST LL_APB2_GRP1_ReleaseReset • APB2RSTR ADCRST LL_APB2_GRP1_ReleaseReset • APB2RSTR SPI1RST LL_APB2_GRP1_ReleaseReset • APB2RSTR USART1RST LL_APB2_GRP1_ReleaseReset • APB2RSTR DBGRST LL_APB2_GRP1_ReleaseReset

LL_APB2_GRP1_EnableClockSleep

Function name	**__STATIC_INLINE void LL_APB2_GRP1_EnableClockSleep (uint32_t Periphs)**
Function description	Enable APB2 peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_TIM21 – LL_APB2_GRP1_PERIPH_TIM22 (*) – LL_APB2_GRP1_PERIPH_ADC1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_USART1 (*) – LL_APB2_GRP1_PERIPH_DBGMCU
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2SMENR SYSCFGSMEN LL_APB2_GRP1_EnableClockSleep • APB2SMENR TIM21SMEN LL_APB2_GRP1_EnableClockSleep • APB2SMENR TIM22SMEN LL_APB2_GRP1_EnableClockSleep • APB2SMENR ADCSMEN LL_APB2_GRP1_EnableClockSleep • APB2SMENR SPI1SMEN LL_APB2_GRP1_EnableClockSleep • APB2SMENR USART1SMEN LL_APB2_GRP1_EnableClockSleep • APB2SMENR DBGSMEN LL_APB2_GRP1_EnableClockSleep

LL_APB2_GRP1_DisableClockSleep

Function name	**__STATIC_INLINE void LL_APB2_GRP1_DisableClockSleep (uint32_t Periphs)**
Function description	Disable APB2 peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_TIM21 – LL_APB2_GRP1_PERIPH_TIM22 (*) – LL_APB2_GRP1_PERIPH_ADC1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_USART1 (*) – LL_APB2_GRP1_PERIPH_DBGMCU
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2SMENR SYSCFGSMEN LL_APB2_GRP1_DisableClockSleep • APB2SMENR TIM21SMEN LL_APB2_GRP1_DisableClockSleep • APB2SMENR TIM22SMEN LL_APB2_GRP1_DisableClockSleep • APB2SMENR ADCSMEN LL_APB2_GRP1_DisableClockSleep • APB2SMENR SPI1SMEN LL_APB2_GRP1_DisableClockSleep • APB2SMENR USART1SMEN LL_APB2_GRP1_DisableClockSleep • APB2SMENR DBGSMEN LL_APB2_GRP1_DisableClockSleep

LL_IOP_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_IOP_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable IOP peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_IOP_GRP1_PERIPH_GPIOA – LL_IOP_GRP1_PERIPH_GPIOB – LL_IOP_GRP1_PERIPH_GPIOC – LL_IOP_GRP1_PERIPH_GPIOD (*) – LL_IOP_GRP1_PERIPH_GPIOE (*) – LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• IOPENR GPIOAEN LL_IOP_GRP1_EnableClock • IOPENR GPIOBEN LL_IOP_GRP1_EnableClock • IOPENR GPIOCEN LL_IOP_GRP1_EnableClock • IOPENR GPIODEN LL_IOP_GRP1_EnableClock

- IOPENR GPIOEEN LL_IOP_GRP1_EnableClock
- IOPENR GPIOHEN LL_IOP_GRP1_EnableClock

LL_IOP_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_IOP_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if IOP peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_IOP_GRP1_PERIPH_GPIOA – LL_IOP_GRP1_PERIPH_GPIOB – LL_IOP_GRP1_PERIPH_GPIOC – LL_IOP_GRP1_PERIPH_GPIOD (*) – LL_IOP_GRP1_PERIPH_GPIOE (*) – LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	• **State:** of Periphs (1 or 0).
Reference Manual to LL API cross reference:	• IOPENR GPIOAEN LL_IOP_GRP1_IsEnabledClock • IOPENR GPIOBEN LL_IOP_GRP1_IsEnabledClock • IOPENR GPIOCEN LL_IOP_GRP1_IsEnabledClock • IOPENR GPIODEN LL_IOP_GRP1_IsEnabledClock • IOPENR GPIOEEN LL_IOP_GRP1_IsEnabledClock • IOPENR GPIOHEN LL_IOP_GRP1_IsEnabledClock

LL_IOP_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_IOP_GRP1_DisableClock (uint32_t Periphs)**
Function description	Disable IOP peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_IOP_GRP1_PERIPH_GPIOA – LL_IOP_GRP1_PERIPH_GPIOB – LL_IOP_GRP1_PERIPH_GPIOC – LL_IOP_GRP1_PERIPH_GPIOD (*) – LL_IOP_GRP1_PERIPH_GPIOE (*) – LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• IOPENR GPIOAEN LL_IOP_GRP1_DisableClock • IOPENR GPIOBEN LL_IOP_GRP1_DisableClock • IOPENR GPIOCEN LL_IOP_GRP1_DisableClock • IOPENR GPIODEN LL_IOP_GRP1_DisableClock • IOPENR GPIOEEN LL_IOP_GRP1_DisableClock • IOPENR GPIOHEN LL_IOP_GRP1_DisableClock

LL_IOP_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_IOP_GRP1_ForceReset (uint32_t Periphs)**
Function description	Disable IOP peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_IOP_GRP1_PERIPH_ALL – LL_IOP_GRP1_PERIPH_GPIOA – LL_IOP_GRP1_PERIPH_GPIOB – LL_IOP_GRP1_PERIPH_GPIOC – LL_IOP_GRP1_PERIPH_GPIOD (*) – LL_IOP_GRP1_PERIPH_GPIOE (*) – LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• IOPRSTR GPIOASMEN LL_IOP_GRP1_ForceReset • IOPRSTR GPIOBSMEN LL_IOP_GRP1_ForceReset • IOPRSTR GPIOCSMEN LL_IOP_GRP1_ForceReset • IOPRSTR GPIODSMEN LL_IOP_GRP1_ForceReset • IOPRSTR GPIOESMEN LL_IOP_GRP1_ForceReset • IOPRSTR GPIOHSMEN LL_IOP_GRP1_ForceReset

LL_IOP_GRP1_ReleaseReset

Function name	**__STATIC_INLINE void LL_IOP_GRP1_ReleaseReset (uint32_t Periphs)**
Function description	Release IOP peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_IOP_GRP1_PERIPH_ALL – LL_IOP_GRP1_PERIPH_GPIOA – LL_IOP_GRP1_PERIPH_GPIOB – LL_IOP_GRP1_PERIPH_GPIOC – LL_IOP_GRP1_PERIPH_GPIOD (*) – LL_IOP_GRP1_PERIPH_GPIOE (*) – LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• IOPRSTR GPIOASMEN LL_IOP_GRP1_ReleaseReset • IOPRSTR GPIOBSMEN LL_IOP_GRP1_ReleaseReset • IOPRSTR GPIOCSMEN LL_IOP_GRP1_ReleaseReset • IOPRSTR GPIODSMEN LL_IOP_GRP1_ReleaseReset • IOPRSTR GPIOESMEN LL_IOP_GRP1_ReleaseReset • IOPRSTR GPIOHSMEN LL_IOP_GRP1_ReleaseReset

LL_IOP_GRP1_EnableClockSleep

Function name	**__STATIC_INLINE void LL_IOP_GRP1_EnableClockSleep (uint32_t Periphs)**
Function description	Enable IOP peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_IOP_GRP1_PERIPH_GPIOA – LL_IOP_GRP1_PERIPH_GPIOB – LL_IOP_GRP1_PERIPH_GPIOC – LL_IOP_GRP1_PERIPH_GPIOD (*) – LL_IOP_GRP1_PERIPH_GPIOE (*) – LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• IOPSMENR GPIOARST LL_IOP_GRP1_EnableClockSleep • IOPSMENR GPIOBRST LL_IOP_GRP1_EnableClockSleep • IOPSMENR GPIOCRST LL_IOP_GRP1_EnableClockSleep • IOPSMENR GPIODRST LL_IOP_GRP1_EnableClockSleep • IOPSMENR GPIOERST LL_IOP_GRP1_EnableClockSleep • IOPSMENR GPIOHRST LL_IOP_GRP1_EnableClockSleep

LL_IOP_GRP1_DisableClockSleep

Function name	**__STATIC_INLINE void LL_IOP_GRP1_DisableClockSleep (uint32_t Periphs)**
Function description	Disable IOP peripherals clock during Low Power (Sleep) mode.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_IOP_GRP1_PERIPH_GPIOA – LL_IOP_GRP1_PERIPH_GPIOB – LL_IOP_GRP1_PERIPH_GPIOC – LL_IOP_GRP1_PERIPH_GPIOD (*) – LL_IOP_GRP1_PERIPH_GPIOE (*) – LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• IOPSMENR GPIOARST LL_IOP_GRP1_DisableClockSleep • IOPSMENR GPIOBRST LL_IOP_GRP1_DisableClockSleep • IOPSMENR GPIOCRST LL_IOP_GRP1_DisableClockSleep • IOPSMENR GPIODRST LL_IOP_GRP1_DisableClockSleep • IOPSMENR GPIOERST LL_IOP_GRP1_DisableClockSleep • IOPSMENR GPIOHRST LL_IOP_GRP1_DisableClockSleep

54.2 BUS Firmware driver defines

54.2.1 BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ALL	
LL_AHB1_GRP1_PERIPH_DMA1	DMA1 clock enable
LL_AHB1_GRP1_PERIPH_MIF	MIF clock enable
LL_AHB1_GRP1_PERIPH_SRAM	Sleep Mode SRAM clock enable
LL_AHB1_GRP1_PERIPH_CRC	CRC clock enable
LL_AHB1_GRP1_PERIPH_TSC	TSC clock enable
LL_AHB1_GRP1_PERIPH_RNG	RNG clock enable

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_ALL	
LL_APB1_GRP1_PERIPH_TIM2	TIM2 clock enable
LL_APB1_GRP1_PERIPH_TIM3	TIM3 clock enable
LL_APB1_GRP1_PERIPH_TIM6	TIM6 clock enable
LL_APB1_GRP1_PERIPH_TIM7	TIM7 clock enable
LL_APB1_GRP1_PERIPH_LCD	LCD clock enable
LL_APB1_GRP1_PERIPH_WWDG	WWDG clock enable
LL_APB1_GRP1_PERIPH_SPI2	SPI2 clock enable
LL_APB1_GRP1_PERIPH_USART2	USART2 clock enable
LL_APB1_GRP1_PERIPH_LPUART1	LPUART1 clock enable
LL_APB1_GRP1_PERIPH_USART4	USART4 clock enable
LL_APB1_GRP1_PERIPH_USART5	USART5 clock enable
LL_APB1_GRP1_PERIPH_I2C1	I2C1 clock enable
LL_APB1_GRP1_PERIPH_I2C2	I2C2 clock enable
LL_APB1_GRP1_PERIPH_USB	USB clock enable
LL_APB1_GRP1_PERIPH_CRS	CRS clock enable
LL_APB1_GRP1_PERIPH_PWR	PWR clock enable
LL_APB1_GRP1_PERIPH_DAC1	DAC clock enable
LL_APB1_GRP1_PERIPH_I2C3	I2C3 clock enable
LL_APB1_GRP1_PERIPH_LPTIM1	LPTIM1 clock enable

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_ALL	
LL_APB2_GRP1_PERIPH_SYSCFG	SYSCFG clock enable
LL_APB2_GRP1_PERIPH_TIM21	TIM21 clock enable
LL_APB2_GRP1_PERIPH_TIM22	TIM22 clock enable

LL_APB2_GRP1_PERIPH_FW	FireWall clock enable
LL_APB2_GRP1_PERIPH_ADC1	ADC1 clock enable
LL_APB2_GRP1_PERIPH_SPI1	SPI1 clock enable
LL_APB2_GRP1_PERIPH_USART1	USART1 clock enable
LL_APB2_GRP1_PERIPH_DBGMCU	DBGMCU clock enable

IOP GRP1 PERIPH

LL_IOP_GRP1_PERIPH_ALL	
LL_IOP_GRP1_PERIPH_GPIOA	GPIO port A control
LL_IOP_GRP1_PERIPH_GPIOB	GPIO port B control
LL_IOP_GRP1_PERIPH_GPIOC	GPIO port C control
LL_IOP_GRP1_PERIPH_GPIOD	GPIO port D control
LL_IOP_GRP1_PERIPH_GPIOE	GPIO port H control
LL_IOP_GRP1_PERIPH_GPIOH	GPIO port H control

55 LL COMP Generic Driver

55.1 COMP Firmware driver registers structures

55.1.1 LL_COMP_InitTypeDef

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***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::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()**.

55.2 COMP Firmware driver API description

55.2.1 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	• **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE()) • **WindowMode:** This parameter can be one of the following values: – LL_COMP_WINDOWMODE_DISABLE – LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON
Return values	• **None**

Reference Manual to LL API cross reference:	• COMP1_CSR COMP1WM LL_COMP_SetCommonWindowMode

LL_COMP_GetCommonWindowMode

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON)**
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	• **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())
Return values	• **Returned:** value can be one of the following values: – LL_COMP_WINDOWMODE_DISABLE – LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1WM LL_COMP_GetCommonWindowMode

LL_COMP_SetPowerMode

Function name	**__STATIC_INLINE void LL_COMP_SetPowerMode (COMP_TypeDef * COMPx, uint32_t PowerMode)**
Function description	Set comparator instance operating mode to adjust power and speed.
Parameters	• **COMPx:** Comparator instance • **PowerMode:** This parameter can be one of the following values: (1) Available only on COMP instance: COMP2. – LL_COMP_POWERMODE_MEDIUMSPEED (1) – LL_COMP_POWERMODE_ULTRALOWPOWER (1)
Return values	• **None**
Reference Manual to LL API cross reference:	• COMP2_CSR COMP2SPEED LL_COMP_SetPowerMode

LL_COMP_GetPowerMode

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetPowerMode (COMP_TypeDef * COMPx)**
Function description	Get comparator instance operating mode to adjust power and speed.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: (1) Available only on COMP instance: COMP2. – LL_COMP_POWERMODE_MEDIUMSPEED (1)

	– LL_COMP_POWERMODE_ULTRALOWPOWER (1)
Notes	• Available only on COMP instance: COMP2.
Reference Manual to LL API cross reference:	• COMP2_CSR COMP2SPEED 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_VREFINT – LL_COMP_INPUT_MINUS_IO1 – LL_COMP_INPUT_MINUS_DAC1_CH1 – LL_COMP_INPUT_MINUS_DAC1_CH2 – LL_COMP_INPUT_MINUS_1_4VREFINT – LL_COMP_INPUT_MINUS_1_2VREFINT – LL_COMP_INPUT_MINUS_3_4VREFINT – LL_COMP_INPUT_MINUS_IO2 • **InputPlus:** This parameter can be one of the following values: (1) Available only on COMP instance: COMP2. (2) Available only on devices STM32L0 category 1. – LL_COMP_INPUT_PLUS_IO1 (1) – LL_COMP_INPUT_PLUS_IO2 (1) – LL_COMP_INPUT_PLUS_IO3 (1) – LL_COMP_INPUT_PLUS_IO4 (1) – LL_COMP_INPUT_PLUS_IO5 (1) – LL_COMP_INPUT_PLUS_IO6 (1)(2)
Return values	• **None**
Notes	• This function shall only be used for COMP2. For setting COMP1 input it is recommended to use LL_COMP_SetInputMinus() Plus (non-inverting) input is not configurable on COMP1. Using this function for COMP1 will corrupt COMP1WM register • On this STM32 serie, specificity if using COMP instance COMP2 with COMP input based on VrefInt (VrefInt or subdivision of VrefInt): scaler bridge is based on VrefInt and requires to enable path from VrefInt (refer to literal SYSCFG_CFGR3_ENBUFLP_VREFINT_COMP).
Reference Manual to LL API cross reference:	• COMP2_CSR COMP2INNSEL LL_COMP_ConfigInputs • COMP2_CSR COMP2INPSEL LL_COMP_ConfigInputs

LL_COMP_SetInputPlus

Function name	**__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)**
Function description	Set comparator input plus (non-inverting).
Parameters	• **COMPx:** Comparator instance • **InputPlus:** This parameter can be one of the following values: (1) Available only on COMP instance: COMP2. (2) Available only on devices STM32L0 category 1. – LL_COMP_INPUT_PLUS_IO1 (1) – LL_COMP_INPUT_PLUS_IO2 (1) – LL_COMP_INPUT_PLUS_IO3 (1) – LL_COMP_INPUT_PLUS_IO4 (1) – LL_COMP_INPUT_PLUS_IO5 (1) – LL_COMP_INPUT_PLUS_IO6 (1)(2)
Return values	• **None**
Notes	• Only COMP2 allows to set the input plus (non-inverting). For COMP1 it is always PA1 IO, except when Windows Mode is selected.
Reference Manual to LL API cross reference:	• COMP2_CSR COMP2INPSEL LL_COMP_SetInputPlus

LL_COMP_GetInputPlus

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (COMP_TypeDef * COMPx)**
Function description	Get comparator input plus (non-inverting).
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: (1) Available only on COMP instance: COMP2. (2) Available only on devices STM32L0 category 1. – LL_COMP_INPUT_PLUS_IO1 (1) – LL_COMP_INPUT_PLUS_IO2 (1) – LL_COMP_INPUT_PLUS_IO3 (1) – LL_COMP_INPUT_PLUS_IO4 (1) – LL_COMP_INPUT_PLUS_IO5 (1) – LL_COMP_INPUT_PLUS_IO6 (1)(2)
Notes	• Only COMP2 allows to set the input plus (non-inverting). For COMP1 it is always PA1 IO, except when Windows Mode is selected.
Reference Manual to LL API cross reference:	• COMP2_CSR COMP2INPSEL LL_COMP_GetInputPlus

LL_COMP_SetInputMinus

Function name	**__STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)**
Function description	Set comparator input minus (inverting).
Parameters	• **COMPx:** Comparator instance • **InputMinus:** This parameter can be one of the following values: (*) Available only on COMP instance: COMP2. – LL_COMP_INPUT_MINUS_VREFINT – LL_COMP_INPUT_MINUS_IO1 – LL_COMP_INPUT_MINUS_DAC1_CH1 – LL_COMP_INPUT_MINUS_DAC1_CH2 – LL_COMP_INPUT_MINUS_1_4VREFINT (*) – LL_COMP_INPUT_MINUS_1_2VREFINT (*) – LL_COMP_INPUT_MINUS_3_4VREFINT (*) – LL_COMP_INPUT_MINUS_IO2 (*)
Return values	• **None**
Notes	• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual. • On this STM32 serie, specificity if using COMP instance COMP2 with COMP input based on VrefInt (VrefInt or subdivision of VrefInt): scaler bridge is based on VrefInt and requires to enable path from VrefInt (refer to literal SYSCFG_CFGR3_ENBUFLP_VREFINT_COMP).
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1INNSEL LL_COMP_SetInputMinus • COMP2_CSR COMP2INNSEL 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: (*) Available only on COMP instance: COMP2. – LL_COMP_INPUT_MINUS_VREFINT – LL_COMP_INPUT_MINUS_IO1 – LL_COMP_INPUT_MINUS_DAC1_CH1 – LL_COMP_INPUT_MINUS_DAC1_CH2 – LL_COMP_INPUT_MINUS_1_4VREFINT (*) – LL_COMP_INPUT_MINUS_1_2VREFINT (*) – LL_COMP_INPUT_MINUS_3_4VREFINT (*) – LL_COMP_INPUT_MINUS_IO2 (*)
Notes	• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
Reference Manual to	• COMP1_CSR COMP1INNSEL LL_COMP_GetInputMinus

LL API cross reference:	• COMP2_CSR COMP2INNSEL LL_COMP_GetInputMinus

LL_COMP_SetOutputLPTIM

Function name	**__STATIC_INLINE void LL_COMP_SetOutputLPTIM (COMP_TypeDef * COMPx, uint32_t OutputLptim)**
Function description	Set comparator output LPTIM.
Parameters	• **COMPx:** Comparator instance • **OutputLptim:** This parameter can be one of the following values: (*) Available only on COMP instance: COMP1. – LL_COMP_OUTPUT_LPTIM1_IN1_COMP1 (*) – LL_COMP_OUTPUT_LPTIM1_IN1_COMP2 (**) – LL_COMP_OUTPUT_LPTIM1_IN2_COMP2 (**) • (**) Available only on COMP instance: COMP2.
Return values	• **None**
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1LPTIMIN1 LL_COMP_SetOutputLPTIM • COMP2_CSR COMP2LPTIMIN1 LL_COMP_SetOutputLPTIM • COMP2_CSR COMP2LPTIMIN2 LL_COMP_SetOutputLPTIM

LL_COMP_GetOutputLPTIM

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetOutputLPTIM (COMP_TypeDef * COMPx)**
Function description	Get comparator output LPTIM.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: (*) Available only on COMP instance: COMP1. – LL_COMP_OUTPUT_LPTIM1_IN1_COMP1 (*) – LL_COMP_OUTPUT_LPTIM1_IN1_COMP2 (**) – LL_COMP_OUTPUT_LPTIM1_IN2_COMP2 (**) • (**) Available only on COMP instance: COMP2.
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1LPTIMIN1 LL_COMP_GetOutputLPTIM • COMP2_CSR COMP2LPTIMIN1 LL_COMP_GetOutputLPTIM • COMP2_CSR COMP2LPTIMIN2 LL_COMP_GetOutputLPTIM

LL_COMP_SetOutputPolarity

Function name	**__STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)**
Function description	Set comparator instance output polarity.
Parameters	• **COMPx:** Comparator instance

	• **OutputPolarity:** This parameter can be one of the following values: – LL_COMP_OUTPUTPOL_NONINVERTED – LL_COMP_OUTPUTPOL_INVERTED
Return values	• **None**
Reference Manual to LL API cross reference:	• COMP COMP1POLARITY LL_COMP_SetOutputPolarity

LL_COMP_GetOutputPolarity

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (COMP_TypeDef * COMPx)**
Function description	Get comparator instance output polarity.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: – LL_COMP_OUTPUTPOL_NONINVERTED – LL_COMP_OUTPUTPOL_INVERTED
Reference Manual to LL API cross reference:	• COMP COMP1POLARITY LL_COMP_GetOutputPolarity

LL_COMP_Enable

Function name	**__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)**
Function description	Enable comparator instance.
Parameters	• **COMPx:** Comparator instance
Return values	• **None**
Notes	• After enable from off state, comparator requires a delay to reach reach propagation delay specification. Refer to device datasheet, parameter "tSTART".
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1EN LL_COMP_Enable • COMP2_CSR COMP2EN LL_COMP_Enable

LL_COMP_Disable

Function name	**__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)**
Function description	Disable comparator instance.
Parameters	• **COMPx:** Comparator instance
Return values	• **None**
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1EN LL_COMP_Disable • COMP2_CSR COMP2EN LL_COMP_Disable

LL_COMP_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_COMP_IsEnabled (COMP_TypeDef * COMPx)**
Function description	Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)
Parameters	• **COMPx:** Comparator instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1EN LL_COMP_IsEnabled • COMP2_CSR COMP2EN LL_COMP_IsEnabled

LL_COMP_Lock

Function name	**__STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)**
Function description	Lock comparator instance.
Parameters	• **COMPx:** Comparator instance
Return values	• **None**
Notes	• Once locked, comparator configuration can be accessed in read-only. • The only way to unlock the comparator is a device hardware reset.
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1LOCK LL_COMP_Lock • COMP2_CSR COMP2LOCK LL_COMP_Lock

LL_COMP_IsLocked

Function name	**__STATIC_INLINE uint32_t LL_COMP_IsLocked (COMP_TypeDef * COMPx)**
Function description	Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).
Parameters	• **COMPx:** Comparator instance
Return values	• **State:** of bit (1 or 0).
Notes	• Once locked, comparator configuration can be accessed in read-only. • The only way to unlock the comparator is a device hardware reset.
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1LOCK LL_COMP_IsLocked • COMP2_CSR COMP2LOCK LL_COMP_IsLocked

LL_COMP_ReadOutputLevel

Function name	**__STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (COMP_TypeDef * COMPx)**

Function description	Read comparator instance output level.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: – LL_COMP_OUTPUT_LEVEL_LOW – LL_COMP_OUTPUT_LEVEL_HIGH
Notes	• The comparator output level depends on the selected polarity (Refer to function LL_COMP_SetOutputPolarity()). If the comparator polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the comparator polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus
Reference Manual to LL API cross reference:	• COMP1_CSR COMP1VALUE LL_COMP_ReadOutputLevel • COMP2_CSR COMP2VALUE LL_COMP_ReadOutputLevel

LL_COMP_DeInit

Function name	**ErrorStatus LL_COMP_DeInit (COMP_TypeDef * COMPx)**
Function description	De-initialize registers of the selected COMP instance to their default reset values.
Parameters	• **COMPx:** COMP instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: COMP registers are de-initialized – ERROR: COMP registers are not de-initialized
Notes	• If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_COMP_Init

Function name	**ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, LL_COMP_InitTypeDef * COMP_InitStruct)**
Function description	Initialize some features of COMP instance.
Parameters	• **COMPx:** COMP instance • **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: COMP registers are initialized – ERROR: COMP registers are not initialized
Notes	• This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.

LL_COMP_StructInit

Function name	**void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)**
Function description	Set each LL_COMP_InitTypeDef field to default value.
Parameters	• **COMP_InitStruct:** pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.
Return values	• **None**

55.3 COMP Firmware driver defines

55.3.1 COMP

Comparator common modes - Window mode

LL_COMP_WINDOWMODE_DISABLE	Window mode disable: Comparators 1 and 2 are independent
LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON	Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

Definitions of COMP hardware constraints delays

LL_COMP_DELAY_STARTUP_US	Delay for COMP startup time
LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US	Delay for COMP voltage scaler stabilization time

Comparator inputs - Input minus (input inverting) selection

LL_COMP_INPUT_MINUS_1_4VREFINT	Comparator input minus connected to 1/4 VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function
LL_COMP_INPUT_MINUS_1_2VREFINT	Comparator input minus connected to 1/2 VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function
LL_COMP_INPUT_MINUS_3_4VREFINT	Comparator input minus connected to 3/4 VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function
LL_COMP_INPUT_MINUS_VREFINT	Comparator input minus connected to VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function
LL_COMP_INPUT_MINUS_DAC1_CH1	Comparator input minus connected to DAC1

	channel 1 (DAC_OUT1)
LL_COMP_INPUT_MINUS_DAC1_CH2	Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)
LL_COMP_INPUT_MINUS_IO1	Comparator input minus connected to IO1 (pin PA0 for COMP1, pin PA2 for COMP2)
LL_COMP_INPUT_MINUS_IO2	Comparator input minus connected to IO2 (pin PB3 for COMP2) (specific to COMP instance: COMP2)

Comparator inputs - Input plus (input non-inverting) selection

LL_COMP_INPUT_PLUS_IO1	Comparator input plus connected to IO1 (pin PA1 for COMP1, pin PA3 for COMP2)
LL_COMP_INPUT_PLUS_IO2	Comparator input plus connected to IO2 (pin PB4 for COMP2) (specific to COMP instance: COMP2)
LL_COMP_INPUT_PLUS_IO3	Comparator input plus connected to IO3 (pin PA5 for COMP2) (specific to COMP instance: COMP2)
LL_COMP_INPUT_PLUS_IO4	Comparator input plus connected to IO4 (pin PB6 for COMP2) (specific to COMP instance: COMP2)
LL_COMP_INPUT_PLUS_IO5	Comparator input plus connected to IO5 (pin PB7 for COMP2) (specific to COMP instance: COMP2)

Comparator output - Output level

LL_COMP_OUTPUT_LEVEL_LOW	Comparator output level low (if the polarity is not inverted, otherwise to be complemented)
LL_COMP_OUTPUT_LEVEL_HIGH	Comparator output level high (if the polarity is not inverted, otherwise to be complemented)

Comparator output - Output polarity

LL_COMP_OUTPUTPOL_NONINVERTED	COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input
LL_COMP_OUTPUTPOL_INVERTED	COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

Comparator output - Output selection specific to LPTIM peripheral

LL_COMP_OUTPUT_LPTIM1_IN1_COMP1	COMP output connected to TIM2 input capture 4
LL_COMP_OUTPUT_LPTIM1_IN1_COMP2	COMP output connected to TIM2 input capture 4
LL_COMP_OUTPUT_LPTIM1_IN2_COMP2	COMP output connected to TIM2 input capture 4

Comparator modes - Power mode

LL_COMP_POWERMODE_ULTRALOWPOWER	COMP power mode to low speed (specific to COMP instance: COMP2)
LL_COMP_POWERMODE_MEDIUMSPEED	COMP power mode to fast speed

(specific to COMP instance: COMP2)

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

56 LL CORTEX Generic Driver

56.1 CORTEX Firmware driver API description

56.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)**
Function description	This function checks if the Systick counter flag is active or not.
Return values	• **State:** of bit (1 or 0).
Notes	• It can be used in timeout function on application side.
Reference Manual to LL API cross reference:	• STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name	**__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)**
Function description	Configures the SysTick clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_SYSTICK_CLKSOURCE_HCLK_DIV8 – LL_SYSTICK_CLKSOURCE_HCLK
Return values	• **None**
Reference Manual to LL API cross reference:	• STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)**
Function description	Get the SysTick clock source.
Return values	• **Returned:** value can be one of the following values: – LL_SYSTICK_CLKSOURCE_HCLK_DIV8 – LL_SYSTICK_CLKSOURCE_HCLK
Reference Manual to LL API cross reference:	• STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name	**__STATIC_INLINE void LL_SYSTICK_EnableIT (void)**
Function description	Enable SysTick exception request.

Return values	• **None**
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name	**__STATIC_INLINE void LL_SYSTICK_DisableIT (void)**
Function description	Disable SysTick exception request.
Return values	• **None**
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)**
Function description	Checks if the SYSTICK interrupt is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name	**__STATIC_INLINE void LL_LPM_EnableSleep (void)**
Function description	Processor uses sleep as its low power mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name	**__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)**
Function description	Processor uses deep sleep as its low power mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name	**__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)**
Function description	Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values	• **None**
Notes	• Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name	**__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)**
Function description	Do not sleep when returning to Thread mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name	**__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)**
Function description	Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.
Return values	• **None**
Reference Manual to LL API cross reference:	• SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name	**__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)**
Function description	Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.
Return values	• **None**
Reference Manual to LL API cross reference:	• SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_CPUID_GetImplementer

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)**
Function description	Get Implementer code.
Return values	• **Value:** should be equal to 0x41 for ARM
Reference Manual to LL API cross reference:	• SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)**
Function description	Get Variant number (The r value in the rnpn product revision identifier)
Return values	• **Value:** between 0 and 255 (0x0: revision 0)
Reference Manual to LL API cross reference:	• SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetArchitecture

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetArchitecture (void)**
Function description	Get Architecture number.
Return values	• **Value:** should be equal to 0xC for Cortex-M0+ devices
Reference Manual to LL API cross reference:	• SCB_CPUID ARCHITECTURE LL_CPUID_GetArchitecture

LL_CPUID_GetParNo

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)**
Function description	Get Part number.
Return values	• **Value:** should be equal to 0xC60 for Cortex-M0+
Reference Manual to LL API cross reference:	• SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)**
Function description	Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)
Return values	• **Value:** between 0 and 255 (0x1: patch 1)
Reference Manual to LL API cross reference:	• SCB_CPUID REVISION LL_CPUID_GetRevision

LL_MPU_Enable

Function name	**__STATIC_INLINE void LL_MPU_Enable (uint32_t Options)**
Function description	Enable MPU with input options.
Parameters	• **Options:** This parameter can be one of the following values: – LL_MPU_CTRL_HFNMI_PRIVDEF_NONE – LL_MPU_CTRL_HARDFAULT_NMI – LL_MPU_CTRL_PRIVILEGED_DEFAULT – LL_MPU_CTRL_HFNMI_PRIVDEF

Return values	• **None**
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Disable

Function name	**__STATIC_INLINE void LL_MPU_Disable (void)**
Function description	Disable MPU.
Return values	• **None**
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_Disable

LL_MPU_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void)**
Function description	Check if MPU is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_IsEnabled

LL_MPU_EnableRegion

Function name	**__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)**
Function description	Enable a MPU region.
Parameters	• **Region:** This parameter can be one of the following values: – LL_MPU_REGION_NUMBER0 – LL_MPU_REGION_NUMBER1 – LL_MPU_REGION_NUMBER2 – LL_MPU_REGION_NUMBER3 – LL_MPU_REGION_NUMBER4 – LL_MPU_REGION_NUMBER5 – LL_MPU_REGION_NUMBER6 – LL_MPU_REGION_NUMBER7
Return values	• **None**
Reference Manual to LL API cross reference:	• MPU_RASR ENABLE LL_MPU_EnableRegion

LL_MPU_ConfigRegion

Function name	**__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t SubRegionDisable, uint32_t Address, uint32_t Attributes)**

Function description	Configure and enable a region.
Parameters	• **Region:** This parameter can be one of the following values: – LL_MPU_REGION_NUMBER0 – LL_MPU_REGION_NUMBER1 – LL_MPU_REGION_NUMBER2 – LL_MPU_REGION_NUMBER3 – LL_MPU_REGION_NUMBER4 – LL_MPU_REGION_NUMBER5 – LL_MPU_REGION_NUMBER6 – LL_MPU_REGION_NUMBER7 • **Address:** Value of region base address • **SubRegionDisable:** Sub-region disable value between Min_Data = 0x00 and Max_Data = 0xFF • **Attributes:** This parameter can be a combination of the following values: – LL_MPU_REGION_SIZE_32B or LL_MPU_REGION_SIZE_64B or LL_MPU_REGION_SIZE_128B or LL_MPU_REGION_SIZE_256B or LL_MPU_REGION_SIZE_512B or LL_MPU_REGION_SIZE_1KB or LL_MPU_REGION_SIZE_2KB or LL_MPU_REGION_SIZE_4KB or LL_MPU_REGION_SIZE_8KB or LL_MPU_REGION_SIZE_16KB or LL_MPU_REGION_SIZE_32KB or LL_MPU_REGION_SIZE_64KB or LL_MPU_REGION_SIZE_128KB or LL_MPU_REGION_SIZE_256KB or LL_MPU_REGION_SIZE_512KB or LL_MPU_REGION_SIZE_1MB or LL_MPU_REGION_SIZE_2MB or LL_MPU_REGION_SIZE_4MB or LL_MPU_REGION_SIZE_8MB or LL_MPU_REGION_SIZE_16MB or LL_MPU_REGION_SIZE_32MB or LL_MPU_REGION_SIZE_64MB or LL_MPU_REGION_SIZE_128MB or LL_MPU_REGION_SIZE_256MB or LL_MPU_REGION_SIZE_512MB or LL_MPU_REGION_SIZE_1GB or LL_MPU_REGION_SIZE_2GB or LL_MPU_REGION_SIZE_4GB – LL_MPU_REGION_NO_ACCESS or LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_PRIV_RW_URO or LL_MPU_REGION_FULL_ACCESS or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_PRIV_RO_URO – LL_MPU_TEX_LEVEL0 or LL_MPU_TEX_LEVEL1 or LL_MPU_TEX_LEVEL2 or LL_MPU_TEX_LEVEL4 – LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE – LL_MPU_ACCESS_SHAREABLE or

	– LL_MPU_ACCESS_NOT_SHAREABLE – LL_MPU_ACCESS_CACHEABLE or LL_MPU_ACCESS_NOT_CACHEABLE – LL_MPU_ACCESS_BUFFERABLE or LL_MPU_ACCESS_NOT_BUFFERABLE
Return values	• **None**
Reference Manual to LL API cross reference:	• MPU_RNR REGION LL_MPU_ConfigRegion • MPU_RBAR REGION LL_MPU_ConfigRegion • MPU_RBAR ADDR LL_MPU_ConfigRegion • MPU_RASR XN LL_MPU_ConfigRegion • MPU_RASR AP LL_MPU_ConfigRegion • MPU_RASR S LL_MPU_ConfigRegion • MPU_RASR C LL_MPU_ConfigRegion • MPU_RASR B LL_MPU_ConfigRegion • MPU_RASR SIZE LL_MPU_ConfigRegion

LL_MPU_DisableRegion

Function name	**__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t Region)**
Function description	Disable a region.
Parameters	• **Region:** This parameter can be one of the following values: – LL_MPU_REGION_NUMBER0 – LL_MPU_REGION_NUMBER1 – LL_MPU_REGION_NUMBER2 – LL_MPU_REGION_NUMBER3 – LL_MPU_REGION_NUMBER4 – LL_MPU_REGION_NUMBER5 – LL_MPU_REGION_NUMBER6 – LL_MPU_REGION_NUMBER7
Return values	• **None**
Reference Manual to LL API cross reference:	• MPU_RNR REGION LL_MPU_DisableRegion • MPU_RASR ENABLE LL_MPU_DisableRegion

56.2 CORTEX Firmware driver defines

56.2.1 CORTEX

MPU Bufferable Access

LL_MPU_ACCESS_BUFFERABLE	Bufferable memory attribute
LL_MPU_ACCESS_NOT_BUFFERABLE	Not Bufferable memory attribute

MPU Cacheable Access

LL_MPU_ACCESS_CACHEABLE	Cacheable memory attribute
LL_MPU_ACCESS_NOT_CACHEABLE	Not Cacheable memory attribute

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8	AHB clock divided by 8 selected as SysTick clock source.
LL_SYSTICK_CLKSOURCE_HCLK	AHB clock selected as SysTick clock source.

MPU Control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE	Disable NMI and privileged SW access
LL_MPU_CTRL_HARDFAULT_NMI	Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers
LL_MPU_CTRL_PRIVILEGED_DEFAULT	Enable privileged software access to default memory map
LL_MPU_CTRL_HFNMI_PRIVDEF	Enable NMI and privileged SW access

MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE	Instruction fetches enabled
LL_MPU_INSTRUCTION_ACCESS_DISABLE	Instruction fetches disabled

MPU Region Number

LL_MPU_REGION_NUMBER0	REGION Number 0
LL_MPU_REGION_NUMBER1	REGION Number 1
LL_MPU_REGION_NUMBER2	REGION Number 2
LL_MPU_REGION_NUMBER3	REGION Number 3
LL_MPU_REGION_NUMBER4	REGION Number 4
LL_MPU_REGION_NUMBER5	REGION Number 5
LL_MPU_REGION_NUMBER6	REGION Number 6
LL_MPU_REGION_NUMBER7	REGION Number 7

MPU Region Privileges

LL_MPU_REGION_NO_ACCESS	No access
LL_MPU_REGION_PRIV_RW	RW privileged (privileged access only)
LL_MPU_REGION_PRIV_RW_URO	RW privileged - RO user (Write in a user program generates a fault)
LL_MPU_REGION_FULL_ACCESS	RW privileged & user (Full access)
LL_MPU_REGION_PRIV_RO	RO privileged (privileged read only)
LL_MPU_REGION_PRIV_RO_URO	RO privileged & user (read only)

MPU Region Size

LL_MPU_REGION_SIZE_32B	32B Size of the MPU protection region
LL_MPU_REGION_SIZE_64B	64B Size of the MPU protection region
LL_MPU_REGION_SIZE_128B	128B Size of the MPU protection region
LL_MPU_REGION_SIZE_256B	256B Size of the MPU protection region
LL_MPU_REGION_SIZE_512B	512B Size of the MPU protection region

LL_MPU_REGION_SIZE_1KB	1KB Size of the MPU protection region
LL_MPU_REGION_SIZE_2KB	2KB Size of the MPU protection region
LL_MPU_REGION_SIZE_4KB	4KB Size of the MPU protection region
LL_MPU_REGION_SIZE_8KB	8KB Size of the MPU protection region
LL_MPU_REGION_SIZE_16KB	16KB Size of the MPU protection region
LL_MPU_REGION_SIZE_32KB	32KB Size of the MPU protection region
LL_MPU_REGION_SIZE_64KB	64KB Size of the MPU protection region
LL_MPU_REGION_SIZE_128KB	128KB Size of the MPU protection region
LL_MPU_REGION_SIZE_256KB	256KB Size of the MPU protection region
LL_MPU_REGION_SIZE_512KB	512KB Size of the MPU protection region
LL_MPU_REGION_SIZE_1MB	1MB Size of the MPU protection region
LL_MPU_REGION_SIZE_2MB	2MB Size of the MPU protection region
LL_MPU_REGION_SIZE_4MB	4MB Size of the MPU protection region
LL_MPU_REGION_SIZE_8MB	8MB Size of the MPU protection region
LL_MPU_REGION_SIZE_16MB	16MB Size of the MPU protection region
LL_MPU_REGION_SIZE_32MB	32MB Size of the MPU protection region
LL_MPU_REGION_SIZE_64MB	64MB Size of the MPU protection region
LL_MPU_REGION_SIZE_128MB	128MB Size of the MPU protection region
LL_MPU_REGION_SIZE_256MB	256MB Size of the MPU protection region
LL_MPU_REGION_SIZE_512MB	512MB Size of the MPU protection region
LL_MPU_REGION_SIZE_1GB	1GB Size of the MPU protection region
LL_MPU_REGION_SIZE_2GB	2GB Size of the MPU protection region
LL_MPU_REGION_SIZE_4GB	4GB Size of the MPU protection region

MPU Shareable Access

LL_MPU_ACCESS_SHAREABLE	Shareable memory attribute
LL_MPU_ACCESS_NOT_SHAREABLE	Not Shareable memory attribute

MPU TEX Level

LL_MPU_TEX_LEVEL0	b000 for TEX bits
LL_MPU_TEX_LEVEL1	b001 for TEX bits
LL_MPU_TEX_LEVEL2	b010 for TEX bits
LL_MPU_TEX_LEVEL4	b100 for TEX bits

57 LL CRC Generic Driver

57.1 CRC Firmware driver API description

57.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name	**__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)**
Function description	Reset the CRC calculation unit.
Parameters	• **CRCx:** CRC Instance
Return values	• **None**
Notes	• If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC_INIT register, otherwise, reset Data Register to its default value.
Reference Manual to LL API cross reference:	• CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_SetPolynomialSize

Function name	**__STATIC_INLINE void LL_CRC_SetPolynomialSize (CRC_TypeDef * CRCx, uint32_t PolySize)**
Function description	Configure size of the polynomial.
Parameters	• **CRCx:** CRC Instance • **PolySize:** This parameter can be one of the following values: – LL_CRC_POLYLENGTH_32B – LL_CRC_POLYLENGTH_16B – LL_CRC_POLYLENGTH_8B – LL_CRC_POLYLENGTH_7B
Return values	• **None**
Reference Manual to LL API cross reference:	• CR POLYSIZE LL_CRC_SetPolynomialSize

LL_CRC_GetPolynomialSize

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (CRC_TypeDef * CRCx)**
Function description	Return size of the polynomial.
Parameters	• **CRCx:** CRC Instance
Return values	• **Returned:** value can be one of the following values: – LL_CRC_POLYLENGTH_32B – LL_CRC_POLYLENGTH_16B

– LL_CRC_POLYLENGTH_8B
– LL_CRC_POLYLENGTH_7B

Reference Manual to LL API cross reference:	• CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseMode

Function name	**__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)**
Function description	Configure the reversal of the bit order of the input data.
Parameters	• **CRCx:** CRC Instance • **ReverseMode:** This parameter can be one of the following values: – LL_CRC_INDATA_REVERSE_NONE – LL_CRC_INDATA_REVERSE_BYTE – LL_CRC_INDATA_REVERSE_HALFWORD – LL_CRC_INDATA_REVERSE_WORD
Return values	• **None**
Reference Manual to LL API cross reference:	• CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (CRC_TypeDef * CRCx)**
Function description	Return type of reversal for input data bit order.
Parameters	• **CRCx:** CRC Instance
Return values	• **Returned:** value can be one of the following values: – LL_CRC_INDATA_REVERSE_NONE – LL_CRC_INDATA_REVERSE_BYTE – LL_CRC_INDATA_REVERSE_HALFWORD – LL_CRC_INDATA_REVERSE_WORD
Reference Manual to LL API cross reference:	• CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name	**__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)**
Function description	Configure the reversal of the bit order of the Output data.
Parameters	• **CRCx:** CRC Instance • **ReverseMode:** This parameter can be one of the following values: – LL_CRC_OUTDATA_REVERSE_NONE – LL_CRC_OUTDATA_REVERSE_BIT

Return values	• **None**
Reference Manual to LL API cross reference:	• CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (CRC_TypeDef * CRCx)**
Function description	Configure the reversal of the bit order of the Output data.
Parameters	• **CRCx:** CRC Instance
Return values	• **Returned:** value can be one of the following values: – LL_CRC_OUTDATA_REVERSE_NONE – LL_CRC_OUTDATA_REVERSE_BIT
Reference Manual to LL API cross reference:	• CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name	**__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)**
Function description	Initialize the Programmable initial CRC value.
Parameters	• **CRCx:** CRC Instance • **InitCrc:** Value to be programmed in Programmable initial CRC value register
Return values	• **None**
Notes	• If the CRC size is less than 32 bits, the least significant bits are used to write the correct value • LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.
Reference Manual to LL API cross reference:	• INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetInitialData (CRC_TypeDef * CRCx)**
Function description	Return current Initial CRC value.
Parameters	• **CRCx:** CRC Instance
Return values	• **Value:** programmed in Programmable initial CRC value register
Notes	• If the CRC size is less than 32 bits, the least significant bits are used to read the correct value
Reference Manual to	• INIT INIT LL_CRC_GetInitialData

LL API cross reference:

LL_CRC_SetPolynomialCoef

Function name	**__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)**
Function description	Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).
Parameters	• **CRCx:** CRC Instance • **PolynomCoef:** Value to be programmed in Programmable Polynomial value register
Return values	• **None**
Notes	• LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter. • Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
Reference Manual to LL API cross reference:	• POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (CRC_TypeDef * CRCx)**
Function description	Return current Programmable polynomial value.
Parameters	• **CRCx:** CRC Instance
Return values	• **Value:** programmed in Programmable Polynomial value register
Notes	• Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
Reference Manual to LL API cross reference:	• POL POL LL_CRC_GetPolynomialCoef

LL_CRC_FeedData32

Function name	**__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)**
Function description	Write given 32-bit data to the CRC calculator.
Parameters	• **CRCx:** CRC Instance • **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values	• **None**
Reference Manual to LL API cross reference:	• DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name	**__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)**
Function description	Write given 16-bit data to the CRC calculator.
Parameters	• **CRCx:** CRC Instance • **InData:** 16 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DR DR LL_CRC_FeedData16

LL_CRC_FeedData8

Function name	**__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)**
Function description	Write given 8-bit data to the CRC calculator.
Parameters	• **CRCx:** CRC Instance • **InData:** 8 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DR DR LL_CRC_FeedData8

LL_CRC_ReadData32

Function name	**__STATIC_INLINE uint32_t LL_CRC_ReadData32 (CRC_TypeDef * CRCx)**
Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (32 bits).
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name	**__STATIC_INLINE uint16_t LL_CRC_ReadData16 (CRC_TypeDef * CRCx)**

Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (16 bits).
Notes	• This function is expected to be used in a 16 bits CRC polynomial size context.
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name	**__STATIC_INLINE uint8_t LL_CRC_ReadData8 (CRC_TypeDef * CRCx)**
Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (8 bits).
Notes	• This function is expected to be used in a 8 bits CRC polynomial size context.
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name	**__STATIC_INLINE uint8_t LL_CRC_ReadData7 (CRC_TypeDef * CRCx)**
Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (7 bits).
Notes	• This function is expected to be used in a 7 bits CRC polynomial size context.
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name	**__STATIC_INLINE uint32_t LL_CRC_Read_IDR (CRC_TypeDef * CRCx)**
Function description	Return data stored in the Independent Data(IDR) register.
Parameters	• **CRCx:** CRC Instance
Return values	• **Value:** stored in CRC_IDR register (General-purpose 8-bit

data register).

Notes	• This register can be used as a temporary storage location for one byte.
Reference Manual to LL API cross reference:	• IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name	**__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)**
Function description	Store data in the Independent Data(IDR) register.
Parameters	• **CRCx:** CRC Instance • **InData:** value to be stored in CRC_IDR register (8-bit) between between Min_Data=0 and Max_Data=0xFF
Return values	• **None**
Notes	• This register can be used as a temporary storage location for one byte.
Reference Manual to LL API cross reference:	• IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name	**ErrorStatus LL_CRC_DeInit (CRC_TypeDef * CRCx)**
Function description	De-initialize CRC registers (Registers restored to their default values).
Parameters	• **CRCx:** CRC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: CRC registers are de-initialized – ERROR: CRC registers are not de-initialized

57.2 CRC Firmware driver defines

57.2.1 CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_INITVALUE	Default CRC computation initialization value

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32_POLY	Default CRC generating polynomial value

Input Data Reverse

LL_CRC_INDATA_REVERSE_NONE	Input Data bit order not affected
LL_CRC_INDATA_REVERSE_BYTE	Input Data bit reversal done by byte
LL_CRC_INDATA_REVERSE_HALFWORD	Input Data bit reversal done by half-word

LL_CRC_INDATA_REVERSE_WORD	Input Data bit reversal done by word

Output Data Reverse

LL_CRC_OUTDATA_REVERSE_NONE	Output Data bit order not affected
LL_CRC_OUTDATA_REVERSE_BIT	Output Data bit reversal done by bit

Polynomial length

LL_CRC_POLYLENGTH_32B	32 bits Polynomial size
LL_CRC_POLYLENGTH_16B	16 bits Polynomial size
LL_CRC_POLYLENGTH_8B	8 bits Polynomial size
LL_CRC_POLYLENGTH_7B	7 bits Polynomial size

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

58 LL CRS Generic Driver

58.1 CRS Firmware driver API description

58.1.1 Detailed description of functions

LL_CRS_EnableFreqErrorCounter

Function name	**__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter (void)**
Function description	Enable Frequency error counter.
Return values	• **None**
Notes	• When this bit is set, the CRS_CFGR register is write-protected and cannot be modified
Reference Manual to LL API cross reference:	• CR CEN LL_CRS_EnableFreqErrorCounter

LL_CRS_DisableFreqErrorCounter

Function name	**__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter (void)**
Function description	Disable Frequency error counter.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR CEN LL_CRS_DisableFreqErrorCounter

LL_CRS_IsEnabledFreqErrorCounter

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter (void)**
Function description	Check if Frequency error counter is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR CEN LL_CRS_IsEnabledFreqErrorCounter

LL_CRS_EnableAutoTrimming

Function name	**__STATIC_INLINE void LL_CRS_EnableAutoTrimming (void)**
Function description	Enable Automatic trimming counter.
Return values	• **None**
Reference Manual to LL API cross	• CR AUTOTRIMEN LL_CRS_EnableAutoTrimming

reference:

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	**__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)**
Function description	Set counter reload value.
Parameters	• **Value:** a number between Min_Data = 0 and Max_Data = 0xFFFF
Return values	• **None**
Notes	• Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro __LL_CRS_CALC_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)
Reference Manual to LL API cross reference:	• CFGR RELOAD LL_CRS_SetReloadCounter

LL_CRS_GetReloadCounter

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void)**
Function description	Get counter reload value.
Return values	• **a:** number between Min_Data = 0 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• CFGR RELOAD LL_CRS_GetReloadCounter

LL_CRS_SetFreqErrorLimit

Function name	**__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)**
Function description	Set frequency error limit.
Parameters	• **Value:** a number between Min_Data = 0 and Max_Data = 255
Return values	• **None**
Notes	• Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT
Reference Manual to LL API cross reference:	• CFGR FELIM LL_CRS_SetFreqErrorLimit

LL_CRS_GetFreqErrorLimit

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void)**
Function description	Get frequency error limit.
Return values	• **A:** number between Min_Data = 0 and Max_Data = 255
Reference Manual to LL API cross	• CFGR FELIM LL_CRS_GetFreqErrorLimit

reference:

LL_CRS_SetSyncDivider

Function name	**__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)**
Function description	Set division factor for SYNC signal.
Parameters	• **Divider:** This parameter can be one of the following values: – LL_CRS_SYNC_DIV_1 – LL_CRS_SYNC_DIV_2 – LL_CRS_SYNC_DIV_4 – LL_CRS_SYNC_DIV_8 – LL_CRS_SYNC_DIV_16 – LL_CRS_SYNC_DIV_32 – LL_CRS_SYNC_DIV_64 – LL_CRS_SYNC_DIV_128
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR SYNCDIV LL_CRS_SetSyncDivider

LL_CRS_GetSyncDivider

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void)**
Function description	Get division factor for SYNC signal.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_SYNC_DIV_1 – LL_CRS_SYNC_DIV_2 – LL_CRS_SYNC_DIV_4 – LL_CRS_SYNC_DIV_8 – LL_CRS_SYNC_DIV_16 – LL_CRS_SYNC_DIV_32 – LL_CRS_SYNC_DIV_64 – LL_CRS_SYNC_DIV_128
Reference Manual to LL API cross reference:	• CFGR SYNCDIV LL_CRS_GetSyncDivider

LL_CRS_SetSyncSignalSource

Function name	**__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)**
Function description	Set SYNC signal source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_CRS_SYNC_SOURCE_GPIO – LL_CRS_SYNC_SOURCE_LSE – LL_CRS_SYNC_SOURCE_USB
Return values	• **None**

Reference Manual to LL API cross reference:	• CFGR SYNCSRC LL_CRS_SetSyncSignalSource

LL_CRS_GetSyncSignalSource

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource (void)**
Function description	Get SYNC signal source.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_SYNC_SOURCE_GPIO – LL_CRS_SYNC_SOURCE_LSE – LL_CRS_SYNC_SOURCE_USB
Reference Manual to LL API cross reference:	• CFGR SYNCSRC LL_CRS_GetSyncSignalSource

LL_CRS_SetSyncPolarity

Function name	**__STATIC_INLINE void LL_CRS_SetSyncPolarity (uint32_t Polarity)**
Function description	Set input polarity for the SYNC signal source.
Parameters	• **Polarity:** This parameter can be one of the following values: – LL_CRS_SYNC_POLARITY_RISING – LL_CRS_SYNC_POLARITY_FALLING
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR SYNCPOL LL_CRS_SetSyncPolarity

LL_CRS_GetSyncPolarity

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity (void)**
Function description	Get input polarity for the SYNC signal source.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_SYNC_POLARITY_RISING – LL_CRS_SYNC_POLARITY_FALLING
Reference Manual to LL API cross reference:	• CFGR SYNCPOL LL_CRS_GetSyncPolarity

LL_CRS_ConfigSynchronization

Function name	**__STATIC_INLINE void LL_CRS_ConfigSynchronization (uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)**
Function description	Configure CRS for the synchronization.

Parameters	• **HSI48CalibrationValue:** a number between Min_Data = 0 and Max_Data = 63 • **ErrorLimitValue:** a number between Min_Data = 0 and Max_Data = 0xFFFF • **ReloadValue:** a number between Min_Data = 0 and Max_Data = 255 • **Settings:** This parameter can be a combination of the following values: – LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128 – LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB – LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING
Return values	• **None**
Reference Manual to LL API cross reference:	• CR TRIM LL_CRS_ConfigSynchronization • CFGR RELOAD LL_CRS_ConfigSynchronization • CFGR FELIM LL_CRS_ConfigSynchronization • CFGR SYNCDIV LL_CRS_ConfigSynchronization • CFGR SYNCSRC LL_CRS_ConfigSynchronization • CFGR SYNCPOL LL_CRS_ConfigSynchronization

LL_CRS_GenerateEvent_SWSYNC

Function name	**__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC (void)**
Function description	Generate software SYNC event.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR SWSYNC LL_CRS_GenerateEvent_SWSYNC

LL_CRS_GetFreqErrorDirection

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection (void)**
Function description	Get the frequency error direction latched in the time of the last SYNC event.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_FREQ_ERROR_DIR_UP – LL_CRS_FREQ_ERROR_DIR_DOWN
Reference Manual to LL API cross reference:	• ISR FEDIR LL_CRS_GetFreqErrorDirection

LL_CRS_GetFreqErrorCapture

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void)**
Function description	Get the frequency error counter value latched in the time of the last SYNC event.
Return values	• **A:** number between Min_Data = 0x0000 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• ISR FECAP LL_CRS_GetFreqErrorCapture

LL_CRS_IsActiveFlag_SYNCOK

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void)**
Function description	Check if SYNC event OK signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK

LL_CRS_IsActiveFlag_SYNCWARN

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void)**
Function description	Check if SYNC warning signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN

LL_CRS_IsActiveFlag_ERR

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void)**
Function description	Check if Synchronization or trimming error signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ERRF LL_CRS_IsActiveFlag_ERR

LL_CRS_IsActiveFlag_ESYNC

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void)**
Function description	Check if Expected SYNC signal occurred or not.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• ISR ESYNCF LL_CRS_IsActiveFlag_ESYNC

LL_CRS_IsActiveFlag_SYNCERR

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR (void)**
Function description	Check if SYNC error signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCERR LL_CRS_IsActiveFlag_SYNCERR

LL_CRS_IsActiveFlag_SYNCMISS

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS (void)**
Function description	Check if SYNC missed error signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCMISS LL_CRS_IsActiveFlag_SYNCMISS

LL_CRS_IsActiveFlag_TRIMOVF

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF (void)**
Function description	Check if Trimming overflow or underflow occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TRIMOVF LL_CRS_IsActiveFlag_TRIMOVF

LL_CRS_ClearFlag_SYNCOK

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void)**
Function description	Clear the SYNC event OK flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR SYNCOKC LL_CRS_ClearFlag_SYNCOK

LL_CRS_ClearFlag_SYNCWARN

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void)**

Function description	Clear the SYNC warning flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR SYNCWARNC LL_CRS_ClearFlag_SYNCWARN

LL_CRS_ClearFlag_ERR

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void)**
Function description	Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ERRC LL_CRS_ClearFlag_ERR

LL_CRS_ClearFlag_ESYNC

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void)**
Function description	Clear Expected SYNC flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ESYNCC LL_CRS_ClearFlag_ESYNC

LL_CRS_EnableIT_SYNCOK

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void)**
Function description	Enable SYNC event OK interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR SYNCOKIE LL_CRS_EnableIT_SYNCOK

LL_CRS_DisableIT_SYNCOK

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void)**
Function description	Disable SYNC event OK interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR SYNCOKIE LL_CRS_DisableIT_SYNCOK

LL_CRS_IsEnabledIT_SYNCOK

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK**

	(void)
Function description	Check if SYNC event OK interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

LL_CRS_EnableIT_SYNCWARN

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void)**
Function description	Enable SYNC warning interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

LL_CRS_DisableIT_SYNCWARN

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void)**
Function description	Disable SYNC warning interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

LL_CRS_IsEnabledIT_SYNCWARN

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void)**
Function description	Check if SYNC warning interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

LL_CRS_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_ERR (void)**
Function description	Enable Synchronization or trimming error interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR ERRIE LL_CRS_EnableIT_ERR

LL_CRS_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_ERR (void)**
Function description	Disable Synchronization or trimming error interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR ERRIE LL_CRS_DisableIT_ERR

LL_CRS_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void)**
Function description	Check if Synchronization or trimming error interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ERRIE LL_CRS_IsEnabledIT_ERR

LL_CRS_EnableIT_ESYNC

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void)**
Function description	Enable Expected SYNC interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR ESYNCIE LL_CRS_EnableIT_ESYNC

LL_CRS_DisableIT_ESYNC

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void)**
Function description	Disable Expected SYNC interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR ESYNCIE LL_CRS_DisableIT_ESYNC

LL_CRS_IsEnabledIT_ESYNC

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void)**
Function description	Check if Expected SYNC interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ESYNCIE LL_CRS_IsEnabledIT_ESYNC

LL_CRS_DeInit

Function name	**ErrorStatus LL_CRS_DeInit (void)**
Function description	De-Initializes CRS peripheral registers to their default reset values.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: CRS registers are de-initialized – ERROR: not applicable

58.2 CRS Firmware driver defines

58.2.1 CRS

Default Values

LL_CRS_RELOADVALUE_DEFAULT

Notes:

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL_CRS_ERRORLIMIT_DEFAULT

LL_CRS_HSI48CALIBRATION_DEFAULT

Notes:

- The default value is 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_DIR_UP	Upcounting direction, the actual frequency is above the target
LL_CRS_FREQ_ERROR_DIR_DOWN	Downcounting direction, the actual frequency is below the target

Get Flags Defines

LL_CRS_ISR_SYNCOKF

LL_CRS_ISR_SYNCWARNF

LL_CRS_ISR_ERRF

LL_CRS_ISR_ESYNCF

LL_CRS_ISR_SYNCERR

LL_CRS_ISR_SYNCMISS

LL_CRS_ISR_TRIMOVF

IT Defines

LL_CRS_CR_SYNCOKIE

LL_CRS_CR_SYNCWARNIE

LL_CRS_CR_ERRIE

LL_CRS_CR_ESYNCIE

Synchronization Signal Divider

LL_CRS_SYNC_DIV_1	Synchro Signal not divided (default)
LL_CRS_SYNC_DIV_2	Synchro Signal divided by 2
LL_CRS_SYNC_DIV_4	Synchro Signal divided by 4
LL_CRS_SYNC_DIV_8	Synchro Signal divided by 8
LL_CRS_SYNC_DIV_16	Synchro Signal divided by 16
LL_CRS_SYNC_DIV_32	Synchro Signal divided by 32
LL_CRS_SYNC_DIV_64	Synchro Signal divided by 64
LL_CRS_SYNC_DIV_128	Synchro Signal divided by 128

Synchronization Signal Polarity

LL_CRS_SYNC_POLARITY_RISING	Synchro Active on rising edge (default)
LL_CRS_SYNC_POLARITY_FALLING	Synchro Active on falling edge

Synchronization Signal Source

LL_CRS_SYNC_SOURCE_GPIO	Synchro Signal 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_CALCULATE_RELOADVALUE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- Reload: value (in Hz)

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

Common Write and read registers Macros

LL_CRS_WriteReg

Description:

- Write a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRS_ReadReg

Description:

- Read a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be read

Return value:

- Register: value

59 LL DAC Generic Driver

59.1 DAC Firmware driver registers structures

59.1.1 LL_DAC_InitTypeDef

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()**.

59.2 DAC Firmware driver API description

59.2.1 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	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following

	values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **TriggerSource:** This parameter can be one of the following values: – LL_DAC_TRIG_SOFTWARE – LL_DAC_TRIG_EXT_TIM2_TRGO – LL_DAC_TRIG_EXT_TIM3_TRGO – LL_DAC_TRIG_EXT_TIM3_CH3 – LL_DAC_TRIG_EXT_TIM6_TRGO – LL_DAC_TRIG_EXT_TIM7_TRGO – LL_DAC_TRIG_EXT_TIM21_TRGO – LL_DAC_TRIG_EXT_EXTI_LINE9
Return values	• **None**
Notes	• For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger(). • To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded. • Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	• CR TSEL1 LL_DAC_SetTriggerSource • CR TSEL2 LL_DAC_SetTriggerSource

LL_DAC_GetTriggerSource

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the conversion trigger source for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_TRIG_SOFTWARE – LL_DAC_TRIG_EXT_TIM2_TRGO – LL_DAC_TRIG_EXT_TIM3_TRGO – LL_DAC_TRIG_EXT_TIM3_CH3 – LL_DAC_TRIG_EXT_TIM6_TRGO – LL_DAC_TRIG_EXT_TIM7_TRGO – LL_DAC_TRIG_EXT_TIM21_TRGO – LL_DAC_TRIG_EXT_EXTI_LINE9
Notes	• For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger(). • Availability of parameters of trigger sources from timer

	depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	• CR TSEL1 LL_DAC_GetTriggerSource • CR TSEL2 LL_DAC_GetTriggerSource

LL_DAC_SetWaveAutoGeneration

Function name	**__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)**
Function description	Set the waveform automatic generation mode for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **WaveAutoGeneration:** This parameter can be one of the following values: – LL_DAC_WAVE_AUTO_GENERATION_NONE – LL_DAC_WAVE_AUTO_GENERATION_NOISE – LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE
Return values	• **None**
Reference Manual to LL API cross reference:	• CR WAVE1 LL_DAC_SetWaveAutoGeneration • CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_GetWaveAutoGeneration

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the waveform automatic generation mode for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_WAVE_AUTO_GENERATION_NONE – LL_DAC_WAVE_AUTO_GENERATION_NOISE – LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE
Reference Manual to LL API cross reference:	• CR WAVE1 LL_DAC_GetWaveAutoGeneration • CR WAVE2 LL_DAC_GetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name	**__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)**
Function description	Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **NoiseLFSRMask:** This parameter can be one of the following values: – LL_DAC_NOISE_LFSR_UNMASK_BIT0 – LL_DAC_NOISE_LFSR_UNMASK_BITS1_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS2_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS3_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS4_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS5_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS6_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS7_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS8_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS9_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS10_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS11_0
Return values	• **None**
Notes	• For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration(). • This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).
Reference Manual to LL API cross reference:	• CR MAMP1 LL_DAC_SetWaveNoiseLFSR • CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1

– LL_DAC_CHANNEL_2 (1)

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveNoiseLFSR
- CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name

__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)

Function description

Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)
- **TriangleAmplitude:** This parameter can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Return values

- **None**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().

	• This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).
Reference Manual to LL API cross reference:	• CR MAMP1 LL_DAC_SetWaveTriangleAmplitude • CR MAMP2 LL_DAC_SetWaveTriangleAmplitude

LL_DAC_GetWaveTriangleAmplitude

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_TRIANGLE_AMPLITUDE_1 – LL_DAC_TRIANGLE_AMPLITUDE_3 – LL_DAC_TRIANGLE_AMPLITUDE_7 – LL_DAC_TRIANGLE_AMPLITUDE_15 – LL_DAC_TRIANGLE_AMPLITUDE_31 – LL_DAC_TRIANGLE_AMPLITUDE_63 – LL_DAC_TRIANGLE_AMPLITUDE_127 – LL_DAC_TRIANGLE_AMPLITUDE_255 – LL_DAC_TRIANGLE_AMPLITUDE_511 – LL_DAC_TRIANGLE_AMPLITUDE_1023 – LL_DAC_TRIANGLE_AMPLITUDE_2047 – LL_DAC_TRIANGLE_AMPLITUDE_4095
Reference Manual to LL API cross reference:	• CR MAMP1 LL_DAC_GetWaveTriangleAmplitude • CR MAMP2 LL_DAC_GetWaveTriangleAmplitude

LL_DAC_SetOutputBuffer

Function name	**__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)**
Function description	Set the output buffer for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)

	• **OutputBuffer:** This parameter can be one of the following values: – LL_DAC_OUTPUT_BUFFER_ENABLE – LL_DAC_OUTPUT_BUFFER_DISABLE
Return values	• **None**
Reference Manual to LL API cross reference:	• CR BOFF1 LL_DAC_SetOutputBuffer • CR BOFF2 LL_DAC_SetOutputBuffer

LL_DAC_GetOutputBuffer

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the output buffer state for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_OUTPUT_BUFFER_ENABLE – LL_DAC_OUTPUT_BUFFER_DISABLE
Reference Manual to LL API cross reference:	• CR BOFF1 LL_DAC_GetOutputBuffer • CR BOFF2 LL_DAC_GetOutputBuffer

LL_DAC_EnableDMAReq

Function name	**__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Enable DAC DMA transfer request of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **None**
Notes	• To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().
Reference Manual to LL API cross reference:	• CR DMAEN1 LL_DAC_EnableDMAReq • CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name	**__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Disable DAC DMA transfer request of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **None**
Notes	• To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().
Reference Manual to LL API cross reference:	• CR DMAEN1 LL_DAC_DisableDMAReq • CR DMAEN2 LL_DAC_DisableDMAReq

LL_DAC_IsDMAReqEnabled

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get DAC DMA transfer request state of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR DMAEN1 LL_DAC_IsDMAReqEnabled • CR DMAEN2 LL_DAC_IsDMAReqEnabled

LL_DAC_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)**
Function description	Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)

- **Register:** This parameter can be one of the following values:
  - LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED
  - LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED
  - LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

Return values
- **DAC:** register address

Notes
- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:
- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name
__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Enable DAC selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

Return values
- **None**

Notes
- After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:
- CR EN1 LL_DAC_Enable
- CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name
__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Disable DAC selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following

	values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR EN1 LL_DAC_Disable • CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsEnabled (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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **None**
Notes	• - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().
Reference Manual to LL API cross	• CR TEN1 LL_DAC_EnableTrigger

reference:	• CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name	**__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Disable DAC trigger of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR TEN1 LL_DAC_DisableTrigger • CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get DAC trigger state of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR TEN1 LL_DAC_IsTriggerEnabled • CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name	**__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Trig DAC conversion by software for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can a combination of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)

Return values	• **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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned • DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)**
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **Data:** Value between Min_Data=0x000 and

Max_Data=0xFFF

Return values	• **None**
Reference Manual to LL API cross reference:	• DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned • DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)**
Function description	Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **Data:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned • DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)**
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF • **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned • DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef ***

	DACx, uint32_t DataChannel1, uint32_t DataChannel2)
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF • **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned • DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)**
Function description	Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x00 and Max_Data=0xFF • **DataChannel2:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned • DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name	**__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Retrieve output data currently generated for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()":

	LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).
Reference Manual to LL API cross reference:	• DOR1 DACC1DOR LL_DAC_RetrieveOutputData • DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_IsActiveFlag_DMAUDR1

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1 (DAC_TypeDef * DACx)**
Function description	Get DAC underrun flag for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1

LL_DAC_IsActiveFlag_DMAUDR2

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2 (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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1

– LL_DAC_CHANNEL_2 (1)
- **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure

Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DAC registers are initialized – ERROR: DAC registers are not initialized
Notes	• 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**

59.3 DAC Firmware driver defines

59.3.1 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_VOLTAGE_SETTLING_US	Delay for DAC channel voltage settling time from DAC channel startup (transition from disable to enable)
LL_DAC_DELAY_VOLTAGE_SETTLING_US	Delay for DAC channel voltage settling time

DAC interruptions

LL_DAC_IT_DMAUDRIE1	DAC channel 1 interruption DMA underrun
LL_DAC_IT_DMAUDRIE2	DAC channel 2 interruption DMA underrun

DAC channel output buffer

LL_DAC_OUTPUT_BUFFER_ENABLE	The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption
LL_DAC_OUTPUT_BUFFER_DISABLE	The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED	DAC channel data holding register 12 bits right aligned
LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED	DAC channel data holding register 12 bits left aligned
LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED	DAC channel data holding register 8 bits right aligned

DAC channel output resolution

LL_DAC_RESOLUTION_12B	DAC channel resolution 12 bits
LL_DAC_RESOLUTION_8B	DAC channel resolution 8 bits

DAC trigger source

LL_DAC_TRIG_SOFTWARE	DAC channel conversion trigger internal (SW start)
LL_DAC_TRIG_EXT_TIM2_TRGO	DAC channel conversion trigger from external IP: TIM2 TRGO.
LL_DAC_TRIG_EXT_TIM3_TRGO	DAC channel conversion trigger from external IP: TIM3 TRGO.
LL_DAC_TRIG_EXT_TIM3_CH3	DAC channel conversion trigger from external IP: TIM3 CH3 event.
LL_DAC_TRIG_EXT_TIM6_TRGO	DAC channel conversion trigger from external IP: TIM6 TRGO.
LL_DAC_TRIG_EXT_TIM7_TRGO	DAC channel conversion trigger from external IP: TIM7 TRGO.
LL_DAC_TRIG_EXT_TIM21_TRGO	DAC channel conversion trigger from external IP: TIM21 TRGO.
LL_DAC_TRIG_EXT_EXTI_LINE9	DAC channel conversion trigger from external IP: external interrupt line 9.

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GENERATION_NONE	DAC channel wave auto generation mode disabled.
LL_DAC_WAVE_AUTO_GENERATION_NOISE	DAC channel wave auto generation mode enabled, set generated noise waveform.
LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE	DAC channel wave auto generation mode enabled, set generated triangle waveform.

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UNMASK_BIT0	Noise wave generation, unmask LFSR bit0, for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS1_0	Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS2_0	Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS3_0	Noise wave generation, unmask LFSR

	bits[3:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS4_0	Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS5_0	Noise wave generation, unmask LFSR bits[5:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS6_0	Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS7_0	Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS8_0	Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS9_0	Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS10_0	Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS11_0	Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPLITUDE_1	Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_3	Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_7	Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_15	Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_31	Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_63	Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_127	Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_255	Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_511	Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_1023	Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_2047	Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_4095	Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

__LL_DAC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get DAC channel number in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (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_SCALE

Description:

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

Parameters:

- __DAC_RESOLUTION__: This parameter can be one of the following values:
  – LL_DAC_RESOLUTION_12B
  – LL_DAC_RESOLUTION_8B

Return value:

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

Notes:

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

__LL_DAC_CALC_VOLTAGE_TO_DATA

Description:

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

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __DAC_VOLTAGE__: Voltage to be generated by DAC channel (unit: mVolt).
- __DAC_RESOLUTION__: This parameter can be one of the following values:
  – LL_DAC_RESOLUTION_12B
  – LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be read

Return value:

- Register: value

60 LL DMA Generic Driver

60.1 DMA Firmware driver registers structures

60.1.1 LL_DMA_InitTypeDef

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()**.

60.2 DMA Firmware driver API description

60.2.1 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	• **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_DisableChannel

LL_DMA_IsEnabledChannel

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if DMA channel is enabled or disabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ConfigTransfer

Function name	**__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)**
Function description	Configure all parameters link to DMA transfer.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **Configuration:** This parameter must be a combination of all

the following values:
- – LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY
- – LL_DMA_MODE_NORMAL or LL_DMA_MODE_CIRCULAR
- – LL_DMA_PERIPH_INCREMENT or LL_DMA_PERIPH_NOINCREMENT
- – LL_DMA_MEMORY_INCREMENT or LL_DMA_MEMORY_NOINCREMENT
- – LL_DMA_PDATAALIGN_BYTE or LL_DMA_PDATAALIGN_HALFWORD or LL_DMA_PDATAALIGN_WORD
- – LL_DMA_MDATAALIGN_BYTE or LL_DMA_MDATAALIGN_HALFWORD or LL_DMA_MDATAALIGN_WORD
- – LL_DMA_PRIORITY_LOW or LL_DMA_PRIORITY_MEDIUM or LL_DMA_PRIORITY_HIGH or LL_DMA_PRIORITY_VERYHIGH

Return values
- **None**

Reference Manual to LL API cross reference:
- CCR DIR LL_DMA_ConfigTransfer
- CCR MEM2MEM LL_DMA_ConfigTransfer
- CCR CIRC LL_DMA_ConfigTransfer
- CCR PINC LL_DMA_ConfigTransfer
- CCR MINC LL_DMA_ConfigTransfer
- CCR PSIZE LL_DMA_ConfigTransfer
- CCR MSIZE LL_DMA_ConfigTransfer
- CCR PL LL_DMA_ConfigTransfer

LL_DMA_SetDataTransferDirection

Function name
__STATIC_INLINE void LL_DMA_SetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)

Function description
Set Data transfer direction (read from peripheral or from memory).

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - – LL_DMA_CHANNEL_1
  - – LL_DMA_CHANNEL_2
  - – LL_DMA_CHANNEL_3
  - – LL_DMA_CHANNEL_4
  - – LL_DMA_CHANNEL_5
  - – LL_DMA_CHANNEL_6
  - – LL_DMA_CHANNEL_7
- **Direction:** This parameter can be one of the following values:
  - – LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - – LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - – LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values
- **None**

Reference Manual to LL API cross reference:	• CCR DIR LL_DMA_SetDataTransferDirection • CCR MEM2MEM LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Data transfer direction (read from peripheral or from memory).
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_DIRECTION_PERIPH_TO_MEMORY – LL_DMA_DIRECTION_MEMORY_TO_PERIPH – LL_DMA_DIRECTION_MEMORY_TO_MEMORY
Reference Manual to LL API cross reference:	• CCR DIR LL_DMA_GetDataTransferDirection • CCR MEM2MEM LL_DMA_GetDataTransferDirection

LL_DMA_SetMode

Function name	**__STATIC_INLINE void LL_DMA_SetMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Mode)**
Function description	Set DMA mode circular or normal.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **Mode:** This parameter can be one of the following values: – LL_DMA_MODE_NORMAL – LL_DMA_MODE_CIRCULAR
Return values	• **None**
Notes	• The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel.
Reference Manual to LL API cross	• CCR CIRC LL_DMA_SetMode

reference:

LL_DMA_GetMode

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get DMA mode circular or normal.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_MODE_NORMAL – LL_DMA_MODE_CIRCULAR
Reference Manual to LL API cross reference:	• CCR CIRC LL_DMA_GetMode

LL_DMA_SetPeriphIncMode

Function name	**__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)**
Function description	Set Peripheral increment mode.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **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_GetPeriphIncMode

LL_DMA_SetMemoryIncMode

Function name	**__STATIC_INLINE void LL_DMA_SetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstIncMode)**
Function description	Set Memory increment mode.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **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_GetMemoryIncMode

LL_DMA_SetPeriphSize

Function name	**__STATIC_INLINE void LL_DMA_SetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcDataSize)**
Function description	Set Peripheral size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **PeriphOrM2MSrcDataSize:** This parameter can be one of the following values: – LL_DMA_PDATAALIGN_BYTE – LL_DMA_PDATAALIGN_HALFWORD – LL_DMA_PDATAALIGN_WORD
Return values	• **None**
Reference Manual to LL API cross reference:	• CCR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Peripheral size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values:

– LL_DMA_CHANNEL_1
– LL_DMA_CHANNEL_2
– LL_DMA_CHANNEL_3
– LL_DMA_CHANNEL_4
– LL_DMA_CHANNEL_5
– LL_DMA_CHANNEL_6
– LL_DMA_CHANNEL_7

Return values	• **Returned:** value can be one of the following values: – LL_DMA_PDATAALIGN_BYTE – LL_DMA_PDATAALIGN_HALFWORD – LL_DMA_PDATAALIGN_WORD
Reference Manual to LL API cross reference:	• CCR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name	**__STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)**
Function description	Set Memory size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **MemoryOrM2MDstDataSize:** This parameter can be one of the following values: – LL_DMA_MDATAALIGN_BYTE – LL_DMA_MDATAALIGN_HALFWORD – LL_DMA_MDATAALIGN_WORD
Return values	• **None**
Reference Manual to LL API cross reference:	• CCR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Memory size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2

	– LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_MDATAALIGN_BYTE – LL_DMA_MDATAALIGN_HALFWORD – LL_DMA_MDATAALIGN_WORD
Reference Manual to LL API cross reference:	• CCR MSIZE LL_DMA_GetMemorySize

LL_DMA_SetChannelPriorityLevel

Function name	**__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)**
Function description	Set Channel priority level.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **Priority:** This parameter can be one of the following values: – LL_DMA_PRIORITY_LOW – LL_DMA_PRIORITY_MEDIUM – LL_DMA_PRIORITY_HIGH – LL_DMA_PRIORITY_VERYHIGH
Return values	• **None**
Reference Manual to LL API cross reference:	• CCR PL LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Channel priority level.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5

	– LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_PRIORITY_LOW – LL_DMA_PRIORITY_MEDIUM – LL_DMA_PRIORITY_HIGH – LL_DMA_PRIORITY_VERYHIGH
Reference Manual to LL API cross reference:	• CCR PL LL_DMA_GetChannelPriorityLevel

LL_DMA_SetDataLength

Function name	**__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t NbData)**
Function description	Set Number of data to transfer.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **NbData:** Between Min_Data = 0 and Max_Data = 0x0000FFFF
Return values	• **None**
Notes	• This action has no effect if channel is enabled.
Reference Manual to LL API cross reference:	• CNDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Number of data to transfer.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes	• Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.
Reference Manual to LL API cross reference:	• CNDTR NDT LL_DMA_GetDataLength

LL_DMA_ConfigAddresses

Function name	**__STATIC_INLINE void LL_DMA_ConfigAddresses (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)**
Function description	Configure the Source and Destination addresses.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **SrcAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF • **DstAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF • **Direction:** This parameter can be one of the following values: – LL_DMA_DIRECTION_PERIPH_TO_MEMORY – LL_DMA_DIRECTION_MEMORY_TO_PERIPH – LL_DMA_DIRECTION_MEMORY_TO_MEMORY
Return values	• **None**
Notes	• Each IP using DMA provides an API to get directly the register adress (LL_PPP_DMA_GetRegAddr)
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_ConfigAddresses • CMAR MA LL_DMA_ConfigAddresses

LL_DMA_SetMemoryAddress

Function name	**__STATIC_INLINE void LL_DMA_SetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)**
Function description	Set the Memory address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4

- – LL_DMA_CHANNEL_5
  - – LL_DMA_CHANNEL_6
  - – LL_DMA_CHANNEL_7
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values	• **None**
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
Reference Manual to LL API cross reference:	• CMAR MA LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name	**__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphAddress)**
Function description	Set the Peripheral address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **PeriphAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	• **None**
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Memory address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3

- LL_DMA_CHANNEL_4
- LL_DMA_CHANNEL_5
- LL_DMA_CHANNEL_6
- LL_DMA_CHANNEL_7

Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
Reference Manual to LL API cross reference:	• CMAR MA LL_DMA_GetMemoryAddress

LL_DMA_GetPeriphAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Peripheral address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name	**__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)**
Function description	Set the Memory to Memory Source address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6

	– LL_DMA_CHANNEL_7 • **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	• **None**
Notes	• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_SetM2MSrcAddress

LL_DMA_SetM2MDstAddress

Function name	**__STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)**
Function description	Set the Memory to Memory Destination address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	• **None**
Notes	• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
Reference Manual to LL API cross reference:	• CMAR MA LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get the Memory to Memory Source address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7

Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_GetM2MSrcAddress

LL_DMA_GetM2MDstAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get the Memory to Memory Destination address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
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	• **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 • **PeriphRequest:** This parameter can be one of the following values: – 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	• **None**
Notes	• Please refer to Reference Manual to get the available mapping of Request value link to Channel Selection.
Reference Manual to LL API cross reference:	• 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	**__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get DMA request for DMA instance on Channel x.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_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:	• 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	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF1 LL_DMA_IsActiveFlag_GI1

LL_DMA_IsActiveFlag_GI2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF2 LL_DMA_IsActiveFlag_GI2

LL_DMA_IsActiveFlag_GI3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF3 LL_DMA_IsActiveFlag_GI3

LL_DMA_IsActiveFlag_GI4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF4 LL_DMA_IsActiveFlag_GI4

LL_DMA_IsActiveFlag_GI5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF5 LL_DMA_IsActiveFlag_GI5

LL_DMA_IsActiveFlag_GI6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF6 LL_DMA_IsActiveFlag_GI6

LL_DMA_IsActiveFlag_GI7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF7 LL_DMA_IsActiveFlag_GI7

LL_DMA_IsActiveFlag_TC1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF1 LL_DMA_IsActiveFlag_TC1

LL_DMA_IsActiveFlag_TC2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF2 LL_DMA_IsActiveFlag_TC2

LL_DMA_IsActiveFlag_TC3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF3 LL_DMA_IsActiveFlag_TC3

LL_DMA_IsActiveFlag_TC4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF4 LL_DMA_IsActiveFlag_TC4

LL_DMA_IsActiveFlag_TC5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF5 LL_DMA_IsActiveFlag_TC5

LL_DMA_IsActiveFlag_TC6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF6 LL_DMA_IsActiveFlag_TC6

LL_DMA_IsActiveFlag_TC7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF7 LL_DMA_IsActiveFlag_TC7

LL_DMA_IsActiveFlag_HT1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF1 LL_DMA_IsActiveFlag_HT1

LL_DMA_IsActiveFlag_HT2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF2 LL_DMA_IsActiveFlag_HT2

LL_DMA_IsActiveFlag_HT3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF3 LL_DMA_IsActiveFlag_HT3

LL_DMA_IsActiveFlag_HT4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF4 LL_DMA_IsActiveFlag_HT4

LL_DMA_IsActiveFlag_HT5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF5 LL_DMA_IsActiveFlag_HT5

LL_DMA_IsActiveFlag_HT6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF6 LL_DMA_IsActiveFlag_HT6

LL_DMA_IsActiveFlag_HT7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF7 LL_DMA_IsActiveFlag_HT7

LL_DMA_IsActiveFlag_TE1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF1 LL_DMA_IsActiveFlag_TE1

LL_DMA_IsActiveFlag_TE2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF2 LL_DMA_IsActiveFlag_TE2

LL_DMA_IsActiveFlag_TE3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF3 LL_DMA_IsActiveFlag_TE3

LL_DMA_IsActiveFlag_TE4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF4 LL_DMA_IsActiveFlag_TE4

LL_DMA_IsActiveFlag_TE5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF5 LL_DMA_IsActiveFlag_TE5

LL_DMA_IsActiveFlag_TE6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF6 LL_DMA_IsActiveFlag_TE6

LL_DMA_IsActiveFlag_TE7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF7 LL_DMA_IsActiveFlag_TE7

LL_DMA_ClearFlag_GI1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CGIF1 LL_DMA_ClearFlag_GI1

LL_DMA_ClearFlag_GI2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CGIF2 LL_DMA_ClearFlag_GI2

LL_DMA_ClearFlag_GI3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CGIF3 LL_DMA_ClearFlag_GI3

LL_DMA_ClearFlag_GI4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CGIF4 LL_DMA_ClearFlag_GI4

LL_DMA_ClearFlag_GI5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CGIF5 LL_DMA_ClearFlag_GI5

LL_DMA_ClearFlag_GI6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CGIF6 LL_DMA_ClearFlag_GI6

LL_DMA_ClearFlag_GI7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CGIF7 LL_DMA_ClearFlag_GI7

LL_DMA_ClearFlag_TC1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTCIF1 LL_DMA_ClearFlag_TC1

LL_DMA_ClearFlag_TC2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTCIF2 LL_DMA_ClearFlag_TC2

LL_DMA_ClearFlag_TC3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTCIF3 LL_DMA_ClearFlag_TC3

LL_DMA_ClearFlag_TC4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTCIF4 LL_DMA_ClearFlag_TC4

LL_DMA_ClearFlag_TC5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTCIF5 LL_DMA_ClearFlag_TC5

LL_DMA_ClearFlag_TC6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTCIF6 LL_DMA_ClearFlag_TC6

LL_DMA_ClearFlag_TC7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTCIF7 LL_DMA_ClearFlag_TC7

LL_DMA_ClearFlag_HT1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CHTIF1 LL_DMA_ClearFlag_HT1

LL_DMA_ClearFlag_HT2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CHTIF2 LL_DMA_ClearFlag_HT2

LL_DMA_ClearFlag_HT3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CHTIF3 LL_DMA_ClearFlag_HT3

LL_DMA_ClearFlag_HT4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CHTIF4 LL_DMA_ClearFlag_HT4

LL_DMA_ClearFlag_HT5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CHTIF5 LL_DMA_ClearFlag_HT5

LL_DMA_ClearFlag_HT6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CHTIF6 LL_DMA_ClearFlag_HT6

LL_DMA_ClearFlag_HT7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CHTIF7 LL_DMA_ClearFlag_HT7

LL_DMA_ClearFlag_TE1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTEIF1 LL_DMA_ClearFlag_TE1

LL_DMA_ClearFlag_TE2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTEIF2 LL_DMA_ClearFlag_TE2

LL_DMA_ClearFlag_TE3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTEIF3 LL_DMA_ClearFlag_TE3

LL_DMA_ClearFlag_TE4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTEIF4 LL_DMA_ClearFlag_TE4

LL_DMA_ClearFlag_TE5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTEIF5 LL_DMA_ClearFlag_TE5

LL_DMA_ClearFlag_TE6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTEIF6 LL_DMA_ClearFlag_TE6

LL_DMA_ClearFlag_TE7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IFCR CTEIF7 LL_DMA_ClearFlag_TE7

LL_DMA_EnableIT_TC

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Transfer complete interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None**
Reference Manual to LL API cross reference:	• CCR TCIE LL_DMA_EnableIT_TC

LL_DMA_EnableIT_HT

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Half transfer interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None**
Reference Manual to LL API cross	• CCR HTIE LL_DMA_EnableIT_HT

reference:

LL_DMA_EnableIT_TE

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Transfer error interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None**
Reference Manual to LL API cross reference:	• CCR TEIE LL_DMA_EnableIT_TE

LL_DMA_DisableIT_TC

Function name	**__STATIC_INLINE void LL_DMA_DisableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable Transfer complete interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None**
Reference Manual to LL API cross reference:	• CCR TCIE LL_DMA_DisableIT_TC

LL_DMA_DisableIT_HT

Function name	**__STATIC_INLINE void LL_DMA_DisableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable Half transfer interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1

- – LL_DMA_CHANNEL_2
- – LL_DMA_CHANNEL_3
- – LL_DMA_CHANNEL_4
- – LL_DMA_CHANNEL_5
- – LL_DMA_CHANNEL_6
- – LL_DMA_CHANNEL_7

Return values	• **None**
Reference Manual to LL API cross reference:	• CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TE

Function name	**__STATIC_INLINE void LL_DMA_DisableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable Transfer error interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None**
Reference Manual to LL API cross reference:	• CCR TEIE LL_DMA_DisableIT_TE

LL_DMA_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if Transfer complete Interrupt is enabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_IsEnabledIT_HT

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if Half transfer Interrupt is enabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TE

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if Transfer error Interrupt is enabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR TEIE LL_DMA_IsEnabledIT_TE

LL_DMA_Init

Function name	**uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)**
Function description	Initialize the DMA registers according to the specified parameters in DMA_InitStruct.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: (*) value not defined in all devices – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2

- – LL_DMA_CHANNEL_3
- – LL_DMA_CHANNEL_4
- – LL_DMA_CHANNEL_5
- – LL_DMA_CHANNEL_6 (*)
- – LL_DMA_CHANNEL_7 (*)
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DMA registers are initialized – ERROR: Not applicable
Notes	• To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : __LL_DMA_GET_INSTANCE __LL_DMA_GET_CHANNEL

LL_DMA_DeInit

Function name	**uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	De-initialize the DMA registers to their default reset values.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: (*) value not defined in all devices – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 (*) – LL_DMA_CHANNEL_7 (*) – LL_DMA_CHANNEL_ALL
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DMA registers are de-initialized – ERROR: DMA registers are not de-initialized

LL_DMA_StructInit

Function name	**void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)**
Function description	Set each LL_DMA_InitTypeDef field to default value.
Parameters	• **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.
Return values	• **None**

60.3 DMA Firmware driver defines

60.3.1 DMA

CHANNEL

LL_DMA_CHANNEL_1	DMA Channel 1

LL_DMA_CHANNEL_2	DMA Channel 2
LL_DMA_CHANNEL_3	DMA Channel 3
LL_DMA_CHANNEL_4	DMA Channel 4
LL_DMA_CHANNEL_5	DMA Channel 5
LL_DMA_CHANNEL_6	DMA Channel 6
LL_DMA_CHANNEL_7	DMA Channel 7
LL_DMA_CHANNEL_ALL	DMA Channel all (used only for function

Clear Flags Defines

LL_DMA_IFCR_CGIF1	Channel 1 global flag
LL_DMA_IFCR_CTCIF1	Channel 1 transfer complete flag
LL_DMA_IFCR_CHTIF1	Channel 1 half transfer flag
LL_DMA_IFCR_CTEIF1	Channel 1 transfer error flag
LL_DMA_IFCR_CGIF2	Channel 2 global flag
LL_DMA_IFCR_CTCIF2	Channel 2 transfer complete flag
LL_DMA_IFCR_CHTIF2	Channel 2 half transfer flag
LL_DMA_IFCR_CTEIF2	Channel 2 transfer error flag
LL_DMA_IFCR_CGIF3	Channel 3 global flag
LL_DMA_IFCR_CTCIF3	Channel 3 transfer complete flag
LL_DMA_IFCR_CHTIF3	Channel 3 half transfer flag
LL_DMA_IFCR_CTEIF3	Channel 3 transfer error flag
LL_DMA_IFCR_CGIF4	Channel 4 global flag
LL_DMA_IFCR_CTCIF4	Channel 4 transfer complete flag
LL_DMA_IFCR_CHTIF4	Channel 4 half transfer flag
LL_DMA_IFCR_CTEIF4	Channel 4 transfer error flag
LL_DMA_IFCR_CGIF5	Channel 5 global flag
LL_DMA_IFCR_CTCIF5	Channel 5 transfer complete flag
LL_DMA_IFCR_CHTIF5	Channel 5 half transfer flag
LL_DMA_IFCR_CTEIF5	Channel 5 transfer error flag
LL_DMA_IFCR_CGIF6	Channel 6 global flag
LL_DMA_IFCR_CTCIF6	Channel 6 transfer complete flag
LL_DMA_IFCR_CHTIF6	Channel 6 half transfer flag
LL_DMA_IFCR_CTEIF6	Channel 6 transfer error flag
LL_DMA_IFCR_CGIF7	Channel 7 global flag
LL_DMA_IFCR_CTCIF7	Channel 7 transfer complete flag
LL_DMA_IFCR_CHTIF7	Channel 7 half transfer flag
LL_DMA_IFCR_CTEIF7	Channel 7 transfer error flag

Transfer Direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY	Peripheral to memory direction
LL_DMA_DIRECTION_MEMORY_TO_PERIPH	Memory to peripheral direction
LL_DMA_DIRECTION_MEMORY_TO_MEMORY	Memory to memory direction

Get Flags Defines

LL_DMA_ISR_GIF1	Channel 1 global flag
LL_DMA_ISR_TCIF1	Channel 1 transfer complete flag
LL_DMA_ISR_HTIF1	Channel 1 half transfer flag
LL_DMA_ISR_TEIF1	Channel 1 transfer error flag
LL_DMA_ISR_GIF2	Channel 2 global flag
LL_DMA_ISR_TCIF2	Channel 2 transfer complete flag
LL_DMA_ISR_HTIF2	Channel 2 half transfer flag
LL_DMA_ISR_TEIF2	Channel 2 transfer error flag
LL_DMA_ISR_GIF3	Channel 3 global flag
LL_DMA_ISR_TCIF3	Channel 3 transfer complete flag
LL_DMA_ISR_HTIF3	Channel 3 half transfer flag
LL_DMA_ISR_TEIF3	Channel 3 transfer error flag
LL_DMA_ISR_GIF4	Channel 4 global flag
LL_DMA_ISR_TCIF4	Channel 4 transfer complete flag
LL_DMA_ISR_HTIF4	Channel 4 half transfer flag
LL_DMA_ISR_TEIF4	Channel 4 transfer error flag
LL_DMA_ISR_GIF5	Channel 5 global flag
LL_DMA_ISR_TCIF5	Channel 5 transfer complete flag
LL_DMA_ISR_HTIF5	Channel 5 half transfer flag
LL_DMA_ISR_TEIF5	Channel 5 transfer error flag
LL_DMA_ISR_GIF6	Channel 6 global flag
LL_DMA_ISR_TCIF6	Channel 6 transfer complete flag
LL_DMA_ISR_HTIF6	Channel 6 half transfer flag
LL_DMA_ISR_TEIF6	Channel 6 transfer error flag
LL_DMA_ISR_GIF7	Channel 7 global flag
LL_DMA_ISR_TCIF7	Channel 7 transfer complete flag
LL_DMA_ISR_HTIF7	Channel 7 half transfer flag
LL_DMA_ISR_TEIF7	Channel 7 transfer error flag

IT Defines

LL_DMA_CCR_TCIE	Transfer complete interrupt

LL_DMA_CCR_HTIE	Half Transfer interrupt
LL_DMA_CCR_TEIE	Transfer error interrupt

Memory data alignment

LL_DMA_MDATAALIGN_BYTE	Memory data alignment : Byte
LL_DMA_MDATAALIGN_HALFWORD	Memory data alignment : HalfWord
LL_DMA_MDATAALIGN_WORD	Memory data alignment : Word

Memory increment mode

LL_DMA_MEMORY_INCREMENT	Memory increment mode Enable
LL_DMA_MEMORY_NOINCREMENT	Memory increment mode Disable

Transfer mode

LL_DMA_MODE_NORMAL	Normal Mode
LL_DMA_MODE_CIRCULAR	Circular Mode

Peripheral data alignment

LL_DMA_PDATAALIGN_BYTE	Peripheral data alignment : Byte
LL_DMA_PDATAALIGN_HALFWORD	Peripheral data alignment : HalfWord
LL_DMA_PDATAALIGN_WORD	Peripheral data alignment : Word

Peripheral increment mode

LL_DMA_PERIPH_INCREMENT	Peripheral increment mode Enable
LL_DMA_PERIPH_NOINCREMENT	Peripheral increment mode Disable

Transfer Priority level

LL_DMA_PRIORITY_LOW	Priority level : Low
LL_DMA_PRIORITY_MEDIUM	Priority level : Medium
LL_DMA_PRIORITY_HIGH	Priority level : High
LL_DMA_PRIORITY_VERYHIGH	Priority level : Very_High

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

61 LL EXTI Generic Driver

61.1 EXTI Firmware driver registers structures

61.1.1 LL_EXTI_InitTypeDef

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***.

61.2 EXTI Firmware driver API description

61.2.1 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
- **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	•	Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	•	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	• **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	• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• 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	• **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	• **None**
Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• RTSR RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_0_31

Function name	**__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_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	• **None**
Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and

	falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• RTSR RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)**
Function description	Check if rising edge trigger is enabled for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_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	• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• RTSR RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_EnableFallingTrig_0_31

Function name	**__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_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	• **None**
Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• FTSR FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_0_31

Function name	**__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)**

Function description	Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_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	• **None**
Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• FTSR FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)**
Function description	Check if falling edge trigger is enabled for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1

- LL_EXTI_LINE_2
- LL_EXTI_LINE_3
- LL_EXTI_LINE_4
- LL_EXTI_LINE_5
- LL_EXTI_LINE_6
- LL_EXTI_LINE_7
- LL_EXTI_LINE_8
- LL_EXTI_LINE_9
- LL_EXTI_LINE_10
- LL_EXTI_LINE_11
- LL_EXTI_LINE_12
- LL_EXTI_LINE_13
- LL_EXTI_LINE_14
- LL_EXTI_LINE_15
- LL_EXTI_LINE_16
- LL_EXTI_LINE_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	• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• 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	• **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	• **None**
Notes	• If the interrupt is enabled on this line in the EXTI_IMR, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR register (by writing a 1 into the bit) • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• SWIER SWIx LL_EXTI_GenerateSWI_0_31

LL_EXTI_IsActiveFlag_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_0_31 (uint32_t ExtiLine)**
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	• **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	• **@note:** This bit is set when the selected edge event arrives on the interrupt
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_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	• **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	• **None**
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_ClearFlag_0_31

LL_EXTI_Init

Function name	**uint32_t LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)**
Function description	Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.
Parameters	• **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: EXTI registers are initialized – ERROR: not applicable

LL_EXTI_DeInit

Function name	**uint32_t LL_EXTI_DeInit (void)**
Function description	De-initialize the EXTI registers to their default reset values.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: EXTI registers are de-initialized – ERROR: not applicable

LL_EXTI_StructInit

Function name	**void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)**
Function description	Set each LL_EXTI_InitTypeDef field to default value.
Parameters	• **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.
Return values	• **None**

61.3 EXTI Firmware driver defines

61.3.1 EXTI

LINE

LL_EXTI_LINE_0	Extended line 0
LL_EXTI_LINE_1	Extended line 1
LL_EXTI_LINE_2	Extended line 2
LL_EXTI_LINE_3	Extended line 3
LL_EXTI_LINE_4	Extended line 4
LL_EXTI_LINE_5	Extended line 5
LL_EXTI_LINE_6	Extended line 6
LL_EXTI_LINE_7	Extended line 7
LL_EXTI_LINE_8	Extended line 8
LL_EXTI_LINE_9	Extended line 9
LL_EXTI_LINE_10	Extended line 10
LL_EXTI_LINE_11	Extended line 11

LL_EXTI_LINE_12	Extended line 12
LL_EXTI_LINE_13	Extended line 13
LL_EXTI_LINE_14	Extended line 14
LL_EXTI_LINE_15	Extended line 15
LL_EXTI_LINE_16	Extended line 16
LL_EXTI_LINE_17	Extended line 17
LL_EXTI_LINE_18	Extended line 18
LL_EXTI_LINE_19	Extended line 19
LL_EXTI_LINE_20	Extended line 20
LL_EXTI_LINE_21	Extended line 21
LL_EXTI_LINE_22	Extended line 22
LL_EXTI_LINE_23	Extended line 23
LL_EXTI_LINE_24	Extended line 24
LL_EXTI_LINE_25	Extended line 25
LL_EXTI_LINE_26	Extended line 26
LL_EXTI_LINE_28	Extended line 28
LL_EXTI_LINE_29	Extended line 29
LL_EXTI_LINE_ALL_0_31	All Extended line not reserved
LL_EXTI_LINE_ALL	All Extended line
LL_EXTI_LINE_NONE	None Extended line

Mode

LL_EXTI_MODE_IT	Interrupt Mode
LL_EXTI_MODE_EVENT	Event Mode
LL_EXTI_MODE_IT_EVENT	Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE	No Trigger Mode
LL_EXTI_TRIGGER_RISING	Trigger Rising Mode
LL_EXTI_TRIGGER_FALLING	Trigger Falling Mode
LL_EXTI_TRIGGER_RISING_FALLING	Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

62 LL GPIO Generic Driver

62.1 GPIO Firmware driver registers structures

62.1.1 LL_GPIO_InitTypeDef

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()**.

62.2 GPIO Firmware driver API description

62.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name	**__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)**
Function description	Configure gpio mode for a dedicated pin on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values:

- LL_GPIO_PIN_0
- LL_GPIO_PIN_1
- LL_GPIO_PIN_2
- LL_GPIO_PIN_3
- LL_GPIO_PIN_4
- LL_GPIO_PIN_5
- LL_GPIO_PIN_6
- LL_GPIO_PIN_7
- LL_GPIO_PIN_8
- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Return values
- **None**

Notes
- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:
- MODER MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name
__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description
Return gpio mode for a dedicated pin on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13

- LL_GPIO_PIN_14
- LL_GPIO_PIN_15

Return values	• **Returned:** value can be one of the following values: – LL_GPIO_MODE_INPUT – LL_GPIO_MODE_OUTPUT – LL_GPIO_MODE_ALTERNATE – LL_GPIO_MODE_ANALOG
Notes	• I/O mode can be Input mode, General purpose output, Alternate function mode or Analog. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• MODER MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinOutputType

Function name	**__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t OutputType)**
Function description	Configure gpio output type for several pins on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL • **OutputType:** This parameter can be one of the following values: – LL_GPIO_OUTPUT_PUSHPULL – LL_GPIO_OUTPUT_OPENDRAIN
Return values	• **None**
Notes	• Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
Reference Manual to LL API cross	• OTYPER OTy LL_GPIO_SetPinOutputType

reference:

LL_GPIO_GetPinOutputType

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio output type for several pins on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_OUTPUT_PUSHPULL – LL_GPIO_OUTPUT_OPENDRAIN
Notes	• Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• OTYPER OTy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinSpeed

Function name	**__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)**
Function description	Configure gpio speed for a dedicated pin on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5

	– LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 • **Speed:** This parameter can be one of the following values: – LL_GPIO_SPEED_FREQ_LOW – LL_GPIO_SPEED_FREQ_MEDIUM – LL_GPIO_SPEED_FREQ_HIGH – LL_GPIO_SPEED_FREQ_VERY_HIGH
Return values	• **None**
Notes	• I/O speed can be Low, Medium, Fast or High speed. • Warning: only one pin can be passed as parameter. • Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.
Reference Manual to LL API cross reference:	• OSPEEDR OSPEEDy LL_GPIO_SetPinSpeed

LL_GPIO_GetPinSpeed

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio speed for a dedicated pin on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_SPEED_FREQ_LOW – LL_GPIO_SPEED_FREQ_MEDIUM

- – LL_GPIO_SPEED_FREQ_HIGH
- – LL_GPIO_SPEED_FREQ_VERY_HIGH

Notes	• I/O speed can be Low, Medium, Fast or High speed. • Warning: only one pin can be passed as parameter. • Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.
Reference Manual to LL API cross reference:	• OSPEEDR OSPEEDy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinPull

Function name	**__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)**
Function description	Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 • **Pull:** This parameter can be one of the following values: – LL_GPIO_PULL_NO – LL_GPIO_PULL_UP – LL_GPIO_PULL_DOWN
Return values	• **None**
Notes	• Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• PUPDR PUPDy LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio pull-up or pull-down for a dedicated pin on a dedicated

port.

Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_PULL_NO – LL_GPIO_PULL_UP – LL_GPIO_PULL_DOWN
Notes	• Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• PUPDR PUPDy LL_GPIO_GetPinPull

LL_GPIO_SetAFPin_0_7

Function name	**__STATIC_INLINE void LL_GPIO_SetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)**
Function description	Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 • **Alternate:** This parameter can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3

  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7

Return values	• **None**
Notes	• Possible values are from AF0 to AF7 depending on target. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• AFRL AFSELy LL_GPIO_SetAFPin_0_7

LL_GPIO_GetAFPin_0_7

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3 – LL_GPIO_AF_4 – LL_GPIO_AF_5 – LL_GPIO_AF_6 – LL_GPIO_AF_7
Reference Manual to LL API cross reference:	• AFRL AFSELy LL_GPIO_GetAFPin_0_7

LL_GPIO_SetAFPin_8_15

Function name	**__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)**
Function description	Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_8

- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15

	• **Alternate:** This parameter can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3 – LL_GPIO_AF_4 – LL_GPIO_AF_5 – LL_GPIO_AF_6 – LL_GPIO_AF_7
Return values	• **None**
Notes	• Possible values are from AF0 to AF7 depending on target. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3 – LL_GPIO_AF_4 – LL_GPIO_AF_5 – LL_GPIO_AF_6 – LL_GPIO_AF_7

Notes	• Possible values are from AF0 to AF7 depending on target.
Reference Manual to LL API cross reference:	• AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name	**__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Lock configuration of several pins for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **None**
Notes	• When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset. • Each lock bit freezes a specific configuration register (control and alternate function registers).
Reference Manual to LL API cross reference:	• LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Return 1 if all pins passed as parameter, of a dedicated port, are locked.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values:

- LL_GPIO_PIN_0
- LL_GPIO_PIN_1
- LL_GPIO_PIN_2
- LL_GPIO_PIN_3
- LL_GPIO_PIN_4
- LL_GPIO_PIN_5
- LL_GPIO_PIN_6
- LL_GPIO_PIN_7
- LL_GPIO_PIN_8
- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15
- LL_GPIO_PIN_ALL

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyPinLocked

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (GPIO_TypeDef * GPIOx)**
Function description	Return 1 if one of the pin of a dedicated port is locked.
Parameters	• **GPIOx:** GPIO Port
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_ReadInputPort

Function name	**__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (GPIO_TypeDef * GPIOx)**
Function description	Return full input data register value for a dedicated port.
Parameters	• **GPIOx:** GPIO Port
Return values	• **Input:** data register value of port
Reference Manual to LL API cross reference:	• IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)**

Function description	Return if input data level for several pins of dedicated port is high or low.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name	**__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)**
Function description	Write output data register for the port.
Parameters	• **GPIOx:** GPIO Port • **PortValue:** Level value for each pin of the port
Return values	• **None**
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name	**__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (GPIO_TypeDef * GPIOx)**
Function description	Return full output data register value for a dedicated port.
Parameters	• **GPIOx:** GPIO Port
Return values	• **Output:** data register value of port
Reference Manual to LL API cross	• ODR ODy LL_GPIO_ReadOutputPort

reference:

LL_GPIO_IsOutputPinSet

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Return if input data level for several pins of dedicated port is high or low.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name	**__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Set several pins to high level on dedicated gpio port.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8

- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15
- LL_GPIO_PIN_ALL

Return values	• **None**
Reference Manual to LL API cross reference:	• BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name	**__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Set several pins to low level on dedicated gpio port.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **None**
Reference Manual to LL API cross reference:	• BRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name	**__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Toggle data value for several pin of dedicated port.
Parameters	• **GPIOx:** GPIO Port

	• **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **None**
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_TogglePin

LL_GPIO_DeInit

Function name	**ErrorStatus LL_GPIO_DeInit (GPIO_TypeDef * GPIOx)**
Function description	De-initialize GPIO registers (Registers restored to their default values).
Parameters	• **GPIOx:** GPIO Port
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: GPIO registers are de-initialized – ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name	**ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)**
Function description	Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.
Parameters	• **GPIOx:** GPIO Port • **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content – ERROR: Not applicable

LL_GPIO_StructInit

Function name	**void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)**
Function description	Set each LL_GPIO_InitTypeDef field to default value.
Parameters	• **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.
Return values	• **None**

62.3 GPIO Firmware driver defines

62.3.1 GPIO

Alternate Function

LL_GPIO_AF_0	Select alternate function 0
LL_GPIO_AF_1	Select alternate function 1
LL_GPIO_AF_2	Select alternate function 2
LL_GPIO_AF_3	Select alternate function 3
LL_GPIO_AF_4	Select alternate function 4
LL_GPIO_AF_5	Select alternate function 5
LL_GPIO_AF_6	Select alternate function 6
LL_GPIO_AF_7	Select alternate function 7

Mode

LL_GPIO_MODE_INPUT	Select input mode
LL_GPIO_MODE_OUTPUT	Select output mode
LL_GPIO_MODE_ALTERNATE	Select alternate function mode
LL_GPIO_MODE_ANALOG	Select analog mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL	Select push-pull as output type
LL_GPIO_OUTPUT_OPENDRAIN	Select open-drain as output type

PIN

LL_GPIO_PIN_0	Select pin 0
LL_GPIO_PIN_1	Select pin 1
LL_GPIO_PIN_2	Select pin 2
LL_GPIO_PIN_3	Select pin 3
LL_GPIO_PIN_4	Select pin 4
LL_GPIO_PIN_5	Select pin 5
LL_GPIO_PIN_6	Select pin 6
LL_GPIO_PIN_7	Select pin 7
LL_GPIO_PIN_8	Select pin 8

LL_GPIO_PIN_9	Select pin 9
LL_GPIO_PIN_10	Select pin 10
LL_GPIO_PIN_11	Select pin 11
LL_GPIO_PIN_12	Select pin 12
LL_GPIO_PIN_13	Select pin 13
LL_GPIO_PIN_14	Select pin 14
LL_GPIO_PIN_15	Select pin 15
LL_GPIO_PIN_ALL	Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO	Select I/O no pull
LL_GPIO_PULL_UP	Select I/O pull up
LL_GPIO_PULL_DOWN	Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW	Select I/O low output speed
LL_GPIO_SPEED_FREQ_MEDIUM	Select I/O medium output speed
LL_GPIO_SPEED_FREQ_HIGH	Select I/O fast output speed
LL_GPIO_SPEED_FREQ_VERY_HIGH	Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants

LL_GPIO_SPEED_LOW

LL_GPIO_SPEED_MEDIUM

LL_GPIO_SPEED_FAST

LL_GPIO_SPEED_HIGH

63 LL I2C Generic Driver

63.1 I2C Firmware driver registers structures

63.1.1 LL_I2C_InitTypeDef

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()**.

63.2 I2C Firmware driver API description

63.2.1 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	• **I2Cx:** I2C Instance. • **AnalogFilter:** This parameter can be one of the following

	values: – LL_I2C_ANALOGFILTER_ENABLE – LL_I2C_ANALOGFILTER_DISABLE • **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.
Return values	• **None**
Notes	• If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_ConfigFilters • CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name	**__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)**
Function description	Configure Digital Noise Filter.
Parameters	• **I2Cx:** I2C Instance. • **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.
Return values	• **None**
Notes	• If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (I2C_TypeDef * I2Cx)**
Function description	Get the current Digital Noise Filter configuration.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• CR1 DNF LL_I2C_GetDigitalFilter

LL_I2C_EnableAnalogFilter

Function name	**__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)**
Function description	Enable Analog Noise Filter.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name	**__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)**
Function description	Disable Analog Noise Filter.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (I2C_TypeDef * I2Cx)**
Function description	Check if Analog Noise Filter is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)**
Function description	Enable DMA transmission requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to	• CR1 TXDMAEN LL_I2C_EnableDMAReq_TX

LL API cross reference:

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	• CR1 RXDMAEN LL_I2C_DisableDMAReq_RX

reference:

LL_I2C_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (I2C_TypeDef * I2Cx)**
Function description	Check if DMA reception requests are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (I2C_TypeDef * I2Cx, uint32_t Direction)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **I2Cx:** I2C Instance • **Direction:** This parameter can be one of the following values: – LL_I2C_DMA_REG_DATA_TRANSMIT – LL_I2C_DMA_REG_DATA_RECEIVE
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• TXDR TXDATA LL_I2C_DMA_GetRegAddr • RXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name	**__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)**
Function description	Enable Clock stretching.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name	**__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)**
Function description	Disable Clock stretching.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (I2C_TypeDef * I2Cx)**
Function description	Check if Clock stretching is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableSlaveByteControl

Function name	**__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)**
Function description	Enable hardware byte control in slave mode.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 SBC LL_I2C_EnableSlaveByteControl

LL_I2C_DisableSlaveByteControl

Function name	**__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)**
Function description	Disable hardware byte control in slave mode.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 SBC LL_I2C_DisableSlaveByteControl

LL_I2C_IsEnabledSlaveByteControl

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (I2C_TypeDef * I2Cx)**

Function description	Check if hardware byte control in slave mode is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 SBC LL_I2C_IsEnabledSlaveByteControl

LL_I2C_EnableWakeUpFromStop

Function name	**__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)**
Function description	Enable Wakeup from STOP.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance. • This bit can only be programmed when Digital Filter is disabled.
Reference Manual to LL API cross reference:	• CR1 WUPEN LL_I2C_EnableWakeUpFromStop

LL_I2C_DisableWakeUpFromStop

Function name	**__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)**
Function description	Disable Wakeup from STOP.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 WUPEN LL_I2C_DisableWakeUpFromStop

LL_I2C_IsEnabledWakeUpFromStop

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (I2C_TypeDef * I2Cx)**
Function description	Check if Wakeup from STOP is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).

Notes	• 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	**__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)**

Function description	Configure the Master to operate in 7-bit or 10-bit addressing mode.
Parameters	• **I2Cx:** I2C Instance. • **AddressingMode:** This parameter can be one of the following values: – LL_I2C_ADDRESSING_MODE_7BIT – LL_I2C_ADDRESSING_MODE_10BIT
Return values	• **None**
Notes	• Changing this bit is not allowed, when the START bit is set.
Reference Manual to LL API cross reference:	• CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (I2C_TypeDef * I2Cx)**
Function description	Get the Master addressing mode.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_ADDRESSING_MODE_7BIT – LL_I2C_ADDRESSING_MODE_10BIT
Reference Manual to LL API cross reference:	• CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name	**__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)**
Function description	Set the Own Address1.
Parameters	• **I2Cx:** I2C Instance. • **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF. • **OwnAddrSize:** This parameter can be one of the following values: – LL_I2C_OWNADDRESS1_7BIT – LL_I2C_OWNADDRESS1_10BIT
Return values	• **None**
Reference Manual to LL API cross reference:	• OAR1 OA1 LL_I2C_SetOwnAddress1 • OAR1 OA1MODE LL_I2C_SetOwnAddress1

LL_I2C_EnableOwnAddress1

Function name	**__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)**

Function description	Enable acknowledge on Own Address1 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• OAR1 OA1EN LL_I2C_EnableOwnAddress1

LL_I2C_DisableOwnAddress1

Function name	**__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)**
Function description	Disable acknowledge on Own Address1 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• OAR1 OA1EN LL_I2C_DisableOwnAddress1

LL_I2C_IsEnabledOwnAddress1

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (I2C_TypeDef * I2Cx)**
Function description	Check if Own Address1 acknowledge is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1

LL_I2C_SetOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)**
Function description	Set the 7bits Own Address2.
Parameters	• **I2Cx:** I2C Instance. • **OwnAddress2:** Value between Min_Data=0 and Max_Data=0x7F. • **OwnAddrMask:** This parameter can be one of the following values: – LL_I2C_OWNADDRESS2_NOMASK – LL_I2C_OWNADDRESS2_MASK01 – LL_I2C_OWNADDRESS2_MASK02 – LL_I2C_OWNADDRESS2_MASK03 – LL_I2C_OWNADDRESS2_MASK04 – LL_I2C_OWNADDRESS2_MASK05 – LL_I2C_OWNADDRESS2_MASK06

– LL_I2C_OWNADDRESS2_MASK07

Return values	• **None**
Notes	• This action has no effect if own address2 is enabled.
Reference Manual to LL API cross reference:	• OAR2 OA2 LL_I2C_SetOwnAddress2 • OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Enable acknowledge on Own Address2 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Disable acknowledge on Own Address2 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Check if Own Address1 acknowledge is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name	**__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)**
Function description	Configure the SDA setup, hold time and the SCL high, low period.

Parameters	• **I2Cx:** I2C Instance. • **Timing:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.
Return values	• **None**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0). • This parameter is computed with the STM32CubeMX Tool.
Reference Manual to LL API cross reference:	• TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (I2C_TypeDef * I2Cx)**
Function description	Get the Timing Prescaler setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• TIMINGR PRESC LL_I2C_GetTimingPrescaler

LL_I2C_GetClockLowPeriod

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (I2C_TypeDef * I2Cx)**
Function description	Get the SCL low period setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• TIMINGR SCLL LL_I2C_GetClockLowPeriod

LL_I2C_GetClockHighPeriod

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (I2C_TypeDef * I2Cx)**
Function description	Get the SCL high period setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• TIMINGR SCLH LL_I2C_GetClockHighPeriod

LL_I2C_GetDataHoldTime

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (I2C_TypeDef * I2Cx)**
Function description	Get the SDA hold time.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• TIMINGR SDADEL LL_I2C_GetDataHoldTime

LL_I2C_GetDataSetupTime

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (I2C_TypeDef * I2Cx)**
Function description	Get the SDA setup time.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• TIMINGR SCLDEL LL_I2C_GetDataSetupTime

LL_I2C_SetMode

Function name	**__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)**
Function description	Configure peripheral mode.
Parameters	• **I2Cx:** I2C Instance. • **PeripheralMode:** This parameter can be one of the following values: – LL_I2C_MODE_I2C – LL_I2C_MODE_SMBUS_HOST – LL_I2C_MODE_SMBUS_DEVICE – LL_I2C_MODE_SMBUS_DEVICE_ARP
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 SMBHEN LL_I2C_SetMode • CR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetMode (I2C_TypeDef * I2Cx)**
Function description	Get peripheral mode.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_MODE_I2C – LL_I2C_MODE_SMBUS_HOST – LL_I2C_MODE_SMBUS_DEVICE – LL_I2C_MODE_SMBUS_DEVICE_ARP
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 SMBHEN LL_I2C_GetMode • CR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Enable SMBus alert (Host or Device mode)
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.
Reference Manual to LL API cross reference:	• CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name	**__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Disable SMBus alert (Host or Device mode)
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.
Reference Manual to LL API cross reference:	• CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Check if SMBus alert (Host or Device mode) is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Enable SMBus Packet Error Calculation (PEC).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 PECEN LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name	**__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Disable SMBus Packet Error Calculation (PEC).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 PECEN LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC**

	(I2C_TypeDef * I2Cx)
Function description	Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 PECEN LL_I2C_IsEnabledSMBusPEC

LL_I2C_ConfigSMBusTimeout

Function name	**__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)**
Function description	Configure the SMBus Clock Timeout.
Parameters	• **I2Cx:** I2C Instance. • **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF. • **TimeoutAMode:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW – LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH • **TimeoutB:**
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/orTimeoutB).
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout • TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout • TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name	**__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)**
Function description	Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).
Parameters	• **I2Cx:** I2C Instance. • **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
Return values	• **None**

Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • These bits can only be programmed when TimeoutA is disabled.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (I2C_TypeDef * I2Cx)**
Function description	Get the SMBus Clock TimeoutA setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0 and Max_Data=0xFFF
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name	**__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)**
Function description	Set the SMBus Clock TimeoutA mode.
Parameters	• **I2Cx:** I2C Instance. • **TimeoutAMode:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW – LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
Return values	• **None**
Notes	• Macro IS_SMBUS_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	Get the SMBus Clock TimeoutA mode.

description	
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_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode

LL_I2C_SetSMBusTimeoutB

Function name	**__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB (I2C_TypeDef * I2Cx, uint32_t TimeoutB)**
Function description	Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).
Parameters	• **I2Cx:** I2C Instance. • **TimeoutB:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
Return values	• **None**
Notes	• Macro IS_SMBUS_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 Extented Cumulative Clock TimeoutB setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0 and Max_Data=0xFFF
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB

LL_I2C_EnableSMBusTimeout

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)**
Function description	Enable the SMBus Clock Timeout.
Parameters	• **I2Cx:** I2C Instance. • **ClockTimeout:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA – LL_I2C_SMBUS_TIMEOUTB – LL_I2C_SMBUS_ALL_TIMEOUT
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout • TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout

LL_I2C_DisableSMBusTimeout

Function name	**__STATIC_INLINE void LL_I2C_DisableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)**
Function description	Disable the SMBus Clock Timeout.
Parameters	• **I2Cx:** I2C Instance. • **ClockTimeout:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA – LL_I2C_SMBUS_TIMEOUTB – LL_I2C_SMBUS_ALL_TIMEOUT
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout • TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (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_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	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 STOPIE LL_I2C_IsEnabledIT_STOP

LL_I2C_EnableIT_TC

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)**
Function description	Enable Transfer Complete interrupt.
Parameters	• **I2Cx:** I2C Instance.

Return values	• **None**
Notes	• Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)
Reference Manual to LL API cross reference:	• CR1 TCIE LL_I2C_EnableIT_TC

LL_I2C_DisableIT_TC

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)**
Function description	Disable Transfer Complete interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)
Reference Manual to LL API cross reference:	• CR1 TCIE LL_I2C_DisableIT_TC

LL_I2C_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (I2C_TypeDef * I2Cx)**
Function description	Check if Transfer Complete interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TCIE LL_I2C_IsEnabledIT_TC

LL_I2C_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)**
Function description	Enable Error interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_SMBUS_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_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	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transmit data register empty flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.
Reference Manual to LL API cross reference:	• ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transmit interrupt flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.
Reference Manual to LL API cross reference:	• ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Receive data register not empty flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.
Reference Manual to LL API cross reference:	• ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Address matched flag (slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When the received slave address matched with one of the enabled slave address.
Reference Manual to LL API cross reference:	• ISR ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_NACK

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Not Acknowledge received flag.
Parameters	• **I2Cx:** I2C Instance.

Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a NACK is received after a byte transmission.
Reference Manual to LL API cross reference:	• ISR NACKF LL_I2C_IsActiveFlag_NACK

LL_I2C_IsActiveFlag_STOP

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Stop detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a Stop condition is detected.
Reference Manual to LL API cross reference:	• ISR STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transfer complete flag (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.
Reference Manual to LL API cross reference:	• ISR TC LL_I2C_IsActiveFlag_TC

LL_I2C_IsActiveFlag_TCR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transfer complete flag (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.
Reference Manual to LL API cross reference:	• ISR TCR LL_I2C_IsActiveFlag_TCR

LL_I2C_IsActiveFlag_BERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Bus error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.
Reference Manual to LL API cross reference:	• ISR BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Arbitration lost flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When arbitration lost.
Reference Manual to LL API cross reference:	• ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Overrun/Underrun flag (slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).
Reference Manual to LL API cross reference:	• ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus PEC error flag in reception.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).

Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.
Reference Manual to LL API cross reference:	• ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus Timeout detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.
Reference Manual to LL API cross reference:	• ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus alert flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.
Reference Manual to LL API cross reference:	• ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Bus Busy flag.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a Start condition is detected.
Reference Manual to LL API cross reference:	• ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)**
Function description	Clear Address Matched flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)**
Function description	Clear Not Acknowledge flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR NACKCF LL_I2C_ClearFlag_NACK

LL_I2C_ClearFlag_STOP

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)**
Function description	Clear Stop detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR STOPCF LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_TXE

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_TXE (I2C_TypeDef * I2Cx)**

Function description	Clear Transmit data register empty flag (TXE).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• This bit can be clear by software in order to flush the transmit data register (TXDR).
Reference Manual to LL API cross reference:	• ISR TXE LL_I2C_ClearFlag_TXE

LL_I2C_ClearFlag_BERR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)**
Function description	Clear Bus error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR BERRCF LL_I2C_ClearFlag_BERR

LL_I2C_ClearFlag_ARLO

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)**
Function description	Clear Arbitration lost flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ARLOCF LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)**
Function description	Clear Overrun/Underrun flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR OVRCF LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name	**__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR**

(I2C_TypeDef * I2Cx)

Function description	Clear SMBus PEC error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_SMBUS_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_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_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableAutoEndMode

Function name	**__STATIC_INLINE void LL_I2C_EnableAutoEndMode (I2C_TypeDef * I2Cx)**
Function description	Enable automatic STOP condition generation (master mode).
Parameters	• **I2Cx:** I2C Instance.

Return values	• **None**
Notes	• Automatic end mode : a STOP condition is automatically sent when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.
Reference Manual to LL API cross reference:	• CR2 AUTOEND LL_I2C_EnableAutoEndMode

LL_I2C_DisableAutoEndMode

Function name	**__STATIC_INLINE void LL_I2C_DisableAutoEndMode (I2C_TypeDef * I2Cx)**
Function description	Disable automatic STOP condition generation (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Software end mode : TC flag is set when NBYTES data are transferre, stretching SCL low.
Reference Manual to LL API cross reference:	• CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (I2C_TypeDef * I2Cx)**
Function description	Check if automatic STOP condition is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name	**__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)**
Function description	Enable reload mode (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.
Reference Manual to LL API cross reference:	• CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name	**__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)**
Function description	Disable reload mode (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).
Reference Manual to LL API cross reference:	• CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (I2C_TypeDef * I2Cx)**
Function description	Check if reload mode is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RELOAD LL_I2C_IsEnabledReloadMode

LL_I2C_SetTransferSize

Function name	**__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)**
Function description	Configure the number of bytes for transfer.
Parameters	• **I2Cx:** I2C Instance. • **TransferSize:** This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
Return values	• **None**
Notes	• Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	• CR2 NBYTES LL_I2C_SetTransferSize

LL_I2C_GetTransferSize

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (I2C_TypeDef * I2Cx)**
Function description	Get the number of bytes configured for transfer.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xFF

Reference Manual to LL API cross reference:	• CR2 NBYTES LL_I2C_GetTransferSize

LL_I2C_AcknowledgeNextData

Function name	**__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)**
Function description	Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.
Parameters	• **I2Cx:** I2C Instance. • **TypeAcknowledge:** This parameter can be one of the following values: – LL_I2C_ACK – LL_I2C_NACK
Return values	• **None**
Notes	• Usage in Slave mode only.
Reference Manual to LL API cross reference:	• CR2 NACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name	**__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)**
Function description	Generate a START or RESTART condition.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.
Reference Manual to LL API cross reference:	• CR2 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name	**__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)**
Function description	Generate a STOP condition after the current byte transfer (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableAuto10BitRead

Function name	**__STATIC_INLINE void LL_I2C_EnableAuto10BitRead (I2C_TypeDef * I2Cx)**
Function description	Enable automatic RESTART Read request condition for 10bit address header (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• The master sends the complete 10bit slave address read sequence : Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.
Reference Manual to LL API cross reference:	• CR2 HEAD10R LL_I2C_EnableAuto10BitRead

LL_I2C_DisableAuto10BitRead

Function name	**__STATIC_INLINE void LL_I2C_DisableAuto10BitRead (I2C_TypeDef * I2Cx)**
Function description	Disable automatic RESTART Read request condition for 10bit address header (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• The master only sends the first 7 bits of 10bit address in Read direction.
Reference Manual to LL API cross reference:	• CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (I2C_TypeDef * I2Cx)**
Function description	Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name	**__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)**
Function description	Configure the transfer direction (master mode).

Parameters	• **I2Cx:** I2C Instance. • **TransferRequest:** This parameter can be one of the following values: – LL_I2C_REQUEST_WRITE – LL_I2C_REQUEST_READ
Return values	• **None**
Notes	• Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	• CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (I2C_TypeDef * I2Cx)**
Function description	Get the transfer direction requested (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_REQUEST_WRITE – LL_I2C_REQUEST_READ
Reference Manual to LL API cross reference:	• CR2 RD_WRN LL_I2C_GetTransferRequest

LL_I2C_SetSlaveAddr

Function name	**__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)**
Function description	Configure the slave address for transfer (master mode).
Parameters	• **I2Cx:** I2C Instance. • **SlaveAddr:** This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.
Return values	• **None**
Notes	• Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	• CR2 SADD LL_I2C_SetSlaveAddr

LL_I2C_GetSlaveAddr

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (I2C_TypeDef * I2Cx)**
Function description	Get the slave address programmed for transfer.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:	• CR2 SADD LL_I2C_GetSlaveAddr

LL_I2C_HandleTransfer

Function name	**__STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)**
Function description	Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).
Parameters	• **I2Cx:** I2C Instance. • **SlaveAddr:** Specifies the slave address to be programmed. • **SlaveAddrSize:** This parameter can be one of the following values: – LL_I2C_ADDRSLAVE_7BIT – LL_I2C_ADDRSLAVE_10BIT • **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=255. • **EndMode:** This parameter can be one of the following values: – LL_I2C_MODE_RELOAD – LL_I2C_MODE_AUTOEND – LL_I2C_MODE_SOFTEND – LL_I2C_MODE_SMBUS_RELOAD – LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC – LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC – LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC – LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC • **Request:** This parameter can be one of the following values: – LL_I2C_GENERATE_NOSTARTSTOP – LL_I2C_GENERATE_STOP – LL_I2C_GENERATE_START_READ – LL_I2C_GENERATE_START_WRITE – LL_I2C_GENERATE_RESTART_7BIT_READ – LL_I2C_GENERATE_RESTART_7BIT_WRITE – LL_I2C_GENERATE_RESTART_10BIT_READ – LL_I2C_GENERATE_RESTART_10BIT_WRITE
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 SADD LL_I2C_HandleTransfer • CR2 ADD10 LL_I2C_HandleTransfer • CR2 RD_WRN LL_I2C_HandleTransfer • CR2 START LL_I2C_HandleTransfer • CR2 STOP LL_I2C_HandleTransfer • CR2 RELOAD LL_I2C_HandleTransfer • CR2 NBYTES LL_I2C_HandleTransfer • CR2 AUTOEND LL_I2C_HandleTransfer • CR2 HEAD10R LL_I2C_HandleTransfer

LL_I2C_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (I2C_TypeDef * I2Cx)**
Function description	Indicate the value of transfer direction (slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_DIRECTION_WRITE – LL_I2C_DIRECTION_READ
Notes	• RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.
Reference Manual to LL API cross reference:	• ISR DIR LL_I2C_GetTransferDirection

LL_I2C_GetAddressMatchCode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode (I2C_TypeDef * I2Cx)**
Function description	Return the slave matched address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x3F
Reference Manual to LL API cross reference:	• ISR ADDCODE LL_I2C_GetAddressMatchCode

LL_I2C_EnableSMBusPECCompare

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)**
Function description	Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None**
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.
Reference Manual to LL API cross reference:	• CR2 PECBYTE LL_I2C_EnableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (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_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Get the SMBus Packet Error byte calculated.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Notes	• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (I2C_TypeDef * I2Cx)**
Function description	Read Receive Data register.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name	**__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)**
Function description	Write in Transmit Data Register .
Parameters	• **I2Cx:** I2C Instance.

	• **Data:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Init

Function name	**uint32_t LL_I2C_Init (I2C_TypeDef * I2Cx, LL_I2C_InitTypeDef * I2C_InitStruct)**
Function description	Initialize the I2C registers according to the specified parameters in I2C_InitStruct.
Parameters	• **I2Cx:** I2C Instance. • **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: I2C registers are initialized – ERROR: Not applicable

LL_I2C_DeInit

Function name	**uint32_t LL_I2C_DeInit (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**

63.3 I2C Firmware driver defines

63.3.1 I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MODE_7BIT	Master operates in 7-bit addressing mode.
LL_I2C_ADDRESSING_MODE_10BIT	Master operates in 10-bit addressing mode.

Slave Address Length

LL_I2C_ADDRSLAVE_7BIT	Slave Address in 7-bit.
LL_I2C_ADDRSLAVE_10BIT	Slave Address in 10-bit.

Analog Filter Selection

LL_I2C_ANALOGFILTER_ENABLE	Analog filter is enabled.
LL_I2C_ANALOGFILTER_DISABLE	Analog filter is disabled.

Clear Flags Defines

LL_I2C_ICR_ADDRCF	Address Matched flag
LL_I2C_ICR_NACKCF	Not Acknowledge flag
LL_I2C_ICR_STOPCF	Stop detection flag
LL_I2C_ICR_BERRCF	Bus error flag
LL_I2C_ICR_ARLOCF	Arbitration Lost flag
LL_I2C_ICR_OVRCF	Overrun/Underrun flag
LL_I2C_ICR_PECCF	PEC error flag
LL_I2C_ICR_TIMOUTCF	Timeout detection flag
LL_I2C_ICR_ALERTCF	Alert flag

Read Write Direction

LL_I2C_DIRECTION_WRITE	Write transfer request by master, slave enters receiver mode.
LL_I2C_DIRECTION_READ	Read transfer request by master, slave enters transmitter mode.

DMA Register Data

LL_I2C_DMA_REG_DATA_TRANSMIT	Get address of data register used for transmission
LL_I2C_DMA_REG_DATA_RECEIVE	Get address of data register used for reception

Start And Stop Generation

LL_I2C_GENERATE_NOSTARTSTOP	Don't Generate Stop and Start condition.
LL_I2C_GENERATE_STOP	Generate Stop condition (Size should be set to 0).
LL_I2C_GENERATE_START_READ	Generate Start for read request.
LL_I2C_GENERATE_START_WRITE	Generate Start for write request.
LL_I2C_GENERATE_RESTART_7BIT_READ	Generate Restart for read request, slave 7Bit address.
LL_I2C_GENERATE_RESTART_7BIT_WRITE	Generate Restart for write request, slave 7Bit address.
LL_I2C_GENERATE_RESTART_10BIT_READ	Generate Restart for read request, slave 10Bit address.
LL_I2C_GENERATE_RESTART_10BIT_WRITE	Generate Restart for write request, slave 10Bit address.

Get Flags Defines

LL_I2C_ISR_TXE	Transmit data register empty
LL_I2C_ISR_TXIS	Transmit interrupt status

LL_I2C_ISR_RXNE	Receive data register not empty
LL_I2C_ISR_ADDR	Address matched (slave mode)
LL_I2C_ISR_NACKF	Not Acknowledge received flag
LL_I2C_ISR_STOPF	Stop detection flag
LL_I2C_ISR_TC	Transfer Complete (master mode)
LL_I2C_ISR_TCR	Transfer Complete Reload
LL_I2C_ISR_BERR	Bus error
LL_I2C_ISR_ARLO	Arbitration lost
LL_I2C_ISR_OVR	Overrun/Underrun (slave mode)
LL_I2C_ISR_PECERR	PEC Error in reception (SMBus mode)
LL_I2C_ISR_TIMEOUT	Timeout detection flag (SMBus mode)
LL_I2C_ISR_ALERT	SMBus alert (SMBus mode)
LL_I2C_ISR_BUSY	Bus busy

Acknowledge Generation

LL_I2C_ACK	ACK is sent after current received byte.
LL_I2C_NACK	NACK is sent after current received byte.

IT Defines

LL_I2C_CR1_TXIE	TX Interrupt enable
LL_I2C_CR1_RXIE	RX Interrupt enable
LL_I2C_CR1_ADDRIE	Address match Interrupt enable (slave only)
LL_I2C_CR1_NACKIE	Not acknowledge received Interrupt enable
LL_I2C_CR1_STOPIE	STOP detection Interrupt enable
LL_I2C_CR1_TCIE	Transfer Complete interrupt enable
LL_I2C_CR1_ERRIE	Error interrupts enable

Transfer End Mode

LL_I2C_MODE_RELOAD	Enable I2C Reload mode.
LL_I2C_MODE_AUTOEND	Enable I2C Automatic end mode with no HW PEC comparison.
LL_I2C_MODE_SOFTEND	Enable I2C Software end mode with no HW PEC comparison.
LL_I2C_MODE_SMBUS_RELOAD	Enable SMBUS Automatic end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC	Enable SMBUS Automatic end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC	Enable SMBUS Software end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC	Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC	Enable SMBUS Software end mode with HW PEC comparison.

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT	Own address 1 is a 7-bit address.
LL_I2C_OWNADDRESS1_10BIT	Own address 1 is a 10-bit address.

Own Address 2 Masks

LL_I2C_OWNADDRESS2_NOMASK	Own Address2 No mask.
LL_I2C_OWNADDRESS2_MASK01	Only Address2 bits[7:2] are compared.
LL_I2C_OWNADDRESS2_MASK02	Only Address2 bits[7:3] are compared.
LL_I2C_OWNADDRESS2_MASK03	Only Address2 bits[7:4] are compared.
LL_I2C_OWNADDRESS2_MASK04	Only Address2 bits[7:5] are compared.
LL_I2C_OWNADDRESS2_MASK05	Only Address2 bits[7:6] are compared.
LL_I2C_OWNADDRESS2_MASK06	Only Address2 bits[7] are compared.
LL_I2C_OWNADDRESS2_MASK07	No comparison is done. All Address2 are acknowledged.

Peripheral Mode

LL_I2C_MODE_I2C	I2C Master or Slave mode
LL_I2C_MODE_SMBUS_HOST	SMBus Host address acknowledge
LL_I2C_MODE_SMBUS_DEVICE	SMBus Device default mode (Default address not acknowledge)
LL_I2C_MODE_SMBUS_DEVICE_ARP	SMBus Device Default address acknowledge

Transfer Request Direction

LL_I2C_REQUEST_WRITE	Master request a write transfer.
LL_I2C_REQUEST_READ	Master request a read transfer.

SMBus TimeoutA Mode SCL SDA Timeout

LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW	TimeoutA is used to detect SCL low level timeout.
LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH	TimeoutA is used to detect both SCL and SDA high level timeout.

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUTA	TimeoutA enable bit
LL_I2C_SMBUS_TIMEOUTB	TimeoutB (extended clock) enable bit
LL_I2C_SMBUS_ALL_TIMEOUT	TimeoutA and TimeoutB (extended clock) enable bits

Convert SDA SCL timings

__LL_I2C_CONVERT_TIMINGS

Description:

- Configure the SDA setup, hold time and the SCL high, low period.

Parameters:

- __PRESCALER__: This parameter must be a value between Min_Data=0 and Max_Data=0xF.
- __DATA_SETUP_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tscldel = (SCLDEL+1)xtpresc)
- __DATA_HOLD_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tsdadel = SDADELxtpresc)
- __CLOCK_HIGH_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tsclh = (SCLH+1)xtpresc)
- __CLOCK_LOW_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tscll = (SCLL+1)xtpresc)

Return value:

- Value: between Min_Data=0 and Max_Data=0xFFFFFFFF

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

64 LL I2S Generic Driver

64.1 I2S Firmware driver registers structures

64.1.1 LL_I2S_InitTypeDef

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()**.

64.2 I2S Firmware driver API description

64.2.1 Detailed description of functions

LL_I2S_Enable

Function name	**__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)**
Function description	Select I2S mode and Enable I2S peripheral.
Parameters	• **SPIx:** SPI Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR I2SMOD LL_I2S_Enable • I2SCFGR I2SE LL_I2S_Enable

LL_I2S_Disable

Function name	**__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)**
Function description	Disable I2S peripheral.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR I2SE LL_I2S_Disable

LL_I2S_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabled (SPI_TypeDef * SPIx)**
Function description	Check if I2S peripheral is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• I2SCFGR I2SE LL_I2S_IsEnabled

LL_I2S_SetDataFormat

Function name	**__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataFormat)**
Function description	Set I2S data frame length.
Parameters	• **SPIx:** SPI Instance • **DataFormat:** This parameter can be one of the following values: – LL_I2S_DATAFORMAT_16B – LL_I2S_DATAFORMAT_16B_EXTENDED – LL_I2S_DATAFORMAT_24B – LL_I2S_DATAFORMAT_32B
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR DATLEN LL_I2S_SetDataFormat • I2SCFGR CHLEN LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (SPI_TypeDef * SPIx)**

Function description	Get I2S data frame length.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_DATAFORMAT_16B – LL_I2S_DATAFORMAT_16B_EXTENDED – LL_I2S_DATAFORMAT_24B – LL_I2S_DATAFORMAT_32B
Reference Manual to LL API cross reference:	• I2SCFGR DATLEN LL_I2S_GetDataFormat • I2SCFGR CHLEN LL_I2S_GetDataFormat

LL_I2S_SetClockPolarity

Function name	**__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)**
Function description	Set I2S clock polarity.
Parameters	• **SPIx:** SPI Instance • **ClockPolarity:** This parameter can be one of the following values: – LL_I2S_POLARITY_LOW – LL_I2S_POLARITY_HIGH
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (SPI_TypeDef * SPIx)**
Function description	Get I2S clock polarity.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_POLARITY_LOW – LL_I2S_POLARITY_HIGH
Reference Manual to LL API cross reference:	• I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name	**__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)**
Function description	Set I2S standard protocol.
Parameters	• **SPIx:** SPI Instance • **Standard:** This parameter can be one of the following values:

- – LL_I2S_STANDARD_PHILIPS
- – LL_I2S_STANDARD_MSB
- – LL_I2S_STANDARD_LSB
- – LL_I2S_STANDARD_PCM_SHORT
- – LL_I2S_STANDARD_PCM_LONG

Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR I2SSTD LL_I2S_SetStandard • I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetStandard (SPI_TypeDef * SPIx)**
Function description	Get I2S standard protocol.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_STANDARD_PHILIPS – LL_I2S_STANDARD_MSB – LL_I2S_STANDARD_LSB – LL_I2S_STANDARD_PCM_SHORT – LL_I2S_STANDARD_PCM_LONG
Reference Manual to LL API cross reference:	• I2SCFGR I2SSTD LL_I2S_GetStandard • I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name	**__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Mode)**
Function description	Set I2S transfer mode.
Parameters	• **SPIx:** SPI Instance • **Mode:** This parameter can be one of the following values: – LL_I2S_MODE_SLAVE_TX – LL_I2S_MODE_SLAVE_RX – LL_I2S_MODE_MASTER_TX – LL_I2S_MODE_MASTER_RX
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (SPI_TypeDef * SPIx)**
Function description	Get I2S transfer mode.

Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_MODE_SLAVE_TX – LL_I2S_MODE_SLAVE_RX – LL_I2S_MODE_MASTER_TX – LL_I2S_MODE_MASTER_RX
Reference Manual to LL API cross reference:	• I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_SetPrescalerLinear

Function name	**__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint8_t PrescalerLinear)**
Function description	Set I2S linear prescaler.
Parameters	• **SPIx:** SPI Instance • **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SPR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (SPI_TypeDef * SPIx)**
Function description	Get I2S linear prescaler.
Parameters	• **SPIx:** SPI Instance
Return values	• **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• I2SPR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name	**__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)**
Function description	Set I2S parity prescaler.
Parameters	• **SPIx:** SPI Instance • **PrescalerParity:** This parameter can be one of the following values: – LL_I2S_PRESCALER_PARITY_EVEN – LL_I2S_PRESCALER_PARITY_ODD
Return values	• **None**

Reference Manual to LL API cross reference:	• I2SPR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (SPI_TypeDef * SPIx)**
Function description	Get I2S parity prescaler.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_PRESCALER_PARITY_EVEN – LL_I2S_PRESCALER_PARITY_ODD
Reference Manual to LL API cross reference:	• I2SPR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name	**__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)**
Function description	Enable the master clock ouput (Pin MCK)
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SPR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name	**__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)**
Function description	Disable the master clock ouput (Pin MCK)
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SPR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (SPI_TypeDef * SPIx)**
Function description	Check if the master clock ouput (Pin MCK) is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• I2SPR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_EnableAsyncStart

Function name	**__STATIC_INLINE void LL_I2S_EnableAsyncStart (SPI_TypeDef * SPIx)**
Function description	Enable asynchronous start.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR ASTRTEN LL_I2S_EnableAsyncStart

LL_I2S_DisableAsyncStart

Function name	**__STATIC_INLINE void LL_I2S_DisableAsyncStart (SPI_TypeDef * SPIx)**
Function description	Disable asynchronous start.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• I2SCFGR ASTRTEN LL_I2S_DisableAsyncStart

LL_I2S_IsEnabledAsyncStart

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledAsyncStart (SPI_TypeDef * SPIx)**
Function description	Check if asynchronous start is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• I2SCFGR ASTRTEN LL_I2S_IsEnabledAsyncStart

LL_I2S_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if Rx buffer is not empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• SR RXNE LL_I2S_IsActiveFlag_RXNE

reference:

LL_I2S_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer is empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TXE LL_I2S_IsActiveFlag_TXE

LL_I2S_IsActiveFlag_BSY

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_BSY (SPI_TypeDef * SPIx)**
Function description	Get busy flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR BSY LL_I2S_IsActiveFlag_BSY

LL_I2S_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Get overrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR OVR LL_I2S_IsActiveFlag_OVR

LL_I2S_IsActiveFlag_UDR

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (SPI_TypeDef * SPIx)**
Function description	Get underrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR UDR LL_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_FRE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Get frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR FRE LL_I2S_IsActiveFlag_FRE

LL_I2S_IsActiveFlag_CHSIDE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_CHSIDE (SPI_TypeDef * SPIx)**
Function description	Get channel side flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• 0: Channel Left has to be transmitted or has been received 1: Channel Right has to be transmitted or has been received It has no significance in PCM mode.
Reference Manual to LL API cross reference:	• SR CHSIDE LL_I2S_IsActiveFlag_CHSIDE

LL_I2S_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Clear overrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR OVR LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_UDR

Function name	**__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)**
Function description	Clear underrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross	• SR UDR LL_I2S_ClearFlag_UDR

reference:

LL_I2S_ClearFlag_FRE

Function name	**__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Clear frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR FRE LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_I2S_EnableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Enable error IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_I2S_EnableIT_ERR

LL_I2S_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_I2S_EnableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Enable Rx buffer not empty IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_I2S_EnableIT_RXNE

LL_I2S_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_I2S_EnableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Enable Tx buffer empty IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to	• CR2 TXEIE LL_I2S_EnableIT_TXE

LL API cross reference:	

LL_I2S_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_I2S_DisableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Disable error IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_I2S_DisableIT_ERR

LL_I2S_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_I2S_DisableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Disable Rx buffer not empty IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_I2S_DisableIT_RXNE

LL_I2S_DisableIT_TXE

Function name	**__STATIC_INLINE void LL_I2S_DisableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Disable Tx buffer empty IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_I2S_DisableIT_TXE

LL_I2S_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_ERR (SPI_TypeDef * SPIx)**
Function description	Check if ERR IT is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CR2 ERRIE LL_I2S_IsEnabledIT_ERR

LL_I2S_IsEnabledIT_RXNE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if RXNE IT is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_I2S_IsEnabledIT_RXNE

LL_I2S_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXE (SPI_TypeDef * SPIx)**
Function description	Check if TXE IT is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_I2S_IsEnabledIT_TXE

LL_I2S_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross	• CR2 RXDMAEN LL_I2S_DisableDMAReq_RX

reference:

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).

64.3 I2S Firmware driver defines

64.3.1 I2S

Audio Frequency

LL_I2S_AUDIOFREQ_192K	Audio Frequency configuration 192000 Hz
LL_I2S_AUDIOFREQ_96K	Audio Frequency configuration 96000 Hz
LL_I2S_AUDIOFREQ_48K	Audio Frequency configuration 48000 Hz
LL_I2S_AUDIOFREQ_44K	Audio Frequency configuration 44100 Hz
LL_I2S_AUDIOFREQ_32K	Audio Frequency configuration 32000 Hz
LL_I2S_AUDIOFREQ_22K	Audio Frequency configuration 22050 Hz
LL_I2S_AUDIOFREQ_16K	Audio Frequency configuration 16000 Hz
LL_I2S_AUDIOFREQ_11K	Audio Frequency configuration 11025 Hz
LL_I2S_AUDIOFREQ_8K	Audio Frequency configuration 8000 Hz
LL_I2S_AUDIOFREQ_DEFAULT	Audio Freq not specified. Register I2SDIV = 2

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_DISABLE	Master clock output is disabled
LL_I2S_MCLK_OUTPUT_ENABLE	Master clock output is enabled

Operation Mode

LL_I2S_MODE_SLAVE_TX	Slave Tx configuration
LL_I2S_MODE_SLAVE_RX	Slave Rx configuration
LL_I2S_MODE_MASTER_TX	Master Tx configuration
LL_I2S_MODE_MASTER_RX	Master Rx configuration

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_PARITY_EVEN	Odd factor: Real divider value is = I2SDIV * 2
LL_I2S_PRESCALER_PARITY_ODD	Odd factor: Real divider value is = (I2SDIV * 2)+1

I2s Standard

LL_I2S_STANDARD_PHILIPS	I2S standard philips
LL_I2S_STANDARD_MSB	MSB justified standard (left justified)
LL_I2S_STANDARD_LSB	LSB justified standard (right justified)
LL_I2S_STANDARD_PCM_SHORT	PCM standard, short frame synchronization
LL_I2S_STANDARD_PCM_LONG	PCM standard, long frame synchronization

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

65 LL IWDG Generic Driver

65.1 IWDG Firmware driver API description

65.1.1 Detailed description of functions

LL_IWDG_Enable

Function name	**__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)**
Function description	Start the Independent Watchdog.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None**
Notes	• Except if the hardware watchdog option is selected
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name	**__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)**
Function description	Reloads IWDG counter with value defined in the reload register.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name	**__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)**
Function description	Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name	**__STATIC_INLINE void LL_IWDG_DisableWriteAccess**

	(IWDG_TypeDef * IWDGx)
Function description	Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name	**__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)**
Function description	Select the prescaler of the IWDG.
Parameters	• **IWDGx:** IWDG Instance • **Prescaler:** This parameter can be one of the following values: – LL_IWDG_PRESCALER_4 – LL_IWDG_PRESCALER_8 – LL_IWDG_PRESCALER_16 – LL_IWDG_PRESCALER_32 – LL_IWDG_PRESCALER_64 – LL_IWDG_PRESCALER_128 – LL_IWDG_PRESCALER_256
Return values	• **None**
Reference Manual to LL API cross reference:	• PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (IWDG_TypeDef * IWDGx)**
Function description	Get the selected prescaler of the IWDG.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **Returned:** value can be one of the following values: – LL_IWDG_PRESCALER_4 – LL_IWDG_PRESCALER_8 – LL_IWDG_PRESCALER_16 – LL_IWDG_PRESCALER_32 – LL_IWDG_PRESCALER_64 – LL_IWDG_PRESCALER_128 – LL_IWDG_PRESCALER_256
Reference Manual to LL API cross reference:	• PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name	**__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)**
Function description	Specify the IWDG down-counter reload value.
Parameters	• **IWDGx:** IWDG Instance • **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF
Return values	• **None**
Reference Manual to LL API cross reference:	• RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name	**__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (IWDG_TypeDef * IWDGx)**
Function description	Get the specified IWDG down-counter reload value.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **Value:** between Min_Data=0 and Max_Data=0x0FFF
Reference Manual to LL API cross reference:	• RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_SetWindow

Function name	**__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)**
Function description	Specify high limit of the window value to be compared to the down-counter.
Parameters	• **IWDGx:** IWDG Instance • **Window:** Value between Min_Data=0 and Max_Data=0x0FFF
Return values	• **None**
Reference Manual to LL API cross reference:	• WINR WIN LL_IWDG_SetWindow

LL_IWDG_GetWindow

Function name	**__STATIC_INLINE uint32_t LL_IWDG_GetWindow (IWDG_TypeDef * IWDGx)**
Function description	Get the high limit of the window value specified.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **Value:** between Min_Data=0 and Max_Data=0x0FFF
Reference Manual to	• WINR WIN LL_IWDG_GetWindow

LL API cross reference:	

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	• SR PVU LL_IWDG_IsReady • SR WVU LL_IWDG_IsReady

reference:
- SR RVU LL_IWDG_IsReady

65.2 IWDG Firmware driver defines

65.2.1 IWDG

Get Flags Defines

LL_IWDG_SR_PVU	Watchdog prescaler value update
LL_IWDG_SR_RVU	Watchdog counter reload value update
LL_IWDG_SR_WVU	Watchdog counter window value update

Prescaler Divider

LL_IWDG_PRESCALER_4	Divider by 4
LL_IWDG_PRESCALER_8	Divider by 8
LL_IWDG_PRESCALER_16	Divider by 16
LL_IWDG_PRESCALER_32	Divider by 32
LL_IWDG_PRESCALER_64	Divider by 64
LL_IWDG_PRESCALER_128	Divider by 128
LL_IWDG_PRESCALER_256	Divider by 256

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

66 LL LPTIM Generic Driver

66.1 LPTIM Firmware driver registers structures

66.1.1 LL_LPTIM_InitTypeDef

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t Waveform***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t LL_LPTIM_InitTypeDef::ClockSource***
  Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of ***LPTIM_LL_EC_CLK_SOURCE***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetClockSource()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Prescaler***
  Specifies the prescaler division ratio. This parameter can be a value of ***LPTIM_LL_EC_PRESCALER***.This feature can be modified afterwards using using unitary function **LL_LPTIM_SetPrescaler()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Waveform***
  Specifies the waveform shape. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_WAVEFORM***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Polarity***
  Specifies waveform polarity. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_POLARITY***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.

66.2 LPTIM Firmware driver API description

66.2.1 Detailed description of functions

LL_LPTIM_Enable

Function name	**__STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)**
Function description	Enable the LPTIM instance.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Notes	• After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.
Reference Manual to LL API cross reference:	• CR ENABLE LL_LPTIM_Enable

LL_LPTIM_Disable

Function name	**__STATIC_INLINE void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)**
Function description	Disable the LPTIM instance.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR ENABLE LL_LPTIM_Disable

LL_LPTIM_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the LPTIM instance is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ENABLE LL_LPTIM_IsEnabled

LL_LPTIM_StartCounter

Function name	**__STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)**
Function description	Starts the LPTIM counter in the desired mode.
Parameters	• **LPTIMx:** Low-Power Timer instance • **OperatingMode:** This parameter can be one of the following values: – LL_LPTIM_OPERATING_MODE_CONTINUOUS – LL_LPTIM_OPERATING_MODE_ONESHOT
Return values	• **None**
Notes	• LPTIM instance must be enabled before starting the counter. • It is possible to change on the fly from One Shot mode to Continuous mode.
Reference Manual to LL API cross reference:	• CR CNTSTRT LL_LPTIM_StartCounter • CR SNGSTRT LL_LPTIM_StartCounter

LL_LPTIM_SetUpdateMode

Function name	**__STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)**
Function description	Set the LPTIM registers update mode (enable/disable register preload)

Parameters	• **LPTIMx:** Low-Power Timer instance • **UpdateMode:** This parameter can be one of the following values: – LL_LPTIM_UPDATE_MODE_IMMEDIATE – LL_LPTIM_UPDATE_MODE_ENDOFPERIOD
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR PRELOAD LL_LPTIM_SetUpdateMode

LL_LPTIM_GetUpdateMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode (LPTIM_TypeDef * LPTIMx)**
Function description	Get the LPTIM registers update mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_UPDATE_MODE_IMMEDIATE – LL_LPTIM_UPDATE_MODE_ENDOFPERIOD
Reference Manual to LL API cross reference:	• CFGR PRELOAD LL_LPTIM_GetUpdateMode

LL_LPTIM_SetAutoReload

Function name	**__STATIC_INLINE void LL_LPTIM_SetAutoReload (LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)**
Function description	Set the auto reload value.
Parameters	• **LPTIMx:** Low-Power Timer instance • **AutoReload:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None**
Notes	• The LPTIMx_ARR register content must only be modified when the LPTIM is enabled • After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag be set, will lead to unpredictable results. • autoreload value be strictly greater than the compare value.
Reference Manual to LL API cross reference:	• ARR ARR LL_LPTIM_SetAutoReload

LL_LPTIM_GetAutoReload

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual auto reload value.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **AutoReload:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• ARR ARR LL_LPTIM_GetAutoReload

LL_LPTIM_SetCompare

Function name	**__STATIC_INLINE void LL_LPTIM_SetCompare (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)**
Function description	Set the compare value.
Parameters	• **LPTIMx:** Low-Power Timer instance • **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None**
Notes	• After a write to the LPTIMx_CMP register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMPOK flag be set, will lead to unpredictable results.
Reference Manual to LL API cross reference:	• CMP CMP LL_LPTIM_SetCompare

LL_LPTIM_GetCompare

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetCompare (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual compare value.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• CMP CMP LL_LPTIM_GetCompare

LL_LPTIM_GetCounter

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetCounter (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual counter value.

Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Counter:** value
Notes	• When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.
Reference Manual to LL API cross reference:	• CNT CNT LL_LPTIM_GetCounter

LL_LPTIM_SetCounterMode

Function name	**__STATIC_INLINE void LL_LPTIM_SetCounterMode (LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)**
Function description	Set the counter mode (selection of the LPTIM counter clock source).
Parameters	• **LPTIMx:** Low-Power Timer instance • **CounterMode:** This parameter can be one of the following values: – LL_LPTIM_COUNTER_MODE_INTERNAL – LL_LPTIM_COUNTER_MODE_EXTERNAL
Return values	• **None**
Notes	• The counter mode can be set only when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR COUNTMODE LL_LPTIM_SetCounterMode

LL_LPTIM_GetCounterMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode (LPTIM_TypeDef * LPTIMx)**
Function description	Get the counter mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_COUNTER_MODE_INTERNAL – LL_LPTIM_COUNTER_MODE_EXTERNAL
Reference Manual to LL API cross reference:	• CFGR COUNTMODE LL_LPTIM_GetCounterMode

LL_LPTIM_ConfigOutput

Function name	**__STATIC_INLINE void LL_LPTIM_ConfigOutput (LPTIM_TypeDef * LPTIMx, uint32_t Waveform, uint32_t Polarity)**

Function description	Configure the LPTIM instance output (LPTIMx_OUT)
Parameters	• **LPTIMx:** Low-Power Timer instance • **Waveform:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_WAVEFORM_PWM – LL_LPTIM_OUTPUT_WAVEFORM_SETONCE • **Polarity:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_POLARITY_REGULAR – LL_LPTIM_OUTPUT_POLARITY_INVERSE
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled. • Regarding the LPTIM output polarity the change takes effect immediately, so the output default value will change immediately after the polarity is re-configured, even before the timer is enabled.
Reference Manual to LL API cross reference:	• CFGR WAVE LL_LPTIM_ConfigOutput • CFGR WAVPOL LL_LPTIM_ConfigOutput

LL_LPTIM_SetWaveform

Function name	**__STATIC_INLINE void LL_LPTIM_SetWaveform (LPTIM_TypeDef * LPTIMx, uint32_t Waveform)**
Function description	Set waveform shape.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Waveform:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_WAVEFORM_PWM – LL_LPTIM_OUTPUT_WAVEFORM_SETONCE
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR WAVE LL_LPTIM_SetWaveform

LL_LPTIM_GetWaveform

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual waveform shape.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_OUTPUT_WAVEFORM_PWM – LL_LPTIM_OUTPUT_WAVEFORM_SETONCE
Reference Manual to LL API cross reference:	• CFGR WAVE LL_LPTIM_GetWaveform

LL_LPTIM_SetPolarity

Function name	**__STATIC_INLINE void LL_LPTIM_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Polarity)**
Function description	Set output polarity.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Polarity:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_POLARITY_REGULAR – LL_LPTIM_OUTPUT_POLARITY_INVERSE
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR WAVPOL LL_LPTIM_SetPolarity

LL_LPTIM_GetPolarity

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetPolarity (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual output polarity.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_OUTPUT_POLARITY_REGULAR – LL_LPTIM_OUTPUT_POLARITY_INVERSE
Reference Manual to LL API cross reference:	• CFGR WAVPOL LL_LPTIM_GetPolarity

LL_LPTIM_SetPrescaler

Function name	**__STATIC_INLINE void LL_LPTIM_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)**
Function description	Set actual prescaler division ratio.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Prescaler:** This parameter can be one of the following values: – LL_LPTIM_PRESCALER_DIV1 – LL_LPTIM_PRESCALER_DIV2 – LL_LPTIM_PRESCALER_DIV4 – LL_LPTIM_PRESCALER_DIV8 – LL_LPTIM_PRESCALER_DIV16 – LL_LPTIM_PRESCALER_DIV32 – LL_LPTIM_PRESCALER_DIV64 – LL_LPTIM_PRESCALER_DIV128
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled. • When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be

	updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.
Reference Manual to LL API cross reference:	• CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual prescaler division ratio.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_PRESCALER_DIV1 – LL_LPTIM_PRESCALER_DIV2 – LL_LPTIM_PRESCALER_DIV4 – LL_LPTIM_PRESCALER_DIV8 – LL_LPTIM_PRESCALER_DIV16 – LL_LPTIM_PRESCALER_DIV32 – LL_LPTIM_PRESCALER_DIV64 – LL_LPTIM_PRESCALER_DIV128
Reference Manual to LL API cross reference:	• CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_EnableTimeout

Function name	**__STATIC_INLINE void LL_LPTIM_EnableTimeout (LPTIM_TypeDef * LPTIMx)**
Function description	Enable the timeout function.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled. • The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. • The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.
Reference Manual to LL API cross reference:	• CFGR TIMOUT LL_LPTIM_EnableTimeout

LL_LPTIM_DisableTimeout

Function name	**__STATIC_INLINE void LL_LPTIM_DisableTimeout (LPTIM_TypeDef * LPTIMx)**

Function description	Disable the timeout function.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled. • A trigger event arriving when the timer is already started will be ignored.
Reference Manual to LL API cross reference:	• CFGR TIMOUT LL_LPTIM_DisableTimeout

LL_LPTIM_IsEnabledTimeout

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout (LPTIM_TypeDef * LPTIMx)**
Function description	Indicate whether the timeout function is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CFGR TIMOUT LL_LPTIM_IsEnabledTimeout

LL_LPTIM_TrigSw

Function name	**__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)**
Function description	Start the LPTIM counter.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR TRIGEN LL_LPTIM_TrigSw

LL_LPTIM_ConfigTrigger

Function name	**__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)**
Function description	Configure the external trigger used as a trigger event for the LPTIM.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Source:** This parameter can be one of the following values: – LL_LPTIM_TRIG_SOURCE_GPIO – LL_LPTIM_TRIG_SOURCE_RTCALARMA

	– LL_LPTIM_TRIG_SOURCE_RTCALARMB – LL_LPTIM_TRIG_SOURCE_RTCTAMP1 – LL_LPTIM_TRIG_SOURCE_RTCTAMP2 – LL_LPTIM_TRIG_SOURCE_RTCTAMP3 – LL_LPTIM_TRIG_SOURCE_COMP1 – LL_LPTIM_TRIG_SOURCE_COMP2 • **Filter:** This parameter can be one of the following values: – LL_LPTIM_TRIG_FILTER_NONE – LL_LPTIM_TRIG_FILTER_2 – LL_LPTIM_TRIG_FILTER_4 – LL_LPTIM_TRIG_FILTER_8 • **Polarity:** This parameter can be one of the following values: – LL_LPTIM_TRIG_POLARITY_RISING – LL_LPTIM_TRIG_POLARITY_FALLING – LL_LPTIM_TRIG_POLARITY_RISING_FALLING
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled. • An internal clock source must be present when a digital filter is required for the trigger.
Reference Manual to LL API cross reference:	• CFGR TRIGSEL LL_LPTIM_ConfigTrigger • CFGR TRGFLT LL_LPTIM_ConfigTrigger • CFGR TRIGEN LL_LPTIM_ConfigTrigger

LL_LPTIM_GetTriggerSource

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual external trigger source.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_TRIG_SOURCE_GPIO – LL_LPTIM_TRIG_SOURCE_RTCALARMA – LL_LPTIM_TRIG_SOURCE_RTCALARMB – LL_LPTIM_TRIG_SOURCE_RTCTAMP1 – LL_LPTIM_TRIG_SOURCE_RTCTAMP2 – LL_LPTIM_TRIG_SOURCE_RTCTAMP3 – LL_LPTIM_TRIG_SOURCE_COMP1 – LL_LPTIM_TRIG_SOURCE_COMP2
Reference Manual to LL API cross reference:	• CFGR TRIGSEL LL_LPTIM_GetTriggerSource

LL_LPTIM_GetTriggerFilter

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual external trigger filter.

Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_TRIG_FILTER_NONE – LL_LPTIM_TRIG_FILTER_2 – LL_LPTIM_TRIG_FILTER_4 – LL_LPTIM_TRIG_FILTER_8
Reference Manual to LL API cross reference:	• CFGR TRGFLT LL_LPTIM_GetTriggerFilter

LL_LPTIM_GetTriggerPolarity

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual external trigger polarity.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_TRIG_POLARITY_RISING – LL_LPTIM_TRIG_POLARITY_FALLING – LL_LPTIM_TRIG_POLARITY_RISING_FALLING
Reference Manual to LL API cross reference:	• CFGR TRIGEN LL_LPTIM_GetTriggerPolarity

LL_LPTIM_SetClockSource

Function name	**__STATIC_INLINE void LL_LPTIM_SetClockSource (LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)**
Function description	Set the source of the clock used by the LPTIM instance.
Parameters	• **LPTIMx:** Low-Power Timer instance • **ClockSource:** This parameter can be one of the following values: – LL_LPTIM_CLK_SOURCE_INTERNAL – LL_LPTIM_CLK_SOURCE_EXTERNAL
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR CKSEL LL_LPTIM_SetClockSource

LL_LPTIM_GetClockSource

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual LPTIM instance clock source.
Parameters	• **LPTIMx:** Low-Power Timer instance

Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_CLK_SOURCE_INTERNAL – LL_LPTIM_CLK_SOURCE_EXTERNAL
Reference Manual to LL API cross reference:	• CFGR CKSEL LL_LPTIM_GetClockSource

LL_LPTIM_ConfigClock

Function name	**__STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)**
Function description	Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.
Parameters	• **LPTIMx:** Low-Power Timer instance • **ClockFilter:** This parameter can be one of the following values: – LL_LPTIM_CLK_FILTER_NONE – LL_LPTIM_CLK_FILTER_2 – LL_LPTIM_CLK_FILTER_4 – LL_LPTIM_CLK_FILTER_8 • **ClockPolarity:** This parameter can be one of the following values: – LL_LPTIM_CLK_POLARITY_RISING – LL_LPTIM_CLK_POLARITY_FALLING – LL_LPTIM_CLK_POLARITY_RISING_FALLING
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled. • When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency. • An internal clock source must be present when a digital filter is required for external clock.
Reference Manual to LL API cross reference:	• CFGR CKFLT LL_LPTIM_ConfigClock • CFGR CKPOL LL_LPTIM_ConfigClock

LL_LPTIM_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual clock polarity.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_CLK_POLARITY_RISING – LL_LPTIM_CLK_POLARITY_FALLING – LL_LPTIM_CLK_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:	• CFGR CKPOL LL_LPTIM_GetClockPolarity

LL_LPTIM_GetClockFilter

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual clock digital filter.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_CLK_FILTER_NONE – LL_LPTIM_CLK_FILTER_2 – LL_LPTIM_CLK_FILTER_4 – LL_LPTIM_CLK_FILTER_8
Reference Manual to LL API cross reference:	• CFGR CKFLT LL_LPTIM_GetClockFilter

LL_LPTIM_SetEncoderMode

Function name	**__STATIC_INLINE void LL_LPTIM_SetEncoderMode (LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)**
Function description	Configure the encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance • **EncoderMode:** This parameter can be one of the following values: – LL_LPTIM_ENCODER_MODE_RISING – LL_LPTIM_ENCODER_MODE_FALLING – LL_LPTIM_ENCODER_MODE_RISING_FALLING
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR CKPOL LL_LPTIM_SetEncoderMode

LL_LPTIM_GetEncoderMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_ENCODER_MODE_RISING – LL_LPTIM_ENCODER_MODE_FALLING – LL_LPTIM_ENCODER_MODE_RISING_FALLING

Reference Manual to LL API cross reference:	• CFGR CKPOL LL_LPTIM_GetEncoderMode

LL_LPTIM_EnableEncoderMode

Function name	**__STATIC_INLINE void LL_LPTIM_EnableEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Enable the encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled. • In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1. • LPTIM instance must be configured in continuous mode prior enabling the encoder mode.
Reference Manual to LL API cross reference:	• CFGR ENC LL_LPTIM_EnableEncoderMode

LL_LPTIM_DisableEncoderMode

Function name	**__STATIC_INLINE void LL_LPTIM_DisableEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Disable the encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR ENC LL_LPTIM_DisableEncoderMode

LL_LPTIM_IsEnabledEncoderMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the LPTIM operates in encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CFGR ENC LL_LPTIM_IsEnabledEncoderMode

LL_LPTIM_ClearFLAG_CMPM

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFLAG_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the compare match flag (CMPMCF)
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR CMPMCF LL_LPTIM_ClearFLAG_CMPM

LL_LPTIM_IsActiveFlag_CMPM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Inform application whether a compare match interrupt has occurred.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMPM LL_LPTIM_IsActiveFlag_CMPM

LL_LPTIM_ClearFLAG_ARRM

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFLAG_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the autoreload match flag (ARRMCF)
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ARRMCF LL_LPTIM_ClearFLAG_ARRM

LL_LPTIM_IsActiveFlag_ARRM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Inform application whether a autoreload match interrupt has occured.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ARRM LL_LPTIM_IsActiveFlag_ARRM

LL_LPTIM_ClearFlag_EXTTRIG

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the external trigger valid edge flag(EXTTRIGCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG

LL_LPTIM_IsActiveFlag_EXTTRIG

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Inform application whether a valid edge on the selected external trigger input has occurred.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG

LL_LPTIM_ClearFlag_CMPOK

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the compare register update interrupt flag (CMPOKCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR CMPOKCF LL_LPTIM_ClearFlag_CMPOK

LL_LPTIM_IsActiveFlag_CMPOK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Informs application whether the APB bus write operation to the LPTIMx_CMP register has been successfully completed; If so, a new one can be initiated.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMPOK LL_LPTIM_IsActiveFlag_CMPOK

LL_LPTIM_ClearFlag_ARROK

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the autoreload register update interrupt flag (ARROKCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ARROKCF LL_LPTIM_ClearFlag_ARROK

LL_LPTIM_IsActiveFlag_ARROK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Informs application whether the APB bus write operation to the LPTIMx_ARR register has been successfully completed; If so, a new one can be initiated.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ARROK LL_LPTIM_IsActiveFlag_ARROK

LL_LPTIM_ClearFlag_UP

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the counter direction change to up interrupt flag (UPCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR UPCF LL_LPTIM_ClearFlag_UP

LL_LPTIM_IsActiveFlag_UP

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• ISR UP LL_LPTIM_IsActiveFlag_UP

reference:

LL_LPTIM_ClearFlag_DOWN

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the counter direction change to down interrupt flag (DOWNCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_EnableIT_CMPM

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Enable compare match interrupt (CMPMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER CMPMIE LL_LPTIM_EnableIT_CMPM

LL_LPTIM_DisableIT_CMPM

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Disable compare match interrupt (CMPMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**

Reference Manual to LL API cross reference:	• IER CMPMIE LL_LPTIM_DisableIT_CMPM

LL_LPTIM_IsEnabledIT_CMPM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the compare match interrupt (CMPMIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER CMPMIE LL_LPTIM_IsEnabledIT_CMPM

LL_LPTIM_EnableIT_ARRM

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Enable autoreload match interrupt (ARRMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Disable autoreload match interrupt (ARRMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the autoreload match interrupt (ARRMIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• IER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Enable external trigger valid edge interrupt (EXTTRIGIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Disable external trigger valid edge interrupt (EXTTRIGIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMPOK

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Enable compare register write completed interrupt (CMPOKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**

Reference Manual to LL API cross reference:	• IER CMPOKIE LL_LPTIM_EnableIT_CMPOK

LL_LPTIM_DisableIT_CMPOK

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Disable compare register write completed interrupt (CMPOKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER CMPOKIE LL_LPTIM_DisableIT_CMPOK

LL_LPTIM_IsEnabledIT_CMPOK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the compare register write completed interrupt (CMPOKIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER CMPOKIE LL_LPTIM_IsEnabledIT_CMPOK

LL_LPTIM_EnableIT_ARROK

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Enable autoreload register write completed interrupt (ARROKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Disable autoreload register write completed interrupt (ARROKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**

Reference Manual to LL API cross reference:	• IER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Enable direction change to up interrupt (UPIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Disable direction change to up interrupt (UPIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the direction change to up interrupt (UPIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• IER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Enable direction change to down interrupt (DOWNIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Disable direction change to down interrupt (DOWNIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• IER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the direction change to down interrupt (DOWNIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

LL_LPTIM_DeInit

Function name	**ErrorStatus LL_LPTIM_DeInit (LPTIM_TypeDef * LPTIMx)**
Function description	Set LPTIMx registers to their reset values.
Parameters	• **LPTIMx:** LP Timer instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPTIMx registers are de-initialized – ERROR: invalid LPTIMx instance

LL_LPTIM_StructInit

Function name	**void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)**
Function description	Set each fields of the LPTIM_InitStruct structure to its default value.
Parameters	• **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure
Return values	• **None**

LL_LPTIM_Init

Function name	**ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, LL_LPTIM_InitTypeDef * LPTIM_InitStruct)**
Function description	Configure the LPTIMx peripheral according to the specified parameters.
Parameters	• **LPTIMx:** LP Timer Instance • **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPTIMx instance has been initialized – ERROR: LPTIMx instance hasn't been initialized
Notes	• LL_LPTIM_Init can only be called when the LPTIM instance is disabled. • LPTIMx can be disabled using unitary function LL_LPTIM_Disable().

66.3 LPTIM Firmware driver defines

66.3.1 LPTIM

Clock Filter

LL_LPTIM_CLK_FILTER_NONE	Any external clock signal level change is considered as a valid transition
LL_LPTIM_CLK_FILTER_2	External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition
LL_LPTIM_CLK_FILTER_4	External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition
LL_LPTIM_CLK_FILTER_8	External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Clock Polarity

LL_LPTIM_CLK_POLARITY_RISING	The rising edge is the active edge used for counting
LL_LPTIM_CLK_POLARITY_FALLING	The falling edge is the active edge used

	for counting
LL_LPTIM_CLK_POLARITY_RISING_FALLING	Both edges are active edges

Clock Source

LL_LPTIM_CLK_SOURCE_INTERNAL	LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)
LL_LPTIM_CLK_SOURCE_EXTERNAL	LPTIM is clocked by an external clock source through the LPTIM external Input1

Counter Mode

LL_LPTIM_COUNTER_MODE_INTERNAL	The counter is incremented following each internal clock pulse
LL_LPTIM_COUNTER_MODE_EXTERNAL	The counter is incremented following each valid clock pulse on the LPTIM external Input1

Encoder Mode

LL_LPTIM_ENCODER_MODE_RISING	The rising edge is the active edge used for counting
LL_LPTIM_ENCODER_MODE_FALLING	The falling edge is the active edge used for counting
LL_LPTIM_ENCODER_MODE_RISING_FALLING	Both edges are active edges

Get Flags Defines

LL_LPTIM_ISR_CMPM	Compare match
LL_LPTIM_ISR_ARRM	Autoreload match
LL_LPTIM_ISR_EXTTRIG	External trigger edge event
LL_LPTIM_ISR_CMPOK	Compare register update OK
LL_LPTIM_ISR_ARROK	Autoreload register update OK
LL_LPTIM_ISR_UP	Counter direction change down to up
LL_LPTIM_ISR_DOWN	Counter direction change up to down

IT Defines

LL_LPTIM_IER_CMPMIE	Compare match Interrupt Enable
LL_LPTIM_IER_ARRMIE	Autoreload match Interrupt Enable
LL_LPTIM_IER_EXTTRIGIE	External trigger valid edge Interrupt Enable
LL_LPTIM_IER_CMPOKIE	Compare register update OK Interrupt Enable
LL_LPTIM_IER_ARROKIE	Autoreload register update OK Interrupt Enable
LL_LPTIM_IER_UPIE	Direction change to UP Interrupt Enable
LL_LPTIM_IER_DOWNIE	Direction change to down Interrupt Enable

Operating Mode

LL_LPTIM_OPERATING_MODE_CONTINUOUS	LP Timer starts in continuous mode
LL_LPTIM_OPERATING_MODE_ONESHOT	LP Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR	The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CMP registers
LL_LPTIM_OUTPUT_POLARITY_INVERSE	The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CMP registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM	LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINOUS or SINGLE
LL_LPTIM_OUTPUT_WAVEFORM_SETONCE	LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1	Prescaler division factor is set to 1
LL_LPTIM_PRESCALER_DIV2	Prescaler division factor is set to 2
LL_LPTIM_PRESCALER_DIV4	Prescaler division factor is set to 4
LL_LPTIM_PRESCALER_DIV8	Prescaler division factor is set to 8
LL_LPTIM_PRESCALER_DIV16	Prescaler division factor is set to 16
LL_LPTIM_PRESCALER_DIV32	Prescaler division factor is set to 32
LL_LPTIM_PRESCALER_DIV64	Prescaler division factor is set to 64
LL_LPTIM_PRESCALER_DIV128	Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE	Any trigger active level change is considered as a valid trigger
LL_LPTIM_TRIG_FILTER_2	Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger
LL_LPTIM_TRIG_FILTER_4	Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger
LL_LPTIM_TRIG_FILTER_8	Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING	LPTIM counter starts when a rising edge is detected
LL_LPTIM_TRIG_POLARITY_FALLING	LPTIM counter starts when a falling edge is detected
LL_LPTIM_TRIG_POLARITY_RISING_FALLING	LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO	External input trigger is connected to TIMx_ETR input
LL_LPTIM_TRIG_SOURCE_RTCALARMA	External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB	External input trigger is connected to RTC Alarm B
LL_LPTIM_TRIG_SOURCE_RTCTAMP1	External input trigger is connected to RTC Tamper 1
LL_LPTIM_TRIG_SOURCE_RTCTAMP2	External input trigger is connected to RTC Tamper 2
LL_LPTIM_TRIG_SOURCE_RTCTAMP3	External input trigger is connected to RTC Tamper 3
LL_LPTIM_TRIG_SOURCE_COMP1	External input trigger is connected to COMP1 output
LL_LPTIM_TRIG_SOURCE_COMP2	External input trigger is connected to COMP2 output

Update Mode

LL_LPTIM_UPDATE_MODE_IMMEDIATE	Preload is disabled: registers are updated after each APB bus write access
LL_LPTIM_UPDATE_MODE_ENDOFPERIOD	preload is enabled: registers are updated at the end of the current LPTIM period

Common Write and read registers Macros

LL_LPTIM_WriteReg

Description:

- Write a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPTIM_ReadReg

Description:

- Read a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

67 LL LPUART Generic Driver

67.1 LPUART Firmware driver registers structures

67.1.1 LL_LPUART_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***
  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***LPUART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***LPUART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***LPUART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

67.2 LPUART Firmware driver API description

67.2.1 Detailed description of functions

LL_LPUART_Enable

Function name	**__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef * LPUARTx)**
Function description	LPUART Enable.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name	**__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)**
Function description	LPUART Disable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Notes	• When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values. • In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.
Reference Manual to LL API cross reference:	• CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (USART_TypeDef * LPUARTx)**
Function description	Indicate if LPUART is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableInStopMode

Function name	**__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)**
Function description	LPUART enabled in STOP Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Notes	• When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock

		selection is HSI or LSE in RCC.
Reference Manual to LL API cross reference:	•	CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name	**__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)**
Function description	LPUART disabled in STOP Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Notes	• When this function is disabled, LPUART is not able to wake up the MCU from Stop mode
Reference Manual to LL API cross reference:	• CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (USART_TypeDef * LPUARTx)**
Function description	Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name	**__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)**
Function description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name	**__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)**

Function description	Receiver Disable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name	**__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)**
Function description	Transmitter Enable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name	**__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)**
Function description	Transmitter Disable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name	**__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)**
Function description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	• **LPUARTx:** LPUART Instance • **TransferDirection:** This parameter can be one of the following values: – LL_LPUART_DIRECTION_NONE – LL_LPUART_DIRECTION_RX – LL_LPUART_DIRECTION_TX – LL_LPUART_DIRECTION_TX_RX
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_SetTransferDirection • CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection (USART_TypeDef * LPUARTx)**
Function description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_DIRECTION_NONE – LL_LPUART_DIRECTION_RX – LL_LPUART_DIRECTION_TX – LL_LPUART_DIRECTION_TX_RX
Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_GetTransferDirection • CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name	**__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)**
Function description	Configure Parity (enabled/disabled and parity mode if enabled)
Parameters	• **LPUARTx:** LPUART Instance • **Parity:** This parameter can be one of the following values: – LL_LPUART_PARITY_NONE – LL_LPUART_PARITY_EVEN – LL_LPUART_PARITY_ODD
Return values	• **None**
Notes	• This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.
Reference Manual to LL API cross reference:	• CR1 PS LL_LPUART_SetParity • CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetParity (USART_TypeDef * LPUARTx)**
Function description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_PARITY_NONE – LL_LPUART_PARITY_EVEN – LL_LPUART_PARITY_ODD
Reference Manual to LL API cross	• CR1 PS LL_LPUART_GetParity

reference:	• CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name	**__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)**
Function description	Set Receiver Wake Up method from Mute mode.
Parameters	• **LPUARTx:** LPUART Instance • **Method:** This parameter can be one of the following values: – LL_LPUART_WAKEUP_IDLELINE – LL_LPUART_WAKEUP_ADDRESSMARK
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (USART_TypeDef * LPUARTx)**
Function description	Return Receiver Wake Up method from Mute mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_WAKEUP_IDLELINE – LL_LPUART_WAKEUP_ADDRESSMARK
Reference Manual to LL API cross reference:	• CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name	**__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)**
Function description	Set Word length (nb of data bits, excluding start and stop bits)
Parameters	• **LPUARTx:** LPUART Instance • **DataWidth:** This parameter can be one of the following values: – LL_LPUART_DATAWIDTH_7B – LL_LPUART_DATAWIDTH_8B – LL_LPUART_DATAWIDTH_9B
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (USART_TypeDef * LPUARTx)**
Function description	Return Word length (i.e.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_DATAWIDTH_7B – LL_LPUART_DATAWIDTH_8B – LL_LPUART_DATAWIDTH_9B
Reference Manual to LL API cross reference:	• CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name	**__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)**
Function description	Allow switch between Mute Mode and Active mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name	**__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)**
Function description	Prevent Mute Mode use.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (USART_TypeDef * LPUARTx)**
Function description	Indicate if switch between Mute Mode and Active mode is allowed.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetStopBitsLength

Function name	**__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)**
Function description	Set the length of the stop bits.
Parameters	• **LPUARTx:** LPUART Instance • **StopBits:** This parameter can be one of the following values: – LL_LPUART_STOPBITS_1 – LL_LPUART_STOPBITS_2
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (USART_TypeDef * LPUARTx)**
Function description	Retrieve the length of the stop bits.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_STOPBITS_1 – LL_LPUART_STOPBITS_2
Reference Manual to LL API cross reference:	• CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name	**__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)**
Function description	Configure Character frame format (Datawidth, Parity control, Stop Bits)
Parameters	• **LPUARTx:** LPUART Instance • **DataWidth:** This parameter can be one of the following values: – LL_LPUART_DATAWIDTH_7B – LL_LPUART_DATAWIDTH_8B – LL_LPUART_DATAWIDTH_9B • **Parity:** This parameter can be one of the following values: – LL_LPUART_PARITY_NONE – LL_LPUART_PARITY_EVEN – LL_LPUART_PARITY_ODD • **StopBits:** This parameter can be one of the following values: – LL_LPUART_STOPBITS_1 – LL_LPUART_STOPBITS_2
Return values	• **None**

Notes	• Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function
Reference Manual to LL API cross reference:	• CR1 PS LL_LPUART_ConfigCharacter • CR1 PCE LL_LPUART_ConfigCharacter • CR1 M LL_LPUART_ConfigCharacter • CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name	**__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)**
Function description	Configure TX/RX pins swapping setting.
Parameters	• **LPUARTx:** LPUART Instance • **SwapConfig:** This parameter can be one of the following values: – LL_LPUART_TXRX_STANDARD – LL_LPUART_TXRX_SWAPPED
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (USART_TypeDef * LPUARTx)**
Function description	Retrieve TX/RX pins swapping configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_TXRX_STANDARD – LL_LPUART_TXRX_SWAPPED
Reference Manual to LL API cross reference:	• CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name	**__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)**
Function description	Configure RX pin active level logic.
Parameters	• **LPUARTx:** LPUART Instance • **PinInvMethod:** This parameter can be one of the following values: – LL_LPUART_RXPIN_LEVEL_STANDARD

	– LL_LPUART_RXPIN_LEVEL_INVERTED
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (USART_TypeDef * LPUARTx)**
Function description	Retrieve RX pin active level logic configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_RXPIN_LEVEL_STANDARD – LL_LPUART_RXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name	**__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)**
Function description	Configure TX pin active level logic.
Parameters	• **LPUARTx:** LPUART Instance • **PinInvMethod:** This parameter can be one of the following values: – LL_LPUART_TXPIN_LEVEL_STANDARD – LL_LPUART_TXPIN_LEVEL_INVERTED
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (USART_TypeDef * LPUARTx)**
Function description	Retrieve TX pin active level logic configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_TXPIN_LEVEL_STANDARD – LL_LPUART_TXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name	**__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)**
Function description	Configure Binary data logic.
Parameters	• **LPUARTx:** LPUART Instance • **DataLogic:** This parameter can be one of the following values: – LL_LPUART_BINARY_LOGIC_POSITIVE – LL_LPUART_BINARY_LOGIC_NEGATIVE
Return values	• **None**
Notes	• Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (USART_TypeDef * LPUARTx)**
Function description	Retrieve Binary data configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_BINARY_LOGIC_POSITIVE – LL_LPUART_BINARY_LOGIC_NEGATIVE
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name	**__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)**
Function description	Configure transfer bit order (either Less or Most Significant Bit First)
Parameters	• **LPUARTx:** LPUART Instance • **BitOrder:** This parameter can be one of the following values: – LL_LPUART_BITORDER_LSBFIRST – LL_LPUART_BITORDER_MSBFIRST
Return values	• **None**
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross	• CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

reference:

LL_LPUART_GetTransferBitOrder

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (USART_TypeDef * LPUARTx)**
Function description	Return transfer bit order (either Less or Most Significant Bit First)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_BITORDER_LSBFIRST – LL_LPUART_BITORDER_MSBFIRST
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross reference:	• CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name	**__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)**
Function description	Set Address of the LPUART node.
Parameters	• **LPUARTx:** LPUART Instance • **AddressLen:** This parameter can be one of the following values: – LL_LPUART_ADDRESS_DETECT_4B – LL_LPUART_ADDRESS_DETECT_7B • **NodeAddress:** 4 or 7 bit Address of the LPUART node.
Return values	• **None**
Notes	• This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection. • 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)
Reference Manual to LL API cross reference:	• CR2 ADD LL_LPUART_ConfigNodeAddress • CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (USART_TypeDef * LPUARTx)**
Function description	Return 8 bit Address of the LPUART node as set in ADD field of CR2.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)
Notes	• If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)
Reference Manual to LL API cross reference:	• CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (USART_TypeDef * LPUARTx)**
Function description	Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_ADDRESS_DETECT_4B – LL_LPUART_ADDRESS_DETECT_7B
Reference Manual to LL API cross reference:	• CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Enable RTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)**

Function description	Disable RTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Enable CTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Disable CTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)**
Function description	Configure HW Flow Control mode (both CTS and RTS)
Parameters	• **LPUARTx:** LPUART Instance • **HardwareFlowControl:** This parameter can be one of the following values: – LL_LPUART_HWCONTROL_NONE – LL_LPUART_HWCONTROL_RTS – LL_LPUART_HWCONTROL_CTS – LL_LPUART_HWCONTROL_RTS_CTS
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_SetHWFlowCtrl • CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Return HW Flow Control configuration (both CTS and RTS)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_HWCONTROL_NONE – LL_LPUART_HWCONTROL_RTS – LL_LPUART_HWCONTROL_CTS – LL_LPUART_HWCONTROL_RTS_CTS
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_GetHWFlowCtrl • CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name	**__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)**
Function description	Enable Overrun detection.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name	**__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)**
Function description	Disable Overrun detection.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (USART_TypeDef * LPUARTx)**
Function description	Indicate if Overrun detection is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to	• CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL API cross reference:

LL_LPUART_SetWKUPType

Function name	**__STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)**
Function description	Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **LPUARTx:** LPUART Instance • **Type:** This parameter can be one of the following values: – LL_LPUART_WAKEUP_ON_ADDRESS – LL_LPUART_WAKEUP_ON_STARTBIT – LL_LPUART_WAKEUP_ON_RXNE
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 WUS LL_LPUART_SetWKUPType

LL_LPUART_GetWKUPType

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (USART_TypeDef * LPUARTx)**
Function description	Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_WAKEUP_ON_ADDRESS – LL_LPUART_WAKEUP_ON_STARTBIT – LL_LPUART_WAKEUP_ON_RXNE
Reference Manual to LL API cross reference:	• CR3 WUS LL_LPUART_GetWKUPType

LL_LPUART_SetBaudRate

Function name	**__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t BaudRate)**
Function description	Configure LPUART BRR register for achieving expected Baud Rate value.
Parameters	• **LPUARTx:** LPUART Instance • **PeriphClk:** Peripheral Clock • **BaudRate:** Baud Rate
Return values	• **None**
Notes	• Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values • Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value != 0).

	• Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].
Reference Manual to LL API cross reference:	• BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk)**
Function description	Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.
Parameters	• **LPUARTx:** LPUART Instance • **PeriphClk:** Peripheral Clock
Return values	• **Baud:** Rate
Notes	• In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
Reference Manual to LL API cross reference:	• BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name	**__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)**
Function description	Enable Single Wire Half-Duplex mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name	**__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)**
Function description	Disable Single Wire Half-Duplex mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (USART_TypeDef * LPUARTx)**
Function description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name	**__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)**
Function description	Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	• **LPUARTx:** LPUART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (USART_TypeDef * LPUARTx)**
Function description	Return DEDT (Driver Enable De-Assertion Time)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** value expressed on 5 bits ([4:0] bits) : c
Reference Manual to LL API cross reference:	• CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name	**__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)**
Function description	Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	• **LPUARTx:** LPUART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None**
Reference Manual to	• CR1 DEAT LL_LPUART_SetDEAssertionTime

LL API cross reference:	

LL_LPUART_GetDEAssertionTime

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (USART_TypeDef * LPUARTx)**
Function description	Return DEAT (Driver Enable Assertion Time)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31
Reference Manual to LL API cross reference:	• CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name	**__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)**
Function description	Enable Driver Enable (DE) Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name	**__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)**
Function description	Disable Driver Enable (DE) Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (USART_TypeDef * LPUARTx)**
Function description	Indicate if Driver Enable (DE) Mode is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR3 DEM LL_LPUART_IsEnabledDEMode

reference:

LL_LPUART_SetDESignalPolarity

Function name	**__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)**
Function description	Select Driver Enable Polarity.
Parameters	• **LPUARTx:** LPUART Instance • **Polarity:** This parameter can be one of the following values: – LL_LPUART_DE_POLARITY_HIGH – LL_LPUART_DE_POLARITY_LOW
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (USART_TypeDef * LPUARTx)**
Function description	Return Driver Enable Polarity.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_DE_POLARITY_HIGH – LL_LPUART_DE_POLARITY_LOW
Reference Manual to LL API cross reference:	• CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Parity Error Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Framing Error Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Noise detected Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART OverRun Error Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART IDLE line detected Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Read Data Register Not Empty Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RXNE LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmission Complete Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmit Data Register Empty Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TXE LL_LPUART_IsActiveFlag_TXE

LL_LPUART_IsActiveFlag_nCTS

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART CTS interrupt Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART CTS Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Busy Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Character Match Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Send Break Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Receive Wake Up from mute mode Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_WKUP

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Wake Up from stop mode Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR WUF LL_LPUART_IsActiveFlag_WKUP

LL_LPUART_IsActiveFlag_TEACK

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmit Enable Acknowledge Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Receive Enable Acknowledge Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_ClearFlag_PE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)**
Function description	Clear Parity Error Flag.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)**
Function description	Clear Framing Error Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)**
Function description	Clear Noise detected Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR NCF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)**
Function description	Clear OverRun Error Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)**
Function description	Clear IDLE line detected Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**

Reference Manual to LL API cross reference:	• ICR IDLECF LL_LPUART_ClearFlag_IDLE

LL_LPUART_ClearFlag_TC

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)**
Function description	Clear Transmission Complete Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)**
Function description	Clear CTS Interrupt Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)**
Function description	Clear Character Match Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_ClearFlag_WKUP

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)**
Function description	Clear Wake Up from stop mode Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross	• ICR WUCF LL_LPUART_ClearFlag_WKUP

reference:

LL_LPUART_EnableIT_IDLE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)**
Function description	Enable IDLE Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_RXNE (USART_TypeDef * LPUARTx)**
Function description	Enable RX Not Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TC

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)**
Function description	Enable Transmission Complete Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_TXE (USART_TypeDef * LPUARTx)**
Function description	Enable TX Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_LPUART_EnableIT_TXE

LL_LPUART_EnableIT_PE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)**
Function description	Enable Parity Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)**
Function description	Enable Character Match Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_ERROR

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)**
Function description	Enable Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)**
Function description	Enable CTS Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**

Reference Manual to LL API cross reference:	• CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_WKUP

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_WKUP (USART_TypeDef * LPUARTx)**
Function description	Enable Wake Up from Stop Mode Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_LPUART_EnableIT_WKUP

LL_LPUART_DisableIT_IDLE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)**
Function description	Disable IDLE Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_RXNE (USART_TypeDef * LPUARTx)**
Function description	Disable RX Not Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TC

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)**
Function description	Disable Transmission Complete Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross	• CR1 TCIE LL_LPUART_DisableIT_TC

reference:

LL_LPUART_DisableIT_TXE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_TXE (USART_TypeDef * LPUARTx)**
Function description	Disable TX Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_LPUART_DisableIT_TXE

LL_LPUART_DisableIT_PE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)**
Function description	Disable Parity Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)**
Function description	Disable Character Match Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_ERROR

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)**
Function description	Disable Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1

	or ORE=1 or NF=1 in the LPUARTx_ISR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)**
Function description	Disable CTS Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_WKUP

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_WKUP (USART_TypeDef * LPUARTx)**
Function description	Disable Wake Up from Stop Mode Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_LPUART_DisableIT_WKUP

LL_LPUART_IsEnabledIT_IDLE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART IDLE Interrupt source is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

LL_LPUART_IsEnabledIT_RXNE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART RX Not Empty Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmission Complete Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART TX Empty Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_LPUART_IsEnabledIT_TXE

LL_LPUART_IsEnabledIT_PE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Parity Error Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Character Match Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_ERROR

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Error Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART CTS Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_WKUP

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_WKUP (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_LPUART_IsEnabledIT_WKUP

LL_LPUART_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)**
Function description	Enable DMA Mode for reception.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)**
Function description	Disable DMA Mode for reception.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (USART_TypeDef * LPUARTx)**
Function description	Check if DMA Mode is enabled for reception.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)**
Function description	Enable DMA Mode for transmission.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)**
Function description	Disable DMA Mode for transmission.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (USART_TypeDef * LPUARTx)**
Function description	Check if DMA Mode is enabled for transmission.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)**
Function description	Enable DMA Disabling on Reception Error.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)**
Function description	Disable DMA Disabling on Reception Error.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (USART_TypeDef * LPUARTx)**
Function description	Indicate if DMA Disabling on Reception Error is disabled.

Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (USART_TypeDef * LPUARTx, uint32_t Direction)**
Function description	Get the LPUART data register address used for DMA transfer.
Parameters	• **LPUARTx:** LPUART Instance • **Direction:** This parameter can be one of the following values: – LL_LPUART_DMA_REG_DATA_TRANSMIT – LL_LPUART_DMA_REG_DATA_RECEIVE
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• RDR RDR LL_LPUART_DMA_GetRegAddr • • TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (USART_TypeDef * LPUARTx)**
Function description	Read Receiver Data register (Receive Data value, 8 bits)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** Value between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name	**__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (USART_TypeDef * LPUARTx)**
Function description	Read Receiver Data register (Receive Data value, 9 bits)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF
Reference Manual to LL API cross reference:	• RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name	**__STATIC_INLINE void LL_LPUART_TransmitData8**

	(USART_TypeDef * LPUARTx, uint8_t Value)
Function description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	• **LPUARTx:** LPUART Instance • **Value:** between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name	**__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	• **LPUARTx:** LPUART Instance • **Value:** between Min_Data=0x00 and Max_Data=0x1FF
Return values	• **None**
Reference Manual to LL API cross reference:	• TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name	**__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)**
Function description	Request Break sending.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name	**__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)**
Function description	Put LPUART in mute mode and set the RWU flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name	**__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)**
Function description	Request a Receive Data flush.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_DeInit

Function name	**ErrorStatus LL_LPUART_DeInit (USART_TypeDef * LPUARTx)**
Function description	De-initialize LPUART registers (Registers restored to their default values).
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPUART registers are de-initialized – ERROR: not applicable

LL_LPUART_Init

Function name	**ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, LL_LPUART_InitTypeDef * LPUART_InitStruct)**
Function description	Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.
Parameters	• **LPUARTx:** LPUART Instance • **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content – ERROR: Problem occurred during LPUART Registers initialization
Notes	• As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned. • Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name	**void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)**
Function description	Set each LL_LPUART_InitTypeDef field to default value.

Parameters	•	**LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.
Return values	•	**None**

67.3 LPUART Firmware driver defines

67.3.1 LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B	4-bit address detection method selected
LL_LPUART_ADDRESS_DETECT_7B	7-bit address detection (in 8-bit data mode) method selected

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE	Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)
LL_LPUART_BINARY_LOGIC_NEGATIVE	Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST	data is transmitted/received with data bit 0 first, following the start bit
LL_LPUART_BITORDER_MSBFIRST	data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF	Parity error flag
LL_LPUART_ICR_FECF	Framing error flag
LL_LPUART_ICR_NCF	Noise detected flag
LL_LPUART_ICR_ORECF	Overrun error flag
LL_LPUART_ICR_IDLECF	Idle line detected flag
LL_LPUART_ICR_TCCF	Transmission complete flag
LL_LPUART_ICR_CTSCF	CTS flag
LL_LPUART_ICR_CMCF	Character match flag
LL_LPUART_ICR_WUCF	Wakeup from Stop mode flag

Datawidth

LL_LPUART_DATAWIDTH_7B	7 bits word length : Start bit, 7 data bits, n stop bits
LL_LPUART_DATAWIDTH_8B	8 bits word length : Start bit, 8 data bits, n stop bits
LL_LPUART_DATAWIDTH_9B	9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH	DE signal is active high
LL_LPUART_DE_POLARITY_LOW	DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE	Transmitter and Receiver are disabled
LL_LPUART_DIRECTION_RX	Transmitter is disabled and Receiver is enabled
LL_LPUART_DIRECTION_TX	Transmitter is enabled and Receiver is disabled
LL_LPUART_DIRECTION_TX_RX	Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT	Get address of data register used for transmission
LL_LPUART_DMA_REG_DATA_RECEIVE	Get address of data register used for reception

Get Flags Defines

LL_LPUART_ISR_PE	Parity error flag
LL_LPUART_ISR_FE	Framing error flag
LL_LPUART_ISR_NE	Noise detected flag
LL_LPUART_ISR_ORE	Overrun error flag
LL_LPUART_ISR_IDLE	Idle line detected flag
LL_LPUART_ISR_RXNE	Read data register not empty flag
LL_LPUART_ISR_TC	Transmission complete flag
LL_LPUART_ISR_TXE	Transmit data register empty flag
LL_LPUART_ISR_CTSIF	CTS interrupt flag
LL_LPUART_ISR_CTS	CTS flag
LL_LPUART_ISR_BUSY	Busy flag
LL_LPUART_ISR_CMF	Character match flag
LL_LPUART_ISR_SBKF	Send break flag
LL_LPUART_ISR_RWU	Receiver wakeup from Mute mode flag
LL_LPUART_ISR_WUF	Wakeup from Stop mode flag
LL_LPUART_ISR_TEACK	Transmit enable acknowledge flag
LL_LPUART_ISR_REACK	Receive enable acknowledge flag

Hardware Control

LL_LPUART_HWCONTROL_NONE	CTS and RTS hardware flow control disabled
LL_LPUART_HWCONTROL_RTS	RTS output enabled, data is only requested when there is space in the receive buffer
LL_LPUART_HWCONTROL_CTS	CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)
LL_LPUART_HWCONTROL_RTS_CTS	CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE	IDLE interrupt enable
LL_LPUART_CR1_RXNEIE	Read data register not empty interrupt enable

LL_LPUART_CR1_TCIE	Transmission complete interrupt enable
LL_LPUART_CR1_TXEIE	Transmit data register empty interrupt enable
LL_LPUART_CR1_PEIE	Parity error
LL_LPUART_CR1_CMIE	Character match interrupt enable
LL_LPUART_CR3_EIE	Error interrupt enable
LL_LPUART_CR3_CTSIE	CTS interrupt enable
LL_LPUART_CR3_WUFIE	Wakeup from Stop mode interrupt enable

Parity Control

LL_LPUART_PARITY_NONE	Parity control disabled
LL_LPUART_PARITY_EVEN	Parity control enabled and Even Parity is selected
LL_LPUART_PARITY_ODD	Parity control enabled and Odd Parity is selected

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD	RX pin signal works using the standard logic levels
LL_LPUART_RXPIN_LEVEL_INVERTED	RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1	1 stop bit
LL_LPUART_STOPBITS_2	2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD	TX pin signal works using the standard logic levels
LL_LPUART_TXPIN_LEVEL_INVERTED	TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD	TX/RX pins are used as defined in standard pinout
LL_LPUART_TXRX_SWAPPED	TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE	LPUART wake up from Mute mode on Idle Line
LL_LPUART_WAKEUP_ADDRESSMARK	LPUART wake up from Mute mode on Address Mark

Wakeup Activation

LL_LPUART_WAKEUP_ON_ADDRESS	Wake up active on address match
LL_LPUART_WAKEUP_ON_STARTBIT	Wake up active on Start bit detection
LL_LPUART_WAKEUP_ON_RXNE	Wake up active on RXNE

Helper Macros

__LL_LPUART_DIV **Description:**

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

68 LL PWR Generic Driver

68.1 PWR Firmware driver API description

68.1.1 Detailed description of functions

LL_PWR_EnableLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_EnableLowPowerRunMode (void)**
Function description	Switch the regulator from main mode to low-power mode.
Return values	• **None**
Notes	• Remind to set the regulator to low power before enabling LowPower run mode (bit LL_PWR_REGU_LPMODES_LOW_POWER).
Reference Manual to LL API cross reference:	• CR LPRUN LL_PWR_EnableLowPowerRunMode

LL_PWR_DisableLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_DisableLowPowerRunMode (void)**
Function description	Switch the regulator from low-power mode to main mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR LPRUN LL_PWR_DisableLowPowerRunMode

LL_PWR_IsEnabledLowPowerRunMode

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledLowPowerRunMode (void)**
Function description	Check if the regulator is in low-power mode.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR LPRUN LL_PWR_IsEnabledLowPowerRunMode

LL_PWR_EnterLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_EnterLowPowerRunMode (void)**
Function descrip	Set voltage regulator to low-power and switch from run main mode to run low-power mode.

tion	
Return values	• **None**
Notes	• This "high level" function is introduced to provide functional compatibility with other families. Notice that the two registers have to be written sequentially, so this function is not atomic. To assure atomicity you can call separately the following functions: LL_PWR_SetRegulModeLP(LL_PWR_REGU_LPMODES_LOW_POWER); LL_PWR_EnableLowPowerRunMode();
Reference Manual to LL API cross reference:	• CR LPSDSR LL_PWR_EnterLowPowerRunMode • CR LPRUN LL_PWR_EnterLowPowerRunMode

LL_PWR_ExitLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_ExitLowPowerRunMode (void)**
Function description	Set voltage regulator to main and switch from run main mode to low-power mode.
Return values	• **None**
Notes	• This "high level" function is introduced to provide functional compatibility with other families. Notice that the two registers have to be written sequentially, so this function is not atomic. To assure atomicity you can call separately the following functions: LL_PWR_DisableLowPowerRunMode();LL_PWR_SetRegulModeLP(LL_PWR_REGU_LPMODES_MAIN);
Reference Manual to LL API cross reference:	• CR LPSDSR LL_PWR_ExitLowPowerRunMode • CR LPRUN LL_PWR_ExitLowPowerRunMode

LL_PWR_SetRegulVoltageScaling

Function name	**__STATIC_INLINE void LL_PWR_SetRegulVoltageScaling (uint32_t VoltageScaling)**
Function description	Set the main internal regulator output voltage.
Parameters	• **VoltageScaling:** This parameter can be one of the following values:

- – LL_PWR_REGU_VOLTAGE_SCALE1
- – LL_PWR_REGU_VOLTAGE_SCALE2
- – LL_PWR_REGU_VOLTAGE_SCALE3

Return values	• **None**
Reference Manual to LL API cross reference:	• CR VOS LL_PWR_SetRegulVoltageScaling

LL_PWR_GetRegulVoltageScaling

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetRegulVoltageScaling (void)**
Function description	Get the main internal regulator output voltage.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_REGU_VOLTAGE_SCALE1 – LL_PWR_REGU_VOLTAGE_SCALE2 – LL_PWR_REGU_VOLTAGE_SCALE3
Reference Manual to LL API cross reference:	• CR VOS LL_PWR_GetRegulVoltageScaling

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:	• CR DBP LL_PWR_EnableBkUpAccess

LL_PWR_DisableBkUpAccess

Function name	**__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)**
Function description	Disable access to the backup domain.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR DBP LL_PWR_DisableBkUpAccess

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	• CR DBP LL_PWR_IsEnabledBkUpAccess

reference:

LL_PWR_SetRegulModeLP

Function name	**__STATIC_INLINE void LL_PWR_SetRegulModeLP (uint32_t RegulMode)**
Function description	Set voltage regulator mode during low power modes.
Parameters	• **RegulMode:** This parameter can be one of the following values: – LL_PWR_REGU_LPMODES_MAIN – LL_PWR_REGU_LPMODES_LOW_POWER
Return values	• **None**
Reference Manual to LL API cross reference:	• CR LPSDSR LL_PWR_SetRegulModeLP

LL_PWR_GetRegulModeLP

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetRegulModeLP (void)**
Function description	Get voltage regulator mode during low power modes.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_REGU_LPMODES_MAIN – LL_PWR_REGU_LPMODES_LOW_POWER
Reference Manual to LL API cross reference:	• CR LPSDSR LL_PWR_GetRegulModeLP

LL_PWR_SetPowerMode

Function name	**__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t PDMode)**
Function description	Set power down mode when CPU enters deepsleep.
Parameters	• **PDMode:** This parameter can be one of the following values: – LL_PWR_MODE_STOP – LL_PWR_MODE_STANDBY
Return values	• **None**
Notes	• Set the regulator to low power (bit LL_PWR_REGU_LPMODES_LOW_POWER) before setting MODE_STOP. If the regulator remains in "main mode", it consumes more power without providing any additional feature. In MODE_STANDBY the regulator is automatically off.
Reference Manual to LL API cross reference:	• CR PDDS LL_PWR_SetPowerMode

LL_PWR_GetPowerMode

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void)**
Function description	Get power down mode when CPU enters deepsleep.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_MODE_STOP – LL_PWR_MODE_STANDBY
Reference Manual to LL API cross reference:	• CR PDDS LL_PWR_GetPowerMode

LL_PWR_SetPVDLevel

Function name	**__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)**
Function description	Configure the voltage threshold detected by the Power Voltage Detector.
Parameters	• **PVDLevel:** This parameter can be one of the following values: – LL_PWR_PVDLEVEL_0 – LL_PWR_PVDLEVEL_1 – LL_PWR_PVDLEVEL_2 – LL_PWR_PVDLEVEL_3 – LL_PWR_PVDLEVEL_4 – LL_PWR_PVDLEVEL_5 – LL_PWR_PVDLEVEL_6 – LL_PWR_PVDLEVEL_7
Return values	• **None**
Reference Manual to LL API cross reference:	• CR PLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)**
Function description	Get the voltage threshold detection.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_PVDLEVEL_0 – LL_PWR_PVDLEVEL_1 – LL_PWR_PVDLEVEL_2 – LL_PWR_PVDLEVEL_3 – LL_PWR_PVDLEVEL_4 – LL_PWR_PVDLEVEL_5 – LL_PWR_PVDLEVEL_6 – LL_PWR_PVDLEVEL_7
Reference Manual to LL API cross reference:	• CR PLS LL_PWR_GetPVDLevel

LL_PWR_EnablePVD

Function name	**__STATIC_INLINE void LL_PWR_EnablePVD (void)**
Function description	Enable Power Voltage Detector.
Return values	• **None**
Reference Manual to LL API cross reference:	• 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: (*) not available on all devices – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 (*)
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

LL_PWR_DisableWakeUpPin

Function name	**__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t**

WakeUpPin)

Function description	Disable the WakeUp PINx functionality.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: (*) not available on all devices – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR EWUP1 LL_PWR_DisableWakeUpPin • • CSR EWUP2 LL_PWR_DisableWakeUpPin • • CSR EWUP3 LL_PWR_DisableWakeUpPin

LL_PWR_IsEnabledWakeUpPin

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)**
Function description	Check if the WakeUp PINx functionality is enabled.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: (*) not available on all devices – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 (*)
Return values	• **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

LL_PWR_EnableUltraLowPower

Function name	**__STATIC_INLINE void LL_PWR_EnableUltraLowPower (void)**
Function description	Enable ultra low-power mode by enabling VREFINT switch off in low-power modes.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR ULP LL_PWR_EnableUltraLowPower

LL_PWR_DisableUltraLowPower

Function name	**__STATIC_INLINE void LL_PWR_DisableUltraLowPower (void)**
Function description	Disable ultra low-power mode by disabling VREFINT switch off in

	low-power modes.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR ULP LL_PWR_DisableUltraLowPower

LL_PWR_IsEnabledUltraLowPower

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledUltraLowPower (void)**
Function description	Check if ultra low-power mode is enabled by checking if VREFINT switch off in low-power modes is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ULP LL_PWR_IsEnabledUltraLowPower

LL_PWR_EnableFastWakeUp

Function name	**__STATIC_INLINE void LL_PWR_EnableFastWakeUp (void)**
Function description	Enable fast wakeup by ignoring VREFINT startup time when exiting from low-power mode.
Return values	• **None**
Notes	• Works in conjunction with ultra low power mode.
Reference Manual to LL API cross reference:	• CR FWU LL_PWR_EnableFastWakeUp

LL_PWR_DisableFastWakeUp

Function name	**__STATIC_INLINE void LL_PWR_DisableFastWakeUp (void)**
Function description	Disable fast wakeup by waiting VREFINT startup time when exiting from low-power mode.
Return values	• **None**
Notes	• Works in conjunction with ultra low power mode.
Reference Manual to LL API cross reference:	• CR FWU LL_PWR_DisableFastWakeUp

LL_PWR_IsEnabledFastWakeUp

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledFastWakeUp (void)**
Function description	Check if fast wakeup is enabled by checking if VREFINT startup time when exiting from low-power mode is ignored.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CR FWU LL_PWR_IsEnabledFastWakeUp

LL_PWR_EnableNVMKeptOff

Function name	**__STATIC_INLINE void LL_PWR_EnableNVMKeptOff (void)**
Function description	Enable non-volatile memory (Flash and EEPROM) keeping off feature when exiting from low-power mode.
Return values	• **None**
Notes	• When enabled, after entering low-power mode (Stop or Standby only), if RUN_PD of FLASH_ACR register is also set, the Flash memory will not be woken up when exiting from deepsleep mode. When enabled, the EEPROM will not be woken up when exiting from low-power mode (if the bit RUN_PD is set)
Reference Manual to LL API cross reference:	• CR DS_EE_KOFF LL_PWR_EnableNVMKeptOff

LL_PWR_DisableNVMKeptOff

Function name	**__STATIC_INLINE void LL_PWR_DisableNVMKeptOff (void)**
Function description	Disable non-volatile memory (Flash and EEPROM) keeping off feature when exiting from low-power mode.
Return values	• **None**
Notes	• When disabled, Flash memory is woken up when exiting from deepsleep mode even if the bit RUN_PD is set
Reference Manual to LL API cross reference:	• CR DS_EE_KOFF LL_PWR_DisableNVMKeptOff

LL_PWR_IsEnabledNVMKeptOff

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledNVMKeptOff (void)**
Function description	Check if non-volatile memory (Flash and EEPROM) keeping off feature when exiting from low-power mode is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR DS_EE_KOFF LL_PWR_IsEnabledNVMKeptOff

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 VrefInt Flag.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR VREFINTRDYF LL_PWR_IsActiveFlag_VREFINTRDY

LL_PWR_IsActiveFlag_VOSF

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VOSF (void)**
Function description	Indicate whether the regulator is ready in the selected voltage range or if its output voltage is still changing to the required voltage level.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR VOSF LL_PWR_IsActiveFlag_VOSF

LL_PWR_IsActiveFlag_REGLPF

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGLPF (void)**
Function description	Indicate whether the regulator is ready in main mode or is in low-power mode.
Return values	• **State:** of bit (1 or 0).
Notes	• Take care, return value "0" means the regulator is ready. Return value "1" means the output voltage range is still changing.
Reference Manual to LL API cross reference:	• CSR REGLPF LL_PWR_IsActiveFlag_REGLPF

LL_PWR_ClearFlag_SB

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_SB (void)**
Function description	Clear Standby Flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• 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	• **None**
Reference Manual to LL API cross reference:	• 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	• **An:** ErrorStatus enumeration value: – SUCCESS: PWR registers are de-initialized – ERROR: not applicable

68.2 PWR Firmware driver defines

68.2.1 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_VREFINTRDYF	VREFINT ready flag
LL_PWR_CSR_VOS	Voltage scaling select flag
LL_PWR_CSR_REGLPF	Regulator low power 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

Mode Power

LL_PWR_MODE_STOP	Enter Stop mode when the CPU enters deepsleep
LL_PWR_MODE_STANDBY	Enter Standby mode when the CPU enters deepsleep

Power Voltage Detector Level

LL_PWR_PVDLEVEL_0	Voltage threshold detected by PVD 1.9 V
LL_PWR_PVDLEVEL_1	Voltage threshold detected by PVD 2.1 V
LL_PWR_PVDLEVEL_2	Voltage threshold detected by PVD 2.3 V
LL_PWR_PVDLEVEL_3	Voltage threshold detected by PVD 2.5 V
LL_PWR_PVDLEVEL_4	Voltage threshold detected by PVD 2.7 V
LL_PWR_PVDLEVEL_5	Voltage threshold detected by PVD 2.9 V
LL_PWR_PVDLEVEL_6	Voltage threshold detected by PVD 3.1 V
LL_PWR_PVDLEVEL_7	External input analog voltage (Compare internally to VREFINT)

Regulator Mode In Low Power Modes

LL_PWR_REGU_LPMODES_MAIN	Voltage regulator in main mode during deepsleep/sleep/low-power run mode
LL_PWR_REGU_LPMODES_LOW_POWER	Voltage regulator in low-power mode during deepsleep/sleep/low-power run mode

Regulator Voltage

LL_PWR_REGU_VOLTAGE_SCALE1	1.8V (range 1)
LL_PWR_REGU_VOLTAGE_SCALE2	1.5V (range 2)
LL_PWR_REGU_VOLTAGE_SCALE3	1.2V (range 3)

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

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

69 LL RCC Generic Driver

69.1 RCC Firmware driver registers structures

69.1.1 LL_RCC_ClocksTypeDef

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
  SYSCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
  HCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
  PCLK1 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
  PCLK2 clock frequency

69.2 RCC Firmware driver API description

69.2.1 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 CSSHSEON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_EnableBypass

Function name	**__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)**
Function description	Enable HSE external oscillator (HSE Bypass)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name	**__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)**

Function description	Disable HSE external oscillator (HSE Bypass)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name	**__STATIC_INLINE void LL_RCC_HSE_Enable (void)**
Function description	Enable HSE crystal oscillator (HSE ON)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name	**__STATIC_INLINE void LL_RCC_HSE_Disable (void)**
Function description	Disable HSE crystal oscillator (HSE ON)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)**
Function description	Check if HSE oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_SetRTC_HSEPrescaler

Function name	**__STATIC_INLINE void LL_RCC_SetRTC_HSEPrescaler (uint32_t Div)**
Function description	Configure the RTC prescaler (divider)
Parameters	• **Div:** This parameter can be one of the following values: – LL_RCC_RTC_HSE_DIV_2 – LL_RCC_RTC_HSE_DIV_4 – LL_RCC_RTC_HSE_DIV_8 – LL_RCC_RTC_HSE_DIV_16
Return values	• **None**
Reference Manual to LL API cross	• CR RTCPRE LL_RCC_SetRTC_HSEPrescaler

reference:

LL_RCC_GetRTC_HSEPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetRTC_HSEPrescaler (void)**
Function description	Get the RTC divider (prescaler)
Return values	• **Returned:** value can be one of the following values: – LL_RCC_RTC_HSE_DIV_2 – LL_RCC_RTC_HSE_DIV_4 – LL_RCC_RTC_HSE_DIV_8 – LL_RCC_RTC_HSE_DIV_16
Reference Manual to LL API cross reference:	• CR RTCPRE LL_RCC_GetRTC_HSEPrescaler

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_EnableInStopMode

Function name	**__STATIC_INLINE void LL_RCC_HSI_EnableInStopMode (void)**
Function description	Enable HSI even in stop mode.
Return values	• **None**
Notes	• HSI oscillator is forced ON even in Stop mode
Reference Manual to LL API cross reference:	• CR HSIKERON LL_RCC_HSI_EnableInStopMode

LL_RCC_HSI_DisableInStopMode

Function name	**__STATIC_INLINE void LL_RCC_HSI_DisableInStopMode (void)**
Function description	Disable HSI in stop mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSIKERON LL_RCC_HSI_DisableInStopMode

LL_RCC_HSI_EnableDivider

Function name	**__STATIC_INLINE void LL_RCC_HSI_EnableDivider (void)**
Function description	Enable HSI Divider (it divides by 4)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSIDIVEN LL_RCC_HSI_EnableDivider

LL_RCC_HSI_DisableDivider

Function name	**__STATIC_INLINE void LL_RCC_HSI_DisableDivider (void)**
Function description	Disable HSI Divider (it divides by 4)
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSIDIVEN LL_RCC_HSI_DisableDivider

LL_RCC_HSI_EnableOutput

Function name	**__STATIC_INLINE void LL_RCC_HSI_EnableOutput (void)**
Function description	Enable HSI Output.
Return values	• **None**

Reference Manual to LL API cross reference:	• CR HSIOUTEN LL_RCC_HSI_EnableOutput

LL_RCC_HSI_DisableOutput

Function name	**__STATIC_INLINE void LL_RCC_HSI_DisableOutput (void)**
Function description	Disable HSI Output.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR HSIOUTEN LL_RCC_HSI_DisableOutput

LL_RCC_HSI_GetCalibration

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)**
Function description	Get HSI Calibration value.
Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0xFF
Notes	• When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value
Reference Manual to LL API cross reference:	• ICSCR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name	**__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)**
Function description	Set HSI Calibration trimming.
Parameters	• **Value:** between Min_Data = 0x00 and Max_Data = 0x1F
Return values	• **None**
Notes	• user-programmable trimming value that is added to the HSICAL • Default 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:	• ICSCR HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)**
Function description	Get HSI Calibration trimming.
Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0x1F

Reference Manual to LL API cross reference:	• ICSCR HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_HSI48_Enable

Function name	**__STATIC_INLINE void LL_RCC_HSI48_Enable (void)**
Function description	Enable HSI48.
Return values	• **None**
Reference Manual to LL API cross reference:	• CRRCR HSI48ON LL_RCC_HSI48_Enable

LL_RCC_HSI48_Disable

Function name	**__STATIC_INLINE void LL_RCC_HSI48_Disable (void)**
Function description	Disable HSI48.
Return values	• **None**
Reference Manual to LL API cross reference:	• CRRCR HSI48ON LL_RCC_HSI48_Disable

LL_RCC_HSI48_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void)**
Function description	Check if HSI48 oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRRCR 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:	• CRRCR HSI48CAL LL_RCC_HSI48_GetCalibration

LL_RCC_HSI48_EnableDivider

Function name	**__STATIC_INLINE void LL_RCC_HSI48_EnableDivider (void)**
Function description	Enable HSI48 Divider (it divides by 6)
Return values	• **None**

Reference Manual to LL API cross reference:	• CRRCR HSI48DIV6OUTEN LL_RCC_HSI48_EnableDivider

LL_RCC_HSI48_DisableDivider

Function name	**__STATIC_INLINE void LL_RCC_HSI48_DisableDivider (void)**
Function description	Disable HSI48 Divider (it divides by 6)
Return values	• **None**
Reference Manual to LL API cross reference:	• CRRCR HSI48DIV6OUTEN LL_RCC_HSI48_DisableDivider

LL_RCC_HSI48_IsDivided

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI48_IsDivided (void)**
Function description	Check if HSI48 Divider is enabled (it divides by 6)
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRRCR HSI48DIV6OUTEN LL_RCC_HSI48_IsDivided

LL_RCC_LSE_Enable

Function name	**__STATIC_INLINE void LL_RCC_LSE_Enable (void)**
Function description	Enable Low Speed External (LSE) crystal.
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name	**__STATIC_INLINE void LL_RCC_LSE_Disable (void)**
Function description	Disable Low Speed External (LSE) crystal.
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_EnableBypass

Function name	**__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)**
Function description	Enable external clock source (LSE bypass).
Return values	• **None**
Reference Manual to	• CSR LSEBYP LL_RCC_LSE_EnableBypass

LL API cross reference:

LL_RCC_LSE_DisableBypass

Function name	**__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)**
Function description	Disable external clock source (LSE bypass).
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SetDriveCapability

Function name	**__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)**
Function description	Set LSE oscillator drive capability.
Parameters	• **LSEDrive:** This parameter can be one of the following values: – LL_RCC_LSEDRIVE_LOW – LL_RCC_LSEDRIVE_MEDIUMLOW – LL_RCC_LSEDRIVE_MEDIUMHIGH – LL_RCC_LSEDRIVE_HIGH
Return values	• **None**
Notes	• The oscillator is in Xtal mode when it is not in bypass mode.
Reference Manual to LL API cross reference:	• CSR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void)**
Function description	Get LSE oscillator drive capability.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_LSEDRIVE_LOW – LL_RCC_LSEDRIVE_MEDIUMLOW – LL_RCC_LSEDRIVE_MEDIUMHIGH – LL_RCC_LSEDRIVE_HIGH
Reference Manual to LL API cross reference:	• CSR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_EnableCSS

Function name	**__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void)**
Function description	Enable Clock security system on LSE.

Return values	• **None**
Reference Manual to LL API cross reference:	• CSR LSECSSON LL_RCC_LSE_EnableCSS

LL_RCC_LSE_DisableCSS

Function name	**__STATIC_INLINE void LL_RCC_LSE_DisableCSS (void)**
Function description	Disable Clock security system on LSE.
Return values	• **None**
Notes	• Clock security system can be disabled only after a LSE failure detection. In that case it MUST be disabled by software.
Reference Manual to LL API cross reference:	• CSR LSECSSON LL_RCC_LSE_DisableCSS

LL_RCC_LSE_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)**
Function description	Check if LSE oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSE_IsCSSDetected

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSE_IsCSSDetected (void)**
Function description	Check if CSS on LSE failure Detection.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LSECSSD LL_RCC_LSE_IsCSSDetected

LL_RCC_LSI_Enable

Function name	**__STATIC_INLINE void LL_RCC_LSI_Enable (void)**
Function description	Enable LSI Oscillator.
Return values	• **None**
Reference Manual to LL API cross reference:	• 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_MSI_Enable

Function name	**__STATIC_INLINE void LL_RCC_MSI_Enable (void)**
Function description	Enable MSI oscillator.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR MSION LL_RCC_MSI_Enable

LL_RCC_MSI_Disable

Function name	**__STATIC_INLINE void LL_RCC_MSI_Disable (void)**
Function description	Disable MSI oscillator.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR MSION LL_RCC_MSI_Disable

LL_RCC_MSI_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_IsReady (void)**
Function description	Check if MSI oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR MSIRDY LL_RCC_MSI_IsReady

LL_RCC_MSI_SetRange

Function name	**__STATIC_INLINE void LL_RCC_MSI_SetRange (uint32_t Range)**
Function description	Configure the Internal Multi Speed oscillator (MSI) clock range in run mode.
Parameters	• **Range:** This parameter can be one of the following values: – LL_RCC_MSIRANGE_0 – LL_RCC_MSIRANGE_1 – LL_RCC_MSIRANGE_2 – LL_RCC_MSIRANGE_3 – LL_RCC_MSIRANGE_4 – LL_RCC_MSIRANGE_5 – LL_RCC_MSIRANGE_6
Return values	• **None**
Reference Manual to LL API cross reference:	• ICSCR MSIRANGE LL_RCC_MSI_SetRange

LL_RCC_MSI_GetRange

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_GetRange (void)**
Function description	Get the Internal Multi Speed oscillator (MSI) clock range in run mode.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_MSIRANGE_0 – LL_RCC_MSIRANGE_1 – LL_RCC_MSIRANGE_2 – LL_RCC_MSIRANGE_3 – LL_RCC_MSIRANGE_4 – LL_RCC_MSIRANGE_5 – LL_RCC_MSIRANGE_6
Reference Manual to LL API cross reference:	• ICSCR MSIRANGE LL_RCC_MSI_GetRange

LL_RCC_MSI_GetCalibration

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibration (void)**
Function description	Get MSI Calibration value.
Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0xFF
Notes	• When MSITRIM is written, MSICAL is updated with the sum of MSITRIM and the factory trim value
Reference Manual to LL API cross reference:	• ICSCR MSICAL LL_RCC_MSI_GetCalibration

LL_RCC_MSI_SetCalibTrimming

Function name	**__STATIC_INLINE void LL_RCC_MSI_SetCalibTrimming (uint32_t Value)**
Function description	Set MSI 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 MSICAL
Reference Manual to LL API cross reference:	• ICSCR MSITRIM LL_RCC_MSI_SetCalibTrimming

LL_RCC_MSI_GetCalibTrimming

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibTrimming (void)**
Function description	Get MSI Calibration trimming.
Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0xFF
Reference Manual to LL API cross reference:	• ICSCR MSITRIM LL_RCC_MSI_GetCalibTrimming

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_MSI – LL_RCC_SYS_CLKSOURCE_HSI – LL_RCC_SYS_CLKSOURCE_HSE – LL_RCC_SYS_CLKSOURCE_PLL
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	• **Returned:** value can be one of the following values: – LL_RCC_SYS_CLKSOURCE_STATUS_MSI – LL_RCC_SYS_CLKSOURCE_STATUS_HSI – LL_RCC_SYS_CLKSOURCE_STATUS_HSE – LL_RCC_SYS_CLKSOURCE_STATUS_PLL

Reference Manual to LL API cross reference:	• CFGR SWS LL_RCC_GetSysClkSource

LL_RCC_SetAHBPrescaler

Function name	**__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)**
Function description	Set AHB prescaler.
Parameters	• **Prescaler:** This parameter can be one of the following values: – LL_RCC_SYSCLK_DIV_1 – LL_RCC_SYSCLK_DIV_2 – LL_RCC_SYSCLK_DIV_4 – LL_RCC_SYSCLK_DIV_8 – LL_RCC_SYSCLK_DIV_16 – LL_RCC_SYSCLK_DIV_64 – LL_RCC_SYSCLK_DIV_128 – LL_RCC_SYSCLK_DIV_256 – LL_RCC_SYSCLK_DIV_512
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR HPRE LL_RCC_SetAHBPrescaler

LL_RCC_SetAPB1Prescaler

Function name	**__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)**
Function description	Set APB1 prescaler.
Parameters	• **Prescaler:** This parameter can be one of the following values: – LL_RCC_APB1_DIV_1 – LL_RCC_APB1_DIV_2 – LL_RCC_APB1_DIV_4 – LL_RCC_APB1_DIV_8 – LL_RCC_APB1_DIV_16
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR PPRE1 LL_RCC_SetAPB1Prescaler

LL_RCC_SetAPB2Prescaler

Function name	**__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)**
Function description	Set APB2 prescaler.
Parameters	• **Prescaler:** This parameter can be one of the following values:

- – LL_RCC_APB2_DIV_1
- – LL_RCC_APB2_DIV_2
- – LL_RCC_APB2_DIV_4
- – LL_RCC_APB2_DIV_8
- – LL_RCC_APB2_DIV_16

Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR PPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_GetAHBPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)**
Function description	Get AHB prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_SYSCLK_DIV_1 – LL_RCC_SYSCLK_DIV_2 – LL_RCC_SYSCLK_DIV_4 – LL_RCC_SYSCLK_DIV_8 – LL_RCC_SYSCLK_DIV_16 – LL_RCC_SYSCLK_DIV_64 – LL_RCC_SYSCLK_DIV_128 – LL_RCC_SYSCLK_DIV_256 – LL_RCC_SYSCLK_DIV_512
Reference Manual to LL API cross reference:	• CFGR HPRE LL_RCC_GetAHBPrescaler

LL_RCC_GetAPB1Prescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)**
Function description	Get APB1 prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_APB1_DIV_1 – LL_RCC_APB1_DIV_2 – LL_RCC_APB1_DIV_4 – LL_RCC_APB1_DIV_8 – LL_RCC_APB1_DIV_16
Reference Manual to LL API cross reference:	• CFGR PPRE1 LL_RCC_GetAPB1Prescaler

LL_RCC_GetAPB2Prescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)**
Function description	Get APB2 prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_APB2_DIV_1

	– LL_RCC_APB2_DIV_2 – LL_RCC_APB2_DIV_4 – LL_RCC_APB2_DIV_8 – LL_RCC_APB2_DIV_16
Reference Manual to LL API cross reference:	• CFGR PPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_SetClkAfterWakeFromStop

Function name	**__STATIC_INLINE void LL_RCC_SetClkAfterWakeFromStop (uint32_t Clock)**
Function description	Set Clock After Wake-Up From Stop mode.
Parameters	• **Clock:** This parameter can be one of the following values: – LL_RCC_STOP_WAKEUPCLOCK_MSI – LL_RCC_STOP_WAKEUPCLOCK_HSI
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR STOPWUCK LL_RCC_SetClkAfterWakeFromStop

LL_RCC_GetClkAfterWakeFromStop

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetClkAfterWakeFromStop (void)**
Function description	Get Clock After Wake-Up From Stop mode.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_STOP_WAKEUPCLOCK_MSI – LL_RCC_STOP_WAKEUPCLOCK_HSI
Reference Manual to LL API cross reference:	• CFGR STOPWUCK LL_RCC_GetClkAfterWakeFromStop

LL_RCC_ConfigMCO

Function name	**__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)**
Function description	Configure MCOx.
Parameters	• **MCOxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_MCO1SOURCE_NOCLOCK – LL_RCC_MCO1SOURCE_SYSCLK – LL_RCC_MCO1SOURCE_HSI – LL_RCC_MCO1SOURCE_MSI – LL_RCC_MCO1SOURCE_HSE – LL_RCC_MCO1SOURCE_PLLCLK – LL_RCC_MCO1SOURCE_LSI – LL_RCC_MCO1SOURCE_LSE

- – LL_RCC_MCO1SOURCE_HSI48 (*)
- **MCOxPrescaler:** This parameter can be one of the following values:
  - – LL_RCC_MCO1_DIV_1
  - – LL_RCC_MCO1_DIV_2
  - – LL_RCC_MCO1_DIV_4
  - – LL_RCC_MCO1_DIV_8
  - – LL_RCC_MCO1_DIV_16

Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR MCOSEL LL_RCC_ConfigMCO • CFGR MCOPRE LL_RCC_ConfigMCO

LL_RCC_SetUSARTClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t USARTxSource)**
Function description	Configure USARTx clock source.
Parameters	• **USARTxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_USART1_CLKSOURCE_PCLK2 (*) – LL_RCC_USART1_CLKSOURCE_SYSCLK (*) – LL_RCC_USART1_CLKSOURCE_HSI (*) – LL_RCC_USART1_CLKSOURCE_LSE (*) – LL_RCC_USART2_CLKSOURCE_PCLK1 – LL_RCC_USART2_CLKSOURCE_SYSCLK – LL_RCC_USART2_CLKSOURCE_HSI – LL_RCC_USART2_CLKSOURCE_LSE
Return values	• **None**
Reference Manual to LL API cross reference:	• CCIPR USARTxSEL LL_RCC_SetUSARTClockSource

LL_RCC_SetLPUARTClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t LPUARTxSource)**
Function description	Configure LPUART1x clock source.
Parameters	• **LPUARTxSource:** This parameter can be one of the following values: – LL_RCC_LPUART1_CLKSOURCE_PCLK1 – LL_RCC_LPUART1_CLKSOURCE_SYSCLK – LL_RCC_LPUART1_CLKSOURCE_HSI – LL_RCC_LPUART1_CLKSOURCE_LSE
Return values	• **None**
Reference Manual to LL API cross reference:	• CCIPR LPUART1SEL LL_RCC_SetLPUARTClockSource

LL_RCC_SetI2CClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t I2CxSource)**
Function description	Configure I2Cx clock source.
Parameters	• **I2CxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_I2C1_CLKSOURCE_PCLK1 – LL_RCC_I2C1_CLKSOURCE_SYSCLK – LL_RCC_I2C1_CLKSOURCE_HSI – LL_RCC_I2C3_CLKSOURCE_PCLK1 (*) – LL_RCC_I2C3_CLKSOURCE_SYSCLK (*) – LL_RCC_I2C3_CLKSOURCE_HSI (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• CCIPR I2CxSEL LL_RCC_SetI2CClockSource

LL_RCC_SetLPTIMClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t LPTIMxSource)**
Function description	Configure LPTIMx clock source.
Parameters	• **LPTIMxSource:** This parameter can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE_PCLK1 – LL_RCC_LPTIM1_CLKSOURCE_LSI – LL_RCC_LPTIM1_CLKSOURCE_HSI – LL_RCC_LPTIM1_CLKSOURCE_LSE
Return values	• **None**
Reference Manual to LL API cross reference:	• CCIPR LPTIMxSEL LL_RCC_SetLPTIMClockSource

LL_RCC_SetRNGClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetRNGClockSource (uint32_t RNGxSource)**
Function description	Configure RNG clock source.
Parameters	• **RNGxSource:** This parameter can be one of the following values: – LL_RCC_RNG_CLKSOURCE_PLL – LL_RCC_RNG_CLKSOURCE_HSI48
Return values	• **None**
Reference Manual to LL API cross reference:	• CCIPR HSI48SEL LL_RCC_SetRNGClockSource

LL_RCC_SetUSBClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t USBxSource)**
Function description	Configure USB clock source.
Parameters	• **USBxSource:** This parameter can be one of the following values: – LL_RCC_RNG_CLKSOURCE_PLL – LL_RCC_RNG_CLKSOURCE_HSI48
Return values	• **None**
Reference Manual to LL API cross reference:	• CCIPR HSI48SEL LL_RCC_SetUSBClockSource

LL_RCC_GetUSARTClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t USARTx)**
Function description	Get USARTx clock source.
Parameters	• **USARTx:** This parameter can be one of the following values: – LL_RCC_USART1_CLKSOURCE (*) – LL_RCC_USART2_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_USART1_CLKSOURCE_PCLK2 (*) – LL_RCC_USART1_CLKSOURCE_SYSCLK (*) – LL_RCC_USART1_CLKSOURCE_HSI (*) – LL_RCC_USART1_CLKSOURCE_LSE (*) – LL_RCC_USART2_CLKSOURCE_PCLK1 – LL_RCC_USART2_CLKSOURCE_SYSCLK – LL_RCC_USART2_CLKSOURCE_HSI – LL_RCC_USART2_CLKSOURCE_LSE
Reference Manual to LL API cross reference:	• CCIPR USARTxSEL LL_RCC_GetUSARTClockSource

LL_RCC_GetLPUARTClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t LPUARTx)**
Function description	Get LPUARTx clock source.
Parameters	• **LPUARTx:** This parameter can be one of the following values: – LL_RCC_LPUART1_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_LPUART1_CLKSOURCE_PCLK1 – LL_RCC_LPUART1_CLKSOURCE_SYSCLK – LL_RCC_LPUART1_CLKSOURCE_HSI

	– LL_RCC_LPUART1_CLKSOURCE_LSE
Reference Manual to LL API cross reference:	• CCIPR LPUART1SEL LL_RCC_GetLPUARTClockSource

LL_RCC_GetI2CClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t I2Cx)**
Function description	Get I2Cx clock source.
Parameters	• **I2Cx:** This parameter can be one of the following values: – LL_RCC_I2C1_CLKSOURCE – LL_RCC_I2C3_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_I2C1_CLKSOURCE_PCLK1 – LL_RCC_I2C1_CLKSOURCE_SYSCLK – LL_RCC_I2C1_CLKSOURCE_HSI – LL_RCC_I2C3_CLKSOURCE_PCLK1 (*) – LL_RCC_I2C3_CLKSOURCE_SYSCLK (*) – LL_RCC_I2C3_CLKSOURCE_HSI (*)
Reference Manual to LL API cross reference:	• CCIPR I2CxSEL LL_RCC_GetI2CClockSource

LL_RCC_GetLPTIMClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t LPTIMx)**
Function description	Get LPTIMx clock source.
Parameters	• **LPTIMx:** This parameter can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE_PCLK1 – LL_RCC_LPTIM1_CLKSOURCE_LSI – LL_RCC_LPTIM1_CLKSOURCE_HSI – LL_RCC_LPTIM1_CLKSOURCE_LSE
Reference Manual to LL API cross reference:	• CCIPR LPTIMxSEL LL_RCC_GetLPTIMClockSource

LL_RCC_GetRNGClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetRNGClockSource (uint32_t RNGx)**
Function description	Get RNGx clock source.
Parameters	• **RNGx:** This parameter can be one of the following values:

	– LL_RCC_RNG_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_RNG_CLKSOURCE_PLL – LL_RCC_RNG_CLKSOURCE_HSI48
Reference Manual to LL API cross reference:	• CCIPR CLK48SEL LL_RCC_GetRNGClockSource

LL_RCC_GetUSBClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t USBx)**
Function description	Get USBx clock source.
Parameters	• **USBx:** This parameter can be one of the following values: – LL_RCC_USB_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_USB_CLKSOURCE_PLL – LL_RCC_USB_CLKSOURCE_HSI48
Reference Manual to LL API cross reference:	• CCIPR CLK48SEL LL_RCC_GetUSBClockSource

LL_RCC_SetRTCClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)**
Function description	Set RTC Clock Source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_RTC_CLKSOURCE_NONE – LL_RCC_RTC_CLKSOURCE_LSE – LL_RCC_RTC_CLKSOURCE_LSI – LL_RCC_RTC_CLKSOURCE_HSE
Return values	• **None**
Notes	• Once the RTC clock source has been selected, it cannot be changed any more unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set). The RTCRST bit can be used to reset them.
Reference Manual to LL API cross reference:	• CSR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)**
Function description	Get RTC Clock Source.
Return values	• **Returned:** value can be one of the following values:

	– LL_RCC_RTC_CLKSOURCE_NONE – LL_RCC_RTC_CLKSOURCE_LSE – LL_RCC_RTC_CLKSOURCE_LSI – LL_RCC_RTC_CLKSOURCE_HSE
Reference Manual to LL API cross reference:	• CSR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name	**__STATIC_INLINE void LL_RCC_EnableRTC (void)**
Function description	Enable RTC.
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR RTCEN LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name	**__STATIC_INLINE void LL_RCC_DisableRTC (void)**
Function description	Disable RTC.
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR RTCEN LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)**
Function description	Check if RTC has been enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name	**__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)**
Function description	Force the Backup domain reset.
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR RTCRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name	**__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)**
Function description	Release the Backup domain reset.
Return values	• **None**
Reference Manual to LL API cross reference:	• CSR RTCRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_PLL_Enable

Function name	**__STATIC_INLINE void LL_RCC_PLL_Enable (void)**
Function description	Enable PLL.
Return values	• **None**
Reference Manual to LL API cross reference:	• CR PLLON LL_RCC_PLL_Enable

LL_RCC_PLL_Disable

Function name	**__STATIC_INLINE void LL_RCC_PLL_Disable (void)**
Function description	Disable PLL.
Return values	• **None**
Notes	• Cannot be disabled if the PLL clock is used as the system clock
Reference Manual to LL API cross reference:	• CR PLLON LL_RCC_PLL_Disable

LL_RCC_PLL_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_IsReady (void)**
Function description	Check if PLL Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR PLLRDY LL_RCC_PLL_IsReady

LL_RCC_PLL_ConfigDomain_SYS

Function name	**__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SYS (uint32_t Source, uint32_t PLLMul, uint32_t PLLDiv)**
Function description	Configure PLL used for SYSCLK Domain.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_PLLSOURCE_HSI – LL_RCC_PLLSOURCE_HSE

	• **PLLMul:** This parameter can be one of the following values: – LL_RCC_PLL_MUL_3 – LL_RCC_PLL_MUL_4 – LL_RCC_PLL_MUL_6 – LL_RCC_PLL_MUL_8 – LL_RCC_PLL_MUL_12 – LL_RCC_PLL_MUL_16 – LL_RCC_PLL_MUL_24 – LL_RCC_PLL_MUL_32 – LL_RCC_PLL_MUL_48 • **PLLDiv:** This parameter can be one of the following values: – LL_RCC_PLL_DIV_2 – LL_RCC_PLL_DIV_3 – LL_RCC_PLL_DIV_4
Return values	• **None**
Reference Manual to LL API cross reference:	• CFGR PLLSRC LL_RCC_PLL_ConfigDomain_SYS • CFGR PLLMUL LL_RCC_PLL_ConfigDomain_SYS • CFGR PLLDIV 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_HSE
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_3 – LL_RCC_PLL_MUL_4 – LL_RCC_PLL_MUL_6 – LL_RCC_PLL_MUL_8 – LL_RCC_PLL_MUL_12 – LL_RCC_PLL_MUL_16 – LL_RCC_PLL_MUL_24 – LL_RCC_PLL_MUL_32 – LL_RCC_PLL_MUL_48
Reference Manual to LL API cross reference:	• CFGR PLLMUL LL_RCC_PLL_GetMultiplicator

LL_RCC_PLL_GetDivider

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_GetDivider (void)**
Function description	Get Division factor for the main PLL and other PLL.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLL_DIV_2 – LL_RCC_PLL_DIV_3 – LL_RCC_PLL_DIV_4
Reference Manual to LL API cross reference:	• CFGR PLLDIV LL_RCC_PLL_GetDivider

LL_RCC_ClearFlag_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)**
Function description	Clear LSI ready interrupt flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• CICR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)**
Function description	Clear LSE ready interrupt flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• CICR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_MSIRDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_MSIRDY (void)**
Function description	Clear MSI ready interrupt flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• CICR MSIRDYC LL_RCC_ClearFlag_MSIRDY

LL_RCC_ClearFlag_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)**
Function description	Clear HSI ready interrupt flag.
Return values	• **None**
Reference Manual to LL API cross	• CICR HSIRDYC LL_RCC_ClearFlag_HSIRDY

reference:

LL_RCC_ClearFlag_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)**
Function description	Clear HSE ready interrupt flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• CICR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_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:	• CICR PLLRDYC LL_RCC_ClearFlag_PLLRDY

LL_RCC_ClearFlag_HSI48RDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void)**
Function description	Clear HSI48 ready interrupt flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• CICR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY

LL_RCC_ClearFlag_HSECSS

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)**
Function description	Clear Clock security system interrupt flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• CICR CSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_ClearFlag_LSECSS

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_LSECSS (void)**
Function description	Clear LSE Clock security system interrupt flag.
Return values	• **None**
Reference Manual to LL API cross reference:	• CICR LSECSSC LL_RCC_ClearFlag_LSECSS

LL_RCC_IsActiveFlag_LSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)**
Function description	Check if LSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)**
Function description	Check if LSE ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_MSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_MSIRDY (void)**
Function description	Check if MSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR MSIRDYF LL_RCC_IsActiveFlag_MSIRDY

LL_RCC_IsActiveFlag_HSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)**
Function description	Check if HSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY (void)**
Function description	Check if HSE ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_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:	• CIFR PLLRDYF LL_RCC_IsActiveFlag_PLLRDY

LL_RCC_IsActiveFlag_HSI48RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY (void)**
Function description	Check if HSI48 ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR HSI48RDYF LL_RCC_IsActiveFlag_HSI48RDY

LL_RCC_IsActiveFlag_HSECSS

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)**
Function description	Check if Clock security system interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR CSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_LSECSS

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSECSS (void)**
Function description	Check if LSE Clock security system interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR LSECSSF LL_RCC_IsActiveFlag_LSECSS

LL_RCC_IsActiveFlag_HSIDIV

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIDIV (void)**

Function description	Check if HSI Divider is enabled (it divides by 4)
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR HSIDIVF LL_RCC_IsActiveFlag_HSIDIV

LL_RCC_IsActiveFlag_FWRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_FWRST (void)**
Function description	Check if RCC flag FW reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR FWRSTF LL_RCC_IsActiveFlag_FWRST

LL_RCC_IsActiveFlag_IWDGRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)**
Function description	Check if RCC flag Independent Watchdog reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_LPWRRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)**
Function description	Check if RCC flag Low Power reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_OBLRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_OBLRST (void)**
Function description	Check if RCC flag is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR OBLRSTF LL_RCC_IsActiveFlag_OBLRST

LL_RCC_IsActiveFlag_PINRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)**
Function description	Check if RCC flag Pin reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_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_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	• CSR RMVF LL_RCC_ClearResetFlags

reference:

LL_RCC_EnableIT_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)**
Function description	Enable LSI ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)**
Function description	Enable LSE ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_MSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_MSIRDY (void)**
Function description	Enable MSI ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER MSIRDYIE LL_RCC_EnableIT_MSIRDY

LL_RCC_EnableIT_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)**
Function description	Enable HSI ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)**
Function description	Enable HSE ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_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:	• CIER PLLRDYIE LL_RCC_EnableIT_PLLRDY

LL_RCC_EnableIT_HSI48RDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void)**
Function description	Enable HSI48 ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER HSI48RDYIE LL_RCC_EnableIT_HSI48RDY

LL_RCC_EnableIT_LSECSS

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSECSS (void)**
Function description	Enable LSE clock security system interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER LSECSSIE LL_RCC_EnableIT_LSECSS

LL_RCC_DisableIT_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)**
Function description	Disable LSI ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)**
Function description	Disable LSE ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_MSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_MSIRDY (void)**
Function description	Disable MSI ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER MSIRDYIE LL_RCC_DisableIT_MSIRDY

LL_RCC_DisableIT_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)**
Function description	Disable HSI ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)**
Function description	Disable HSE ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_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:	• CIER PLLRDYIE LL_RCC_DisableIT_PLLRDY

LL_RCC_DisableIT_HSI48RDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void)**
Function description	Disable HSI48 ready interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER HSI48RDYIE LL_RCC_DisableIT_HSI48RDY

LL_RCC_DisableIT_LSECSS

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSECSS (void)**
Function description	Disable LSE clock security system interrupt.
Return values	• **None**
Reference Manual to LL API cross reference:	• CIER LSECSSIE LL_RCC_DisableIT_LSECSS

LL_RCC_IsEnabledIT_LSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void)**
Function description	Checks if LSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY

LL_RCC_IsEnabledIT_LSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void)**
Function description	Checks if LSE ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER LSERDYIE LL_RCC_IsEnabledIT_LSERDY

LL_RCC_IsEnabledIT_MSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_MSIRDY (void)**
Function description	Checks if MSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER MSIRDYIE LL_RCC_IsEnabledIT_MSIRDY

LL_RCC_IsEnabledIT_HSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void)**
Function description	Checks if HSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CIER HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY

reference:

LL_RCC_IsEnabledIT_HSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY (void)**
Function description	Checks if HSE ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_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:	• CIER PLLRDYIE LL_RCC_IsEnabledIT_PLLRDY

LL_RCC_IsEnabledIT_HSI48RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI48RDY (void)**
Function description	Checks if HSI48 ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER HSI48RDYIE LL_RCC_IsEnabledIT_HSI48RDY

LL_RCC_IsEnabledIT_LSECSS

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSECSS (void)**
Function description	Checks if LSECSS interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER LSECSSIE LL_RCC_IsEnabledIT_LSECSS

LL_RCC_DeInit

Function name	**ErrorStatus LL_RCC_DeInit (void)**
Function description	Reset the RCC clock configuration to the default reset state.
Return values	• **An:** ErrorStatus enumeration value:

- – SUCCESS: RCC registers are de-initialized
- – ERROR: not applicable

Notes	• The default reset state of the clock configuration is given below: MSI ON and used as system clock sourceHSE, HSI and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO OFFAll interrupts disabled • This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name	**void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)**
Function description	Return the frequencies of different on chip clocks; System, AHB, APB1 and APB2 buses clocks.
Parameters	• **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies
Return values	• **None**
Notes	• Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 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: (*) value not defined in all devices. – LL_RCC_USART1_CLKSOURCE – LL_RCC_USART2_CLKSOURCE (*)
Return values	• **USART:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetI2CClockFreq

Function name	**uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)**
Function description	Return I2Cx clock frequency.
Parameters	• **I2CxSource:** This parameter can be one of the following values: (*) value not defined in all devices – LL_RCC_I2C1_CLKSOURCE – LL_RCC_I2C3_CLKSOURCE (*)
Return values	• **I2C:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that HSI oscillator is not ready

LL_RCC_GetLPUARTClockFreq

Function name	**uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)**
Function description	Return LPUARTx clock frequency.
Parameters	• **LPUARTxSource:** This parameter can be one of the following values: – LL_RCC_LPUART1_CLKSOURCE
Return values	• **LPUART:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetLPTIMClockFreq

Function name	**uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)**
Function description	Return LPTIMx clock frequency.
Parameters	• **LPTIMxSource:** This parameter can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE
Return values	• **LPTIM:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetUSBClockFreq

Function name	**uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)**
Function description	Return USBx clock frequency.
Parameters	• **USBxSource:** This parameter can be one of the following values: – LL_RCC_USB_CLKSOURCE
Return values	• **USB:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI48) or PLL is not ready – LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

69.3 RCC Firmware driver defines

69.3.1 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

APB high-speed prescaler (APB2)

LL_RCC_APB2_DIV_1	HCLK not divided
LL_RCC_APB2_DIV_2	HCLK divided by 2
LL_RCC_APB2_DIV_4	HCLK divided by 4
LL_RCC_APB2_DIV_8	HCLK divided by 8
LL_RCC_APB2_DIV_16	HCLK divided by 16

Clear Flags Defines

LL_RCC_CICR_LSIRDYC	LSI Ready Interrupt Clear
LL_RCC_CICR_LSERDYC	LSE Ready Interrupt Clear
LL_RCC_CICR_HSIRDYC	HSI Ready Interrupt Clear
LL_RCC_CICR_HSERDYC	HSE Ready Interrupt Clear
LL_RCC_CICR_PLLRDYC	PLL Ready Interrupt Clear
LL_RCC_CICR_MSIRDYC	MSI Ready Interrupt Clear
LL_RCC_CICR_HSI48RDYC	HSI48 Ready Interrupt Clear
LL_RCC_CICR_LSECSSC	LSE Clock Security System Interrupt Clear
LL_RCC_CICR_CSSC	Clock Security System Interrupt Clear

Get Flags Defines

LL_RCC_CIFR_LSIRDYF	LSI Ready Interrupt flag
LL_RCC_CIFR_LSERDYF	LSE Ready Interrupt flag
LL_RCC_CIFR_HSIRDYF	HSI Ready Interrupt flag
LL_RCC_CIFR_HSERDYF	HSE Ready Interrupt flag
LL_RCC_CIFR_PLLRDYF	PLL Ready Interrupt flag
LL_RCC_CIFR_MSIRDYF	MSI Ready Interrupt flag
LL_RCC_CIFR_HSI48RDYF	HSI48 Ready Interrupt flag
LL_RCC_CIFR_LSECSSF	LSE Clock Security System Interrupt flag
LL_RCC_CIFR_CSSF	Clock Security System Interrupt flag
LL_RCC_CSR_FWRSTF	Firewall reset 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_WWDGRSTF	Window watchdog reset flag
LL_RCC_CSR_LPWRRSTF	Low-Power reset flag

Peripheral I2C get clock source

LL_RCC_I2C1_CLKSOURCE	I2C1 clock source selection bits
LL_RCC_I2C3_CLKSOURCE	I2C3 clock source selection bits

Peripheral I2C clock source selection

LL_RCC_I2C1_CLKSOURCE_PCLK1	PCLK1 selected as I2C1 clock
LL_RCC_I2C1_CLKSOURCE_SYSCLK	SYSCLK selected as I2C1 clock
LL_RCC_I2C1_CLKSOURCE_HSI	HSI selected as I2C1 clock
LL_RCC_I2C3_CLKSOURCE_PCLK1	PCLK1 selected as I2C3 clock
LL_RCC_I2C3_CLKSOURCE_SYSCLK	SYSCLK selected as I2C3 clock
LL_RCC_I2C3_CLKSOURCE_HSI	HSI selected as I2C3 clock

IT Defines

LL_RCC_CIER_LSIRDYIE	LSI Ready Interrupt Enable
LL_RCC_CIER_LSERDYIE	LSE Ready Interrupt Enable
LL_RCC_CIER_HSIRDYIE	HSI Ready Interrupt Enable
LL_RCC_CIER_HSERDYIE	HSE Ready Interrupt Enable
LL_RCC_CIER_PLLRDYIE	PLL Ready Interrupt Enable
LL_RCC_CIER_MSIRDYIE	MSI Ready Interrupt Enable
LL_RCC_CIER_HSI48RDYIE	HSI48 Ready Interrupt Enable
LL_RCC_CIER_LSECSSIE	LSE CSS Interrupt Enable

Peripheral LPTIM get clock source

LL_RCC_LPTIM1_CLKSOURCE	LPTIM1 clock source selection bits

Peripheral LPTIM clock source selection

LL_RCC_LPTIM1_CLKSOURCE_PCLK1	PCLK1 selected as LPTIM1 clock
LL_RCC_LPTIM1_CLKSOURCE_LSI	LSI selected as LPTIM1 clock
LL_RCC_LPTIM1_CLKSOURCE_HSI	HSI selected as LPTIM1 clock
LL_RCC_LPTIM1_CLKSOURCE_LSE	LSE selected as LPTIM1 clock

Peripheral LPUART get clock source

LL_RCC_LPUART1_CLKSOURCE	LPUART1 clock source selection bits

Peripheral LPUART clock source selection

LL_RCC_LPUART1_CLKSOURCE_PCLK1	PCLK1 selected as LPUART1 clock
LL_RCC_LPUART1_CLKSOURCE_SYSCLK	SYSCLK selected as LPUART1 clock
LL_RCC_LPUART1_CLKSOURCE_HSI	HSI selected as LPUART1 clock
LL_RCC_LPUART1_CLKSOURCE_LSE	LSE selected as LPUART1 clock

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW	Xtal mode lower driving capability

LL_RCC_LSEDRIVE_MEDIUMLOW	Xtal mode medium low driving capability
LL_RCC_LSEDRIVE_MEDIUMHIGH	Xtal mode medium high driving capability
LL_RCC_LSEDRIVE_HIGH	Xtal mode higher driving capability

MCO1 SOURCE selection

LL_RCC_MCO1SOURCE_NOCLOCK	MCO output disabled, no clock on MCO
LL_RCC_MCO1SOURCE_SYSCLK	SYSCLK selection as MCO source
LL_RCC_MCO1SOURCE_HSI	HSI selection as MCO source
LL_RCC_MCO1SOURCE_MSI	MSI selection as MCO source
LL_RCC_MCO1SOURCE_HSE	HSE selection as MCO source
LL_RCC_MCO1SOURCE_LSI	LSI selection as MCO source
LL_RCC_MCO1SOURCE_LSE	LSE selection as MCO source
LL_RCC_MCO1SOURCE_HSI48	HSI48 selection as MCO source
LL_RCC_MCO1SOURCE_PLLCLK	PLLCLK selection as MCO source

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

MSI clock ranges

LL_RCC_MSIRANGE_0	MSI = 65.536 KHz
LL_RCC_MSIRANGE_1	MSI = 131.072 KHz
LL_RCC_MSIRANGE_2	MSI = 262.144 KHz
LL_RCC_MSIRANGE_3	MSI = 524.288 KHz
LL_RCC_MSIRANGE_4	MSI = 1.048 MHz
LL_RCC_MSIRANGE_5	MSI = 2.097 MHz
LL_RCC_MSIRANGE_6	MSI = 4.194 MHz

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_FREQUENCY_NO	No clock enabled for the peripheral
LL_RCC_PERIPH_FREQUENCY_NA	Frequency cannot be provided as external clock

PLL SOURCE

LL_RCC_PLLSOURCE_HSI	HSI clock selected as PLL entry clock source
LL_RCC_PLLSOURCE_HSE	HSE clock selected as PLL entry clock source

PLL division factor

LL_RCC_PLL_DIV_2	PLL clock output = PLLVCO / 2
LL_RCC_PLL_DIV_3	PLL clock output = PLLVCO / 3
LL_RCC_PLL_DIV_4	PLL clock output = PLLVCO / 4

PLL Multiplicator factor

LL_RCC_PLL_MUL_3	PLL input clock * 3
LL_RCC_PLL_MUL_4	PLL input clock * 4
LL_RCC_PLL_MUL_6	PLL input clock * 6
LL_RCC_PLL_MUL_8	PLL input clock * 8
LL_RCC_PLL_MUL_12	PLL input clock * 12
LL_RCC_PLL_MUL_16	PLL input clock * 16
LL_RCC_PLL_MUL_24	PLL input clock * 24
LL_RCC_PLL_MUL_32	PLL input clock * 32
LL_RCC_PLL_MUL_48	PLL input clock * 48

Peripheral RNG get clock source

LL_RCC_RNG_CLKSOURCE	HSI48 RC clock source selection bit for RNG

Peripheral RNG clock source selection

LL_RCC_RNG_CLKSOURCE_PLL	PLL selected as RNG clock
LL_RCC_RNG_CLKSOURCE_HSI48	HSI48 selected as RNG clock

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE	No clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_LSE	LSE oscillator clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_LSI	LSI oscillator clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_HSE	HSE oscillator clock divided by a programmable prescaler (selection through

RTC HSE Prescaler

LL_RCC_RTC_HSE_DIV_2	HSE is divided by 2 for RTC clock
LL_RCC_RTC_HSE_DIV_4	HSE is divided by 4 for RTC clock
LL_RCC_RTC_HSE_DIV_8	HSE is divided by 8 for RTC clock
LL_RCC_RTC_HSE_DIV_16	HSE is divided by 16 for RTC clock

Wakeup from Stop and CSS backup clock selection

LL_RCC_STOP_WAKEUPCLOCK_MSI	MSI selection after wake-up from STOP
LL_RCC_STOP_WAKEUPCLOCK_HSI	HSI selection after wake-up from STOP

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_CLKSOURCE_MSI	MSI selection as system clock
LL_RCC_SYS_CLKSOURCE_HSI	HSI selection as system clock
LL_RCC_SYS_CLKSOURCE_HSE	HSE selection as system clock
LL_RCC_SYS_CLKSOURCE_PLL	PLL selection as system clock

System clock switch status

LL_RCC_SYS_CLKSOURCE_STATUS_MSI	MSI used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_HSI	HSI used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_HSE	HSE used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_PLL	PLL used as system clock

Peripheral USART get clock source

LL_RCC_USART1_CLKSOURCE	USART1 clock source selection bits
LL_RCC_USART2_CLKSOURCE	USART2 clock source selection bits

Peripheral USART clock source selection

LL_RCC_USART1_CLKSOURCE_PCLK2	PCLK2 selected as USART1 clock
LL_RCC_USART1_CLKSOURCE_SYSCLK	SYSCLK selected as USART1 clock
LL_RCC_USART1_CLKSOURCE_HSI	HSI selected as USART1 clock
LL_RCC_USART1_CLKSOURCE_LSE	LSE selected as USART1 clock
LL_RCC_USART2_CLKSOURCE_PCLK1	PCLK1 selected as USART2 clock
LL_RCC_USART2_CLKSOURCE_SYSCLK	SYSCLK selected as USART2 clock
LL_RCC_USART2_CLKSOURCE_HSI	HSI selected as USART2 clock
LL_RCC_USART2_CLKSOURCE_LSE	LSE selected as USART2 clock

Peripheral USB get clock source

LL_RCC_USB_CLKSOURCE	HSI48 RC clock source selection bit for USB

Peripheral USB clock source selection

LL_RCC_USB_CLKSOURCE_PLL	PLL selected as USB clock

LL_RCC_USB_CLKSOURCE_HSI48 HSI48 selected as USB clock

Calculate frequencies

__LL_RCC_CALC_PLLCLK_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLMUL__: This parameter can be one of the following values:
  - LL_RCC_PLL_MUL_3
  - LL_RCC_PLL_MUL_4
  - LL_RCC_PLL_MUL_6
  - LL_RCC_PLL_MUL_8
  - LL_RCC_PLL_MUL_12
  - LL_RCC_PLL_MUL_16
  - LL_RCC_PLL_MUL_24
  - LL_RCC_PLL_MUL_32
  - LL_RCC_PLL_MUL_48
- __PLLDIV__: This parameter can be one of the following values:
  - LL_RCC_PLL_DIV_2
  - LL_RCC_PLL_DIV_3
  - LL_RCC_PLL_DIV_4

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_FREQ (HSE_VALUE, LL_RCC_PLL_GetMultiplicator (), LL_RCC_PLL_GetDivider ());

__LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency (based on MSI/HSE/HSI/PLLCLK)
- __AHBPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

Notes:

- : __AHBPRESCALER__ be retrieved by LL_RCC_GetAHBPrescaler ex: __LL_RCC_CALC_HCLK_FREQ(LL_RCC_GetAHBPrescaler())

__LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

Notes:

- : __APB1PRESCALER__ be retrieved by LL_RCC_GetAPB1Prescaler ex: __LL_RCC_CALC_PCLK1_FREQ(LL_RCC_GetAPB1Prescaler())

__LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return value:

- PCLK2: clock frequency (in Hz)

Notes:

- : __APB2PRESCALER__ be retrieved by LL_RCC_GetAPB2Prescaler ex: __LL_RCC_CALC_PCLK2_FREQ(LL_RCC_GetAPB2Prescaler())

__LL_RCC_CALC_MSI_FREQ

Description:

- Helper macro to calculate the MSI frequency (in Hz)

Parameters:

- __MSIRANGE__: This parameter can be one of the following values:
  - LL_RCC_MSIRANGE_0
  - LL_RCC_MSIRANGE_1
  - LL_RCC_MSIRANGE_2
  - LL_RCC_MSIRANGE_3
  - LL_RCC_MSIRANGE_4
  - LL_RCC_MSIRANGE_5
  - LL_RCC_MSIRANGE_6

Return value:

- MSI: clock frequency (in Hz)

Notes:

- : __MSIRANGE__can be retrieved by LL_RCC_MSI_GetRange ex: __LL_RCC_CALC_MSI_FREQ(LL_RCC_MSI_GetRange())

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

70 LL RNG Generic Driver

70.1 RNG Firmware driver API description

70.1.1 Detailed description of functions

LL_RNG_Enable

Function name	**__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)**
Function description	Enable Random Number Generation.
Parameters	• **RNGx:** RNG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR RNGEN LL_RNG_Enable

LL_RNG_Disable

Function name	**__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)**
Function description	Disable Random Number Generation.
Parameters	• **RNGx:** RNG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR RNGEN LL_RNG_Disable

LL_RNG_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsEnabled (RNG_TypeDef * RNGx)**
Function description	Check if Random Number Generator is enabled.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR RNGEN LL_RNG_IsEnabled

LL_RNG_IsActiveFlag_DRDY

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (RNG_TypeDef * RNGx)**
Function description	Indicate if the RNG Data ready Flag is set or not.

Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR DRDY LL_RNG_IsActiveFlag_DRDY

LL_RNG_IsActiveFlag_CECS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Clock Error Current Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CECS LL_RNG_IsActiveFlag_CECS

LL_RNG_IsActiveFlag_SECS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Seed Error Current Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR SECS LL_RNG_IsActiveFlag_SECS

LL_RNG_IsActiveFlag_CEIS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Clock Error Interrupt Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CEIS LL_RNG_IsActiveFlag_CEIS

LL_RNG_IsActiveFlag_SEIS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Seed Error Interrupt Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR SEIS LL_RNG_IsActiveFlag_SEIS

LL_RNG_ClearFlag_CEIS

Function name	**__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)**
Function description	Clear Clock Error interrupt Status (CEIS) Flag.
Parameters	• **RNGx:** RNG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR CEIS LL_RNG_ClearFlag_CEIS

LL_RNG_ClearFlag_SEIS

Function name	**__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)**
Function description	Clear Seed Error interrupt Status (SEIS) Flag.
Parameters	• **RNGx:** RNG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR SEIS LL_RNG_ClearFlag_SEIS

LL_RNG_EnableIT

Function name	**__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)**
Function description	Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	• **RNGx:** RNG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR IE LL_RNG_EnableIT

LL_RNG_DisableIT

Function name	**__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)**
Function description	Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	• **RNGx:** RNG Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• CR IE LL_RNG_DisableIT

LL_RNG_IsEnabledIT

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (RNG_TypeDef * RNGx)**
Function description	Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR IE LL_RNG_IsEnabledIT

LL_RNG_ReadRandData32

Function name	**__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (RNG_TypeDef * RNGx)**
Function description	Return32-bit Random Number value.
Parameters	• **RNGx:** RNG Instance
Return values	• **Generated:** 32-bit random value
Reference Manual to LL API cross reference:	• DR RNDATA LL_RNG_ReadRandData32

LL_RNG_DeInit

Function name	**ErrorStatus LL_RNG_DeInit (RNG_TypeDef * RNGx)**
Function description	De-initialize RNG registers (Registers restored to their default values).
Parameters	• **RNGx:** RNG Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RNG registers are de-initialized – ERROR: not applicable

70.2 RNG Firmware driver defines

70.2.1 RNG

Get Flags Defines

LL_RNG_SR_DRDY	Register contains valid random data
LL_RNG_SR_CECS	Clock error current status
LL_RNG_SR_SECS	Seed error current status

LL_RNG_SR_CEIS Clock error interrupt status

LL_RNG_SR_SEIS Seed error interrupt status

IT Defines

LL_RNG_CR_IE RNG Interrupt enable

Common Write and read registers Macros

LL_RNG_WriteReg

Description:

- Write a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RNG_ReadReg

Description:

- Read a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be read

Return value:

- Register: value

71 LL RTC Generic Driver

71.1 RTC Firmware driver registers structures

71.1.1 LL_RTC_InitTypeDef

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()**.

71.1.2 LL_RTC_TimeTypeDef

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()**.

71.1.3 LL_RTC_DateTypeDef

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()**.

71.1.4 LL_RTC_AlarmTypeDef

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_LL_EC_ALMA_MASK*** for ALARM A or ***RTC_LL_EC_ALMB_MASK*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A or **LL_RTC_ALMB_SetMask()** for ALARM B
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A or ***RTC_LL_EC_ALMB_WEEKDAY_SELECTION*** for ALARM BThis feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A or **LL_RTC_ALMB_EnableWeekday()** or **LL_RTC_ALMB_DisableWeekday()** for ALARM B

- ***uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min_Data = 1 and Max_Data = 31.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetDay()** for ALARM A or **LL_RTC_ALMB_SetDay()** for ALARM B.If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of ***RTC_LL_EC_WEEKDAY***.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetWeekDay()** for ALARM A or **LL_RTC_ALMB_SetWeekDay()** for ALARM B.

71.2 RTC Firmware driver API description

71.2.1 Detailed description of functions

LL_RTC_SetHourFormat

Function name	**__STATIC_INLINE void LL_RTC_SetHourFormat (RTC_TypeDef * RTCx, uint32_t HourFormat)**
Function description	Set Hours format (24 hour/day or AM/PM hour format)
Parameters	• **RTCx:** RTC Instance • **HourFormat:** This parameter can be one of the following values: – LL_RTC_HOURFORMAT_24HOUR – LL_RTC_HOURFORMAT_AMPM
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function)
Reference Manual to LL API cross reference:	• CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (RTC_TypeDef * RTCx)**
Function description	Get Hours format (24 hour/day or AM/PM hour format)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_HOURFORMAT_24HOUR – LL_RTC_HOURFORMAT_AMPM
Reference Manual to LL API cross reference:	• CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name	**__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)**

Function description	Select the flag to be routed to RTC_ALARM output.
Parameters	• **RTCx:** RTC Instance • **AlarmOutput:** This parameter can be one of the following values: – LL_RTC_ALARMOUT_DISABLE – LL_RTC_ALARMOUT_ALMA – LL_RTC_ALARMOUT_ALMB – LL_RTC_ALARMOUT_WAKEUP
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (RTC_TypeDef * RTCx)**
Function description	Get the flag to be routed to RTC_ALARM output.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALARMOUT_DISABLE – LL_RTC_ALARMOUT_ALMA – LL_RTC_ALARMOUT_ALMB – LL_RTC_ALARMOUT_WAKEUP
Reference Manual to LL API cross reference:	• CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name	**__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)**
Function description	Set RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	• **RTCx:** RTC Instance • **Output:** This parameter can be one of the following values: – LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN – LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL
Return values	• **None**
Notes	• Used only when RTC_ALARM is mapped on PC13
Reference Manual to LL API cross reference:	• OR ALARMOUTTYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (RTC_TypeDef * RTCx)**
Function description	Get RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN – LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL
Notes	• used only when RTC_ALARM is mapped on PC13
Reference Manual to LL API cross reference:	• OR ALARMOUTTYPE LL_RTC_GetAlarmOutputType

LL_RTC_EnableInitMode

Function name	**__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)**
Function description	Enable initialization mode.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.
Reference Manual to LL API cross reference:	• ISR INIT LL_RTC_EnableInitMode

LL_RTC_DisableInitMode

Function name	**__STATIC_INLINE void LL_RTC_DisableInitMode (RTC_TypeDef * RTCx)**
Function description	Disable initialization mode (Free running mode)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR INIT LL_RTC_DisableInitMode

LL_RTC_SetOutputPolarity

Function name	**__STATIC_INLINE void LL_RTC_SetOutputPolarity (RTC_TypeDef * RTCx, uint32_t Polarity)**
Function description	Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is

asserted)

Parameters	• **RTCx:** RTC Instance • **Polarity:** This parameter can be one of the following values: – LL_RTC_OUTPUTPOLARITY_PIN_HIGH – LL_RTC_OUTPUTPOLARITY_PIN_LOW
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (RTC_TypeDef * RTCx)**
Function description	Get Output polarity.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_OUTPUTPOLARITY_PIN_HIGH – LL_RTC_OUTPUTPOLARITY_PIN_LOW
Reference Manual to LL API cross reference:	• CR POL LL_RTC_GetOutputPolarity

LL_RTC_EnableShadowRegBypass

Function name	**__STATIC_INLINE void LL_RTC_EnableShadowRegBypass (RTC_TypeDef * RTCx)**
Function description	Enable Bypass the shadow registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR BYPSHAD LL_RTC_EnableShadowRegBypass

LL_RTC_DisableShadowRegBypass

Function name	**__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)**
Function description	Disable Bypass the shadow registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**

Reference Manual to LL API cross reference:	• CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (RTC_TypeDef * RTCx)**
Function description	Check if Shadow registers bypass is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

LL_RTC_EnableRefClock

Function name	**__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)**
Function description	Enable RTC_REFIN reference clock detection (50 or 60 Hz)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function)
Reference Manual to LL API cross reference:	• CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name	**__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)**
Function description	Disable RTC_REFIN reference clock detection (50 or 60 Hz)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function)
Reference Manual to LL API cross reference:	• CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name	**__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)**
Function description	Set Asynchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance • **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7F
Return values	• **None**
Reference Manual to LL API cross reference:	• PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name	**__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)**
Function description	Set Synchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance • **SynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7FFF
Return values	• **None**
Reference Manual to LL API cross reference:	• PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (RTC_TypeDef * RTCx)**
Function description	Get Asynchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data = 0 and Max_Data = 0x7F
Reference Manual to LL API cross reference:	• PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (RTC_TypeDef * RTCx)**
Function description	Get Synchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data = 0 and Max_Data = 0x7FFF
Reference Manual to LL API cross	• PRER PREDIV_S LL_RTC_GetSynchPrescaler

reference:

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_EnableOutRemap

Function name	**__STATIC_INLINE void LL_RTC_EnableOutRemap (RTC_TypeDef * RTCx)**
Function description	Enable RTC_OUT remap.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• OR OUT_RMP LL_RTC_EnableOutRemap

LL_RTC_DisableOutRemap

Function name	**__STATIC_INLINE void LL_RTC_DisableOutRemap (RTC_TypeDef * RTCx)**
Function description	Disable RTC_OUT remap.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• OR OUT_RMP LL_RTC_DisableOutRemap

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	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection

	function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function) • helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format
Reference Manual to LL API cross reference:	• TR HT LL_RTC_TIME_SetHour • TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (RTC_TypeDef * RTCx)**
Function description	Get Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• if 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 hour from BCD to Binary format
Reference Manual to LL API cross reference:	• TR HT LL_RTC_TIME_GetHour • TR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name	**__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)**
Function description	Set Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function) • helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format
Reference Manual to LL API cross reference:	• TR MNT LL_RTC_TIME_SetMinute • TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• if 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 minute from BCD to Binary format
Reference Manual to LL API cross reference:	• TR MNT LL_RTC_TIME_GetMinute • TR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name	**__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)**
Function description	Set Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function) • helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	• TR ST LL_RTC_TIME_SetSecond • TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• if 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 BCK 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 BCK LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (RTC_TypeDef * RTCx)**
Function description	Check if RTC Day Light Saving stored operation has been enabled

or not.

Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR BCK LL_RTC_TIME_IsDayLightStoreEnabled

LL_RTC_TIME_DecHour

Function name	**__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)**
Function description	Subtract 1 hour (winter time change)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR SUB1H LL_RTC_TIME_DecHour

LL_RTC_TIME_IncHour

Function name	**__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)**
Function description	Add 1 hour (summer time change)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (RTC_TypeDef * RTCx)**
Function description	Get Sub second value in the synchronous prescaler counter.
Parameters	• **RTCx:** RTC Instance
Return values	• **Sub:** second value (number between 0 and 65535)
Notes	• You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula: ==> Seconds fraction ratio * time_unit= [(SecondFraction-

	SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS.
Reference Manual to LL API cross reference:	• SSR SS LL_RTC_TIME_GetSubSecond

LL_RTC_TIME_Synchronize

Function name	**__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)**
Function description	Synchronize to a remote clock with a high degree of precision.
Parameters	• **RTCx:** RTC Instance • **ShiftSecond:** This parameter can be one of the following values: – LL_RTC_SHIFT_SECOND_DELAY – LL_RTC_SHIFT_SECOND_ADVANCE • **Fraction:** Number of Seconds Fractions (any value from 0 to 0x7FFF)
Return values	• **None**
Notes	• This operation effectively subtracts from (delays) or advance the clock of a fraction of a second. • Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • When REFCKON is set, firmware must not write to Shift control register.
Reference Manual to LL API cross reference:	• SHIFTR ADD1S LL_RTC_TIME_Synchronize • SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name	**__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)**
Function description	Set Year in BCD format.
Parameters	• **RTCx:** RTC Instance • **Year:** Value between Min_Data=0x00 and Max_Data=0x99
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format
Reference Manual to LL API cross reference:	• DR YT LL_RTC_DATE_SetYear • DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear**

	(RTC_TypeDef * RTCx)
Function description	Get Year in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x99
Notes	• if 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:	• DR YT LL_RTC_DATE_GetYear • DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name	**__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)**
Function description	Set Week day.
Parameters	• **RTCx:** RTC Instance • **WeekDay:** This parameter can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Return values	• **None**
Reference Manual to LL API cross reference:	• DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (RTC_TypeDef * RTCx)**
Function description	Get Week day.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY

Notes	• if 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_APRIL – LL_RTC_MONTH_MAY – LL_RTC_MONTH_JUNE – LL_RTC_MONTH_JULY – LL_RTC_MONTH_AUGUST – LL_RTC_MONTH_SEPTEMBER – LL_RTC_MONTH_OCTOBER – LL_RTC_MONTH_NOVEMBER – LL_RTC_MONTH_DECEMBER
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Month from binary to BCD format
Reference Manual to LL API cross reference:	• DR MT LL_RTC_DATE_SetMonth • DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (RTC_TypeDef * RTCx)**
Function description	Get Month in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_MONTH_JANUARY – LL_RTC_MONTH_FEBRUARY – LL_RTC_MONTH_MARCH – LL_RTC_MONTH_APRIL – LL_RTC_MONTH_MAY – LL_RTC_MONTH_JUNE – LL_RTC_MONTH_JULY – LL_RTC_MONTH_AUGUST – LL_RTC_MONTH_SEPTEMBER – LL_RTC_MONTH_OCTOBER

– LL_RTC_MONTH_NOVEMBER
– LL_RTC_MONTH_DECEMBER

Notes	• if 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:	• DR MT LL_RTC_DATE_GetMonth • DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name	**__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)**
Function description	Set Day in BCD format.
Parameters	• **RTCx:** RTC Instance • **Day:** Value between Min_Data=0x01 and Max_Data=0x31
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format
Reference Manual to LL API cross reference:	• DR DT LL_RTC_DATE_SetDay • DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (RTC_TypeDef * RTCx)**
Function description	Get Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31
Notes	• if 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_APRIL – LL_RTC_MONTH_MAY – LL_RTC_MONTH_JUNE – LL_RTC_MONTH_JULY – LL_RTC_MONTH_AUGUST – LL_RTC_MONTH_SEPTEMBER – LL_RTC_MONTH_OCTOBER – LL_RTC_MONTH_NOVEMBER – LL_RTC_MONTH_DECEMBER • **Year:** Value between Min_Data=0x00 and Max_Data=0x99
Return values	• **None**
Reference Manual to LL API cross reference:	• DR WDU LL_RTC_DATE_Config • DR MT LL_RTC_DATE_Config • DR MU LL_RTC_DATE_Config • DR DT LL_RTC_DATE_Config • DR DU LL_RTC_DATE_Config • DR YT LL_RTC_DATE_Config • DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_Get (RTC_TypeDef * RTCx)**
Function description	Get date (WeekDay, Day, Month and Year) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_YEAR, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.
Reference Manual to LL API cross reference:	• DR WDU LL_RTC_DATE_Get • DR MT LL_RTC_DATE_Get • DR MU LL_RTC_DATE_Get

- DR DT LL_RTC_DATE_Get
- DR DU LL_RTC_DATE_Get
- DR YT LL_RTC_DATE_Get
- DR YU LL_RTC_DATE_Get

LL_RTC_ALMA_Enable

Function name	**__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)**
Function description	Enable Alarm A.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name	**__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)**
Function description	Disable Alarm A.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAE LL_RTC_ALMA_Disable

LL_RTC_ALMA_SetMask

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Specify the Alarm A masks.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: – LL_RTC_ALMA_MASK_NONE – LL_RTC_ALMA_MASK_DATEWEEKDAY – LL_RTC_ALMA_MASK_HOURS – LL_RTC_ALMA_MASK_MINUTES – LL_RTC_ALMA_MASK_SECONDS – LL_RTC_ALMA_MASK_ALL
Return values	• **None**

Reference Manual to LL API cross reference:	• ALRMAR MSK4 LL_RTC_ALMA_SetMask • ALRMAR MSK3 LL_RTC_ALMA_SetMask • ALRMAR MSK2 LL_RTC_ALMA_SetMask • ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (RTC_TypeDef * RTCx)**
Function description	Get the Alarm A masks.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be can be a combination of the following values: – LL_RTC_ALMA_MASK_NONE – LL_RTC_ALMA_MASK_DATEWEEKDAY – LL_RTC_ALMA_MASK_HOURS – LL_RTC_ALMA_MASK_MINUTES – LL_RTC_ALMA_MASK_SECONDS – LL_RTC_ALMA_MASK_ALL
Reference Manual to LL API cross reference:	• ALRMAR MSK4 LL_RTC_ALMA_GetMask • ALRMAR MSK3 LL_RTC_ALMA_GetMask • ALRMAR MSK2 LL_RTC_ALMA_GetMask • ALRMAR MSK1 LL_RTC_ALMA_GetMask

LL_RTC_ALMA_EnableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)**
Function description	Enable AlarmA Week day selection (DU[3:0] represents the week day.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)**
Function description	Disable AlarmA Week day selection (DU[3:0] represents the date)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)**
Function description	Set ALARM A Day in BCD format.
Parameters	• **RTCx:** RTC Instance • **Day:** Value between Min_Data=0x01 and Max_Data=0x31
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR DT LL_RTC_ALMA_SetDay • ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR DT LL_RTC_ALMA_GetDay • ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)**
Function description	Set ALARM A Weekday.
Parameters	• **RTCx:** RTC Instance • **WeekDay:** This parameter can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Weekday.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Reference Manual to LL API cross reference:	• ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)**
Function description	Set Alarm A time format (AM/24-hour or PM notation)
Parameters	• **RTCx:** RTC Instance • **TimeFormat:** This parameter can be one of the following values: – LL_RTC_ALMA_TIME_FORMAT_AM – LL_RTC_ALMA_TIME_FORMAT_PM
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (RTC_TypeDef * RTCx)**
Function description	Get Alarm A time format (AM or PM notation)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALMA_TIME_FORMAT_AM – LL_RTC_ALMA_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	• ALRMAR PM LL_RTC_ALMA_GetTimeFormat

LL_RTC_ALMA_SetHour

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)**
Function description	Set ALARM A Hours in BCD format.
Parameters	• **RTCx:** RTC Instance • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR HT LL_RTC_ALMA_SetHour • ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR HT LL_RTC_ALMA_GetHour • ALRMAR HU LL_RTC_ALMA_GetHour

LL_RTC_ALMA_SetMinute

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)**
Function description	Set ALARM A Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR MNT LL_RTC_ALMA_SetMinute • ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR MNT LL_RTC_ALMA_GetMinute • ALRMAR MNU LL_RTC_ALMA_GetMinute

LL_RTC_ALMA_SetSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)**
Function description	Set ALARM A Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR ST LL_RTC_ALMA_SetSecond • ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR ST LL_RTC_ALMA_GetSecond • ALRMAR SU LL_RTC_ALMA_GetSecond

LL_RTC_ALMA_ConfigTime

Function name	**__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)**

Function description	Set Alarm A Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance • **Format12_24:** This parameter can be one of the following values: – LL_RTC_ALMA_TIME_FORMAT_AM – LL_RTC_ALMA_TIME_FORMAT_PM • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23 • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59 • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMAR PM LL_RTC_ALMA_ConfigTime • ALRMAR HT LL_RTC_ALMA_ConfigTime • ALRMAR HU LL_RTC_ALMA_ConfigTime • ALRMAR MNT LL_RTC_ALMA_ConfigTime • ALRMAR MNU LL_RTC_ALMA_ConfigTime • ALRMAR ST LL_RTC_ALMA_ConfigTime • ALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of hours, minutes and seconds.
Notes	• helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.
Reference Manual to LL API cross reference:	• ALRMAR HT LL_RTC_ALMA_GetTime • ALRMAR HU LL_RTC_ALMA_GetTime • ALRMAR MNT LL_RTC_ALMA_GetTime • ALRMAR MNU LL_RTC_ALMA_GetTime • ALRMAR ST LL_RTC_ALMA_GetTime • ALRMAR SU LL_RTC_ALMA_GetTime

LL_RTC_ALMA_SetSubSecondMask

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Set Alarm A Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance • **Mask:** Value between Min_Data=0x00 and Max_Data=0xF
Return values	• **None**
Notes	• This register can be written only when ALRAE is reset in

		RTC_CR register, or in initialization mode.
Reference Manual to LL API cross reference:	•	ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (RTC_TypeDef * RTCx)**
Function description	Get Alarm A Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xF
Reference Manual to LL API cross reference:	• ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetSubSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)**
Function description	Set Alarm A Sub seconds value.
Parameters	• **RTCx:** RTC Instance • **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (RTC_TypeDef * RTCx)**
Function description	Get Alarm A Sub seconds value.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x7FFF
Reference Manual to LL API cross reference:	• ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name	**__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)**
Function description	Enable Alarm B.
Parameters	• **RTCx:** RTC Instance

Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBE LL_RTC_ALMB_Enable

LL_RTC_ALMB_Disable

Function name	**__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)**
Function description	Disable Alarm B.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBE LL_RTC_ALMB_Disable

LL_RTC_ALMB_SetMask

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Specify the Alarm B masks.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: – LL_RTC_ALMB_MASK_NONE – LL_RTC_ALMB_MASK_DATEWEEKDAY – LL_RTC_ALMB_MASK_HOURS – LL_RTC_ALMB_MASK_MINUTES – LL_RTC_ALMB_MASK_SECONDS – LL_RTC_ALMB_MASK_ALL
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMBR MSK4 LL_RTC_ALMB_SetMask • ALRMBR MSK3 LL_RTC_ALMB_SetMask • ALRMBR MSK2 LL_RTC_ALMB_SetMask • ALRMBR MSK1 LL_RTC_ALMB_SetMask

LL_RTC_ALMB_GetMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask (RTC_TypeDef * RTCx)**
Function description	Get the Alarm B masks.
Parameters	• **RTCx:** RTC Instance

Return values	• **Returned:** value can be can be a combination of the following values: – LL_RTC_ALMB_MASK_NONE – LL_RTC_ALMB_MASK_DATEWEEKDAY – LL_RTC_ALMB_MASK_HOURS – LL_RTC_ALMB_MASK_MINUTES – LL_RTC_ALMB_MASK_SECONDS – LL_RTC_ALMB_MASK_ALL
Reference Manual to LL API cross reference:	• ALRMBR MSK4 LL_RTC_ALMB_GetMask • ALRMBR MSK3 LL_RTC_ALMB_GetMask • ALRMBR MSK2 LL_RTC_ALMB_GetMask • ALRMBR MSK1 LL_RTC_ALMB_GetMask

LL_RTC_ALMB_EnableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMB_EnableWeekday (RTC_TypeDef * RTCx)**
Function description	Enable AlarmB Week day selection (DU[3:0] represents the week day.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)**
Function description	Disable AlarmB Week day selection (DU[3:0] represents the date)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)**
Function description	Set ALARM B Day in BCD format.
Parameters	• **RTCx:** RTC Instance • **Day:** Value between Min_Data=0x01 and Max_Data=0x31
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:	• ALRMBR DT LL_RTC_ALMB_SetDay • ALRMBR DU LL_RTC_ALMB_SetDay

LL_RTC_ALMB_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR DT LL_RTC_ALMB_GetDay • ALRMBR DU LL_RTC_ALMB_GetDay

LL_RTC_ALMB_SetWeekDay

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)**
Function description	Set ALARM B Weekday.
Parameters	• **RTCx:** RTC Instance • **WeekDay:** This parameter can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Weekday.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY

  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:	• ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)**
Function description	Set ALARM B time format (AM/24-hour or PM notation)
Parameters	• **RTCx:** RTC Instance • **TimeFormat:** This parameter can be one of the following values: – LL_RTC_ALMB_TIME_FORMAT_AM – LL_RTC_ALMB_TIME_FORMAT_PM
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (RTC_TypeDef * RTCx)**
Function description	Get ALARM B time format (AM or PM notation)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALMB_TIME_FORMAT_AM – LL_RTC_ALMB_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	• ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)**
Function description	Set ALARM B Hours in BCD format.
Parameters	• **RTCx:** RTC Instance • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format
Reference Manual to	• ALRMBR HT LL_RTC_ALMB_SetHour

LL API cross reference:	• ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR HT LL_RTC_ALMB_GetHour • ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)**
Function description	Set ALARM B Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance • **Minutes:** between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMBR MNT LL_RTC_ALMB_SetMinute • ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR MNT LL_RTC_ALMB_GetMinute • ALRMBR MNU LL_RTC_ALMB_GetMinute

LL_RTC_ALMB_SetSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)**
Function description	Set ALARM B Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMBR ST LL_RTC_ALMB_SetSecond • ALRMBR SU LL_RTC_ALMB_SetSecond

LL_RTC_ALMB_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR ST LL_RTC_ALMB_GetSecond • ALRMBR SU LL_RTC_ALMB_GetSecond

LL_RTC_ALMB_ConfigTime

Function name	**__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)**
Function description	Set Alarm B Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance • **Format12_24:** This parameter can be one of the following values: – LL_RTC_ALMB_TIME_FORMAT_AM – LL_RTC_ALMB_TIME_FORMAT_PM • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23 • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59 • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None**
Reference Manual to	• ALRMBR PM LL_RTC_ALMB_ConfigTime

LL API cross reference:	• ALRMBR HT LL_RTC_ALMB_ConfigTime • ALRMBR HU LL_RTC_ALMB_ConfigTime • ALRMBR MNT LL_RTC_ALMB_ConfigTime • ALRMBR MNU LL_RTC_ALMB_ConfigTime • ALRMBR ST LL_RTC_ALMB_ConfigTime • ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of hours, minutes and seconds.
Notes	• helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.
Reference Manual to LL API cross reference:	• ALRMBR HT LL_RTC_ALMB_GetTime • ALRMBR HU LL_RTC_ALMB_GetTime • ALRMBR MNT LL_RTC_ALMB_GetTime • ALRMBR MNU LL_RTC_ALMB_GetTime • ALRMBR ST LL_RTC_ALMB_GetTime • ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Set Alarm B Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance • **Mask:** Value between Min_Data=0x00 and Max_Data=0xF
Return values	• **None**
Notes	• This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.
Reference Manual to LL API cross reference:	• ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask

LL_RTC_ALMB_GetSubSecondMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecondMask (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xF
Reference Manual to	• ALRMBSSR MASKSS LL_RTC_ALMB_GetSubSecondMask

LL API cross reference:

LL_RTC_ALMB_SetSubSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)**
Function description	Set Alarm B Sub seconds value.
Parameters	• **RTCx:** RTC Instance • **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF
Return values	• **None**
Reference Manual to LL API cross reference:	• ALRMBSSR SS LL_RTC_ALMB_SetSubSecond

LL_RTC_ALMB_GetSubSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Sub seconds value.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x7FFF
Reference Manual to LL API cross reference:	• ALRMBSSR SS LL_RTC_ALMB_GetSubSecond

LL_RTC_TS_Enable

Function name	**__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)**
Function description	Enable Timestamp.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR TSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name	**__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)**
Function description	Disable Timestamp.
Parameters	• **RTCx:** RTC Instance

Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• 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_GetActiveEdge

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	• **Returned:** value can be one of the following values: – LL_RTC_TS_TIME_FORMAT_AM – LL_RTC_TS_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	• TSTR PM LL_RTC_TS_GetTimeFormat

LL_RTC_TS_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format
Reference Manual to LL API cross reference:	• TSTR HT LL_RTC_TS_GetHour • TSTR HU LL_RTC_TS_GetHour

LL_RTC_TS_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format
Reference Manual to LL API cross reference:	• TSTR MNT LL_RTC_TS_GetMinute • TSTR MNU LL_RTC_TS_GetMinute

LL_RTC_TS_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross	• TSTR ST LL_RTC_TS_GetSecond • TSTR SU LL_RTC_TS_GetSecond

reference:

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	• **Returned:** value can be one of the following values: – LL_RTC_MONTH_JANUARY – LL_RTC_MONTH_FEBRUARY – LL_RTC_MONTH_MARCH

- – LL_RTC_MONTH_APRIL
- – LL_RTC_MONTH_MAY
- – LL_RTC_MONTH_JUNE
- – LL_RTC_MONTH_JULY
- – LL_RTC_MONTH_AUGUST
- – LL_RTC_MONTH_SEPTEMBER
- – LL_RTC_MONTH_OCTOBER
- – LL_RTC_MONTH_NOVEMBER
- – LL_RTC_MONTH_DECEMBER

Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format
Reference Manual to LL API cross reference:	• TSDR MT LL_RTC_TS_GetMonth • TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	• TSDR DT LL_RTC_TS_GetDay • TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (RTC_TypeDef * RTCx)**
Function description	Get Timestamp date (WeekDay, Day and Month) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of Weekday, Day and Month
Notes	• helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.
Reference Manual to LL API cross reference:	• 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:	• TAMPCR 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:	• TAMPCR 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	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_1 (*) – LL_RTC_TAMPER_2 – LL_RTC_TAMPER_3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1E LL_RTC_TAMPER_Enable • TAMPCR TAMP2E LL_RTC_TAMPER_Enable • TAMPCR TAMP3E LL_RTC_TAMPER_Enable

LL_RTC_TAMPER_Disable

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_Disable (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Clear RTC_TAMPx input detection.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_1 (*) – LL_RTC_TAMPER_2 – LL_RTC_TAMPER_3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1E LL_RTC_TAMPER_Disable • TAMPCR TAMP2E LL_RTC_TAMPER_Disable • TAMPCR TAMP3E LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_EnableMask

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnableMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Enable Tamper mask flag.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_MASK_TAMPER1 (*) – LL_RTC_TAMPER_MASK_TAMPER2 – LL_RTC_TAMPER_MASK_TAMPER3 (*)
Return values	• **None**
Notes	• Associated Tamper IT must not enabled when tamper mask is set.
Reference Manual to LL API cross reference:	• TAMPCR TAMP1MF LL_RTC_TAMPER_EnableMask • TAMPCR TAMP2MF LL_RTC_TAMPER_EnableMask • TAMPCR TAMP3MF LL_RTC_TAMPER_EnableMask

LL_RTC_TAMPER_DisableMask

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisableMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Disable Tamper mask flag.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_MASK_TAMPER1 (*) – LL_RTC_TAMPER_MASK_TAMPER2 – LL_RTC_TAMPER_MASK_TAMPER3 (*)
Return values	• **None**
Reference Manual to	• TAMPCR TAMP1MF LL_RTC_TAMPER_DisableMask

LL API cross reference:	• TAMPCR TAMP2MF LL_RTC_TAMPER_DisableMask • TAMPCR TAMP3MF LL_RTC_TAMPER_DisableMask

LL_RTC_TAMPER_EnableEraseBKP

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Enable backup register erase after Tamper event detection.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_NOERASE_TAMPER1 (*) – LL_RTC_TAMPER_NOERASE_TAMPER2 – LL_RTC_TAMPER_NOERASE_TAMPER3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1NOERASE LL_RTC_TAMPER_EnableEraseBKP • TAMPCR TAMP2NOERASE LL_RTC_TAMPER_EnableEraseBKP • TAMPCR TAMP3NOERASE LL_RTC_TAMPER_EnableEraseBKP

LL_RTC_TAMPER_DisableEraseBKP

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Disable backup register erase after Tamper event detection.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_NOERASE_TAMPER1 (*) – LL_RTC_TAMPER_NOERASE_TAMPER2 – LL_RTC_TAMPER_NOERASE_TAMPER3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1NOERASE LL_RTC_TAMPER_DisableEraseBKP • TAMPCR TAMP2NOERASE LL_RTC_TAMPER_DisableEraseBKP • TAMPCR TAMP3NOERASE LL_RTC_TAMPER_DisableEraseBKP

LL_RTC_TAMPER_DisablePullUp

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (RTC_TypeDef * RTCx)**
Function description	Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)
Parameters	• **RTCx:** RTC Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (RTC_TypeDef * RTCx)**
Function description	Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp

LL_RTC_TAMPER_SetPrecharge

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (RTC_TypeDef * RTCx, uint32_t Duration)**
Function description	Set RTC_TAMPx precharge duration.
Parameters	• **RTCx:** RTC Instance • **Duration:** This parameter can be one of the following values: – LL_RTC_TAMPER_DURATION_1RTCCLK – LL_RTC_TAMPER_DURATION_2RTCCLK – LL_RTC_TAMPER_DURATION_4RTCCLK – LL_RTC_TAMPER_DURATION_8RTCCLK
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge

LL_RTC_TAMPER_GetPrecharge

Function name	**__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMPx precharge duration.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TAMPER_DURATION_1RTCCLK – LL_RTC_TAMPER_DURATION_2RTCCLK – LL_RTC_TAMPER_DURATION_4RTCCLK – LL_RTC_TAMPER_DURATION_8RTCCLK
Reference Manual to LL API cross reference:	• TAMPCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge

LL_RTC_TAMPER_SetFilterCount

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (RTC_TypeDef * RTCx, uint32_t FilterCount)**
Function description	Set RTC_TAMPx filter count.
Parameters	• **RTCx:** RTC Instance • **FilterCount:** This parameter can be one of the following values: – LL_RTC_TAMPER_FILTER_DISABLE – LL_RTC_TAMPER_FILTER_2SAMPLE – LL_RTC_TAMPER_FILTER_4SAMPLE – LL_RTC_TAMPER_FILTER_8SAMPLE
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMPFLT LL_RTC_TAMPER_SetFilterCount

LL_RTC_TAMPER_GetFilterCount

Function name	**__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMPx filter count.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TAMPER_FILTER_DISABLE – LL_RTC_TAMPER_FILTER_2SAMPLE – LL_RTC_TAMPER_FILTER_4SAMPLE – LL_RTC_TAMPER_FILTER_8SAMPLE
Reference Manual to LL API cross reference:	• TAMPCR TAMPFLT LL_RTC_TAMPER_GetFilterCount

LL_RTC_TAMPER_SetSamplingFreq

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (RTC_TypeDef * RTCx, uint32_t SamplingFreq)**
Function description	Set Tamper sampling frequency.
Parameters	• **RTCx:** RTC Instance • **SamplingFreq:** This parameter can be one of the following values: – LL_RTC_TAMPER_SAMPLFREQDIV_32768 – LL_RTC_TAMPER_SAMPLFREQDIV_16384 – LL_RTC_TAMPER_SAMPLFREQDIV_8192 – LL_RTC_TAMPER_SAMPLFREQDIV_4096 – LL_RTC_TAMPER_SAMPLFREQDIV_2048 – LL_RTC_TAMPER_SAMPLFREQDIV_1024 – LL_RTC_TAMPER_SAMPLFREQDIV_512 – LL_RTC_TAMPER_SAMPLFREQDIV_256

Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name	**__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (RTC_TypeDef * RTCx)**
Function description	Get Tamper sampling frequency.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TAMPER_SAMPLFREQDIV_32768 – LL_RTC_TAMPER_SAMPLFREQDIV_16384 – LL_RTC_TAMPER_SAMPLFREQDIV_8192 – LL_RTC_TAMPER_SAMPLFREQDIV_4096 – LL_RTC_TAMPER_SAMPLFREQDIV_2048 – LL_RTC_TAMPER_SAMPLFREQDIV_1024 – LL_RTC_TAMPER_SAMPLFREQDIV_512 – LL_RTC_TAMPER_SAMPLFREQDIV_256
Reference Manual to LL API cross reference:	• TAMPCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Enable Active level for Tamper input.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_ACTIVELEVEL_TAMP1 (*) – LL_RTC_TAMPER_ACTIVELEVEL_TAMP2 – LL_RTC_TAMPER_ACTIVELEVEL_TAMP3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel • TAMPCR TAMP2TRG LL_RTC_TAMPER_EnableActiveLevel • TAMPCR TAMP3TRG LL_RTC_TAMPER_EnableActiveLevel

LL_RTC_TAMPER_DisableActiveLevel

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Disable Active level for Tamper input.

Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_RTC_TAMPER_ACTIVELEVEL_TAMP1 (*) – LL_RTC_TAMPER_ACTIVELEVEL_TAMP2 – LL_RTC_TAMPER_ACTIVELEVEL_TAMP3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel • TAMPCR TAMP2TRG LL_RTC_TAMPER_DisableActiveLevel • TAMPCR 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	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name	**__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)**
Function description	Select Wakeup clock.
Parameters	• **RTCx:** RTC Instance • **WakeupClock:** This parameter can be one of the following values: – LL_RTC_WAKEUPCLOCK_DIV_16 – LL_RTC_WAKEUPCLOCK_DIV_8 – LL_RTC_WAKEUPCLOCK_DIV_4 – LL_RTC_WAKEUPCLOCK_DIV_2 – LL_RTC_WAKEUPCLOCK_CKSPRE – LL_RTC_WAKEUPCLOCK_CKSPRE_WUT
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ISR WUTWF bit = 1
Reference Manual to LL API cross reference:	• CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name	**__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (RTC_TypeDef * RTCx)**
Function description	Get Wakeup clock.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WAKEUPCLOCK_DIV_16 – LL_RTC_WAKEUPCLOCK_DIV_8 – LL_RTC_WAKEUPCLOCK_DIV_4 – LL_RTC_WAKEUPCLOCK_DIV_2 – LL_RTC_WAKEUPCLOCK_CKSPRE – LL_RTC_WAKEUPCLOCK_CKSPRE_WUT
Reference Manual to LL API cross reference:	• CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name	**__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload**

	(RTC_TypeDef * RTCx, uint32_t Value)
Function description	Set Wakeup auto-reload value.
Parameters	• **RTCx:** RTC Instance • **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None**
Notes	• Bit can be written only when WUTWF is set to 1 in RTC_ISR
Reference Manual to LL API cross reference:	• WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name	**__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (RTC_TypeDef * RTCx)**
Function description	Get Wakeup auto-reload value.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BAK_SetRegister

Function name	**__STATIC_INLINE void LL_RTC_BAK_SetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)**
Function description	Writes a data in a specified RTC Backup data register.
Parameters	• **RTCx:** RTC Instance • **BackupRegister:** This parameter can be one of the following values: – LL_RTC_BKP_DR0 – LL_RTC_BKP_DR1 – LL_RTC_BKP_DR2 – LL_RTC_BKP_DR3 – LL_RTC_BKP_DR4 • **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• BKPxR BKP LL_RTC_BAK_SetRegister

LL_RTC_BAK_GetRegister

Function name	**__STATIC_INLINE uint32_t LL_RTC_BAK_GetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister)**

Function description	Reads data from the specified RTC Backup data Register.
Parameters	• **RTCx:** RTC Instance • **BackupRegister:** This parameter can be one of the following values: – LL_RTC_BKP_DR0 – LL_RTC_BKP_DR1 – LL_RTC_BKP_DR2 – LL_RTC_BKP_DR3 – LL_RTC_BKP_DR4
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• BKPxR BKP LL_RTC_BAK_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)**
Function description	Set Calibration output frequency (1 Hz or 512 Hz)
Parameters	• **RTCx:** RTC Instance • **Frequency:** This parameter can be one of the following values: – LL_RTC_CALIB_OUTPUT_NONE – LL_RTC_CALIB_OUTPUT_1HZ – LL_RTC_CALIB_OUTPUT_512HZ
Return values	• **None**
Notes	• Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR COE LL_RTC_CAL_SetOutputFreq • CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name	**__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (RTC_TypeDef * RTCx)**
Function description	Get Calibration output frequency (1 Hz or 512 Hz)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_CALIB_OUTPUT_NONE – LL_RTC_CALIB_OUTPUT_1HZ – LL_RTC_CALIB_OUTPUT_512HZ
Reference Manual to LL API cross reference:	• CR COE LL_RTC_CAL_GetOutputFreq • CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)**
Function description	Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)
Parameters	• **RTCx:** RTC Instance • **Pulse:** This parameter can be one of the following values: – LL_RTC_CALIB_INSERTPULSE_NONE – LL_RTC_CALIB_INSERTPULSE_SET
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • Bit can be written only when RECALPF is set to 0 in RTC_ISR
Reference Manual to LL API cross reference:	• CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name	**__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (RTC_TypeDef * RTCx)**
Function description	Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CALR CALP LL_RTC_CAL_IsPulseInserted

LL_RTC_CAL_SetPeriod

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)**
Function description	Set the calibration cycle period.
Parameters	• **RTCx:** RTC Instance • **Period:** This parameter can be one of the following values: – LL_RTC_CALIB_PERIOD_32SEC – LL_RTC_CALIB_PERIOD_16SEC – LL_RTC_CALIB_PERIOD_8SEC
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • Bit can be written only when RECALPF is set to 0 in RTC_ISR
Reference Manual to	• CALR CALW8 LL_RTC_CAL_SetPeriod

LL API cross reference:	• 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_GetPeriod • CALR CALW16 LL_RTC_CAL_GetPeriod

LL_RTC_CAL_SetMinus

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)**
Function description	Set 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_GetMinus

LL_RTC_IsActiveFlag_RECALP

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (RTC_TypeDef * RTCx)**
Function description	Get Recalibration pending Flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TAMP3

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMP3 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_TAMP2

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMP2 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP1

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMP1 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TSOV

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (RTC_TypeDef * RTCx)**
Function description	Get Time-stamp overflow flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (RTC_TypeDef * RTCx)**
Function description	Get Time-stamp flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (RTC_TypeDef * RTCx)**
Function description	Get Wakeup timer flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRB

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (RTC_TypeDef * RTCx)**
Function description	Get Alarm B flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ALRBF LL_RTC_IsActiveFlag_ALRB

LL_RTC_IsActiveFlag_ALRA

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (RTC_TypeDef * RTCx)**
Function description	Get Alarm A flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_ClearFlag_TAMP3

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Clear RTC_TAMP3 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR TAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_TAMP2

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Clear RTC_TAMP2 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR TAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP1

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Clear RTC_TAMP1 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR TAMP1F LL_RTC_ClearFlag_TAMP1

LL_RTC_ClearFlag_TSOV

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)**
Function description	Clear Time-stamp overflow flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR TSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)**
Function description	Clear Time-stamp flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR TSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)**
Function description	Clear Wakeup timer flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR WUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRB

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)**
Function description	Clear Alarm B flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR ALRBF LL_RTC_ClearFlag_ALRB

LL_RTC_ClearFlag_ALRA

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)**
Function description	Clear Alarm A flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR ALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_IsActiveFlag_INIT

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (RTC_TypeDef * RTCx)**
Function description	Get Initialization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (RTC_TypeDef * RTCx)**
Function description	Get Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)**
Function description	Clear Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ISR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (RTC_TypeDef * RTCx)**
Function description	Get Initialization status flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (RTC_TypeDef * RTCx)**
Function description	Get Shift operation pending flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (RTC_TypeDef * RTCx)**
Function description	Get Wakeup timer write flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRBW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBW (RTC_TypeDef * RTCx)**
Function description	Get Alarm B write flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ALRBWF LL_RTC_IsActiveFlag_ALRBW

LL_RTC_IsActiveFlag_ALRAW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW (RTC_TypeDef * RTCx)**
Function description	Get Alarm A write flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ALRAWF LL_RTC_IsActiveFlag_ALRAW

LL_RTC_EnableIT_TS

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)**
Function description	Enable Time-stamp interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR TSIE LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)**
Function description	Disable Time-stamp interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• 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_ALRB

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)**
Function description	Enable Alarm B interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBIE LL_RTC_EnableIT_ALRB

LL_RTC_DisableIT_ALRB

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)**
Function description	Disable Alarm B interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBIE LL_RTC_DisableIT_ALRB

LL_RTC_EnableIT_ALRA

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)**
Function description	Enable Alarm A interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAIE LL_RTC_EnableIT_ALRA

LL_RTC_DisableIT_ALRA

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)**
Function description	Disable Alarm A interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_EnableIT_TAMP3

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Enable Tamper 3 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP3IE LL_RTC_EnableIT_TAMP3

LL_RTC_DisableIT_TAMP3

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Disable Tamper 3 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to	• TAMPCR TAMP3IE LL_RTC_DisableIT_TAMP3

LL API cross reference:

LL_RTC_EnableIT_TAMP2

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Enable Tamper 2 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP2IE LL_RTC_EnableIT_TAMP2

LL_RTC_DisableIT_TAMP2

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Disable Tamper 2 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP2IE LL_RTC_DisableIT_TAMP2

LL_RTC_EnableIT_TAMP1

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Enable Tamper 1 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1IE LL_RTC_EnableIT_TAMP1

LL_RTC_DisableIT_TAMP1

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Disable Tamper 1 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross	• TAMPCR TAMP1IE LL_RTC_DisableIT_TAMP1

reference:	

LL_RTC_EnableIT_TAMP

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP (RTC_TypeDef * RTCx)**
Function description	Enable all Tamper Interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMPIE LL_RTC_EnableIT_TAMP

LL_RTC_DisableIT_TAMP

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP (RTC_TypeDef * RTCx)**
Function description	Disable all Tamper Interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• TAMPCR TAMPIE LL_RTC_DisableIT_TAMP

LL_RTC_IsEnabledIT_TS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (RTC_TypeDef * RTCx)**
Function description	Check if Time-stamp interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• 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_ALRB

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (RTC_TypeDef * RTCx)**
Function description	Check if Alarm B interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ALRBIE LL_RTC_IsEnabledIT_ALRB

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_TAMP3

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Check if Tamper 3 interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• TAMPCR TAMP3IE LL_RTC_IsEnabledIT_TAMP3

LL_RTC_IsEnabledIT_TAMP2

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Check if Tamper 2 interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• TAMPCR TAMP2IE LL_RTC_IsEnabledIT_TAMP2

LL_RTC_IsEnabledIT_TAMP1

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Check if Tamper 1 interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• TAMPCR TAMP1IE LL_RTC_IsEnabledIT_TAMP1

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:	• TAMPCR 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	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are de-initialized – ERROR: RTC registers are not de-initialized
Notes	• 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	• **RTCx:** RTC Instance • **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are initialized – ERROR: RTC registers are not initialized

Notes	• The RTC Prescaler register is write protected and can be written in initialization mode only.

LL_RTC_StructInit

Function name	**void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)**
Function description	Set each LL_RTC_InitTypeDef field to default value.
Parameters	• **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.
Return values	• **None**

LL_RTC_TIME_Init

Function name	**ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)**
Function description	Set the RTC current time.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values: – LL_RTC_FORMAT_BIN – LL_RTC_FORMAT_BCD • **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC Time register is configured – ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name	**void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)**
Function description	Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).
Parameters	• **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.
Return values	• **None**

LL_RTC_DATE_Init

Function name	**ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)**
Function description	Set the RTC current date.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values:

- – LL_RTC_FORMAT_BIN
- – LL_RTC_FORMAT_BCD
- **RTC_DateStruct:** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Day register is configured
  - ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name	**void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)**
Function description	Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)
Parameters	• **RTC_DateStruct:** pointer to a LL_RTC_DateTypeDef structure which will be initialized.
Return values	• **None**

LL_RTC_ALMA_Init

Function name	**ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**
Function description	Set the RTC Alarm A.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values: – LL_RTC_FORMAT_BIN – LL_RTC_FORMAT_BCD • **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ALARMA registers are configured – ERROR: ALARMA registers are not configured
Notes	• The Alarm register can only be written when the corresponding Alarm is disabled (Use LL_RTC_ALMA_Disable function).

LL_RTC_ALMB_Init

Function name	**ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**
Function description	Set the RTC Alarm B.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values: – LL_RTC_FORMAT_BIN

	– LL_RTC_FORMAT_BCD • **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ALARMB registers are configured – ERROR: ALARMB registers are not configured
Notes	• The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).

LL_RTC_ALMA_StructInit

Function name	**void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**
Function description	Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).
Parameters	• **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.
Return values	• **None**

LL_RTC_ALMB_StructInit

Function name	**void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**
Function description	Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).
Parameters	• **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.
Return values	• **None**

LL_RTC_EnterInitMode

Function name	**ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)**
Function description	Enters the RTC Initialization mode.
Parameters	• **RTCx:** RTC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC is in Init mode – ERROR: RTC is not in Init mode
Notes	• The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_ExitInitMode

Function name	**ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)**
Function description	Exit the RTC Initialization mode.

Parameters	• **RTCx:** RTC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC exited from in Init mode – ERROR: Not applicable
Notes	• When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles. • The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_WaitForSynchro

Function name	**ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)**
Function description	Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	• **RTCx:** RTC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are synchronised – ERROR: RTC registers are not synchronised
Notes	• The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function. • To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

71.3 RTC Firmware driver defines

71.3.1 RTC

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE	Output disabled
LL_RTC_ALARMOUT_ALMA	Alarm A output enabled
LL_RTC_ALARMOUT_ALMB	Alarm B output enabled
LL_RTC_ALARMOUT_WAKEUP	Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN	RTC_ALARM, when mapped on PC13, is open-drain output
LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL	RTC_ALARM, when mapped on PC13, is push-pull output

ALARMA MASK

LL_RTC_ALMA_MASK_NONE	No masks applied on Alarm A
LL_RTC_ALMA_MASK_DATEWEEKDAY	Date/day do not care in Alarm A comparison

LL_RTC_ALMA_MASK_HOURS	Hours do not care in Alarm A comparison
LL_RTC_ALMA_MASK_MINUTES	Minutes do not care in Alarm A comparison
LL_RTC_ALMA_MASK_SECONDS	Seconds do not care in Alarm A comparison
LL_RTC_ALMA_MASK_ALL	Masks all

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_ALMA_TIME_FORMAT_PM	PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEEKDAYSEL_DATE	Alarm A Date is selected
LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY	Alarm A WeekDay is selected

ALARMB MASK

LL_RTC_ALMB_MASK_NONE	No masks applied on Alarm B
LL_RTC_ALMB_MASK_DATEWEEKDAY	Date/day do not care in Alarm B comparison
LL_RTC_ALMB_MASK_HOURS	Hours do not care in Alarm B comparison
LL_RTC_ALMB_MASK_MINUTES	Minutes do not care in Alarm B comparison
LL_RTC_ALMB_MASK_SECONDS	Seconds do not care in Alarm B comparison
LL_RTC_ALMB_MASK_ALL	Masks all

ALARMB TIME FORMAT

LL_RTC_ALMB_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_ALMB_TIME_FORMAT_PM	PM

RTC Alarm B Date WeekDay

LL_RTC_ALMB_DATEWEEKDAYSEL_DATE	Alarm B Date is selected
LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY	Alarm B WeekDay is selected

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_INSERTPULSE_NONE	No RTCCLK pulses are added
LL_RTC_CALIB_INSERTPULSE_SET	One RTCCLK pulse is effectively inserted every 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration output

LL_RTC_CALIB_OUTPUT_NONE	Calibration output disabled
LL_RTC_CALIB_OUTPUT_1HZ	Calibration output is 512 Hz

LL_RTC_CALIB_OUTPUT_512HZ	Calibration output is 1 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC	Use a 32-second calibration cycle period
LL_RTC_CALIB_PERIOD_16SEC	Use a 16-second calibration cycle period
LL_RTC_CALIB_PERIOD_8SEC	Use a 8-second calibration cycle period

FORMAT

LL_RTC_FORMAT_BIN	Binary data format
LL_RTC_FORMAT_BCD	BCD data format

Get Flags Defines

LL_RTC_ISR_RECALPF

LL_RTC_ISR_TAMP3F

LL_RTC_ISR_TAMP2F

LL_RTC_ISR_TAMP1F

LL_RTC_ISR_TSOVF

LL_RTC_ISR_TSF

LL_RTC_ISR_WUTF

LL_RTC_ISR_ALRBF

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_ALRBWF

LL_RTC_ISR_ALRAWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR	24 hour/day format
LL_RTC_HOURFORMAT_AMPM	AM/PM hour format

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRBIE

LL_RTC_CR_ALRAIE

LL_RTC_TAMPCR_TAMP3IE

LL_RTC_TAMPCR_TAMP2IE

LL_RTC_TAMPCR_TAMP1IE

LL_RTC_TAMPCR_TAMPIE

MONTH

LL_RTC_MONTH_JANUARY	January
LL_RTC_MONTH_FEBRUARY	February
LL_RTC_MONTH_MARCH	March
LL_RTC_MONTH_APRIL	April
LL_RTC_MONTH_MAY	May
LL_RTC_MONTH_JUNE	June
LL_RTC_MONTH_JULY	July
LL_RTC_MONTH_AUGUST	August
LL_RTC_MONTH_SEPTEMBER	September
LL_RTC_MONTH_OCTOBER	October
LL_RTC_MONTH_NOVEMBER	November
LL_RTC_MONTH_DECEMBER	December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH	Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)
LL_RTC_OUTPUTPOLARITY_PIN_LOW	Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

LL_RTC_SHIFT_SECOND_ADVANCE

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_ACTIVELEVEL_TAMP1	RTC_TAMP1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event
LL_RTC_TAMPER_ACTIVELEVEL_TAMP2	RTC_TAMP2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event
LL_RTC_TAMPER_ACTIVELEVEL_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_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
LL_RTC_TAMPER_DURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
LL_RTC_TAMPER_DURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
LL_RTC_TAMPER_DURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE	Tamper filter is disabled
LL_RTC_TAMPER_FILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at the active level
LL_RTC_TAMPER_FILTER_4SAMPLE	Tamper is activated after 4 consecutive samples at the active level
LL_RTC_TAMPER_FILTER_8SAMPLE	Tamper is activated after 8 consecutive samples at the active level.

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1	Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware.The backup registers are not erased
LL_RTC_TAMPER_MASK_TAMPER2	Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.
LL_RTC_TAMPER_MASK_TAMPER3	Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased

TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_TAMPER1	Tamper 1 event does not erase the backup registers.
LL_RTC_TAMPER_NOERASE_TAMPER2	Tamper 2 event does not erase the backup registers.
LL_RTC_TAMPER_NOERASE_TAMPER3	Tamper 3 event does not erase the backup registers.

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPLFREQDIV_32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
LL_RTC_TAMPER_SAMPLFREQDIV_16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
LL_RTC_TAMPER_SAMPLFREQDIV_8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
LL_RTC_TAMPER_SAMPLFREQDIV_4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
LL_RTC_TAMPER_SAMPLFREQDIV_2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

LL_RTC_TAMPER_SAMPLFREQDIV_1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
LL_RTC_TAMPER_SAMPLFREQDIV_512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
LL_RTC_TAMPER_SAMPLFREQDIV_256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDGE_RISING	RTC_TS input rising edge generates a time-stamp event
LL_RTC_TIMESTAMP_EDGE_FALLING	RTC_TS input falling edge generates a time-stamp even

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24	AM or 24-hour format
LL_RTC_TIME_FORMAT_PM	PM

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_TS_TIME_FORMAT_PM	PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16	RTC/16 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_8	RTC/8 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_4	RTC/4 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_2	RTC/2 clock is selected
LL_RTC_WAKEUPCLOCK_CKSPRE	ck_spre (usually 1 Hz) clock is selected
LL_RTC_WAKEUPCLOCK_CKSPRE_WUT	ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY	Monday
LL_RTC_WEEKDAY_TUESDAY	Tuesday
LL_RTC_WEEKDAY_WEDNESDAY	Wednesday
LL_RTC_WEEKDAY_THURSDAY	Thrusday
LL_RTC_WEEKDAY_FRIDAY	Friday
LL_RTC_WEEKDAY_SATURDAY	Saturday
LL_RTC_WEEKDAY_SUNDAY	Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY

- LL_RTC_MONTH_JUNE
- LL_RTC_MONTH_JULY
- LL_RTC_MONTH_AUGUST
- LL_RTC_MONTH_SEPTEMBER
- LL_RTC_MONTH_OCTOBER
- LL_RTC_MONTH_NOVEMBER
- LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

Common Write and read registers Macros

LL_RTC_WriteReg

Description:

- Write a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be written

- __VALUE__: Value to be written in the register

Return value:

- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be read

Return value:

- Register: value

72 LL SPI Generic Driver

72.1 SPI Firmware driver registers structures

72.1.1 LL_SPI_InitTypeDef

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()**.

72.2 SPI Firmware driver API description

72.2.1 Detailed description of functions

LL_SPI_Enable

Function name	**__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)**
Function description	Enable SPI peripheral.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name	**__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)**
Function description	Disable SPI peripheral.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• When disabling the SPI, follow the procedure described in the Reference Manual.
Reference Manual to LL API cross reference:	• CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabled (SPI_TypeDef * SPIx)**
Function description	Check if SPI peripheral is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR1 SPE LL_SPI_IsEnabled

reference:

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	• CR2 FRF LL_SPI_SetStandard

reference:

LL_SPI_GetStandard

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetStandard (SPI_TypeDef * SPIx)**
Function description	Get serial protocol used.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_PROTOCOL_MOTOROLA – LL_SPI_PROTOCOL_TI
Reference Manual to LL API cross reference:	• CR2 FRF LL_SPI_GetStandard

LL_SPI_SetClockPhase

Function name	**__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)**
Function description	Set clock phase.
Parameters	• **SPIx:** SPI Instance • **ClockPhase:** This parameter can be one of the following values: – LL_SPI_PHASE_1EDGE – LL_SPI_PHASE_2EDGE
Return values	• **None**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 CPHA LL_SPI_SetClockPhase

LL_SPI_GetClockPhase

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (SPI_TypeDef * SPIx)**
Function description	Get clock phase.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_PHASE_1EDGE – LL_SPI_PHASE_2EDGE
Reference Manual to LL API cross reference:	• CR1 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name	**__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)**
Function description	Set clock polarity.
Parameters	• **SPIx:** SPI Instance • **ClockPolarity:** This parameter can be one of the following values: – LL_SPI_POLARITY_LOW – LL_SPI_POLARITY_HIGH
Return values	• **None**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (SPI_TypeDef * SPIx)**
Function description	Get clock polarity.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_POLARITY_LOW – LL_SPI_POLARITY_HIGH
Reference Manual to LL API cross reference:	• CR1 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetBaudRatePrescaler

Function name	**__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t BaudRate)**
Function description	Set baud rate prescaler.
Parameters	• **SPIx:** SPI Instance • **BaudRate:** This parameter can be one of the following values: – LL_SPI_BAUDRATEPRESCALER_DIV2 – LL_SPI_BAUDRATEPRESCALER_DIV4 – LL_SPI_BAUDRATEPRESCALER_DIV8 – LL_SPI_BAUDRATEPRESCALER_DIV16 – LL_SPI_BAUDRATEPRESCALER_DIV32 – LL_SPI_BAUDRATEPRESCALER_DIV64 – LL_SPI_BAUDRATEPRESCALER_DIV128 – LL_SPI_BAUDRATEPRESCALER_DIV256
Return values	• **None**

Notes	• These bits should not be changed when communication is ongoing. SPI BaudRate = fPCLK/Prescaler.
Reference Manual to LL API cross reference:	• CR1 BR LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (SPI_TypeDef * SPIx)**
Function description	Get baud rate prescaler.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_BAUDRATEPRESCALER_DIV2 – LL_SPI_BAUDRATEPRESCALER_DIV4 – LL_SPI_BAUDRATEPRESCALER_DIV8 – LL_SPI_BAUDRATEPRESCALER_DIV16 – LL_SPI_BAUDRATEPRESCALER_DIV32 – LL_SPI_BAUDRATEPRESCALER_DIV64 – LL_SPI_BAUDRATEPRESCALER_DIV128 – LL_SPI_BAUDRATEPRESCALER_DIV256
Reference Manual to LL API cross reference:	• CR1 BR LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name	**__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)**
Function description	Set transfer bit order.
Parameters	• **SPIx:** SPI Instance • **BitOrder:** This parameter can be one of the following values: – LL_SPI_LSB_FIRST – LL_SPI_MSB_FIRST
Return values	• **None**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 LSBFIRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (SPI_TypeDef * SPIx)**
Function description	Get transfer bit order.
Parameters	• **SPIx:** SPI Instance

Return values	• **Returned:** value can be one of the following values: – LL_SPI_LSB_FIRST – LL_SPI_MSB_FIRST
Reference Manual to LL API cross reference:	• CR1 LSBFIRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name	**__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)**
Function description	Set transfer direction mode.
Parameters	• **SPIx:** SPI Instance • **TransferDirection:** This parameter can be one of the following values: – LL_SPI_FULL_DUPLEX – LL_SPI_SIMPLEX_RX – LL_SPI_HALF_DUPLEX_RX – LL_SPI_HALF_DUPLEX_TX
Return values	• **None**
Notes	• For Half-Duplex mode, Rx Direction is set by default. In master mode, the MOSI pin is used and in slave mode, the MISO pin is used for Half-Duplex.
Reference Manual to LL API cross reference:	• CR1 RXONLY LL_SPI_SetTransferDirection • CR1 BIDIMODE LL_SPI_SetTransferDirection • CR1 BIDIOE LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (SPI_TypeDef * SPIx)**
Function description	Get transfer direction mode.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_FULL_DUPLEX – LL_SPI_SIMPLEX_RX – LL_SPI_HALF_DUPLEX_RX – LL_SPI_HALF_DUPLEX_TX
Reference Manual to LL API cross reference:	• CR1 RXONLY LL_SPI_GetTransferDirection • CR1 BIDIMODE LL_SPI_GetTransferDirection • CR1 BIDIOE LL_SPI_GetTransferDirection

LL_SPI_SetDataWidth

Function name	**__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)**
Function description	Set frame data width.

Parameters	• **SPIx:** SPI Instance • **DataWidth:** This parameter can be one of the following values: – LL_SPI_DATAWIDTH_8BIT – LL_SPI_DATAWIDTH_16BIT
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 DFF LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (SPI_TypeDef * SPIx)**
Function description	Get frame data width.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_DATAWIDTH_8BIT – LL_SPI_DATAWIDTH_16BIT
Reference Manual to LL API cross reference:	• CR1 DFF LL_SPI_GetDataWidth

LL_SPI_EnableCRC

Function name	**__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)**
Function description	Enable CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name	**__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)**
Function description	Disable CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (SPI_TypeDef * SPIx)**
Function description	Check if CRC is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCNext

Function name	**__STATIC_INLINE void LL_SPI_SetCRCNext (SPI_TypeDef * SPIx)**
Function description	Set CRCNext to transfer CRC on the line.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• This bit has to be written as soon as the last data is written in the SPIx_DR register.
Reference Manual to LL API cross reference:	• CR1 CRCNEXT LL_SPI_SetCRCNext

LL_SPI_SetCRCPolynomial

Function name	**__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)**
Function description	Set polynomial for CRC calculation.
Parameters	• **SPIx:** SPI Instance • **CRCPoly:** This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• CRCPR CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial**

	(SPI_TypeDef * SPIx)
Function description	Get polynomial for CRC calculation.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• CRCPR CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_GetRxCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (SPI_TypeDef * SPIx)**
Function description	Get Rx CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)**
Function description	Get Tx CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_SetNSSMode

Function name	**__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)**
Function description	Set NSS mode.
Parameters	• **SPIx:** SPI Instance • **NSS:** This parameter can be one of the following values: – LL_SPI_NSS_SOFT – LL_SPI_NSS_HARD_INPUT – LL_SPI_NSS_HARD_OUTPUT
Return values	• **None**
Notes	• LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.

Reference Manual to LL API cross reference:	• CR1 SSM LL_SPI_SetNSSMode • • CR2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (SPI_TypeDef * SPIx)**
Function description	Get NSS mode.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_NSS_SOFT – LL_SPI_NSS_HARD_INPUT – LL_SPI_NSS_HARD_OUTPUT
Reference Manual to LL API cross reference:	• CR1 SSM LL_SPI_GetNSSMode • • CR2 SSOE LL_SPI_GetNSSMode

LL_SPI_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if Rx buffer is not empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR RXNE LL_SPI_IsActiveFlag_RXNE

LL_SPI_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer is empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TXE LL_SPI_IsActiveFlag_TXE

LL_SPI_IsActiveFlag_CRCERR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (SPI_TypeDef * SPIx)**
Function description	Get CRC error flag.
Parameters	• **SPIx:** SPI Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CRCERR LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (SPI_TypeDef * SPIx)**
Function description	Get mode fault error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Get overrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_BSY

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_BSY (SPI_TypeDef * SPIx)**
Function description	Get busy flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• The BSY flag is cleared under any one of the following conditions: -When the SPI is correctly disabled -When a fault is detected in Master mode (MODF bit set to 1) -In Master mode, when it finishes a data transmission and no new data is ready to be sent -In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.
Reference Manual to LL API cross reference:	• SR BSY LL_SPI_IsActiveFlag_BSY

LL_SPI_IsActiveFlag_FRE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Get frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR FRE LL_SPI_IsActiveFlag_FRE

LL_SPI_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	• SR OVR LL_SPI_ClearFlag_OVR

LL API cross reference:	

LL_SPI_ClearFlag_FRE

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Clear frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• Clearing this flag is done by reading SPIx_SR register
Reference Manual to LL API cross reference:	• SR FRE LL_SPI_ClearFlag_FRE

LL_SPI_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Enable error interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_EnableIT_ERR

LL_SPI_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Enable Rx buffer not empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_EnableIT_RXNE

LL_SPI_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Enable Tx buffer empty interrupt.

Parameters	•	**SPIx:** SPI Instance
Return values	•	**None**
Reference Manual to LL API cross reference:	•	CR2 TXEIE LL_SPI_EnableIT_TXE

LL_SPI_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Disable error interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_DisableIT_ERR

LL_SPI_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Disable Rx buffer not empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_DisableIT_RXNE

LL_SPI_DisableIT_TXE

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Disable Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_DisableIT_TXE

LL_SPI_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_ERR (SPI_TypeDef * SPIx)**

Function description	Check if error interrupt is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_IsEnabledIT_ERR

LL_SPI_IsEnabledIT_RXNE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if Rx buffer not empty interrupt is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_IsEnabledIT_RXNE

LL_SPI_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_IsEnabledIT_TXE

LL_SPI_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_EnableDMAReq_RX

LL_SPI_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Rx.

Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Rx is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Tx is enabled.
Parameters	• **SPIx:** SPI Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr (SPI_TypeDef * SPIx)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **SPIx:** SPI Instance
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• DR DR LL_SPI_DMA_GetRegAddr

LL_SPI_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)**
Function description	Read 8-Bits in the data register.
Parameters	• **SPIx:** SPI Instance
Return values	• **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• DR DR LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name	**__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)**
Function description	Read 16-Bits in the data register.
Parameters	• **SPIx:** SPI Instance
Return values	• **RxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• DR DR LL_SPI_ReceiveData16

LL_SPI_TransmitData8

Function name	**__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)**
Function description	Write 8-Bits in the data register.
Parameters	• **SPIx:** SPI Instance

	• **TxData:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DR DR LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name	**__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)**
Function description	Write 16-Bits in the data register.
Parameters	• **SPIx:** SPI Instance • **TxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• DR DR LL_SPI_TransmitData16

LL_SPI_DeInit

Function name	**ErrorStatus LL_SPI_DeInit (SPI_TypeDef * SPIx)**
Function description	De-initialize the SPI registers to their default reset values.
Parameters	• **SPIx:** SPI Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: SPI registers are de-initialized – ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name	**ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)**
Function description	Initialize the SPI registers according to the specified parameters in SPI_InitStruct.
Parameters	• **SPIx:** SPI Instance • **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value. (Return always SUCCESS)
Notes	• As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI 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**

72.3 SPI Firmware driver defines

72.3.1 SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_DIV2	BaudRate control equal to fPCLK/2
LL_SPI_BAUDRATEPRESCALER_DIV4	BaudRate control equal to fPCLK/4
LL_SPI_BAUDRATEPRESCALER_DIV8	BaudRate control equal to fPCLK/8
LL_SPI_BAUDRATEPRESCALER_DIV16	BaudRate control equal to fPCLK/16
LL_SPI_BAUDRATEPRESCALER_DIV32	BaudRate control equal to fPCLK/32
LL_SPI_BAUDRATEPRESCALER_DIV64	BaudRate control equal to fPCLK/64
LL_SPI_BAUDRATEPRESCALER_DIV128	BaudRate control equal to fPCLK/128
LL_SPI_BAUDRATEPRESCALER_DIV256	BaudRate control equal to fPCLK/256

Transmission Bit Order

LL_SPI_LSB_FIRST	Data is transmitted/received with the LSB first
LL_SPI_MSB_FIRST	Data is transmitted/received with the MSB first

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE	CRC calculation disabled
LL_SPI_CRCCALCULATION_ENABLE	CRC calculation enabled

Datawidth

LL_SPI_DATAWIDTH_8BIT	Data length for SPI transfer: 8 bits
LL_SPI_DATAWIDTH_16BIT	Data length for SPI transfer: 16 bits

Get Flags Defines

LL_SPI_SR_RXNE	Rx buffer not empty flag
LL_SPI_SR_TXE	Tx buffer empty flag
LL_SPI_SR_BSY	Busy flag
LL_SPI_SR_UDR	Underrun flag
LL_SPI_SR_CRCERR	CRC error flag
LL_SPI_SR_MODF	Mode fault flag
LL_SPI_SR_OVR	Overrun flag
LL_SPI_SR_FRE	TI mode frame format error flag

IT Defines

LL_SPI_CR2_RXNEIE	Rx buffer not empty interrupt enable

LL_SPI_CR2_TXEIE	Tx buffer empty interrupt enable
LL_SPI_CR2_ERRIE	Error interrupt enable

Operation Mode

LL_SPI_MODE_MASTER	Master configuration
LL_SPI_MODE_SLAVE	Slave configuration

Slave Select Pin Mode

LL_SPI_NSS_SOFT	NSS managed internally. NSS pin not used and free
LL_SPI_NSS_HARD_INPUT	NSS pin used in Input. Only used in Master mode
LL_SPI_NSS_HARD_OUTPUT	NSS pin used in Output. Only used in Slave mode as chip select

Clock Phase

LL_SPI_PHASE_1EDGE	First clock transition is the first data capture edge
LL_SPI_PHASE_2EDGE	Second clock transition is the first data capture edge

Clock Polarity

LL_SPI_POLARITY_LOW	Clock to 0 when idle
LL_SPI_POLARITY_HIGH	Clock to 1 when idle

Serial Protocol

LL_SPI_PROTOCOL_MOTOROLA	Motorola mode. Used as default value
LL_SPI_PROTOCOL_TI	TI mode

Transfer Mode

LL_SPI_FULL_DUPLEX	Full-Duplex mode. Rx and Tx transfer on 2 lines
LL_SPI_SIMPLEX_RX	Simplex Rx mode. Rx transfer only on 1 line
LL_SPI_HALF_DUPLEX_RX	Half-Duplex Rx mode. Rx transfer on 1 line
LL_SPI_HALF_DUPLEX_TX	Half-Duplex Tx mode. Tx transfer on 1 line

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

73 LL SYSTEM Generic Driver

73.1 SYSTEM Firmware driver API description

73.1.1 Detailed description of functions

LL_SYSCFG_SetRemapMemory

Function name	**__STATIC_INLINE void LL_SYSCFG_SetRemapMemory (uint32_t Memory)**
Function description	Set memory mapping at address 0x00000000.
Parameters	• **Memory:** This parameter can be one of the following values: – LL_SYSCFG_REMAP_FLASH – LL_SYSCFG_REMAP_SYSTEMFLASH – LL_SYSCFG_REMAP_SRAM
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 MEM_MODE LL_SYSCFG_SetRemapMemory

LL_SYSCFG_GetRemapMemory

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetRemapMemory (void)**
Function description	Get memory mapping at address 0x00000000.
Return values	• **Returned:** value can be one of the following values: – LL_SYSCFG_REMAP_FLASH – LL_SYSCFG_REMAP_SYSTEMFLASH – LL_SYSCFG_REMAP_SRAM
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 MEM_MODE LL_SYSCFG_GetRemapMemory

LL_SYSCFG_SetFlashBankMode

Function name	**__STATIC_INLINE void LL_SYSCFG_SetFlashBankMode (uint32_t Bank)**
Function description	Select Flash bank mode (Bank flashed at 0x08000000)
Parameters	• **Bank:** This parameter can be one of the following values: – LL_SYSCFG_BANKMODE_BANK1 – LL_SYSCFG_BANKMODE_BANK2
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 UFB LL_SYSCFG_SetFlashBankMode

LL_SYSCFG_GetFlashBankMode

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashBankMode (void)**
Function description	Get Flash bank mode (Bank flashed at 0x08000000)
Return values	• **Returned:** value can be one of the following values: – LL_SYSCFG_BANKMODE_BANK1 – LL_SYSCFG_BANKMODE_BANK2
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 UFB LL_SYSCFG_GetFlashBankMode

LL_SYSCFG_GetBootMode

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetBootMode (void)**
Function description	Get Boot mode selected by the boot pins status bits.
Return values	• **Returned:** value can be one of the following values: – LL_SYSCFG_BOOTMODE_FLASH – LL_SYSCFG_BOOTMODE_SYSTEMFLASH – LL_SYSCFG_BOOTMODE_SRAM
Notes	• It indicates the boot mode selected by the boot pins. Bit 9 corresponds to the complement of nBOOT1 bit in the FLASH_OPTR register. Its value is defined in the option bytes. Bit 8 corresponds to the value sampled on the BOOT0 pin.
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 BOOT_MODE LL_SYSCFG_GetBootMode

LL_SYSCFG_EnableFirewall

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableFirewall (void)**
Function description	Firewall protection enabled.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 FWDIS LL_SYSCFG_EnableFirewall

LL_SYSCFG_IsEnabledFirewall

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledFirewall (void)**
Function description	Check if Firewall protection is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 FWDIS LL_SYSCFG_IsEnabledFirewall

LL_SYSCFG_SetVLCDRailConnection

Function name	**__STATIC_INLINE void LL_SYSCFG_SetVLCDRailConnection (uint32_t IoPinConnect)**
Function description	Set VLCD rail connection to optional external capacitor.
Parameters	• **IoPinConnect:** This parameter can be a combination of the following values: (*) value not defined in all devices – LL_SYSCFG_CAPA_VLCD1_PB12 – LL_SYSCFG_CAPA_VLCD1_PE11(*) – LL_SYSCFG_CAPA_VLCD2_PB2 – LL_SYSCFG_CAPA_VLCD3_PB0 – LL_SYSCFG_CAPA_VLCD3_PE12(*)
Return values	• **None**
Notes	• One to three external capacitors can be connected to pads to do VLCD biasing. LCD_VLCD1 rail can be connected to PB12 or PE11(*),LCD_VLCD2 rail can be connected to PB2,LCD_VLCD3 rail can be connected to PB0 or PE12(*)
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 CAPA LL_SYSCFG_SetVLCDRailConnection

LL_SYSCFG_GetVLCDRailConnection

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetVLCDRailConnection (void)**
Function description	Get VLCD rail connection configuration.
Return values	• **Returned:** value can be a combination of the following values: (*) value not defined in all devices – LL_SYSCFG_CAPA_VLCD1_PB12 – LL_SYSCFG_CAPA_VLCD1_PE11(*) – LL_SYSCFG_CAPA_VLCD2_PB2 – LL_SYSCFG_CAPA_VLCD3_PB0 – LL_SYSCFG_CAPA_VLCD3_PE12(*)
Notes	• One to three external capacitors can be connected to pads to do VLCD biasing. LCD_VLCD1 rail can be connected to PB12 or PE11(*),LCD_VLCD2 rail can be connected to PB2,LCD_VLCD3 rail can be connected to PB0 or PE12(*)
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 CAPA LL_SYSCFG_GetVLCDRailConnection

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: (*) value not defined in all devices – 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_I2C3 (*)

Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 I2C_PBx_FMP LL_SYSCFG_EnableFastModePlus • SYSCFG_CFGR2 I2Cx_FMP LL_SYSCFG_EnableFastModePlus

LL_SYSCFG_DisableFastModePlus

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableFastModePlus (uint32_t ConfigFastModePlus)**
Function description	Disable the I2C fast mode plus driving capability.
Parameters	• **ConfigFastModePlus:** This parameter can be a combination of the following values: (*) value not defined in all devices – 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_I2C3 (*)
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 I2C_PBx_FMP LL_SYSCFG_DisableFastModePlus • SYSCFG_CFGR2 I2Cx_FMP LL_SYSCFG_DisableFastModePlus

LL_SYSCFG_VREFINT_SetConnection

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_SetConnection (uint32_t IoPinConnect)**
Function description	Select which pad is connected to VREFINT_ADC.
Parameters	• **IoPinConnect:** This parameter can be one of the following values: – LL_SYSCFG_VREFINT_CONNECT_NONE – LL_SYSCFG_VREFINT_CONNECT_IO1 – LL_SYSCFG_VREFINT_CONNECT_IO2 – LL_SYSCFG_VREFINT_CONNECT_IO1_IO2
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 SEL_VREF_OUT LL_SYSCFG_VREFINT_SetConnection

LL_SYSCFG_VREFINT_GetConnection

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_VREFINT_GetConnection (void)**
Function description	Get pad connection to VREFINT_ADC.
Return values	• **Returned:** value can be one of the following values: – LL_SYSCFG_VREFINT_CONNECT_NONE – LL_SYSCFG_VREFINT_CONNECT_IO1 – LL_SYSCFG_VREFINT_CONNECT_IO2 – LL_SYSCFG_VREFINT_CONNECT_IO1_IO2
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 SEL_VREF_OUT LL_SYSCFG_VREFINT_GetConnection

LL_SYSCFG_VREFINT_EnableADC

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_EnableADC (void)**
Function description	Buffer used to generate VREFINT reference for ADC enable.
Return values	• **None**
Notes	• The VrefInit buffer to ADC through internal path is also enabled using function LL_ADC_SetCommonPathInternalCh() with parameter LL_ADC_PATH_INTERNAL_VREFINT
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENBUF_VREFINT_ADC LL_SYSCFG_VREFINT_EnableADC

LL_SYSCFG_VREFINT_DisableADC

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_DisableADC (void)**
Function description	Buffer used to generate VREFINT reference for ADC disable.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENBUF_VREFINT_ADC LL_SYSCFG_VREFINT_DisableADC

LL_SYSCFG_TEMPSENSOR_Enable

Function name	**__STATIC_INLINE void LL_SYSCFG_TEMPSENSOR_Enable (void)**
Function description	Buffer used to generate temperature sensor reference for ADC enable.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENBUF_SENSOR_ADC LL_SYSCFG_TEMPSENSOR_Enable

LL_SYSCFG_TEMPSENSOR_Disable

Function name	**__STATIC_INLINE void LL_SYSCFG_TEMPSENSOR_Disable (void)**
Function description	Buffer used to generate temperature sensor reference for ADC disable.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENBUF_SENSOR_ADC LL_SYSCFG_TEMPSENSOR_Disable

LL_SYSCFG_VREFINT_EnableCOMP

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_EnableCOMP (void)**
Function description	Buffer used to generate VREFINT reference for comparator enable.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENBUF_VREFINT_COMP LL_SYSCFG_VREFINT_EnableCOMP

LL_SYSCFG_VREFINT_DisableCOMP

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_DisableCOMP (void)**
Function description	Buffer used to generate VREFINT reference for comparator disable.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENBUF_VREFINT_COMP LL_SYSCFG_VREFINT_DisableCOMP

LL_SYSCFG_VREFINT_EnableHSI48

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_EnableHSI48 (void)**
Function description	Buffer used to generate VREFINT reference for HSI48 oscillator enable.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENREF_HSI48 LL_SYSCFG_VREFINT_EnableHSI48

LL_SYSCFG_VREFINT_DisableHSI48

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_DisableHSI48 (void)**

Function description	Buffer used to generate VREFINT reference for HSI48 oscillator disable.
Return values	• **None**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 ENREF_HSI48 LL_SYSCFG_VREFINT_DisableHSI48

LL_SYSCFG_VREFINT_IsReady

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_VREFINT_IsReady (void)**
Function description	Check if VREFINT is ready or not.
Return values	• **State:** of bit (1 or 0).
Notes	• When set, it indicates that VREFINT is available for BOR, PVD and LCD
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 VREFINT_RDYF LL_SYSCFG_VREFINT_IsReady

LL_SYSCFG_VREFINT_Lock

Function name	**__STATIC_INLINE void LL_SYSCFG_VREFINT_Lock (void)**
Function description	Lock the whole content of SYSCFG_CFGR3 register.
Return values	• **None**
Notes	• After SYSCFG_CFGR3 register lock, only read access available. Only system hardware reset unlocks SYSCFG_CFGR3 register.
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 REF_LOCK LL_SYSCFG_VREFINT_Lock

LL_SYSCFG_VREFINT_IsLocked

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_VREFINT_IsLocked (void)**
Function description	Check if SYSCFG_CFGR3 register is locked (only read access) or not.
Return values	• **State:** of bit (1 or 0).
Notes	• When set, it indicates that SYSCFG_CFGR3 register is locked, only read access available
Reference Manual to LL API cross reference:	• SYSCFG_CFGR3 REF_LOCK LL_SYSCFG_VREFINT_IsLocked

LL_SYSCFG_SetEXTISource

Function name	**__STATIC_INLINE void LL_SYSCFG_SetEXTISource (uint32_t**

	Port, uint32_t Line)
Function description	Configure source input for the EXTI external interrupt.
Parameters	• **Port:** This parameter can be one of the following values: (*) value not defined in all devices – LL_SYSCFG_EXTI_PORTA – LL_SYSCFG_EXTI_PORTB – LL_SYSCFG_EXTI_PORTC – LL_SYSCFG_EXTI_PORTD (*) – LL_SYSCFG_EXTI_PORTE (*) – LL_SYSCFG_EXTI_PORTH (*) • **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	• **None**
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_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: (*) value not defined in all devices – LL_SYSCFG_EXTI_PORTA – LL_SYSCFG_EXTI_PORTB – LL_SYSCFG_EXTI_PORTC – LL_SYSCFG_EXTI_PORTD (*) – LL_SYSCFG_EXTI_PORTE (*) – LL_SYSCFG_EXTI_PORTH (*)
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_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=0x7FF (ex: L053 -> 0x417, L073 -> 0x447)
Reference Manual to LL API cross	• DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

reference:

LL_DBGMCU_GetRevisionID

Function name	**__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)**
Function description	Return the device revision identifier.
Return values	• **Values:** between Min_Data=0x00 and Max_Data=0xFFFF
Notes	• This field indicates the revision of the device.
Reference Manual to LL API cross reference:	• DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableDBGSleepMode

Function name	**__STATIC_INLINE void LL_DBGMCU_EnableDBGSleepMode (void)**
Function description	Enable the Debug Module during SLEEP mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_SLEEP LL_DBGMCU_EnableDBGSleepMode

LL_DBGMCU_DisableDBGSleepMode

Function name	**__STATIC_INLINE void LL_DBGMCU_DisableDBGSleepMode (void)**
Function description	Disable the Debug Module during SLEEP mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_SLEEP LL_DBGMCU_DisableDBGSleepMode

LL_DBGMCU_EnableDBGStopMode

Function name	**__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void)**
Function description	Enable the Debug Module during STOP mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode

LL_DBGMCU_DisableDBGStopMode

Function name	**__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void)**

Function description	Disable the Debug Module during STOP mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode

LL_DBGMCU_EnableDBGStandbyMode

Function name	**__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void)**
Function description	Enable the Debug Module during STANDBY mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode

LL_DBGMCU_DisableDBGStandbyMode

Function name	**__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void)**
Function description	Disable the Debug Module during STANDBY mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)**
Function description	Freeze APB1 peripherals (group1 peripherals)
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices – 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_RTC_STOP – LL_DBGMCU_APB1_GRP1_WWDG_STOP – LL_DBGMCU_APB1_GRP1_IWDG_STOP – LL_DBGMCU_APB1_GRP1_I2C1_STOP – LL_DBGMCU_APB1_GRP1_I2C2_STOP (*) – LL_DBGMCU_APB1_GRP1_I2C3_STOP (*) – LL_DBGMCU_APB1_GRP1_LPTIM1_STOP
Return values	• **None**
Reference Manual to LL API cross	• APB1FZ DBG_TIM2_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_TIM3_STOP

reference:	LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_TIM6_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_TIM7_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_I2C1_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_I2C2_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_I2C3_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph • APB1FZ DBG_LPTIMER_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph

LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periphs)**
Function description	Unfreeze APB1 peripherals (group1 peripherals)
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices – 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_RTC_STOP – LL_DBGMCU_APB1_GRP1_WWDG_STOP – LL_DBGMCU_APB1_GRP1_IWDG_STOP – LL_DBGMCU_APB1_GRP1_I2C1_STOP – LL_DBGMCU_APB1_GRP1_I2C2_STOP (*) – LL_DBGMCU_APB1_GRP1_I2C3_STOP (*) – LL_DBGMCU_APB1_GRP1_LPTIM1_STOP
Return values	• **None**
Reference Manual to LL API cross reference:	• APB1FZ DBG_TIM2_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph • APB1FZ DBG_TIM3_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph • APB1FZ DBG_TIM6_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph • APB1FZ DBG_TIM7_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph • APB1FZ DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph • APB1FZ DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

- APB1FZ DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- APB1FZ DBG_I2C1_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- APB1FZ DBG_I2C2_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- APB1FZ DBG_I2C3_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- APB1FZ DBG_LPTIMER_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

LL_DBGMCU_APB2_GRP1_FreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_FreezePeriph (uint32_t Periphs)**
Function description	Freeze APB2 peripherals.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices – LL_DBGMCU_APB2_GRP1_TIM22_STOP (*) – LL_DBGMCU_APB2_GRP1_TIM21_STOP
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2FZ DBG_TIM22_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph • APB2FZ DBG_TIM21_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_UnFreezePeriph (uint32_t Periphs)**
Function description	Unfreeze APB2 peripherals.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices – LL_DBGMCU_APB2_GRP1_TIM22_STOP (*) – LL_DBGMCU_APB2_GRP1_TIM21_STOP
Return values	• **None**
Reference Manual to LL API cross reference:	• APB2FZ DBG_TIM22_STOP LL_DBGMCU_APB2_GRP1_UnFreezePeriph • APB2FZ DBG_TIM21_STOP LL_DBGMCU_APB2_GRP1_UnFreezePeriph

LL_FLASH_SetLatency

Function name	**__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)**
Function description	Set FLASH Latency.
Parameters	• **Latency:** This parameter can be one of the following values: – LL_FLASH_LATENCY_0

– LL_FLASH_LATENCY_1

Return values	• **None**
Reference Manual to LL API cross reference:	• FLASH_ACR LATENCY LL_FLASH_SetLatency

LL_FLASH_GetLatency

Function name	**__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)**
Function description	Get FLASH Latency.
Return values	• **Returned:** value can be one of the following values: – LL_FLASH_LATENCY_0 – LL_FLASH_LATENCY_1
Reference Manual to LL API cross reference:	• FLASH_ACR LATENCY LL_FLASH_GetLatency

LL_FLASH_EnablePrefetch

Function name	**__STATIC_INLINE void LL_FLASH_EnablePrefetch (void)**
Function description	Enable Prefetch.
Return values	• **None**
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTEN LL_FLASH_EnablePrefetch

LL_FLASH_DisablePrefetch

Function name	**__STATIC_INLINE void LL_FLASH_DisablePrefetch (void)**
Function description	Disable Prefetch.
Return values	• **None**
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTEN LL_FLASH_DisablePrefetch

LL_FLASH_IsPrefetchEnabled

Function name	**__STATIC_INLINE uint32_t LL_FLASH_IsPrefetchEnabled (void)**
Function description	Check if Prefetch buffer is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTEN LL_FLASH_IsPrefetchEnabled

LL_FLASH_EnableRunPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_EnableRunPowerDown (void)**
Function description	Enable Flash Power-down mode during run mode or Low-power run mode.
Return values	• **None**
Notes	• Flash memory can be put in power-down mode only when the code is executed from RAM • Flash must not be accessed when power down is enabled • Flash must not be put in power-down while a program or an erase operation is on-going
Reference Manual to LL API cross reference:	• FLASH_ACR RUN_PD LL_FLASH_EnableRunPowerDown • FLASH_PDKEYR PDKEY1 LL_FLASH_EnableRunPowerDown • FLASH_PDKEYR PDKEY2 LL_FLASH_EnableRunPowerDown

LL_FLASH_DisableRunPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_DisableRunPowerDown (void)**
Function description	Disable Flash Power-down mode during run mode or Low-power run mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• FLASH_ACR RUN_PD LL_FLASH_DisableRunPowerDown • FLASH_PDKEYR PDKEY1 LL_FLASH_DisableRunPowerDown • FLASH_PDKEYR PDKEY2 LL_FLASH_DisableRunPowerDown

LL_FLASH_EnableSleepPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_EnableSleepPowerDown (void)**
Function description	Enable Flash Power-down mode during Sleep or Low-power sleep mode.
Return values	• **None**
Notes	• Flash must not be put in power-down while a program or an erase operation is on-going
Reference Manual to LL API cross reference:	• FLASH_ACR SLEEP_PD LL_FLASH_EnableSleepPowerDown

LL_FLASH_DisableSleepPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_DisableSleepPowerDown (void)**

Function description	Disable Flash Power-down mode during Sleep or Low-power sleep mode.
Return values	• **None**
Reference Manual to LL API cross reference:	• FLASH_ACR SLEEP_PD LL_FLASH_DisableSleepPowerDown

LL_FLASH_EnableBuffers

Function name	**__STATIC_INLINE void LL_FLASH_EnableBuffers (void)**
Function description	Enable buffers used as a cache during read access.
Return values	• **None**
Reference Manual to LL API cross reference:	• FLASH_ACR DISAB_BUF LL_FLASH_EnableBuffers

LL_FLASH_DisableBuffers

Function name	**__STATIC_INLINE void LL_FLASH_DisableBuffers (void)**
Function description	Disable buffers used as a cache during read access.
Return values	• **None**
Notes	• When disabled, every read will access the NVM even for an address already read (for example, the previous address).
Reference Manual to LL API cross reference:	• FLASH_ACR DISAB_BUF LL_FLASH_DisableBuffers

LL_FLASH_EnablePreRead

Function name	**__STATIC_INLINE void LL_FLASH_EnablePreRead (void)**
Function description	Enable pre-read.
Return values	• **None**
Notes	• When enabled, the memory interface stores the last address read as data and tries to read the next one when no other read or write or prefetch operation is ongoing. It is automatically disabled every time the buffers are disabled.
Reference Manual to LL API cross reference:	• FLASH_ACR PRE_READ LL_FLASH_EnablePreRead

LL_FLASH_DisablePreRead

Function name	**__STATIC_INLINE void LL_FLASH_DisablePreRead (void)**
Function description	Disable pre-read.
Return values	• **None**

Reference Manual to LL API cross reference:	• FLASH_ACR PRE_READ LL_FLASH_DisablePreRead

73.2 SYSTEM Firmware driver defines

73.2.1 SYSTEM

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP	TIM2 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM3_STOP	TIM3 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM6_STOP	TIM6 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM7_STOP	TIM7 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_RTC_STOP	RTC Calendar frozen when core is halted
LL_DBGMCU_APB1_GRP1_WWDG_STOP	Debug Window Watchdog stopped when Core is halted
LL_DBGMCU_APB1_GRP1_IWDG_STOP	Debug Independent Watchdog stopped when Core is halted
LL_DBGMCU_APB1_GRP1_I2C1_STOP	I2C1 SMBUS timeout mode stopped when Core is halted
LL_DBGMCU_APB1_GRP1_I2C2_STOP	I2C2 SMBUS timeout mode stopped when Core is halted
LL_DBGMCU_APB1_GRP1_I2C3_STOP	I2C3 SMBUS timeout mode stopped when Core is halted
LL_DBGMCU_APB1_GRP1_LPTIM1_STOP	LPTIM1 counter stopped when core is halted

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM22_STOP	TIM22 counter stopped when core is halted
LL_DBGMCU_APB2_GRP1_TIM21_STOP	TIM21 counter stopped when core is halted

SYSCFG Bank Mode

LL_SYSCFG_BANKMODE_BANK1	Flash Bank1 mapped at 0x08000000 (and aliased at 0x00000000), Flash Bank2 mapped at 0x08018000 (and aliased at 0x00018000), Data EEPROM Bank1 mapped at 0x08080000 (and aliased at 0x00080000), Data EEPROM Bank2 mapped at 0x08080C00 (and aliased at 0x00080C00)
LL_SYSCFG_BANKMODE_BANK2	Flash Bank2 mapped at 0x08000000 (and aliased at 0x00000000), Flash Bank1 mapped at 0x08018000 (and aliased at 0x00018000), Data EEPROM Bank2 mapped at 0x08080000 (and aliased at 0x00080000), Data EEPROM Bank1 mapped at 0x08080C00 (and aliased at 0x00080C00)

SYSCFG Boot Mode

LL_SYSCFG_BOOTMODE_FLASH	Main Flash memory boot mode

LL_SYSCFG_BOOTMODE_SYSTEMFLASH	System Flash memory boot mode
LL_SYSCFG_BOOTMODE_SRAM	SRAM boot mode

SYSCFG VLCD Rail Connection

LL_SYSCFG_CAPA_VLCD2_PB2	Connect PB2 pin to LCD_VLCD2 rails supply voltage
LL_SYSCFG_CAPA_VLCD1_PB12	Connect PB12 pin to LCD_VLCD1 rails supply voltage
LL_SYSCFG_CAPA_VLCD3_PB0	Connect PB0 pin to LCD_VLCD3 rails supply voltage
LL_SYSCFG_CAPA_VLCD1_PE11	Connect PE11 pin to LCD_VLCD1 rails supply voltage
LL_SYSCFG_CAPA_VLCD3_PE12	Connect PE12 pin to LCD_VLCD3 rails supply voltage

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_PORTH	EXTI PORT H

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMODEPLUS_PB6	Enable Fast Mode Plus on PB6

LL_SYSCFG_I2C_FASTMODEPLUS_PB7	Enable Fast Mode Plus on PB7
LL_SYSCFG_I2C_FASTMODEPLUS_PB8	Enable Fast Mode Plus on PB8
LL_SYSCFG_I2C_FASTMODEPLUS_PB9	Enable Fast Mode Plus on PB9
LL_SYSCFG_I2C_FASTMODEPLUS_I2C1	Enable Fast Mode Plus on I2C1 pins
LL_SYSCFG_I2C_FASTMODEPLUS_I2C2	Enable Fast Mode Plus on I2C2 pins
LL_SYSCFG_I2C_FASTMODEPLUS_I2C3	Enable Fast Mode Plus on I2C3 pins

FLASH LATENCY

LL_FLASH_LATENCY_0	FLASH Zero Latency cycle
LL_FLASH_LATENCY_1	FLASH One Latency cycle

SYSCFG Memory Remap

LL_SYSCFG_REMAP_FLASH	Main Flash memory mapped at 0x00000000
LL_SYSCFG_REMAP_SYSTEMFLASH	System Flash memory mapped at 0x00000000
LL_SYSCFG_REMAP_SRAM	SRAM mapped at 0x00000000

SYSCFG VREFINT Control

LL_SYSCFG_VREFINT_CONNECT_NONE	No pad connected to VREFINT_ADC
LL_SYSCFG_VREFINT_CONNECT_IO1	PB0 connected to VREFINT_ADC
LL_SYSCFG_VREFINT_CONNECT_IO2	PB1 connected to VREFINT_ADC
LL_SYSCFG_VREFINT_CONNECT_IO1_IO2	PB0 and PB1 connected to VREFINT_ADC

74 LL TIM Generic Driver

74.1 TIM Firmware driver registers structures

74.1.1 LL_TIM_InitTypeDef

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***

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()**.

74.1.2 LL_TIM_OC_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***

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::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()**.

74.1.3 LL_TIM_IC_InitTypeDef

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***
  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

74.1.4 LL_TIM_ENCODER_InitTypeDef

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***
  Specifies the encoder resolution (x2 or x4). This parameter can be a value of ***TIM_LL_EC_ENCODERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of

TIM_LL_EC_IC_POLARITY.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***
  Specifies the TI1 input source This parameter can be a value of *TIM_LL_EC_ACTIVEINPUT*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. This parameter can be a value of *TIM_LL_EC_ICPSC*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of *TIM_LL_EC_IC_FILTER*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***
  Specifies the active edge of TI2 input. This parameter can be a value of *TIM_LL_EC_IC_POLARITY*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***
  Specifies the TI2 input source This parameter can be a value of *TIM_LL_EC_ACTIVEINPUT*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***
  Specifies the TI2 input prescaler value. This parameter can be a value of *TIM_LL_EC_ICPSC*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***
  Specifies the TI2 input filter. This parameter can be a value of *TIM_LL_EC_IC_FILTER*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

74.2 TIM Firmware driver API description

74.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name	**__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)**
Function description	Enable timer counter.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name	**__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)**
Function description	Disable timer counter.

Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (TIM_TypeDef * TIMx)**
Function description	Indicates whether the timer counter is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name	**__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Enable update event generation.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name	**__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Disable update event generation.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Indicates whether update event generation is enabled.
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name	**__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)**
Function description	Set update event source.
Parameters	• **TIMx:** Timer instance • **UpdateSource:** This parameter can be one of the following values: – LL_TIM_UPDATESOURCE_REGULAR – LL_TIM_UPDATESOURCE_COUNTER
Return values	• **None**
Notes	• Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller • Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.
Reference Manual to LL API cross reference:	• CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (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_REPETITIVE
Reference Manual to LL API cross reference:	• CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name	**__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)**
Function description	Set the timer counter counting mode.
Parameters	• **TIMx:** Timer instance • **CounterMode:** This parameter can be one of the following values: – LL_TIM_COUNTERMODE_UP – LL_TIM_COUNTERMODE_DOWN – LL_TIM_COUNTERMODE_CENTER_UP – LL_TIM_COUNTERMODE_CENTER_DOWN – LL_TIM_COUNTERMODE_CENTER_UP_DOWN
Return values	• **None**
Notes	• Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
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	• **Returned:** value can be one of the following values: – LL_TIM_COUNTERMODE_UP – LL_TIM_COUNTERMODE_DOWN – LL_TIM_COUNTERMODE_CENTER_UP – LL_TIM_COUNTERMODE_CENTER_DOWN – LL_TIM_COUNTERMODE_CENTER_UP_DOWN
Notes	• Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
Reference Manual to LL API cross reference:	• CR1 DIR LL_TIM_GetCounterMode • CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name	**__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)**
Function description	Enable auto-reload (ARR) preload.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name	**__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)**
Function description	Disable auto-reload (ARR) preload.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (TIM_TypeDef * TIMx)**
Function description	Indicates whether auto-reload (ARR) preload is enabled.
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name	**__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)**
Function description	Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.
Parameters	• **TIMx:** Timer instance • **ClockDivision:** This parameter can be one of the following values: – LL_TIM_CLOCKDIVISION_DIV1 – LL_TIM_CLOCKDIVISION_DIV2 – LL_TIM_CLOCKDIVISION_DIV4
Return values	• **None**
Notes	• Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.
Reference Manual to LL API cross reference:	• CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (TIM_TypeDef * TIMx)**
Function description	Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_CLOCKDIVISION_DIV1 – LL_TIM_CLOCKDIVISION_DIV2 – LL_TIM_CLOCKDIVISION_DIV4
Notes	• Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.
Reference Manual to LL API cross reference:	• CR1 CKD LL_TIM_GetClockDivision

LL_TIM_SetCounter

Function name	**__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef ***

	TIMx, uint32_t Counter)
Function description	Set the counter value.
Parameters	• **TIMx:** Timer instance • **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF)
Return values	• **None**
Reference Manual to LL API cross reference:	• CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetCounter (TIM_TypeDef * TIMx)**
Function description	Get the counter value.
Parameters	• **TIMx:** Timer instance
Return values	• **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF)
Reference Manual to LL API cross reference:	• CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetDirection (TIM_TypeDef * TIMx)**
Function description	Get the current direction of the counter.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_COUNTERDIRECTION_UP – LL_TIM_COUNTERDIRECTION_DOWN
Reference Manual to LL API cross reference:	• CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name	**__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)**
Function description	Set the prescaler value.
Parameters	• **TIMx:** Timer instance • **Prescaler:** between Min_Data=0 and Max_Data=65535
Return values	• **None**
Notes	• The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1). • The prescaler can be changed on the fly as this control

		register is buffered. The new prescaler ratio is taken into account at the next update event.
	•	Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter
Reference Manual to LL API cross reference:	•	PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (TIM_TypeDef * TIMx)**
Function description	Get the prescaler value.
Parameters	• **TIMx:** Timer instance
Return values	• **Prescaler:** value between Min_Data=0 and Max_Data=65535
Reference Manual to LL API cross reference:	• PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name	**__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)**
Function description	Set the auto-reload value.
Parameters	• **TIMx:** Timer instance • **AutoReload:** between Min_Data=0 and Max_Data=65535
Return values	• **None**
Notes	• The counter is blocked while the auto-reload value is null. • Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter
Reference Manual to LL API cross reference:	• ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (TIM_TypeDef * TIMx)**
Function description	Get the auto-reload value.
Parameters	• **TIMx:** Timer instance
Return values	• **Auto-reload:** value
Reference Manual to LL API cross reference:	• ARR ARR LL_TIM_GetAutoReload

LL_TIM_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_EnableChannel

Function name	**__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)**
Function description	Enable capture/compare channels.
Parameters	• **TIMx:** Timer instance • **Channels:** This parameter can be a combination of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **None**
Reference Manual to LL API cross reference:	• CCER CC1E LL_TIM_CC_EnableChannel • CCER CC2E LL_TIM_CC_EnableChannel • CCER CC3E LL_TIM_CC_EnableChannel • CCER CC4E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name	**__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)**
Function description	Disable capture/compare channels.
Parameters	• **TIMx:** Timer instance • **Channels:** This parameter can be a combination of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **None**
Reference Manual to LL API cross reference:	• CCER CC1E LL_TIM_CC_DisableChannel • CCER CC2E LL_TIM_CC_DisableChannel • CCER CC3E LL_TIM_CC_DisableChannel • CCER CC4E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name	**__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (TIM_TypeDef * TIMx, uint32_t Channels)**
Function description	Indicate whether channel(s) is(are) enabled.
Parameters	• **TIMx:** Timer instance • **Channels:** This parameter can be a combination of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCER CC1E LL_TIM_CC_IsEnabledChannel • CCER CC2E LL_TIM_CC_IsEnabledChannel • CCER CC3E LL_TIM_CC_IsEnabledChannel • CCER CC4E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name	**__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)**
Function description	Configure an output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **Configuration:** This parameter must be a combination of all

	the following values: – LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW
Return values	• **None**
Reference Manual to LL API cross reference:	• CCMR1 CC1S LL_TIM_OC_ConfigOutput • CCMR1 CC2S LL_TIM_OC_ConfigOutput • CCMR2 CC3S LL_TIM_OC_ConfigOutput • CCMR2 CC4S LL_TIM_OC_ConfigOutput • CCER CC1P LL_TIM_OC_ConfigOutput • CCER CC2P LL_TIM_OC_ConfigOutput • CCER CC3P LL_TIM_OC_ConfigOutput • CCER CC4P 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	• **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_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:	• 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	**__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)**
Function description	Set the polarity of an output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **Polarity:** This parameter can be one of the following values: – LL_TIM_OCPOLARITY_HIGH – LL_TIM_OCPOLARITY_LOW
Return values	• **None**
Reference Manual to LL API cross reference:	• CCER CC1P LL_TIM_OC_SetPolarity • CCER CC2P LL_TIM_OC_SetPolarity • CCER CC3P 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_CH2 – LL_TIM_CHANNEL_CH3 – 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 CC2P LL_TIM_OC_GetPolarity • CCER CC3P LL_TIM_OC_GetPolarity • CCER CC4P LL_TIM_OC_GetPolarity

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	**__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast**

	(TIM_TypeDef * TIMx, uint32_t Channel)
Function description	Indicates whether fast mode is enabled for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCMR1 OC1FE LL_TIM_OC_IsEnabledFast • CCMR1 OC2FE LL_TIM_OC_IsEnabledFast • CCMR2 OC3FE LL_TIM_OC_IsEnabledFast • CCMR2 OC4FE LL_TIM_OC_IsEnabledFast •

LL_TIM_OC_EnablePreload

Function name	**__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Enable compare register (TIMx_CCRx) preload for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **None**
Reference Manual to LL API cross reference:	• CCMR1 OC1PE LL_TIM_OC_EnablePreload • CCMR1 OC2PE LL_TIM_OC_EnablePreload • CCMR2 OC3PE LL_TIM_OC_EnablePreload • CCMR2 OC4PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name	**__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Disable compare register (TIMx_CCRx) preload for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **None**
Reference Manual to	• CCMR1 OC1PE LL_TIM_OC_DisablePreload

LL API cross reference:	• CCMR1 OC2PE LL_TIM_OC_DisablePreload • CCMR2 OC3PE LL_TIM_OC_DisablePreload • CCMR2 OC4PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload • CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload • CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload • CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload •

LL_TIM_OC_EnableClear

Function name	**__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Enable clearing the output channel on an external event.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **None**
Notes	• This function can only be used in Output compare and PWM modes. It does not work in Forced mode. • Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.
Reference Manual to LL API cross reference:	• CCMR1 OC1CE LL_TIM_OC_EnableClear • CCMR1 OC2CE LL_TIM_OC_EnableClear • CCMR2 OC3CE LL_TIM_OC_EnableClear • CCMR2 OC4CE LL_TIM_OC_EnableClear

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_SetCompareCH1

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)**

Function description	Set compare value for output channel 1 (TIMx_CCR1).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None**
Notes	• Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 2 (TIMx_CCR2).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None**
Notes	• Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 3 (TIMx_CCR3).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None**
Notes	• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 4 (TIMx_CCR4).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None**
Notes	• Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_GetCompareCH1

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR1) set for output channel 1.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR2) set for output channel 2.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR3) set for output channel 3.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR4) set for output channel 4.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_IC_Config

Function name	**__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)**
Function description	Configure input channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **Configuration:** This parameter must be a combination of all the following values: – LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC – LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8 – LL_TIM_IC_FILTER_FDIV1 or ... or

LL_TIM_IC_FILTER_FDIV32_N8
– LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE

Return values
- **None**

Reference Manual to LL API cross reference:
- CCMR1 CC1S LL_TIM_IC_Config
- CCMR1 IC1PSC LL_TIM_IC_Config
- CCMR1 IC1F LL_TIM_IC_Config
- CCMR1 CC2S LL_TIM_IC_Config
- CCMR1 IC2PSC LL_TIM_IC_Config
- CCMR1 IC2F LL_TIM_IC_Config
- CCMR2 CC3S LL_TIM_IC_Config
- CCMR2 IC3PSC LL_TIM_IC_Config
- CCMR2 IC3F LL_TIM_IC_Config
- CCMR2 CC4S LL_TIM_IC_Config
- CCMR2 IC4PSC LL_TIM_IC_Config
- CCMR2 IC4F LL_TIM_IC_Config
- CCER CC1P LL_TIM_IC_Config
- CCER CC1NP LL_TIM_IC_Config
- CCER CC2P LL_TIM_IC_Config
- CCER CC2NP LL_TIM_IC_Config
- CCER CC3P LL_TIM_IC_Config
- CCER CC3NP LL_TIM_IC_Config
- CCER CC4P LL_TIM_IC_Config
- CCER CC4NP LL_TIM_IC_Config

LL_TIM_IC_SetActiveInput

Function name
__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)

Function description
Set the active input.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
- **ICActiveInput:** This parameter can be one of the following values:
  – LL_TIM_ACTIVEINPUT_DIRECTTI
  – LL_TIM_ACTIVEINPUT_INDIRECTTI
  – LL_TIM_ACTIVEINPUT_TRC

Return values
- **None**

Reference Manual to LL API cross reference:
- CCMR1 CC1S LL_TIM_IC_SetActiveInput
- CCMR1 CC2S LL_TIM_IC_SetActiveInput
- CCMR2 CC3S LL_TIM_IC_SetActiveInput
- CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the current active input.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_ACTIVEINPUT_DIRECTTI – LL_TIM_ACTIVEINPUT_INDIRECTTI – LL_TIM_ACTIVEINPUT_TRC
Reference Manual to LL API cross reference:	• CCMR1 CC1S LL_TIM_IC_GetActiveInput • CCMR1 CC2S LL_TIM_IC_GetActiveInput • CCMR2 CC3S LL_TIM_IC_GetActiveInput • CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name	**__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)**
Function description	Set the prescaler of input channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **ICPrescaler:** This parameter can be one of the following values: – LL_TIM_ICPSC_DIV1 – LL_TIM_ICPSC_DIV2 – LL_TIM_ICPSC_DIV4 – LL_TIM_ICPSC_DIV8
Return values	• **None**
Reference Manual to LL API cross reference:	• CCMR1 IC1PSC LL_TIM_IC_SetPrescaler • CCMR1 IC2PSC LL_TIM_IC_SetPrescaler • CCMR2 IC3PSC LL_TIM_IC_SetPrescaler • CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the current prescaler value acting on an input channel.

Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_ICPSC_DIV1 – LL_TIM_ICPSC_DIV2 – LL_TIM_ICPSC_DIV4 – LL_TIM_ICPSC_DIV8
Reference Manual to LL API cross reference:	• CCMR1 IC1PSC LL_TIM_IC_GetPrescaler • CCMR1 IC2PSC LL_TIM_IC_GetPrescaler • CCMR2 IC3PSC LL_TIM_IC_GetPrescaler • CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name	**__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)**
Function description	Set the input filter duration.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **ICFilter:** This parameter can be one of the following values: – LL_TIM_IC_FILTER_FDIV1 – LL_TIM_IC_FILTER_FDIV1_N2 – LL_TIM_IC_FILTER_FDIV1_N4 – LL_TIM_IC_FILTER_FDIV1_N8 – LL_TIM_IC_FILTER_FDIV2_N6 – LL_TIM_IC_FILTER_FDIV2_N8 – LL_TIM_IC_FILTER_FDIV4_N6 – LL_TIM_IC_FILTER_FDIV4_N8 – LL_TIM_IC_FILTER_FDIV8_N6 – LL_TIM_IC_FILTER_FDIV8_N8 – LL_TIM_IC_FILTER_FDIV16_N5 – LL_TIM_IC_FILTER_FDIV16_N6 – LL_TIM_IC_FILTER_FDIV16_N8 – LL_TIM_IC_FILTER_FDIV32_N5 – LL_TIM_IC_FILTER_FDIV32_N6 – LL_TIM_IC_FILTER_FDIV32_N8
Return values	• **None**
Reference Manual to LL API cross reference:	• CCMR1 IC1F LL_TIM_IC_SetFilter • CCMR1 IC2F LL_TIM_IC_SetFilter • CCMR2 IC3F LL_TIM_IC_SetFilter • CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the input filter duration.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_IC_FILTER_FDIV1 – LL_TIM_IC_FILTER_FDIV1_N2 – LL_TIM_IC_FILTER_FDIV1_N4 – LL_TIM_IC_FILTER_FDIV1_N8 – LL_TIM_IC_FILTER_FDIV2_N6 – LL_TIM_IC_FILTER_FDIV2_N8 – LL_TIM_IC_FILTER_FDIV4_N6 – LL_TIM_IC_FILTER_FDIV4_N8 – LL_TIM_IC_FILTER_FDIV8_N6 – LL_TIM_IC_FILTER_FDIV8_N8 – LL_TIM_IC_FILTER_FDIV16_N5 – LL_TIM_IC_FILTER_FDIV16_N6 – LL_TIM_IC_FILTER_FDIV16_N8 – LL_TIM_IC_FILTER_FDIV32_N5 – LL_TIM_IC_FILTER_FDIV32_N6 – LL_TIM_IC_FILTER_FDIV32_N8
Reference Manual to LL API cross reference:	• CCMR1 IC1F LL_TIM_IC_GetFilter • CCMR1 IC2F LL_TIM_IC_GetFilter • CCMR2 IC3F LL_TIM_IC_GetFilter • CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name	**__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)**
Function description	Set the input channel polarity.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **ICPolarity:** This parameter can be one of the following values: – LL_TIM_IC_POLARITY_RISING – LL_TIM_IC_POLARITY_FALLING – LL_TIM_IC_POLARITY_BOTHEDGE

Return values	• **None**
Reference Manual to LL API cross reference:	• CCER CC1P LL_TIM_IC_SetPolarity • CCER CC1NP LL_TIM_IC_SetPolarity • CCER CC2P LL_TIM_IC_SetPolarity • CCER CC2NP LL_TIM_IC_SetPolarity • CCER CC3P LL_TIM_IC_SetPolarity • CCER CC3NP LL_TIM_IC_SetPolarity • CCER CC4P LL_TIM_IC_SetPolarity • CCER CC4NP LL_TIM_IC_SetPolarity

LL_TIM_IC_GetPolarity

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the current input channel polarity.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_IC_POLARITY_RISING – LL_TIM_IC_POLARITY_FALLING – LL_TIM_IC_POLARITY_BOTHEDGE
Reference Manual to LL API cross reference:	• CCER CC1P LL_TIM_IC_GetPolarity • CCER CC1NP LL_TIM_IC_GetPolarity • CCER CC2P LL_TIM_IC_GetPolarity • CCER CC2NP LL_TIM_IC_GetPolarity • CCER CC3P LL_TIM_IC_GetPolarity • CCER CC3NP LL_TIM_IC_GetPolarity • CCER CC4P LL_TIM_IC_GetPolarity • CCER CC4NP LL_TIM_IC_GetPolarity

LL_TIM_IC_EnableXORCombination

Function name	**__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)**
Function description	Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Notes	• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.
Reference Manual to LL API cross reference:	• CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name	**__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)**
Function description	Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Notes	• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.
Reference Manual to LL API cross reference:	• CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (TIM_TypeDef * TIMx)**
Function description	Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.
Reference Manual to LL API cross reference:	• CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 1.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (TIM_TypeDef * TIMx)**

Function description	Get captured value for input channel 2.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 3.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 4.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name	**__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)**
Function description	Enable external clock mode 2.

Parameters	• **TIMx:** Timer instance
Return values	• **None**
Notes	• When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal. • Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name	**__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)**
Function description	Disable external clock mode 2.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Notes	• Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (TIM_TypeDef * TIMx)**
Function description	Indicate whether external clock mode 2 is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name	**__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)**
Function description	Set the clock source of the counter clock.

Parameters	• **TIMx:** Timer instance • **ClockSource:** This parameter can be one of the following values: – LL_TIM_CLOCKSOURCE_INTERNAL – LL_TIM_CLOCKSOURCE_EXT_MODE1 – LL_TIM_CLOCKSOURCE_EXT_MODE2
Return values	• **None**
Notes	• when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function. • Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1. • Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR SMS LL_TIM_SetClockSource • SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name	**__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)**
Function description	Set the encoder interface mode.
Parameters	• **TIMx:** Timer instance • **EncoderMode:** This parameter can be one of the following values: – LL_TIM_ENCODERMODE_X2_TI1 – LL_TIM_ENCODERMODE_X2_TI2 – LL_TIM_ENCODERMODE_X4_TI12
Return values	• **None**
Notes	• Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.
Reference Manual to LL API cross reference:	• SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name	**__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)**
Function description	Set the trigger output (TRGO) used for timer synchronization .
Parameters	• **TIMx:** Timer instance • **TimerSynchronization:** This parameter can be one of the

following values:
- LL_TIM_TRGO_RESET
- LL_TIM_TRGO_ENABLE
- LL_TIM_TRGO_UPDATE
- LL_TIM_TRGO_CC1IF
- LL_TIM_TRGO_OC1REF
- LL_TIM_TRGO_OC2REF
- LL_TIM_TRGO_OC3REF
- LL_TIM_TRGO_OC4REF

Return values	• **None**
Notes	• Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.
Reference Manual to LL API cross reference:	• CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_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	• **None**
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (TIM_TypeDef * TIMx)**
Function description	Indicates whether the Master/Slave mode is enabled.

Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name	**__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)**
Function description	Configure the external trigger (ETR) input.
Parameters	• **TIMx:** Timer instance • **ETRPolarity:** This parameter can be one of the following values: – LL_TIM_ETR_POLARITY_NONINVERTED – LL_TIM_ETR_POLARITY_INVERTED • **ETRPrescaler:** This parameter can be one of the following values: – LL_TIM_ETR_PRESCALER_DIV1 – LL_TIM_ETR_PRESCALER_DIV2 – LL_TIM_ETR_PRESCALER_DIV4 – LL_TIM_ETR_PRESCALER_DIV8 • **ETRFilter:** This parameter can be one of the following values: – LL_TIM_ETR_FILTER_FDIV1 – LL_TIM_ETR_FILTER_FDIV1_N2 – LL_TIM_ETR_FILTER_FDIV1_N4 – LL_TIM_ETR_FILTER_FDIV1_N8 – LL_TIM_ETR_FILTER_FDIV2_N6 – LL_TIM_ETR_FILTER_FDIV2_N8 – LL_TIM_ETR_FILTER_FDIV4_N6 – LL_TIM_ETR_FILTER_FDIV4_N8 – LL_TIM_ETR_FILTER_FDIV8_N6 – LL_TIM_ETR_FILTER_FDIV8_N8 – LL_TIM_ETR_FILTER_FDIV16_N5 – LL_TIM_ETR_FILTER_FDIV16_N6 – LL_TIM_ETR_FILTER_FDIV16_N8 – LL_TIM_ETR_FILTER_FDIV32_N5 – LL_TIM_ETR_FILTER_FDIV32_N6 – LL_TIM_ETR_FILTER_FDIV32_N8
Return values	• **None**
Notes	• Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.
Reference Manual to	• SMCR ETP LL_TIM_ConfigETR

LL API cross reference:	• SMCR ETPS LL_TIM_ConfigETR • SMCR ETF LL_TIM_ConfigETR

LL_TIM_ConfigDMABurst

Function name: **__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength)**

Function description: Configures the timer DMA burst feature.

Parameters:

- **TIMx:** Timer instance
- **DMABurstBaseAddress:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_BASEADDR_CR1
  - LL_TIM_DMABURST_BASEADDR_CR2
  - LL_TIM_DMABURST_BASEADDR_SMCR
  - LL_TIM_DMABURST_BASEADDR_DIER
  - LL_TIM_DMABURST_BASEADDR_SR
  - LL_TIM_DMABURST_BASEADDR_EGR
  - LL_TIM_DMABURST_BASEADDR_CCMR1
  - LL_TIM_DMABURST_BASEADDR_CCMR2
  - LL_TIM_DMABURST_BASEADDR_CCER
  - LL_TIM_DMABURST_BASEADDR_CNT
  - LL_TIM_DMABURST_BASEADDR_PSC
  - LL_TIM_DMABURST_BASEADDR_ARR
  - LL_TIM_DMABURST_BASEADDR_RCR
  - LL_TIM_DMABURST_BASEADDR_CCR1
  - LL_TIM_DMABURST_BASEADDR_CCR2
  - LL_TIM_DMABURST_BASEADDR_CCR3
  - LL_TIM_DMABURST_BASEADDR_CCR4
  - LL_TIM_DMABURST_BASEADDR_BDTR
  - LL_TIM_DMABURST_BASEADDR_CCMR3
  - LL_TIM_DMABURST_BASEADDR_CCR5
  - LL_TIM_DMABURST_BASEADDR_CCR6
  - LL_TIM_DMABURST_BASEADDR_OR1
  - LL_TIM_DMABURST_BASEADDR_OR2
  - LL_TIM_DMABURST_BASEADDR_OR3
- **DMABurstLength:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_LENGTH_1TRANSFER
  - LL_TIM_DMABURST_LENGTH_2TRANSFERS
  - LL_TIM_DMABURST_LENGTH_3TRANSFERS
  - LL_TIM_DMABURST_LENGTH_4TRANSFERS
  - LL_TIM_DMABURST_LENGTH_5TRANSFERS
  - LL_TIM_DMABURST_LENGTH_6TRANSFERS
  - LL_TIM_DMABURST_LENGTH_7TRANSFERS
  - LL_TIM_DMABURST_LENGTH_8TRANSFERS
  - LL_TIM_DMABURST_LENGTH_9TRANSFERS
  - LL_TIM_DMABURST_LENGTH_10TRANSFERS
  - LL_TIM_DMABURST_LENGTH_11TRANSFERS
  - LL_TIM_DMABURST_LENGTH_12TRANSFERS
  - LL_TIM_DMABURST_LENGTH_13TRANSFERS
  - LL_TIM_DMABURST_LENGTH_14TRANSFERS

	– LL_TIM_DMABURST_LENGTH_15TRANSFERS – LL_TIM_DMABURST_LENGTH_16TRANSFERS – LL_TIM_DMABURST_LENGTH_17TRANSFERS – LL_TIM_DMABURST_LENGTH_18TRANSFERS
Return values	• **None**
Notes	• Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.
Reference Manual to LL API cross reference:	• DCR DBL LL_TIM_ConfigDMABurst • DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_SetRemap

Function name	**__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)**
Function description	Remap TIM inputs (input channel, internal/external triggers).
Parameters	• **TIMx:** Timer instance • **Remap:** Remap params depends on the TIMx. Description available only in CHM version of the User Manual (not in .pdf). Otherwise see Reference Manual description of OR registers.
Return values	• **None**
Notes	• Macro IS_TIM_REMAP_INSTANCE(TIMx) can be used to check whether or not a some timer inputs can be remapped.
Reference Manual to LL API cross reference:	• TIM2_OR ETR_RMP LL_TIM_SetRemap • TIM2_OR TI4_RMP LL_TIM_SetRemap • TIM21_OR ETR_RMP LL_TIM_SetRemap • TIM21_OR TI1_RMP LL_TIM_SetRemap • TIM21_OR TI2_RMP LL_TIM_SetRemap • TIM22_OR ETR_RMP LL_TIM_SetRemap • TIM22_OR TI1_RMP LL_TIM_SetRemap • TIM3_OR ETR_RMP LL_TIM_SetRemap • TIM3_OR TI1_RMP LL_TIM_SetRemap • TIM3_OR TI2_RMP LL_TIM_SetRemap • TIM3_OR TI4_RMP LL_TIM_SetRemap

LL_TIM_SetOCRefClearInputSource

Function name	**__STATIC_INLINE void LL_TIM_SetOCRefClearInputSource (TIM_TypeDef * TIMx, uint32_t OCRefClearInputSource)**
Function description	Set the OCREF clear input source.
Parameters	• **TIMx:** Timer instance • **OCRefClearInputSource:** This parameter can be one of the following values: – LL_TIM_OCREF_CLR_INT_NC – LL_TIM_OCREF_CLR_INT_ETR

Return values	• **None**
Notes	• The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUT • This function can only be used in Output compare and PWM modes.
Reference Manual to LL API cross reference:	• SMCR OCCS LL_TIM_SetOCRefClearInputSource

LL_TIM_ClearFlag_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)**
Function description	Clear the update interrupt flag (UIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 1 interrupt flag (CC1F).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1**

	(TIM_TypeDef * TIMx)
Function description	Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 2 interrupt flag (CC2F).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 3 interrupt flag (CC3F).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 4 interrupt flag (CC4F).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (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_TRIG

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)**
Function description	Clear the trigger interrupt flag (TIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_CC1OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (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	• SR CC4OF LL_TIM_ClearFlag_CC4OVR

reference:

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	**__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Enable update interrupt (UIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Disable update interrupt (UIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicates whether the update interrupt (UIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• DIER UIE LL_TIM_IsEnabledIT_UPDATE

reference:

LL_TIM_EnableIT_CC1

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 1 interrupt (CC1IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 1 interrupt (CC1IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (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	• DIER CC2IE LL_TIM_EnableIT_CC2

reference:

LL_TIM_DisableIT_CC2

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 2 interrupt (CC2IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 3 interrupt (CC3IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 3 interrupt (CC3IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross	• DIER CC3IE LL_TIM_DisableIT_CC3

reference:

LL_TIM_IsEnabledIT_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 4 interrupt (CC4IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 4 interrupt (CC4IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (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	• DIER CC4IE LL_TIM_IsEnabledIT_CC4

reference:

LL_TIM_EnableIT_TRIG

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Enable trigger interrupt (TIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Disable trigger interrupt (TIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicates whether the trigger interrupt (TIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableDMAReq_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Enable update DMA request (UDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Disable update DMA request (UDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicates whether the update DMA request (UDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 1 DMA request (CC1DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 1 DMA request (CC1DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 2 DMA request (CC2DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 2 DMA request (CC2DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 3 DMA request (CC3DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 3 DMA request (CC3DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

LL_TIM_EnableDMAReq_CC4

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 4 DMA request (CC4DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 4 DMA request (CC4DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_TRIG

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Enable trigger interrupt (TDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Disable trigger interrupt (TDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicates whether the trigger interrupt (TDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)**
Function description	Generate an update event.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 1 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 2 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 3 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 4 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_TRIG

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)**
Function description	Generate trigger event.
Parameters	• **TIMx:** Timer instance
Return values	• **None**
Reference Manual to LL API cross reference:	• EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_DeInit

Function name	**ErrorStatus LL_TIM_DeInit (TIM_TypeDef * TIMx)**
Function description	Set TIMx registers to their reset values.
Parameters	• **TIMx:** Timer instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx registers are de-initialized – ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name	**void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)**
Function description	Set the fields of the timebase unit configuration data structure to

	their default values.
Parameters	• **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (timebase unit configuration data structure)
Return values	• **None**

LL_TIM_Init

Function name	**ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, LL_TIM_InitTypeDef * TIM_InitStruct)**
Function description	Configure the TIMx timebase unit.
Parameters	• **TIMx:** Timer Instance • **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx timebase unit configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx registers are de-initialized – ERROR: not applicable

LL_TIM_OC_StructInit

Function name	**void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)**
Function description	Set the fields of the TIMx output channel configuration data structure to their default values.
Parameters	• **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)
Return values	• **None**

LL_TIM_OC_Init

Function name	**ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)**
Function description	Configure the TIMx output channel.
Parameters	• **TIMx:** Timer Instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx output channel is initialized – ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name	**void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef ***

	TIM_ICInitStruct)
Function description	Set the fields of the TIMx input channel configuration data structure to their default values.
Parameters	• **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)
Return values	• **None**

LL_TIM_IC_Init

Function name	**ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)**
Function description	Configure the TIMx input channel.
Parameters	• **TIMx:** Timer Instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx output channel is initialized – ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name	**void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)**
Function description	Fills each TIM_EncoderInitStruct field with its default value.
Parameters	• **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)
Return values	• **None**

LL_TIM_ENCODER_Init

Function name	**ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)**
Function description	Configure the encoder interface of the timer instance.
Parameters	• **TIMx:** Timer Instance • **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx registers are de-initialized – ERROR: not applicable

74.3 TIM Firmware driver defines

74.3.1 TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI	ICx is mapped on TIx
LL_TIM_ACTIVEINPUT_INDIRECTTI	ICx is mapped on TIy
LL_TIM_ACTIVEINPUT_TRC	ICx is mapped on TRC

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC	CCx DMA request sent when CCx event occurs
LL_TIM_CCDMAREQUEST_UPDATE	CCx DMA requests sent when update event occurs

Channel

LL_TIM_CHANNEL_CH1	Timer input/output channel 1
LL_TIM_CHANNEL_CH2	Timer input/output channel 2
LL_TIM_CHANNEL_CH3	Timer input/output channel 3
LL_TIM_CHANNEL_CH4	Timer input/output channel 4

Clock Division

LL_TIM_CLOCKDIVISION_DIV1	tDTS=tCK_INT
LL_TIM_CLOCKDIVISION_DIV2	tDTS=2*tCK_INT
LL_TIM_CLOCKDIVISION_DIV4	tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL	The timer is clocked by the internal clock provided from the RCC
LL_TIM_CLOCKSOURCE_EXT_MODE1	Counter counts at each rising or falling edge on a selected inpu t
LL_TIM_CLOCKSOURCE_EXT_MODE2	Counter counts at each rising or falling edge on the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP	Timer counter counts up
LL_TIM_COUNTERDIRECTION_DOWN	Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP	Counter used as upcounter
LL_TIM_COUNTERMODE_DOWN	Counter used as downcounter
LL_TIM_COUNTERMODE_CENTER_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_BASEADDR_CR1	TIMx_CR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CR2	TIMx_CR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_SMCR	TIMx_SMCR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_DIER	TIMx_DIER register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_SR	TIMx_SR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_EGR	TIMx_EGR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR1	TIMx_CCMR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR2	TIMx_CCMR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCER	TIMx_CCER register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CNT	TIMx_CNT register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_PSC	TIMx_PSC register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_ARR	TIMx_ARR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_RCR	TIMx_RCR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR1	TIMx_CCR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR2	TIMx_CCR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR3	TIMx_CCR3 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR4	TIMx_CCR4 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_BDTR	TIMx_BDTR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR3	TIMx_CCMR3 register is the DMA base

	address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR5	TIMx_CCR5 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR6	TIMx_CCR6 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR1	TIMx_OR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR2	TIMx_OR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR3	TIMx_OR3 register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER	Transfer is done to 1 register starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_2TRANSFERS	Transfer is done to 2 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_3TRANSFERS	Transfer is done to 3 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_4TRANSFERS	Transfer is done to 4 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_5TRANSFERS	Transfer is done to 5 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_6TRANSFERS	Transfer is done to 6 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_7TRANSFERS	Transfer is done to 7 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_8TRANSFERS	Transfer is done to 1 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_9TRANSFERS	Transfer is done to 9 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_10TRANSFERS	Transfer is done to 10 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_11TRANSFERS	Transfer is done to 11 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_12TRANSFERS	Transfer is done to 12 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_13TRANSFERS	Transfer is done to 13 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_14TRANSFERS	Transfer is done to 14 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_15TRANSFERS	Transfer is done to 15 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_16TRANSFERS	Transfer is done to 16 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_17TRANSFERS	Transfer is done to 17 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_18TRANSFERS	Transfer is done to 18 registers starting from the DMA burst base address

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1	Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level
LL_TIM_ENCODERMODE_X2_TI2	Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level
LL_TIM_ENCODERMODE_X4_TI12	Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input l

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1	No filter, sampling is done at fDTS
LL_TIM_ETR_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2
LL_TIM_ETR_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_ETR_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_ETR_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_ETR_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_ETR_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_ETR_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_ETR_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=8
LL_TIM_ETR_FILTER_FDIV8_N8	fSAMPLING=fDTS/16, N=5
LL_TIM_ETR_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=6
LL_TIM_ETR_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=8
LL_TIM_ETR_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=5
LL_TIM_ETR_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_ETR_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6
LL_TIM_ETR_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED	ETR is non-inverted, active at high level or rising edge
LL_TIM_ETR_POLARITY_INVERTED	ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1	ETR prescaler OFF
LL_TIM_ETR_PRESCALER_DIV2	ETR frequency is divided by 2
LL_TIM_ETR_PRESCALER_DIV4	ETR frequency is divided by 4
LL_TIM_ETR_PRESCALER_DIV8	ETR frequency is divided by 8

Get Flags Defines

LL_TIM_SR_UIF	Update interrupt flag
LL_TIM_SR_CC1IF	Capture/compare 1 interrupt flag
LL_TIM_SR_CC2IF	Capture/compare 2 interrupt flag
LL_TIM_SR_CC3IF	Capture/compare 3 interrupt flag
LL_TIM_SR_CC4IF	Capture/compare 4 interrupt flag
LL_TIM_SR_TIF	Trigger interrupt flag
LL_TIM_SR_CC1OF	Capture/Compare 1 overcapture flag
LL_TIM_SR_CC2OF	Capture/Compare 2 overcapture flag
LL_TIM_SR_CC3OF	Capture/Compare 3 overcapture flag
LL_TIM_SR_CC4OF	Capture/Compare 4 overcapture flag

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1	No prescaler, capture is done each time an edge is detected on the capture input
LL_TIM_ICPSC_DIV2	Capture is done once every 2 events
LL_TIM_ICPSC_DIV4	Capture is done once every 4 events
LL_TIM_ICPSC_DIV8	Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1	No filter, sampling is done at fDTS
LL_TIM_IC_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2
LL_TIM_IC_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_IC_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_IC_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_IC_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_IC_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_IC_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_IC_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=6
LL_TIM_IC_FILTER_FDIV8_N8	fSAMPLING=fDTS/8, N=8
LL_TIM_IC_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=5
LL_TIM_IC_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=6
LL_TIM_IC_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=8
LL_TIM_IC_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_IC_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6
LL_TIM_IC_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING	The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted

LL_TIM_IC_POLARITY_FALLING	The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted
LL_TIM_IC_POLARITY_BOTHEDGE	The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

IT Defines

LL_TIM_DIER_UIE	Update interrupt enable
LL_TIM_DIER_CC1IE	Capture/compare 1 interrupt enable
LL_TIM_DIER_CC2IE	Capture/compare 2 interrupt enable
LL_TIM_DIER_CC3IE	Capture/compare 3 interrupt enable
LL_TIM_DIER_CC4IE	Capture/compare 4 interrupt enable
LL_TIM_DIER_TIE	Trigger interrupt enable

Output Configuration Mode

LL_TIM_OCMODE_FROZEN	The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level
LL_TIM_OCMODE_ACTIVE	OCyREF is forced high on compare match
LL_TIM_OCMODE_INACTIVE	OCyREF is forced low on compare match
LL_TIM_OCMODE_TOGGLE	OCyREF toggles on compare match
LL_TIM_OCMODE_FORCED_INACTIVE	OCyREF is forced low
LL_TIM_OCMODE_FORCED_ACTIVE	OCyREF is forced high
LL_TIM_OCMODE_PWM1	In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.
LL_TIM_OCMODE_PWM2	In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH	OCxactive high
LL_TIM_OCPOLARITY_LOW	OCxactive low

OCREF clear input selection

LL_TIM_OCREF_CLR_INT_NC	OCREF_CLR_INT is not connected
LL_TIM_OCREF_CLR_INT_ETR	OCREF_CLR_INT is connected to ETRF

Output Configuration State

LL_TIM_OCSTATE_DISABLE	OCx is not active
LL_TIM_OCSTATE_ENABLE	OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE	Counter is not stopped at update event

LL_TIM_ONEPULSEMODE_REPETITIVE	Counter stops counting at the next update event

Slave Mode

LL_TIM_SLAVEMODE_DISABLED	Slave mode disabled
LL_TIM_SLAVEMODE_RESET	Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter
LL_TIM_SLAVEMODE_GATED	Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high
LL_TIM_SLAVEMODE_TRIGGER	Trigger Mode - The counter starts at a rising edge of the trigger TRGI

TIM21 External Trigger Remap

LL_TIM_TIM21_ETR_RMP_GPIO	TIM21_ETR is connected to Ored GPIO1
LL_TIM_TIM21_ETR_RMP_COMP2	TIM21_ETR is connected to COMP2_OUT
LL_TIM_TIM21_ETR_RMP_COMP1	TIM21_ETR is connected to COMP1_OUT
LL_TIM_TIM21_ETR_RMP_LSE	TIM21_ETR is connected to LSE

TIM21 External Input Ch1 Remap

LL_TIM_TIM21_TI1_RMP_GPIO	TIM21_TI1 is connected to Ored GPIO1
LL_TIM_TIM21_TI1_RMP_RTC_WK	TIM21_TI1 is connected to RTC_WAKEUP
LL_TIM_TIM21_TI1_RMP_HSE_RTC	TIM21_TI1 is connected to HSE_RTC
LL_TIM_TIM21_TI1_RMP_MSI	TIM21_TI1 is connected to MSI
LL_TIM_TIM21_TI1_RMP_LSE	TIM21_TI1 is connected to LSE
LL_TIM_TIM21_TI1_RMP_LSI	TIM21_TI1 is connected to LSI
LL_TIM_TIM21_TI1_RMP_COMP1	TIM21_TI1 is connected to COMP1_OUT
LL_TIM_TIM21_TI1_RMP_MCO	TIM21_TI1 is connected to MCO

TIM21 External Input Ch2 Remap

LL_TIM_TIM21_TI2_RMP_GPIO	TIM21_TI2 is connected to Ored GPIO1
LL_TIM_TIM21_TI2_RMP_COMP2	TIM21_TI2 is connected to COMP2_OUT

TIM22 External Trigger Remap

LL_TIM_TIM22_ETR_RMP_GPIO	TIM22_ETR is connected to GPIO
LL_TIM_TIM22_ETR_RMP_COMP2	TIM22_ETR is connected to COMP2_OUT
LL_TIM_TIM22_ETR_RMP_COMP1	TIM22_ETR is connected to COMP1_OUT
LL_TIM_TIM22_ETR_RMP_LSE	TIM22_ETR is connected to LSE

TIM22 External Input Ch1 Remap

LL_TIM_TIM22_TI1_RMP_GPIO1	TIM22_TI1 is connected to GPIO1
LL_TIM_TIM22_TI1_RMP_COMP2	TIM22_TI1 is connected to COMP2_OUT
LL_TIM_TIM22_TI1_RMP_COMP1	TIM22_TI1 is connected to COMP1_OUT
LL_TIM_TIM22_TI1_RMP_GPIO2	TIM22_TI1 is connected to GPIO2

TIM2 External Trigger Remap

LL_TIM_TIM2_ETR_RMP_GPIO	TIM2_ETR is connected to Ored GPIO
LL_TIM_TIM2_ETR_RMP_HSI	TIM2_ETR is connected to HSI
LL_TIM_TIM2_ETR_RMP_HSI48	TIM2_ETR is connected to HSI48
LL_TIM_TIM2_ETR_RMP_LSE	TIM2_ETR is connected to LSE
LL_TIM_TIM2_ETR_RMP_COMP2	TIM2_ETR is connected to COMP2_OUT
LL_TIM_TIM2_ETR_RMP_COMP1	TIM2_ETR is connected to COMP1_OUT

TIM2 Timer Input Ch4 Remap

LL_TIM_TIM2_TI4_RMP_GPIO	TIM2 input capture 4 is connected to GPIO
LL_TIM_TIM2_TI4_RMP_COMP2	TIM2 input capture 4 is connected to COMP2_OUT
LL_TIM_TIM2_TI4_RMP_COMP1	TIM2 input capture 4 is connected to COMP1_OUT

TIM3 External Trigger Remap

LL_TIM_TIM3_ETR_RMP_GPIO	TIM3_ETR is connected to GPIO
LL_TIM_TIM3_ETR_RMP_HSI48DIV6	TIM3_ETR is connected to HSI48 divided by 6

TIM3 External Inputs Remap

LL_TIM_TIM3_TI_RMP_TI1_USB_SOF	TIM3_TI1 input is connected to USB_SOF
LL_TIM_TIM3_TI_RMP_TI1_GPIO	TIM3_TI1 input is connected to PE3, PA6, PC6 or PB4
LL_TIM_TIM3_TI_RMP_TI2_GPIO_DEF	Mapping PB5 to TIM22_CH2
LL_TIM_TIM3_TI_RMP_TI2_GPIOB5_AF4	Mapping PB5 to TIM3_CH2
LL_TIM_TIM3_TI_RMP_TI4_GPIO_DEF	Mapping PC9 to USB_OE
LL_TIM_TIM3_TI_RMP_TI4_GPIOC9_AF2	Mapping PC9 to TIM3_CH4

Trigger Output

LL_TIM_TRGO_RESET	UG bit from the TIMx_EGR register is used as trigger output
LL_TIM_TRGO_ENABLE	Counter Enable signal (CNT_EN) is used as trigger output
LL_TIM_TRGO_UPDATE	Update event is used as trigger output
LL_TIM_TRGO_CC1IF	CC1 capture or a compare match is used as trigger output
LL_TIM_TRGO_OC1REF	OC1REF signal is used as trigger output
LL_TIM_TRGO_OC2REF	OC2REF signal is used as trigger output
LL_TIM_TRGO_OC3REF	OC3REF signal is used as trigger output
LL_TIM_TRGO_OC4REF	OC4REF signal is used as trigger output

Trigger Selection

LL_TIM_TS_ITR0	Internal Trigger 0 (ITR0) is used as trigger input
LL_TIM_TS_ITR1	Internal Trigger 1 (ITR1) is used as trigger input
LL_TIM_TS_ITR2	Internal Trigger 2 (ITR2) is used as trigger input
LL_TIM_TS_ITR3	Internal Trigger 3 (ITR3) is used as trigger input
LL_TIM_TS_TI1F_ED	TI1 Edge Detector (TI1F_ED) is used as trigger input
LL_TIM_TS_TI1FP1	Filtered Timer Input 1 (TI1FP1) is used as trigger input

LL_TIM_TS_TI2FP2	Filtered Timer Input 2 (TI12P2) is used as trigger input
LL_TIM_TS_ETRF	Filtered external Trigger (ETRF) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR	Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request
LL_TIM_UPDATESOURCE_COUNTER	Only counter overflow/underflow generates an update request

Exported_Macros

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive

delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

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

75 LL USART Generic Driver

75.1 USART Firmware driver registers structures

75.1.1 LL_USART_InitTypeDef

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()**.

75.1.2 LL_USART_ClockInitTypeDef

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.

75.2 USART Firmware driver API description

75.2.1 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	**__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (USART_TypeDef * USARTx)**
Function description	Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name	**__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)**
Function description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name	**__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)**
Function description	Receiver Disable.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name	**__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)**

Function description	Transmitter Enable.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name	**__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)**
Function description	Transmitter Disable.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name	**__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)**
Function description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	• **USARTx:** USART Instance • **TransferDirection:** This parameter can be one of the following values: – LL_USART_DIRECTION_NONE – LL_USART_DIRECTION_RX – LL_USART_DIRECTION_TX – LL_USART_DIRECTION_TX_RX
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_SetTransferDirection • CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (USART_TypeDef * USARTx)**
Function description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_DIRECTION_NONE – LL_USART_DIRECTION_RX – LL_USART_DIRECTION_TX

	– LL_USART_DIRECTION_TX_RX
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_GetTransferDirection • CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name	**__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)**
Function description	Configure Parity (enabled/disabled and parity mode if enabled).
Parameters	• **USARTx:** USART Instance • **Parity:** This parameter can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD
Return values	• **None**
Notes	• This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_SetParity • CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name	**__STATIC_INLINE uint32_t LL_USART_GetParity (USART_TypeDef * USARTx)**
Function description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_GetParity • CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name	**__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)**
Function description	Set Receiver Wake Up method from Mute mode.
Parameters	• **USARTx:** USART Instance • **Method:** This parameter can be one of the following values: – LL_USART_WAKEUP_IDLELINE

- – LL_USART_WAKEUP_ADDRESSMARK

Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name	**__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (USART_TypeDef * USARTx)**
Function description	Return Receiver Wake Up method from Mute mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_WAKEUP_IDLELINE – LL_USART_WAKEUP_ADDRESSMARK
Reference Manual to LL API cross reference:	• CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name	**__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)**
Function description	Set Word length (i.e.
Parameters	• **USARTx:** USART Instance • **DataWidth:** This parameter can be one of the following values: – LL_USART_DATAWIDTH_7B – LL_USART_DATAWIDTH_8B – LL_USART_DATAWIDTH_9B
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 M0 LL_USART_SetDataWidth • CR1 M1 LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDataWidth (USART_TypeDef * USARTx)**
Function description	Return Word length (i.e.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_DATAWIDTH_7B – LL_USART_DATAWIDTH_8B – LL_USART_DATAWIDTH_9B
Reference Manual to LL API cross	• CR1 M0 LL_USART_GetDataWidth

reference:	• 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	• **USARTx:** USART Instance • **OverSampling:** This parameter can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8
Return values	• **None**

Reference Manual to LL API cross reference:	• CR1 OVER8 LL_USART_SetOverSampling

LL_USART_GetOverSampling

Function name	**__STATIC_INLINE uint32_t LL_USART_GetOverSampling (USART_TypeDef * USARTx)**
Function description	Return Oversampling mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8
Reference Manual to LL API cross reference:	• CR1 OVER8 LL_USART_GetOverSampling

LL_USART_SetLastClkPulseOutput

Function name	**__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)**
Function description	Configure if Clock pulse of the last data bit is output to the SCLK pin or not.
Parameters	• **USARTx:** USART Instance • **LastBitClockPulse:** This parameter can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT – LL_USART_LASTCLKPULSE_OUTPUT
Return values	• **None**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name	**__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (USART_TypeDef * USARTx)**
Function description	Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT – LL_USART_LASTCLKPULSE_OUTPUT

Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name	**__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)**
Function description	Select the phase of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance • **ClockPhase:** This parameter can be one of the following values: – LL_USART_PHASE_1EDGE – LL_USART_PHASE_2EDGE
Return values	• **None**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name	**__STATIC_INLINE uint32_t LL_USART_GetClockPhase (USART_TypeDef * USARTx)**
Function description	Return phase of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_PHASE_1EDGE – LL_USART_PHASE_2EDGE
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name	**__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)**
Function description	Select the polarity of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance • **ClockPolarity:** This parameter can be one of the following values: – LL_USART_POLARITY_LOW – LL_USART_POLARITY_HIGH
Return values	• **None**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (USART_TypeDef * USARTx)**
Function description	Return polarity of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_POLARITY_LOW – LL_USART_POLARITY_HIGH
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name	**__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)**
Function description	Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)
Parameters	• **USARTx:** USART Instance • **Phase:** This parameter can be one of the following values: – LL_USART_PHASE_1EDGE

	– LL_USART_PHASE_2EDGE • **Polarity:** This parameter can be one of the following values: – LL_USART_POLARITY_LOW – LL_USART_POLARITY_HIGH • **LBCPOutput:** This parameter can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT – LL_USART_LASTCLKPULSE_OUTPUT
Return values	• **None**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function
Reference Manual to LL API cross reference:	• CR2 CPHA LL_USART_ConfigClock • CR2 CPOL LL_USART_ConfigClock • CR2 LBCL LL_USART_ConfigClock

LL_USART_EnableSCLKOutput

Function name	**__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)**
Function description	Enable Clock output on SCLK pin.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name	**__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)**
Function description	Disable Clock output on SCLK pin.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to	• CR2 CLKEN LL_USART_DisableSCLKOutput

LL API cross reference:	

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:	• CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name	**__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)**
Function description	Set the length of the stop bits.
Parameters	• **USARTx:** USART Instance • **StopBits:** This parameter can be one of the following values: – LL_USART_STOPBITS_0_5 – LL_USART_STOPBITS_1 – LL_USART_STOPBITS_1_5 – LL_USART_STOPBITS_2
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name	**__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (USART_TypeDef * USARTx)**
Function description	Retrieve the length of the stop bits.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_STOPBITS_0_5 – LL_USART_STOPBITS_1 – LL_USART_STOPBITS_1_5 – LL_USART_STOPBITS_2
Reference Manual to LL API cross reference:	• CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name	**__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)**
Function description	Configure Character frame format (Datawidth, Parity control, Stop Bits)
Parameters	• **USARTx:** USART Instance • **DataWidth:** This parameter can be one of the following values: – LL_USART_DATAWIDTH_7B – LL_USART_DATAWIDTH_8B – LL_USART_DATAWIDTH_9B • **Parity:** This parameter can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD • **StopBits:** This parameter can be one of the following values: – LL_USART_STOPBITS_0_5 – LL_USART_STOPBITS_1 – LL_USART_STOPBITS_1_5 – LL_USART_STOPBITS_2
Return values	• **None**
Notes	• Call of this function is equivalent to following function call sequence : Data Width configuration using LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_ConfigCharacter • CR1 PCE LL_USART_ConfigCharacter • CR1 M0 LL_USART_ConfigCharacter • CR1 M1 LL_USART_ConfigCharacter • CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetTXRXSwap

Function name	**__STATIC_INLINE void LL_USART_SetTXRXSwap (USART_TypeDef * USARTx, uint32_t SwapConfig)**
Function description	Configure TX/RX pins swapping setting.
Parameters	• **USARTx:** USART Instance • **SwapConfig:** This parameter can be one of the following values: – LL_USART_TXRX_STANDARD – LL_USART_TXRX_SWAPPED
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 SWAP LL_USART_SetTXRXSwap

LL_USART_GetTXRXSwap

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap (USART_TypeDef * USARTx)**
Function description	Retrieve TX/RX pins swapping configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_TXRX_STANDARD – LL_USART_TXRX_SWAPPED
Reference Manual to LL API cross reference:	• CR2 SWAP LL_USART_GetTXRXSwap

LL_USART_SetRXPinLevel

Function name	**__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)**
Function description	Configure RX pin active level logic.
Parameters	• **USARTx:** USART Instance • **PinInvMethod:** This parameter can be one of the following values: – LL_USART_RXPIN_LEVEL_STANDARD – LL_USART_RXPIN_LEVEL_INVERTED
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RXINV LL_USART_SetRXPinLevel

LL_USART_GetRXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (USART_TypeDef * USARTx)**
Function description	Retrieve RX pin active level logic configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_RXPIN_LEVEL_STANDARD – LL_USART_RXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 RXINV LL_USART_GetRXPinLevel

LL_USART_SetTXPinLevel

Function name	**__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)**
Function description	Configure TX pin active level logic.
Parameters	• **USARTx:** USART Instance

	• **PinInvMethod:** This parameter can be one of the following values: – LL_USART_TXPIN_LEVEL_STANDARD – LL_USART_TXPIN_LEVEL_INVERTED
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 TXINV LL_USART_SetTXPinLevel

LL_USART_GetTXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (USART_TypeDef * USARTx)**
Function description	Retrieve TX pin active level logic configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_TXPIN_LEVEL_STANDARD – LL_USART_TXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 TXINV LL_USART_GetTXPinLevel

LL_USART_SetBinaryDataLogic

Function name	**__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)**
Function description	Configure Binary data logic.
Parameters	• **USARTx:** USART Instance • **DataLogic:** This parameter can be one of the following values: – LL_USART_BINARY_LOGIC_POSITIVE – LL_USART_BINARY_LOGIC_NEGATIVE
Return values	• **None**
Notes	• Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_USART_SetBinaryDataLogic

LL_USART_GetBinaryDataLogic

Function name	**__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (USART_TypeDef * USARTx)**
Function description	Retrieve Binary data configuration.
Parameters	• **USARTx:** USART Instance

Return values	• **Returned:** value can be one of the following values: – LL_USART_BINARY_LOGIC_POSITIVE – LL_USART_BINARY_LOGIC_NEGATIVE
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_USART_GetBinaryDataLogic

LL_USART_SetTransferBitOrder

Function name	**__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)**
Function description	Configure transfer bit order (either Less or Most Significant Bit First)
Parameters	• **USARTx:** USART Instance • **BitOrder:** This parameter can be one of the following values: – LL_USART_BITORDER_LSBFIRST – LL_USART_BITORDER_MSBFIRST
Return values	• **None**
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross reference:	• CR2 MSBFIRST LL_USART_SetTransferBitOrder

LL_USART_GetTransferBitOrder

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (USART_TypeDef * USARTx)**
Function description	Return transfer bit order (either Less or Most Significant Bit First)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_BITORDER_LSBFIRST – LL_USART_BITORDER_MSBFIRST
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross reference:	• CR2 MSBFIRST LL_USART_GetTransferBitOrder

LL_USART_EnableAutoBaudRate

Function name	**__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)**
Function description	Enable Auto Baud-Rate Detection.
Parameters	• **USARTx:** USART Instance

Return values	•	**None**
Notes	•	Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	•	CR2 ABREN LL_USART_EnableAutoBaudRate

LL_USART_DisableAutoBaudRate

Function name	**__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)**
Function description	Disable Auto Baud-Rate Detection.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 ABREN LL_USART_DisableAutoBaudRate

LL_USART_IsEnabledAutoBaud

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (USART_TypeDef * USARTx)**
Function description	Indicate if Auto Baud-Rate Detection mechanism is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 ABREN LL_USART_IsEnabledAutoBaud

LL_USART_SetAutoBaudRateMode

Function name	**__STATIC_INLINE void LL_USART_SetAutoBaudRateMode (USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)**
Function description	Set Auto Baud-Rate mode bits.
Parameters	• **USARTx:** USART Instance • **AutoBaudRateMode:** This parameter can be one of the following

	values: – LL_USART_AUTOBAUD_DETECT_ON_STARTBIT – LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE – LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME – LL_USART_AUTOBAUD_DETECT_ON_55_FRAME
Return values	• **None**
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 ABRMODE LL_USART_SetAutoBaudRateMode

LL_USART_GetAutoBaudRateMode

Function name	**__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (USART_TypeDef * USARTx)**
Function description	Return Auto Baud-Rate mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_AUTOBAUD_DETECT_ON_STARTBIT – LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE – LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME – LL_USART_AUTOBAUD_DETECT_ON_55_FRAME
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 ABRMODE LL_USART_GetAutoBaudRateMode

LL_USART_EnableRxTimeout

Function name	**__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)**
Function description	Enable Receiver Timeout.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name	**__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)**
Function description	Disable Receiver Timeout.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (USART_TypeDef * USARTx)**
Function description	Indicate if Receiver Timeout feature is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name	**__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)**
Function description	Set Address of the USART node.
Parameters	• **USARTx:** USART Instance • **AddressLen:** This parameter can be one of the following values: – LL_USART_ADDRESS_DETECT_4B – LL_USART_ADDRESS_DETECT_7B • **NodeAddress:** 4 or 7 bit Address of the USART node.
Return values	• **None**
Notes	• This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection. • 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on

match)

Reference Manual to LL API cross reference:	• CR2 ADD LL_USART_ConfigNodeAddress • CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name	**__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (USART_TypeDef * USARTx)**
Function description	Return 8 bit Address of the USART node as set in ADD field of CR2.
Parameters	• **USARTx:** USART Instance
Return values	• **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)
Notes	• If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)
Reference Manual to LL API cross reference:	• CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name	**__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (USART_TypeDef * USARTx)**
Function description	Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_ADDRESS_DETECT_4B – LL_USART_ADDRESS_DETECT_7B
Reference Manual to LL API cross reference:	• CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Enable RTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Disable RTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Enable CTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Disable CTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)**
Function description	Configure HW Flow Control mode (both CTS and RTS)
Parameters	• **USARTx:** USART Instance • **HardwareFlowControl:** This parameter can be one of the following values: – LL_USART_HWCONTROL_NONE – LL_USART_HWCONTROL_RTS – LL_USART_HWCONTROL_CTS – LL_USART_HWCONTROL_RTS_CTS
Return values	• **None**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_SetHWFlowCtrl • CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name	**__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Return HW Flow Control configuration (both CTS and RTS)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_HWCONTROL_NONE – LL_USART_HWCONTROL_RTS – LL_USART_HWCONTROL_CTS – LL_USART_HWCONTROL_RTS_CTS
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_GetHWFlowCtrl • CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_EnableOneBitSamp

Function name	**__STATIC_INLINE void LL_USART_EnableOneBitSamp (USART_TypeDef * USARTx)**
Function description	Enable One bit sampling method.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross	• CR3 ONEBIT LL_USART_EnableOneBitSamp

reference:

LL_USART_DisableOneBitSamp

Function name	**__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTx)**
Function description	Disable One bit sampling method.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 ONEBIT LL_USART_DisableOneBitSamp

LL_USART_IsEnabledOneBitSamp

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (USART_TypeDef * USARTx)**
Function description	Indicate if One bit sampling method is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 ONEBIT LL_USART_IsEnabledOneBitSamp

LL_USART_EnableOverrunDetect

Function name	**__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTx)**
Function description	Enable Overrun detection.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_USART_EnableOverrunDetect

LL_USART_DisableOverrunDetect

Function name	**__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTx)**
Function description	Disable Overrun detection.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_USART_DisableOverrunDetect

LL_USART_IsEnabledOverrunDetect

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (USART_TypeDef * USARTx)**
Function description	Indicate if Overrun detection is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_USART_IsEnabledOverrunDetect

LL_USART_SetWKUPType

Function name	**__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTx, uint32_t Type)**
Function description	Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **USARTx:** USART Instance • **Type:** This parameter can be one of the following values: – LL_USART_WAKEUP_ON_ADDRESS – LL_USART_WAKEUP_ON_STARTBIT – LL_USART_WAKEUP_ON_RXNE
Return values	• **None**
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUS LL_USART_SetWKUPType

LL_USART_GetWKUPType

Function name	**__STATIC_INLINE uint32_t LL_USART_GetWKUPType (USART_TypeDef * USARTx)**
Function description	Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_WAKEUP_ON_ADDRESS – LL_USART_WAKEUP_ON_STARTBIT – LL_USART_WAKEUP_ON_RXNE
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR3 WUS LL_USART_GetWKUPType

reference:

LL_USART_SetBaudRate

Function name	**__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t OverSampling, uint32_t BaudRate)**
Function description	Configure USART BRR register for achieving expected Baud Rate value.
Parameters	• **USARTx:** USART Instance • **PeriphClk:** Peripheral Clock • **OverSampling:** This parameter can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8 • **BaudRate:** Baud Rate
Return values	• **None**
Notes	• Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values • Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)
Reference Manual to LL API cross reference:	• BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name	**__STATIC_INLINE uint32_t LL_USART_GetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t OverSampling)**
Function description	Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.
Parameters	• **USARTx:** USART Instance • **PeriphClk:** Peripheral Clock • **OverSampling:** This parameter can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8
Return values	• **Baud:** Rate
Notes	• In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
Reference Manual to LL API cross reference:	• BRR BRR LL_USART_GetBaudRate

LL_USART_SetRxTimeout

Function name	**__STATIC_INLINE void LL_USART_SetRxTimeout (USART_TypeDef * USARTx, uint32_t Timeout)**
Function description	Set Receiver Time Out Value (expressed in nb of bits duration)
Parameters	• **USARTx:** USART Instance • **Timeout:** Value between Min_Data=0x00 and Max_Data=0x00FFFFFF
Return values	• **None**
Reference Manual to LL API cross reference:	• RTOR RTO LL_USART_SetRxTimeout

LL_USART_GetRxTimeout

Function name	**__STATIC_INLINE uint32_t LL_USART_GetRxTimeout (USART_TypeDef * USARTx)**
Function description	Get Receiver Time Out Value (expressed in nb of bits duration)
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x00FFFFFF
Reference Manual to LL API cross reference:	• RTOR RTO LL_USART_GetRxTimeout

LL_USART_SetBlockLength

Function name	**__STATIC_INLINE void LL_USART_SetBlockLength (USART_TypeDef * USARTx, uint32_t BlockLength)**
Function description	Set Block Length value in reception.
Parameters	• **USARTx:** USART Instance • **BlockLength:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• RTOR BLEN LL_USART_SetBlockLength

LL_USART_GetBlockLength

Function name	**__STATIC_INLINE uint32_t LL_USART_GetBlockLength (USART_TypeDef * USARTx)**
Function description	Get Block Length value in reception.
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to	• RTOR BLEN LL_USART_GetBlockLength

LL API cross reference:

LL_USART_EnableIrda

Function name	**__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)**
Function description	Enable IrDA mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name	**__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)**
Function description	Disable IrDA mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (USART_TypeDef * USARTx)**
Function description	Indicate if IrDA mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name	**__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)**
Function description	Configure IrDA Power Mode (Normal or Low Power)
Parameters	• **USARTx:** USART Instance • **PowerMode:** This parameter can be one of the following values: – LL_USART_IRDA_POWER_NORMAL – LL_USART_IRDA_POWER_LOW
Return values	• **None**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name	**__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (USART_TypeDef * USARTx)**
Function description	Retrieve IrDA Power Mode configuration (Normal or Low Power)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_IRDA_POWER_NORMAL – LL_USART_PHASE_2EDGE
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name	**__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)**
Function description	Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	• **USARTx:** USART Instance • **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name	**__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (USART_TypeDef * USARTx)**
Function description	Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	• **USARTx:** USART Instance
Return values	• **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name	**__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Enable Smartcard NACK transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name	**__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Disable Smartcard NACK transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR3 NACK LL_USART_DisableSmartcardNACK

reference:

LL_USART_IsEnabledSmartcardNACK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Indicate if Smartcard NACK transmission is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name	**__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)**
Function description	Enable Smartcard mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name	**__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)**
Function description	Disable Smartcard mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (USART_TypeDef * USARTx)**
Function description	Indicate if Smartcard mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardAutoRetryCount

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)**
Function description	Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	• **USARTx:** USART Instance • **AutoRetryCount:** Value between Min_Data=0 and Max_Data=7
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance. • This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number or erroneous reception trials, before generating a reception error (RXNE and PE bits set)
Reference Manual to LL API cross reference:	• CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount

LL_USART_GetSmartcardAutoRetryCount

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (USART_TypeDef * USARTx)**
Function description	Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)

Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount

LL_USART_SetSmartcardPrescaler

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)**
Function description	Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	• **USARTx:** USART Instance • **PrescalerValue:** Value between Min_Data=0 and Max_Data=31
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (USART_TypeDef * USARTx)**
Function description	Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)**
Function description	Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters	• **USARTx:** USART Instance • **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (USART_TypeDef * USARTx)**
Function description	Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)
Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name	**__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)**
Function description	Enable Single Wire Half-Duplex mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name	**__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)**
Function description	Disable Single Wire Half-Duplex mode.

Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (USART_TypeDef * USARTx)**
Function description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_SetLINBrkDetectionLen

Function name	**__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)**
Function description	Set LIN Break Detection Length.
Parameters	• **USARTx:** USART Instance • **LINBDLength:** This parameter can be one of the following values: – LL_USART_LINBREAK_DETECT_10B – LL_USART_LINBREAK_DETECT_11B
Return values	• **None**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name	**__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (USART_TypeDef * USARTx)**

Function description	Return LIN Break Detection Length.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_LINBREAK_DETECT_10B – LL_USART_LINBREAK_DETECT_11B
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name	**__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)**
Function description	Enable LIN mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name	**__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)**
Function description	Disable LIN mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (USART_TypeDef * USARTx)**
Function description	Indicate if LIN mode is enabled.

Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_SetDEDeassertionTime

Function name	**__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)**
Function description	Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	• **USARTx:** USART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None**
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 DEDT LL_USART_SetDEDeassertionTime

LL_USART_GetDEDeassertionTime

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (USART_TypeDef * USARTx)**
Function description	Return DEDT (Driver Enable De-Assertion Time)
Parameters	• **USARTx:** USART Instance
Return values	• **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 DEDT LL_USART_GetDEDeassertionTime

LL_USART_SetDEAssertionTime

Function name	**__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)**
Function description	Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters	• **USARTx:** USART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None**
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 DEAT LL_USART_SetDEAssertionTime

LL_USART_GetDEAssertionTime

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime (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,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. • Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function • Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigAsyncMode • CR2 CLKEN LL_USART_ConfigAsyncMode • CR3 SCEN LL_USART_ConfigAsyncMode • CR3 IREN LL_USART_ConfigAsyncMode • CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name	**__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Synchronous Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• In Synchronous mode, the following bits must be kept

	cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode. • Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function • Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigSyncMode • CR2 CLKEN LL_USART_ConfigSyncMode • CR3 SCEN LL_USART_ConfigSyncMode • CR3 IREN LL_USART_ConfigSyncMode • CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name	**__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in LIN Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode. • Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function • Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_ConfigLINMode • CR2 STOP LL_USART_ConfigLINMode • CR2 LINEN LL_USART_ConfigLINMode • CR3 IREN LL_USART_ConfigLINMode • CR3 SCEN LL_USART_ConfigLINMode • CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name	**__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Half Duplex Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode. • Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function • Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigHalfDuplexMode • CR2 CLKEN LL_USART_ConfigHalfDuplexMode • CR3 HDSEL LL_USART_ConfigHalfDuplexMode • CR3 SCEN LL_USART_ConfigHalfDuplexMode • CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name	**__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Smartcard Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• In Smartcard mode, the following bits must be kept cleared:

	LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN). • Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function • Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigSmartcardMode • CR2 STOP LL_USART_ConfigSmartcardMode • CR2 CLKEN LL_USART_ConfigSmartcardMode • CR3 HDSEL LL_USART_ConfigSmartcardMode • CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name	**__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Irda Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit). • Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function • Other remaining configurations items related to Irda Mode (as

	Baud Rate, Word length, Power mode, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigIrdaMode • CR2 CLKEN LL_USART_ConfigIrdaMode • CR2 STOP LL_USART_ConfigIrdaMode • CR3 SCEN LL_USART_ConfigIrdaMode • CR3 HDSEL LL_USART_ConfigIrdaMode • CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name	**__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. • Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function • Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigMultiProcessMode • CR2 CLKEN LL_USART_ConfigMultiProcessMode • CR3 SCEN LL_USART_ConfigMultiProcessMode • CR3 HDSEL LL_USART_ConfigMultiProcessMode • CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (USART_TypeDef * USARTx)**
Function description	Check if the USART Parity Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• ISR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (USART_TypeDef * USARTx)**
Function description	Check if the USART Framing Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (USART_TypeDef * USARTx)**
Function description	Check if the USART Noise error detected Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR NF LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (USART_TypeDef * USARTx)**
Function description	Check if the USART OverRun Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (USART_TypeDef * USARTx)**
Function description	Check if the USART IDLE line detected Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• ISR IDLE LL_USART_IsActiveFlag_IDLE

reference:

LL_USART_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE (USART_TypeDef * USARTx)**
Function description	Check if the USART Read Data Register Not Empty 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 RXNE LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmission Complete Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmit Data Register Empty 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 TXE LL_USART_IsActiveFlag_TXE

LL_USART_IsActiveFlag_LBD

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (USART_TypeDef * USARTx)**
Function description	Check if the USART LIN Break Detection Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx

	instance.
Reference Manual to LL API cross reference:	• ISR LBDF LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (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	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR EOBF LL_USART_IsActiveFlag_EOB

LL_USART_IsActiveFlag_ABRE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (USART_TypeDef * USARTx)**
Function description	Check if the USART Auto-Baud Rate Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR ABRE LL_USART_IsActiveFlag_ABRE

LL_USART_IsActiveFlag_ABR

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (USART_TypeDef * USARTx)**
Function description	Check if the USART Auto-Baud Rate Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR ABRF LL_USART_IsActiveFlag_ABR

LL_USART_IsActiveFlag_BUSY

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (USART_TypeDef * USARTx)**
Function description	Check if the USART Busy Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR BUSY LL_USART_IsActiveFlag_BUSY

LL_USART_IsActiveFlag_CM

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (USART_TypeDef * USARTx)**
Function description	Check if the USART Character Match Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMF LL_USART_IsActiveFlag_CM

LL_USART_IsActiveFlag_SBK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (USART_TypeDef * USARTx)**
Function description	Check if the USART Send Break Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SBKF LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (USART_TypeDef * USARTx)**
Function description	Check if the USART Receive Wake Up from mute mode Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RWU LL_USART_IsActiveFlag_RWU

LL_USART_IsActiveFlag_WKUP

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (USART_TypeDef * USARTx)**
Function description	Check if the USART Wake Up from stop mode Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR WUF LL_USART_IsActiveFlag_WKUP

LL_USART_IsActiveFlag_TEACK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmit Enable Acknowledge Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (USART_TypeDef * USARTx)**
Function description	Check if the USART Receive Enable Acknowledge Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_ClearFlag_PE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)**
Function description	Clear Parity Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**

Reference Manual to LL API cross reference:	• ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)**
Function description	Clear Framing Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)**
Function description	Clear Noise detected Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR NCF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)**
Function description	Clear OverRun Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)**
Function description	Clear IDLE line detected Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross	• ICR IDLECF LL_USART_ClearFlag_IDLE

reference:

LL_USART_ClearFlag_TC

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)**
Function description	Clear Transmission Complete Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_LBD

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)**
Function description	Clear LIN Break Detection Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)**
Function description	Clear CTS Interrupt Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_RTO (USART_TypeDef * USARTx)**
Function description	Clear Receiver Time Out Flag.

Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR RTOCF LL_USART_ClearFlag_RTO

LL_USART_ClearFlag_EOB

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_EOB (USART_TypeDef * USARTx)**
Function description	Clear End Of Block Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR EOBCF LL_USART_ClearFlag_EOB

LL_USART_ClearFlag_CM

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_CM (USART_TypeDef * USARTx)**
Function description	Clear Character Match Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• ICR CMCF LL_USART_ClearFlag_CM

LL_USART_ClearFlag_WKUP

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_WKUP (USART_TypeDef * USARTx)**
Function description	Clear Wake Up from stop mode Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR WUCF LL_USART_ClearFlag_WKUP

LL_USART_EnableIT_IDLE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)**
Function description	Enable IDLE Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_RXNE (USART_TypeDef * USARTx)**
Function description	Enable RX Not Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TC

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)**
Function description	Enable Transmission Complete Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TXE (USART_TypeDef * USARTx)**
Function description	Enable TX Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_USART_EnableIT_TXE

LL_USART_EnableIT_PE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)**
Function description	Enable Parity Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name	**__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)**
Function description	Enable Character Match Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name	**__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)**
Function description	Enable Receiver Timeout Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name	**__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)**
Function description	Enable End Of Block Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR1 EOBIE LL_USART_EnableIT_EOB

reference:

LL_USART_EnableIT_LBD

Function name	**__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)**
Function description	Enable LIN Break Detection Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name	**__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)**
Function description	Enable Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name	**__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)**
Function description	Enable CTS Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_EnableIT_WKUP

Function name	**__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)**
Function description	Enable Wake Up from Stop Mode Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_USART_EnableIT_WKUP

LL_USART_DisableIT_IDLE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)**
Function description	Disable IDLE Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_RXNE (USART_TypeDef * USARTx)**
Function description	Disable RX Not Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TC

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)**
Function description	Disable Transmission Complete Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross	• CR1 TCIE LL_USART_DisableIT_TC

reference:

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	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmission Complete Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE (USART_TypeDef * USARTx)**
Function description	Check if the USART TX Empty Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_USART_IsEnabledIT_TXE

LL_USART_IsEnabledIT_PE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (USART_TypeDef * USARTx)**
Function description	Check if the USART Parity Error Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_CM

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (USART_TypeDef * USARTx)**
Function description	Check if the USART Character Match Interrupt is enabled or disabled.

Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 CMIE LL_USART_IsEnabledIT_CM

LL_USART_IsEnabledIT_RTO

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (USART_TypeDef * USARTx)**
Function description	Check if the USART Receiver Timeout Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RTOIE LL_USART_IsEnabledIT_RTO

LL_USART_IsEnabledIT_EOB

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (USART_TypeDef * USARTx)**
Function description	Check if the USART End Of Block Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 EOBIE LL_USART_IsEnabledIT_EOB

LL_USART_IsEnabledIT_LBD

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (USART_TypeDef * USARTx)**
Function description	Check if the USART LIN Break Detection Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR2 LBDIE LL_USART_IsEnabledIT_LBD

reference:

LL_USART_IsEnabledIT_ERROR

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (USART_TypeDef * USARTx)**
Function description	Check if the USART Error Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (USART_TypeDef * USARTx)**
Function description	Check if the USART CTS Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_WKUP

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (USART_TypeDef * USARTx)**
Function description	Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_USART_IsEnabledIT_WKUP

LL_USART_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_USART_EnableDMAReq_RX**

	(USART_TypeDef * USARTx)
Function description	Enable DMA Mode for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Disable DMA Mode for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Check if DMA Mode is enabled for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Enable DMA Mode for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)**

Function description	Disable DMA Mode for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Check if DMA Mode is enabled for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_EnableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_USART_EnableDMADeactOnRxErr (USART_TypeDef * USARTx)**
Function description	Enable DMA Disabling on Reception Error.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_USART_EnableDMADeactOnRxErr

LL_USART_DisableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_USART_DisableDMADeactOnRxErr (USART_TypeDef * USARTx)**
Function description	Disable DMA Disabling on Reception Error.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_USART_DisableDMADeactOnRxErr

LL_USART_IsEnabledDMADeactOnRxErr

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (USART_TypeDef * USARTx)**

Function description	Indicate if DMA Disabling on Reception Error is disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

LL_USART_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (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	**__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	• **USARTx:** USART Instance • **Value:** between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None**
Reference Manual to LL API cross reference:	• TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name	**__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	• **USARTx:** USART Instance • **Value:** between Min_Data=0x00 and Max_Data=0x1FF
Return values	• **None**
Reference Manual to LL API cross reference:	• TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name	**__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)**
Function description	Request an Automatic Baud Rate measurement on next received data frame.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name	**__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)**
Function description	Request Break sending.
Parameters	• **USARTx:** USART Instance

Return values	• **None**
Reference Manual to LL API cross reference:	• RQR SBKRQ LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name	**__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)**
Function description	Put USART in mute mode and set the RWU flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• RQR MMRQ LL_USART_RequestEnterMuteMode

LL_USART_RequestRxDataFlush

Function name	**__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)**
Function description	Request a Receive Data flush.
Parameters	• **USARTx:** USART Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• RQR RXFRQ LL_USART_RequestRxDataFlush

LL_USART_RequestTxDataFlush

Function name	**__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)**
Function description	Request a Transmit data flush.
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:	• RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_DeInit

Function name	**ErrorStatus LL_USART_DeInit (USART_TypeDef * USARTx)**
Function description	De-initialize USART registers (Registers restored to their default values).

Parameters	• **USARTx:** USART Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: USART registers are de-initialized – ERROR: USART registers are not de-initialized

LL_USART_Init

Function name	**ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, LL_USART_InitTypeDef * USART_InitStruct)**
Function description	Initialize USART registers according to the specified parameters in USART_InitStruct.
Parameters	• **USARTx:** USART Instance • **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: USART registers are initialized according to USART_InitStruct content – ERROR: Problem occurred during USART Registers initialization
Notes	• As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART 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	• **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.
Return values	• **None**

LL_USART_ClockInit

Function name	**ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, LL_USART_ClockInitTypeDef * USART_ClockInitStruct)**
Function description	Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.
Parameters	• **USARTx:** USART Instance • **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.
Return values	• **An:** ErrorStatus enumeration value:

- SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content
- ERROR: Problem occurred during USART Registers initialization

Notes	• As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART 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	• **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.
Return values	• **None**

75.3 USART Firmware driver defines

75.3.1 USART

Address Length Detection

LL_USART_ADDRESS_DETECT_4B	4-bit address detection method selected
LL_USART_ADDRESS_DETECT_7B	7-bit address detection (in 8-bit data mode) method selected

Autobaud Detection

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT	Measurement of the start bit is used to detect the baud rate
LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE	Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start10xxxxxx
LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME	0x7F frame detection
LL_USART_AUTOBAUD_DETECT_ON_55_FRAME	0x55 frame detection

Binary Data Inversion

LL_USART_BINARY_LOGIC_POSITIVE	Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)
LL_USART_BINARY_LOGIC_NEGATIVE	Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_USART_BITORDER_LSBFIRST	data is transmitted/received with data bit 0 first, following the start bit
LL_USART_BITORDER_MSBFIRST	data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF	Parity error 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_DISABLE	Clock signal not provided
LL_USART_CLOCK_ENABLE	Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7B	7 bits word length : Start bit, 7 data bits, n stop bits
LL_USART_DATAWIDTH_8B	8 bits word length : Start bit, 8 data bits, n stop bits
LL_USART_DATAWIDTH_9B	9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_USART_DE_POLARITY_HIGH	DE signal is active high
LL_USART_DE_POLARITY_LOW	DE signal is active low

Communication Direction

LL_USART_DIRECTION_NONE	Transmitter and Receiver are disabled
LL_USART_DIRECTION_RX	Transmitter is disabled and Receiver is enabled
LL_USART_DIRECTION_TX	Transmitter is enabled and Receiver is disabled
LL_USART_DIRECTION_TX_RX	Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DATA_TRANSMIT	Get address of data register used for transmission
LL_USART_DMA_REG_DATA_RECEIVE	Get address of data register used for reception

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
LL_USART_ISR_CTSIF	CTS interrupt flag
LL_USART_ISR_CTS	CTS flag
LL_USART_ISR_RTOF	Receiver timeout flag
LL_USART_ISR_EOBF	End of block flag
LL_USART_ISR_ABRE	Auto baud rate error flag
LL_USART_ISR_ABRF	Auto baud rate flag
LL_USART_ISR_BUSY	Busy flag
LL_USART_ISR_CMF	Character match flag
LL_USART_ISR_SBKF	Send break flag
LL_USART_ISR_RWU	Receiver wakeup from Mute mode flag
LL_USART_ISR_WUF	Wakeup from Stop mode flag
LL_USART_ISR_TEACK	Transmit enable acknowledge flag
LL_USART_ISR_REACK	Receive enable acknowledge flag

Hardware Control

LL_USART_HWCONTROL_NONE	CTS and RTS hardware flow control disabled
LL_USART_HWCONTROL_RTS	RTS output enabled, data is only requested when there is space in the receive buffer
LL_USART_HWCONTROL_CTS	CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)
LL_USART_HWCONTROL_RTS_CTS	CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL	IrDA normal power mode
LL_USART_IRDA_POWER_LOW	IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE	IDLE interrupt enable
LL_USART_CR1_RXNEIE	Read data register not empty interrupt enable
LL_USART_CR1_TCIE	Transmission complete interrupt enable
LL_USART_CR1_TXEIE	Transmit data register empty interrupt enable

LL_USART_CR1_PEIE	Parity error
LL_USART_CR1_CMIE	Character match interrupt enable
LL_USART_CR1_RTOIE	Receiver timeout interrupt enable
LL_USART_CR1_EOBIE	End of Block interrupt enable
LL_USART_CR2_LBDIE	LIN break detection interrupt enable
LL_USART_CR3_EIE	Error interrupt enable
LL_USART_CR3_CTSIE	CTS interrupt enable
LL_USART_CR3_WUFIE	Wakeup from Stop mode interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT	The clock pulse of the last data bit is not output to the SCLK pin
LL_USART_LASTCLKPULSE_OUTPUT	The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B	10-bit break detection method selected
LL_USART_LINBREAK_DETECT_11B	11-bit break detection method selected

Oversampling

LL_USART_OVERSAMPLING_16	Oversampling by 16
LL_USART_OVERSAMPLING_8	Oversampling by 8

Parity Control

LL_USART_PARITY_NONE	Parity control disabled
LL_USART_PARITY_EVEN	Parity control enabled and Even Parity is selected
LL_USART_PARITY_ODD	Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE	The first clock transition is the first data capture edge
LL_USART_PHASE_2EDGE	The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW	Steady low value on SCLK pin outside transmission window
LL_USART_POLARITY_HIGH	Steady high value on SCLK pin outside transmission window

RX Pin Active Level Inversion

LL_USART_RXPIN_LEVEL_STANDARD	RX pin signal works using the standard logic levels
LL_USART_RXPIN_LEVEL_INVERTED	RX pin signal values are inverted.

Stop Bits

LL_USART_STOPBITS_0_5	0.5 stop bit
LL_USART_STOPBITS_1	1 stop bit

LL_USART_STOPBITS_1_5	1.5 stop bits
LL_USART_STOPBITS_2	2 stop bits

TX Pin Active Level Inversion

LL_USART_TXPIN_LEVEL_STANDARD	TX pin signal works using the standard logic levels
LL_USART_TXPIN_LEVEL_INVERTED	TX pin signal values are inverted.

TX RX Pins Swap

LL_USART_TXRX_STANDARD	TX/RX pins are used as defined in standard pinout
LL_USART_TXRX_SWAPPED	TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLELINE	USART wake up from Mute mode on Idle Line
LL_USART_WAKEUP_ADDRESSMARK	USART wake up from Mute mode on Address Mark

Wakeup Activation

LL_USART_WAKEUP_ON_ADDRESS	Wake up active on address match
LL_USART_WAKEUP_ON_STARTBIT	Wake up active on Start bit detection
LL_USART_WAKEUP_ON_RXNE	Wake up active on RXNE

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLING16

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __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

76 LL UTILS Generic Driver

76.1 UTILS Firmware driver registers structures

76.1.1 LL_UTILS_PLLInitTypeDef

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_PLL_DIV***This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.

76.1.2 LL_UTILS_ClkInitTypeDef

Data Fields

- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_LL_EC_SYSCLK_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.

76.2 UTILS Firmware driver API description

76.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 32000000 Hz.

This section contains the following APIs:

- ***LL_SetSystemCoreClock()***
- ***LL_PLL_ConfigSystemClock_HSI()***

- *LL_PLL_ConfigSystemClock_HSE()*

76.2.2 Detailed description of functions

LL_GetUID_Word0

Function name	**__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)**
Function description	Get Word0 of the unique device identifier (UID based on 96 bits)
Return values	• **UID[31:0]:**

LL_GetUID_Word1

Function name	**__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)**
Function description	Get Word1 of the unique device identifier (UID based on 96 bits)
Return values	• **UID[63:32]:**

LL_GetUID_Word2

Function name	**__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)**
Function description	Get Word2 of the unique device identifier (UID based on 96 bits)
Return values	• **UID[95:64]:**

LL_GetFlashSize

Function name	**__STATIC_INLINE uint32_t LL_GetFlashSize (void)**
Function description	Get Flash memory size.
Return values	• **FLASH_SIZE[15:0]:** Flash memory size
Notes	• This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.

LL_InitTick

Function name	**__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)**
Function description	This function configures the Cortex-M SysTick source of the timebase.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro) • **Ticks:** Number of ticks
Return values	• **None**
Notes	• When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name	**void LL_Init1msTick (uint32_t HCLKFrequency)**
Function description	This function configures the Cortex-M SysTick source to have 1ms timebase.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz
Return values	• **None**
Notes	• When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service. • HCLK frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_mDelay

Function name	**void LL_mDelay (uint32_t Delay)**
Function description	This function provides accurate delay (in milliseconds) based on SysTick counter flag.
Parameters	• **Delay:** specifies the delay time length, in milliseconds.
Return values	• **None**
Notes	• When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service. • To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name	**void LL_SetSystemCoreClock (uint32_t HCLKFrequency)**
Function description	This function sets directly SystemCoreClock CMSIS variable.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
Return values	• **None**
Notes	• Variable can be calculated also through SystemCoreClockUpdate function.

LL_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	• **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: PLL output frequency = ((HSI frequency * PLLMul) / PLLDiv)PLLMul: The application software must set correctly the PLL multiplication factor to avoid exceeding 96 MHz as PLLVCO when the product is in range 1,48 MHz as PLLVCO when the product is in range 2,24 MHz when the product is in range 3 • FLASH latency can be modified through this function.

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 = 1000000 and Max_Data = 24000000 • **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: PLL output frequency = ((HSE frequency * PLLMul) / PLLDiv)PLLMul: The application software must set correctly the PLL multiplication factor to avoid exceeding 96 MHz as PLLVCO when the product is in range 1,48 MHz as PLLVCO when the product is in range 2,24 MHz when the product is in range 3 • FLASH latency can be modified through this function.

76.3 UTILS Firmware driver defines

76.3.1 UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF	HSE Bypass is not enabled
LL_UTILS_HSEBYPASS_ON	HSE Bypass is enabled

77 LL WWDG Generic Driver

77.1 WWDG Firmware driver API description

77.1.1 Detailed description of functions

LL_WWDG_Enable

Function name	**__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)**
Function description	Enable Window Watchdog.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **None**
Notes	• It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.
Reference Manual to LL API cross reference:	• CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (WWDG_TypeDef * WWDGx)**
Function description	Checks if Window Watchdog is enabled.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name	**__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)**
Function description	Set the Watchdog counter value to provided value (7-bits T[6:0])
Parameters	• **WWDGx:** WWDG Instance • **Counter:** 0..0x7F (7 bit counter value)
Return values	• **None**
Notes	• When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6

	becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)
Reference Manual to LL API cross reference:	• CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name	**__STATIC_INLINE uint32_t LL_WWDG_GetCounter (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 timebase 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 x 2expWDGTB) PCLK cycles
Reference Manual to LL API cross reference:	• CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (WWDG_TypeDef * WWDGx)**
Function description	Return current Watchdog Prescaler Value.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **Returned:** value can be one of the following values: – LL_WWDG_PRESCALER_1 – LL_WWDG_PRESCALER_2 – LL_WWDG_PRESCALER_4

	– LL_WWDG_PRESCALER_8
Reference Manual to LL API cross reference:	• CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name	**__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)**
Function description	Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).
Parameters	• **WWDGx:** WWDG Instance • **Window:** 0x00..0x7F (7 bit Window value)
Return values	• **None**
Notes	• This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.
Reference Manual to LL API cross reference:	• CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name	**__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)**
Function description	Return current Watchdog Window Value (7 bits value)
Parameters	• **WWDGx:** WWDG Instance
Return values	• **7:** bit Watchdog Window value
Reference Manual to LL API cross reference:	• CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name	**__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)**
Function description	Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **State:** of bit (1 or 0).
Notes	• This bit is set by hardware when the counter has reached the

	value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.
Reference Manual to LL API cross reference:	• SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name	**__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)**
Function description	Clear WWDG Early Wakeup Interrupt Flag (EWIF)
Parameters	• **WWDGx:** WWDG Instance
Return values	• **None**
Reference Manual to LL API cross reference:	• SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name	**__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)**
Function description	Enable the Early Wakeup Interrupt.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **None**
Notes	• When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset
Reference Manual to LL API cross reference:	• CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name	**__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (WWDG_TypeDef * WWDGx)**
Function description	Check if Early Wakeup Interrupt is enabled.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CFR EWI LL_WWDG_IsEnabledIT_EWKUP

77.2 WWDG Firmware driver defines

77.2.1 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

78 Correspondence between API registers and API low-layer driver functions

78.1 ADC

Table 25: Correspondence between ADC registers and ADC low-layer driver functions

Register	Field	Function
CALFACT	CALFACT	LL_ADC_GetCalibrationFactor
		LL_ADC_SetCalibrationFactor
CCR	LFMEN	LL_ADC_GetCommonFrequencyMode
		LL_ADC_SetCommonFrequencyMode
	PRESC	LL_ADC_GetCommonClock
		LL_ADC_SetCommonClock
	TSEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
	VLCDEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
	VREFEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
CFGR1	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

Register	Field	Function
	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
	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
CFGR2	CKMODE	LL_ADC_GetClock
		LL_ADC_SetClock
	OVSE	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	OVSR	LL_ADC_ConfigOverSamplingRatioShift
		LL_ADC_GetOverSamplingRatio
	OVSS	LL_ADC_ConfigOverSamplingRatioShift
		LL_ADC_GetOverSamplingShift
	TOVS	LL_ADC_GetOverSamplingDiscont
		LL_ADC_SetOverSamplingDiscont
CHSELR	CHSEL0	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL1	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL10	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem

Register	Field	Function
		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
		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

Register	Field	Function
		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
		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*

Register	Field	Function
		LL_ADC_REG_StartConversion
	ADSTP	LL_ADC_REG_IsStopConversionOngoing
		LL_ADC_REG_StopConversion
	ADVREGEN	LL_ADC_DisableInternalRegulator
		LL_ADC_EnableInternalRegulator
		LL_ADC_IsInternalRegulatorEnabled
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
	EOCALIE	LL_ADC_DisableIT_EOCAL
		LL_ADC_EnableIT_EOCAL
		LL_ADC_IsEnabledIT_EOCAL
	EOCIE	LL_ADC_DisableIT_EOC
		LL_ADC_EnableIT_EOC
		LL_ADC_IsEnabledIT_EOC
	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
ISR	ADRDY	LL_ADC_ClearFlag_ADRDY
		LL_ADC_IsActiveFlag_ADRDY
	AWD	LL_ADC_ClearFlag_AWD1

Register	Field	Function
		LL_ADC_IsActiveFlag_AWD1
	EOC	*LL_ADC_ClearFlag_EOC*
		LL_ADC_IsActiveFlag_EOC
	EOCAL	*LL_ADC_ClearFlag_EOCAL*
		LL_ADC_IsActiveFlag_EOCAL
	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_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT	*LL_ADC_ConfigAnalogWDThresholds*
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds

78.2 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
	CRYPEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMAEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	MIFEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	RNGEN	*LL_AHB1_GRP1_DisableClock*

Register	Field	Function
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	TSCEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
AHBRSTR	CRCRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	CRYPRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	DMARST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	MIFRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	RNGRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	TSCRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
AHBSMENR	CRCSMEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
	CRYPSMEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
	DMASMEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
	MIFSMEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
	RNGSMEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
	SRAMSMEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
	TSCSMEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
APB1ENR	CRSEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	DACEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock

Register	Field	Function
		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
	I2C3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	LCDEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	LPTIM1EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	LPUART1EN	*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
	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
	TIM7EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock

Register	Field	Function
		LL_APB1_GRP1_IsEnabledClock
	USART2EN	*LL_APB1_GRP1_DisableClock*
		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
	USBEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	WWDGEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
APB1RSTR	CRSRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	DACRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	I2C1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	I2C2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	I2C3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	LCDRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	LPTIM1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	LPUART1RST	*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

Register	Field	Function
	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
	TIM7RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USART2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USART4RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	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
APB1SMENR	CRSSMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	DACSMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	I2C1SMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	I2C2SMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	I2C3SMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	LCDSMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	LPTIM1SMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	LPUART1SMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep
	PWRSMEN	*LL_APB1_GRP1_DisableClockSleep*
		LL_APB1_GRP1_EnableClockSleep

Register	Field	Function
	SPI2SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	TIM2SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	TIM3SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	TIM6SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	TIM7SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	USART2SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	USART4SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	USART5SMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	USBSMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
	WWDGSMEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
APB2ENR	ADCEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	DBGEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	FWEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SPI1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SYSCFGEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM21EN	LL_APB2_GRP1_DisableClock

Register	Field	Function
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM22EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	USART1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
APB2RSTR	ADCRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	DBGRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SPI1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SYSCFGRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM21RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM22RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	USART1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
APB2SMENR	ADCSMEN	*LL_APB2_GRP1_DisableClockSleep*
		LL_APB2_GRP1_EnableClockSleep
	DBGSMEN	*LL_APB2_GRP1_DisableClockSleep*
		LL_APB2_GRP1_EnableClockSleep
	SPI1SMEN	*LL_APB2_GRP1_DisableClockSleep*
		LL_APB2_GRP1_EnableClockSleep
	SYSCFGSMEN	*LL_APB2_GRP1_DisableClockSleep*
		LL_APB2_GRP1_EnableClockSleep
	TIM21SMEN	*LL_APB2_GRP1_DisableClockSleep*
		LL_APB2_GRP1_EnableClockSleep
	TIM22SMEN	*LL_APB2_GRP1_DisableClockSleep*
		LL_APB2_GRP1_EnableClockSleep
	USART1SMEN	*LL_APB2_GRP1_DisableClockSleep*
		LL_APB2_GRP1_EnableClockSleep

Register	Field	Function
IOPENR	GPIOAEN	*LL_IOP_GRP1_DisableClock*
		LL_IOP_GRP1_EnableClock
		LL_IOP_GRP1_IsEnabledClock
	GPIOBEN	*LL_IOP_GRP1_DisableClock*
		LL_IOP_GRP1_EnableClock
		LL_IOP_GRP1_IsEnabledClock
	GPIOCEN	*LL_IOP_GRP1_DisableClock*
		LL_IOP_GRP1_EnableClock
		LL_IOP_GRP1_IsEnabledClock
	GPIODEN	*LL_IOP_GRP1_DisableClock*
		LL_IOP_GRP1_EnableClock
		LL_IOP_GRP1_IsEnabledClock
	GPIOEEN	*LL_IOP_GRP1_DisableClock*
		LL_IOP_GRP1_EnableClock
		LL_IOP_GRP1_IsEnabledClock
	GPIOHEN	*LL_IOP_GRP1_DisableClock*
		LL_IOP_GRP1_EnableClock
		LL_IOP_GRP1_IsEnabledClock
IOPRSTR	GPIOASMEN	*LL_IOP_GRP1_ForceReset*
		LL_IOP_GRP1_ReleaseReset
	GPIOBSMEN	*LL_IOP_GRP1_ForceReset*
		LL_IOP_GRP1_ReleaseReset
	GPIOCSMEN	*LL_IOP_GRP1_ForceReset*
		LL_IOP_GRP1_ReleaseReset
	GPIODSMEN	*LL_IOP_GRP1_ForceReset*
		LL_IOP_GRP1_ReleaseReset
	GPIOESMEN	*LL_IOP_GRP1_ForceReset*
		LL_IOP_GRP1_ReleaseReset
	GPIOHSMEN	*LL_IOP_GRP1_ForceReset*
		LL_IOP_GRP1_ReleaseReset
IOPSMENR	GPIOARST	*LL_IOP_GRP1_DisableClockSleep*
		LL_IOP_GRP1_EnableClockSleep
	GPIOBRST	*LL_IOP_GRP1_DisableClockSleep*
		LL_IOP_GRP1_EnableClockSleep
	GPIOCRST	*LL_IOP_GRP1_DisableClockSleep*
		LL_IOP_GRP1_EnableClockSleep

Register	Field	Function
	GPIODRST	*LL_IOP_GRP1_DisableClockSleep*
		LL_IOP_GRP1_EnableClockSleep
	GPIOERST	*LL_IOP_GRP1_DisableClockSleep*
		LL_IOP_GRP1_EnableClockSleep
	GPIOHRST	*LL_IOP_GRP1_DisableClockSleep*
		LL_IOP_GRP1_EnableClockSleep

78.3 COMP

Table 27: Correspondence between COMP registers and COMP low-layer driver functions

Register	Field	Function
COMP	COMP1POLARITY	*LL_COMP_GetOutputPolarity*
		LL_COMP_SetOutputPolarity
COMP1_CSR	COMP1EN	*LL_COMP_Disable*
		LL_COMP_Enable
		LL_COMP_IsEnabled
	COMP1INNSEL	*LL_COMP_GetInputMinus*
		LL_COMP_SetInputMinus
	COMP1LOCK	*LL_COMP_IsLocked*
		LL_COMP_Lock
	COMP1LPTIMIN1	*LL_COMP_GetOutputLPTIM*
		LL_COMP_SetOutputLPTIM
	COMP1VALUE	*LL_COMP_ReadOutputLevel*
	COMP1WM	*LL_COMP_GetCommonWindowMode*
		LL_COMP_SetCommonWindowMode
COMP2_CSR	COMP2EN	*LL_COMP_Disable*
		LL_COMP_Enable
		LL_COMP_IsEnabled
	COMP2INNSEL	*LL_COMP_ConfigInputs*
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	COMP2INPSEL	*LL_COMP_ConfigInputs*
		LL_COMP_GetInputPlus
		LL_COMP_SetInputPlus
	COMP2LOCK	*LL_COMP_IsLocked*
		LL_COMP_Lock
	COMP2LPTIMIN1	*LL_COMP_GetOutputLPTIM*

Register	Field	Function
		LL_COMP_SetOutputLPTIM
	COMP2LPTIMIN2	LL_COMP_GetOutputLPTIM
		LL_COMP_SetOutputLPTIM
	COMP2SPEED	LL_COMP_GetPowerMode
		LL_COMP_SetPowerMode
	COMP2VALUE	LL_COMP_ReadOutputLevel

78.4 CORTEX

Table 28: Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
MPU_CTRL	ENABLE	LL_MPU_Disable
		LL_MPU_Enable
		LL_MPU_IsEnabled
MPU_RASR	AP	LL_MPU_ConfigRegion
	B	LL_MPU_ConfigRegion
	C	LL_MPU_ConfigRegion
	ENABLE	LL_MPU_DisableRegion
		LL_MPU_EnableRegion
	S	LL_MPU_ConfigRegion
	SIZE	LL_MPU_ConfigRegion
	XN	LL_MPU_ConfigRegion
MPU_RBAR	ADDR	LL_MPU_ConfigRegion
	REGION	LL_MPU_ConfigRegion
MPU_RNR	REGION	LL_MPU_ConfigRegion
		LL_MPU_DisableRegion
SCB_CPUID	ARCHITECTURE	LL_CPUID_GetArchitecture
	IMPLEMENTER	LL_CPUID_GetImplementer
	PARTNO	LL_CPUID_GetParNo
	REVISION	LL_CPUID_GetRevision
	VARIANT	LL_CPUID_GetVariant
SCB_SCR	SEVEONPEND	LL_LPM_DisableEventOnPend
		LL_LPM_EnableEventOnPend
	SLEEPDEEP	LL_LPM_EnableDeepSleep
		LL_LPM_EnableSleep
	SLEEPONEXIT	LL_LPM_DisableSleepOnExit
		LL_LPM_EnableSleepOnExit

Register	Field	Function
STK_CTRL	CLKSOURCE	*LL_SYSTICK_GetClkSource*
		LL_SYSTICK_SetClkSource
	COUNTFLAG	*LL_SYSTICK_IsActiveCounterFlag*
	TICKINT	*LL_SYSTICK_DisableIT*
		LL_SYSTICK_EnableIT
		LL_SYSTICK_IsEnabledIT

78.5 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

78.6 CRS

Table 30: Correspondence between CRS registers and CRS low-layer driver functions

Register	Field	Function
CFGR	FELIM	*LL_CRS_ConfigSynchronization*

Register	Field	Function
		LL_CRS_GetFreqErrorLimit
		LL_CRS_SetFreqErrorLimit
	RELOAD	LL_CRS_ConfigSynchronization
		LL_CRS_GetReloadCounter
		LL_CRS_SetReloadCounter
	SYNCDIV	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncDivider
		LL_CRS_SetSyncDivider
	SYNCPOL	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncPolarity
		LL_CRS_SetSyncPolarity
	SYNCSRC	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncSignalSource
		LL_CRS_SetSyncSignalSource
CR	AUTOTRIMEN	LL_CRS_DisableAutoTrimming
		LL_CRS_EnableAutoTrimming
		LL_CRS_IsEnabledAutoTrimming
	CEN	LL_CRS_DisableFreqErrorCounter
		LL_CRS_EnableFreqErrorCounter
		LL_CRS_IsEnabledFreqErrorCounter
	ERRIE	LL_CRS_DisableIT_ERR
		LL_CRS_EnableIT_ERR
		LL_CRS_IsEnabledIT_ERR
	ESYNCIE	LL_CRS_DisableIT_ESYNC
		LL_CRS_EnableIT_ESYNC
		LL_CRS_IsEnabledIT_ESYNC
	SWSYNC	LL_CRS_GenerateEvent_SWSYNC
	SYNCOKIE	LL_CRS_DisableIT_SYNCOK
		LL_CRS_EnableIT_SYNCOK
		LL_CRS_IsEnabledIT_SYNCOK
	SYNCWARNIE	LL_CRS_DisableIT_SYNCWARN
		LL_CRS_EnableIT_SYNCWARN
		LL_CRS_IsEnabledIT_SYNCWARN
	TRIM	LL_CRS_ConfigSynchronization
		LL_CRS_GetHSI48SmoothTrimming
		LL_CRS_SetHSI48SmoothTrimming

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*

78.7 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

Register	Field	Function
	EN2	*LL_DAC_Disable*
		LL_DAC_Enable
		LL_DAC_IsEnabled
	MAMP1	*LL_DAC_GetWaveNoiseLFSR*
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MAMP2	*LL_DAC_GetWaveNoiseLFSR*
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	TEN1	*LL_DAC_DisableTrigger*
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TEN2	*LL_DAC_DisableTrigger*
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TSEL1	*LL_DAC_GetTriggerSource*
		LL_DAC_SetTriggerSource
	TSEL2	*LL_DAC_GetTriggerSource*
		LL_DAC_SetTriggerSource
	WAVE1	*LL_DAC_GetWaveAutoGeneration*
		LL_DAC_SetWaveAutoGeneration
	WAVE2	*LL_DAC_GetWaveAutoGeneration*
		LL_DAC_SetWaveAutoGeneration
DHR12L1	DACC1DHR	*LL_DAC_ConvertData12LeftAligned*
		LL_DAC_DMA_GetRegAddr
DHR12L2	DACC2DHR	*LL_DAC_ConvertData12LeftAligned*
		LL_DAC_DMA_GetRegAddr
DHR12LD	DACC1DHR	*LL_DAC_ConvertDualData12LeftAligned*
	DACC2DHR	*LL_DAC_ConvertDualData12LeftAligned*
DHR12R1	DACC1DHR	*LL_DAC_ConvertData12RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR12R2	DACC2DHR	*LL_DAC_ConvertData12RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR12RD	DACC1DHR	*LL_DAC_ConvertDualData12RightAligned*

Register	Field	Function
	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*
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*

78.8 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

Register	Field	Function
		LL_DMA_SetMemoryIncMode
	MSIZE	LL_DMA_ConfigTransfer
		LL_DMA_GetMemorySize
		LL_DMA_SetMemorySize
	PINC	LL_DMA_ConfigTransfer
		LL_DMA_GetPeriphIncMode
		LL_DMA_SetPeriphIncMode
	PL	LL_DMA_ConfigTransfer
		LL_DMA_GetChannelPriorityLevel
		LL_DMA_SetChannelPriorityLevel
	PSIZE	LL_DMA_ConfigTransfer
		LL_DMA_GetPeriphSize
		LL_DMA_SetPeriphSize
	TCIE	LL_DMA_DisableIT_TC
		LL_DMA_EnableIT_TC
		LL_DMA_IsEnabledIT_TC
	TEIE	LL_DMA_DisableIT_TE
		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

Register	Field	Function
		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*
	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*

Register	Field	Function
	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*
	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*

78.9 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

Register	Field	Function
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*

78.10 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
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

Register	Field	Function
OTYPER	OTy	LL_GPIO_GetPinOutputType
		LL_GPIO_SetPinOutputType
PUPDR	PUPDy	LL_GPIO_GetPinPull
		LL_GPIO_SetPinPull

78.11 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
	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

Register	Field	Function
	PECEN	LL_I2C_DisableSMBusPEC
		LL_I2C_EnableSMBusPEC
		LL_I2C_IsEnabledSMBusPEC
	RXDMAEN	LL_I2C_DisableDMAReq_RX
		LL_I2C_EnableDMAReq_RX
		LL_I2C_IsEnabledDMAReq_RX
	RXIE	LL_I2C_DisableIT_RX
		LL_I2C_EnableIT_RX
		LL_I2C_IsEnabledIT_RX
	SBC	LL_I2C_DisableSlaveByteControl
		LL_I2C_EnableSlaveByteControl
		LL_I2C_IsEnabledSlaveByteControl
	SMBDEN	LL_I2C_GetMode
		LL_I2C_SetMode
	SMBHEN	LL_I2C_GetMode
		LL_I2C_SetMode
	STOPIE	LL_I2C_DisableIT_STOP
		LL_I2C_EnableIT_STOP
		LL_I2C_IsEnabledIT_STOP
	TCIE	LL_I2C_DisableIT_TC
		LL_I2C_EnableIT_TC
		LL_I2C_IsEnabledIT_TC
	TXDMAEN	LL_I2C_DisableDMAReq_TX
		LL_I2C_EnableDMAReq_TX
		LL_I2C_IsEnabledDMAReq_TX
	TXIE	LL_I2C_DisableIT_TX
		LL_I2C_EnableIT_TX
		LL_I2C_IsEnabledIT_TX
	WUPEN	LL_I2C_DisableWakeUpFromStop
		LL_I2C_EnableWakeUpFromStop
		LL_I2C_IsEnabledWakeUpFromStop
CR2	ADD10	LL_I2C_GetMasterAddressingMode
		LL_I2C_HandleTransfer
		LL_I2C_SetMasterAddressingMode
	AUTOEND	LL_I2C_DisableAutoEndMode
		LL_I2C_EnableAutoEndMode

Register	Field	Function
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAutoEndMode
	HEAD10R	LL_I2C_DisableAuto10BitRead
		LL_I2C_EnableAuto10BitRead
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAuto10BitRead
	NACK	LL_I2C_AcknowledgeNextData
	NBYTES	LL_I2C_GetTransferSize
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferSize
	PECBYTE	LL_I2C_EnableSMBusPECCompare
		LL_I2C_IsEnabledSMBusPECCompare
	RD_WRN	LL_I2C_GetTransferRequest
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferRequest
	RELOAD	LL_I2C_DisableReloadMode
		LL_I2C_EnableReloadMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledReloadMode
	SADD	LL_I2C_GetSlaveAddr
		LL_I2C_HandleTransfer
		LL_I2C_SetSlaveAddr
	START	LL_I2C_GenerateStartCondition
		LL_I2C_HandleTransfer
	STOP	LL_I2C_GenerateStopCondition
		LL_I2C_HandleTransfer
ICR	ADDRCF	LL_I2C_ClearFlag_ADDR
	ALERTCF	LL_I2C_ClearSMBusFlag_ALERT
	ARLOCF	LL_I2C_ClearFlag_ARLO
	BERRCF	LL_I2C_ClearFlag_BERR
	NACKCF	LL_I2C_ClearFlag_NACK
	OVRCF	LL_I2C_ClearFlag_OVR
	PECCF	LL_I2C_ClearSMBusFlag_PECERR
	STOPCF	LL_I2C_ClearFlag_STOP
	TIMOUTCF	LL_I2C_ClearSMBusFlag_TIMEOUT
ISR	ADDCODE	LL_I2C_GetAddressMatchCode

Register	Field	Function
	ADDR	LL_I2C_IsActiveFlag_ADDR
	ALERT	LL_I2C_IsActiveSMBusFlag_ALERT
	ARLO	LL_I2C_IsActiveFlag_ARLO
	BERR	LL_I2C_IsActiveFlag_BERR
	BUSY	LL_I2C_IsActiveFlag_BUSY
	DIR	LL_I2C_GetTransferDirection
	NACKF	LL_I2C_IsActiveFlag_NACK
	OVR	LL_I2C_IsActiveFlag_OVR
	PECERR	LL_I2C_IsActiveSMBusFlag_PECERR
	RXNE	LL_I2C_IsActiveFlag_RXNE
	STOPF	LL_I2C_IsActiveFlag_STOP
	TC	LL_I2C_IsActiveFlag_TC
	TCR	LL_I2C_IsActiveFlag_TCR
	TIMEOUT	LL_I2C_IsActiveSMBusFlag_TIMEOUT
	TXE	LL_I2C_ClearFlag_TXE
		LL_I2C_IsActiveFlag_TXE
	TXIS	LL_I2C_IsActiveFlag_TXIS
OAR1	OA1	LL_I2C_SetOwnAddress1
	OA1EN	LL_I2C_DisableOwnAddress1
		LL_I2C_EnableOwnAddress1
		LL_I2C_IsEnabledOwnAddress1
	OA1MODE	LL_I2C_SetOwnAddress1
OAR2	OA2	LL_I2C_SetOwnAddress2
	OA2EN	LL_I2C_DisableOwnAddress2
		LL_I2C_EnableOwnAddress2
		LL_I2C_IsEnabledOwnAddress2
	OA2MSK	LL_I2C_SetOwnAddress2
PECR	PEC	LL_I2C_GetSMBusPEC
RXDR	RXDATA	LL_I2C_DMA_GetRegAddr
		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
	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

78.12 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

Register	Field	Function
DR	DR	*LL_I2S_ReceiveData16*
		LL_I2S_TransmitData16
I2SCFGR	ASTRTEN	*LL_I2S_DisableAsyncStart*
		LL_I2S_EnableAsyncStart
		LL_I2S_IsEnabledAsyncStart
	CHLEN	*LL_I2S_GetDataFormat*
		LL_I2S_SetDataFormat
	CKPOL	*LL_I2S_GetClockPolarity*
		LL_I2S_SetClockPolarity
	DATLEN	*LL_I2S_GetDataFormat*
		LL_I2S_SetDataFormat
	I2SCFG	*LL_I2S_GetTransferMode*
		LL_I2S_SetTransferMode
	I2SE	*LL_I2S_Disable*
		LL_I2S_Enable
		LL_I2S_IsEnabled
	I2SMOD	*LL_I2S_Enable*
	I2SSTD	*LL_I2S_GetStandard*
		LL_I2S_SetStandard
	PCMSYNC	*LL_I2S_GetStandard*
		LL_I2S_SetStandard
I2SPR	I2SDIV	*LL_I2S_GetPrescalerLinear*
		LL_I2S_SetPrescalerLinear
	MCKOE	*LL_I2S_DisableMasterClock*
		LL_I2S_EnableMasterClock
		LL_I2S_IsEnabledMasterClock
	ODD	*LL_I2S_GetPrescalerParity*
		LL_I2S_SetPrescalerParity
SR	BSY	*LL_I2S_IsActiveFlag_BSY*
	CHSIDE	*LL_I2S_IsActiveFlag_CHSIDE*
	FRE	*LL_I2S_ClearFlag_FRE*
		LL_I2S_IsActiveFlag_FRE
	OVR	*LL_I2S_ClearFlag_OVR*
		LL_I2S_IsActiveFlag_OVR
	RXNE	*LL_I2S_IsActiveFlag_RXNE*
	TXE	*LL_I2S_IsActiveFlag_TXE*

Register	Field	Function
	UDR	*LL_I2S_ClearFlag_UDR*
		LL_I2S_IsActiveFlag_UDR

78.13 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
		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

78.14 LPTIM

Table 38: Correspondence between LPTIM registers and LPTIM low-layer driver functions

Register	Field	Function
ARR	ARR	*LL_LPTIM_GetAutoReload*
		LL_LPTIM_SetAutoReload
CFGR	CKFLT	*LL_LPTIM_ConfigClock*
		LL_LPTIM_GetClockFilter
	CKPOL	*LL_LPTIM_ConfigClock*
		LL_LPTIM_GetClockPolarity
		LL_LPTIM_GetEncoderMode
		LL_LPTIM_SetEncoderMode
	CKSEL	*LL_LPTIM_GetClockSource*

Register	Field	Function
		LL_LPTIM_SetClockSource
	COUNTMODE	LL_LPTIM_GetCounterMode
		LL_LPTIM_SetCounterMode
	ENC	LL_LPTIM_DisableEncoderMode
		LL_LPTIM_EnableEncoderMode
		LL_LPTIM_IsEnabledEncoderMode
	PRELOAD	LL_LPTIM_GetUpdateMode
		LL_LPTIM_SetUpdateMode
	PRESC	LL_LPTIM_GetPrescaler
		LL_LPTIM_SetPrescaler
	TIMOUT	LL_LPTIM_DisableTimeout
		LL_LPTIM_EnableTimeout
		LL_LPTIM_IsEnabledTimeout
	TRGFLT	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerFilter
	TRIGEN	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerPolarity
		LL_LPTIM_TrigSw
	TRIGSEL	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerSource
	WAVE	LL_LPTIM_ConfigOutput
		LL_LPTIM_GetWaveform
		LL_LPTIM_SetWaveform
	WAVPOL	LL_LPTIM_ConfigOutput
		LL_LPTIM_GetPolarity
		LL_LPTIM_SetPolarity
CMP	CMP	LL_LPTIM_GetCompare
		LL_LPTIM_SetCompare
CNT	CNT	LL_LPTIM_GetCounter
CR	CNTSTRT	LL_LPTIM_StartCounter
	ENABLE	LL_LPTIM_Disable
		LL_LPTIM_Enable
		LL_LPTIM_IsEnabled
	SNGSTRT	LL_LPTIM_StartCounter
ICR	ARRMCF	LL_LPTIM_ClearFLAG_ARRM
	ARROKCF	LL_LPTIM_ClearFlag_ARROK

Register	Field	Function
	CMPMCF	*LL_LPTIM_ClearFLAG_CMPM*
	CMPOKCF	*LL_LPTIM_ClearFlag_CMPOK*
	DOWNCF	*LL_LPTIM_ClearFlag_DOWN*
	EXTTRIGCF	*LL_LPTIM_ClearFlag_EXTTRIG*
	UPCF	*LL_LPTIM_ClearFlag_UP*
IER	ARRMIE	*LL_LPTIM_DisableIT_ARRM*
		LL_LPTIM_EnableIT_ARRM
		LL_LPTIM_IsEnabledIT_ARRM
	ARROKIE	*LL_LPTIM_DisableIT_ARROK*
		LL_LPTIM_EnableIT_ARROK
		LL_LPTIM_IsEnabledIT_ARROK
	CMPMIE	*LL_LPTIM_DisableIT_CMPM*
		LL_LPTIM_EnableIT_CMPM
		LL_LPTIM_IsEnabledIT_CMPM
	CMPOKIE	*LL_LPTIM_DisableIT_CMPOK*
		LL_LPTIM_EnableIT_CMPOK
		LL_LPTIM_IsEnabledIT_CMPOK
	DOWNIE	*LL_LPTIM_DisableIT_DOWN*
		LL_LPTIM_EnableIT_DOWN
		LL_LPTIM_IsEnabledIT_DOWN
	EXTTRIGIE	*LL_LPTIM_DisableIT_EXTTRIG*
		LL_LPTIM_EnableIT_EXTTRIG
		LL_LPTIM_IsEnabledIT_EXTTRIG
	UPIE	*LL_LPTIM_DisableIT_UP*
		LL_LPTIM_EnableIT_UP
		LL_LPTIM_IsEnabledIT_UP
ISR	ARRM	*LL_LPTIM_IsActiveFlag_ARRM*
	ARROK	*LL_LPTIM_IsActiveFlag_ARROK*
	CMPM	*LL_LPTIM_IsActiveFlag_CMPM*
	CMPOK	*LL_LPTIM_IsActiveFlag_CMPOK*
	DOWN	*LL_LPTIM_IsActiveFlag_DOWN*
	EXTTRIG	*LL_LPTIM_IsActiveFlag_EXTTRIG*
	UP	*LL_LPTIM_IsActiveFlag_UP*

78.15 LPUART

Table 39: Correspondence between LPUART registers and LPUART low-layer driver functions

Register	Field	Function
BRR	BRR	*LL_LPUART_GetBaudRate*
		LL_LPUART_SetBaudRate
CR1	CMIE	*LL_LPUART_DisableIT_CM*
		LL_LPUART_EnableIT_CM
		LL_LPUART_IsEnabledIT_CM
	DEAT	*LL_LPUART_GetDEAssertionTime*
		LL_LPUART_SetDEAssertionTime
	DEDT	*LL_LPUART_GetDEDeassertionTime*
		LL_LPUART_SetDEDeassertionTime
	IDLEIE	*LL_LPUART_DisableIT_IDLE*
		LL_LPUART_EnableIT_IDLE
		LL_LPUART_IsEnabledIT_IDLE
	M	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetDataWidth
		LL_LPUART_SetDataWidth
	MME	*LL_LPUART_DisableMuteMode*
		LL_LPUART_EnableMuteMode
		LL_LPUART_IsEnabledMuteMode
	PCE	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	PEIE	*LL_LPUART_DisableIT_PE*
		LL_LPUART_EnableIT_PE
		LL_LPUART_IsEnabledIT_PE
	PS	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	RE	*LL_LPUART_DisableDirectionRx*
		LL_LPUART_EnableDirectionRx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	RXNEIE	*LL_LPUART_DisableIT_RXNE*
		LL_LPUART_EnableIT_RXNE
		LL_LPUART_IsEnabledIT_RXNE

Register	Field	Function
	TCIE	*LL_LPUART_DisableIT_TC*
		LL_LPUART_EnableIT_TC
		LL_LPUART_IsEnabledIT_TC
	TE	*LL_LPUART_DisableDirectionTx*
		LL_LPUART_EnableDirectionTx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	TXEIE	*LL_LPUART_DisableIT_TXE*
		LL_LPUART_EnableIT_TXE
		LL_LPUART_IsEnabledIT_TXE
	UE	*LL_LPUART_Disable*
		LL_LPUART_Enable
		LL_LPUART_IsEnabled
	UESM	*LL_LPUART_DisableInStopMode*
		LL_LPUART_EnableInStopMode
		LL_LPUART_IsEnabledInStopMode
	WAKE	*LL_LPUART_GetWakeUpMethod*
		LL_LPUART_SetWakeUpMethod
CR2	ADD	*LL_LPUART_ConfigNodeAddress*
		LL_LPUART_GetNodeAddress
	ADDM7	*LL_LPUART_ConfigNodeAddress*
		LL_LPUART_GetNodeAddressLen
	DATAINV	*LL_LPUART_GetBinaryDataLogic*
		LL_LPUART_SetBinaryDataLogic
	MSBFIRST	*LL_LPUART_GetTransferBitOrder*
		LL_LPUART_SetTransferBitOrder
	RXINV	*LL_LPUART_GetRXPinLevel*
		LL_LPUART_SetRXPinLevel
	STOP	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetStopBitsLength
		LL_LPUART_SetStopBitsLength
	SWAP	*LL_LPUART_GetTXRXSwap*
		LL_LPUART_SetTXRXSwap
	TXINV	*LL_LPUART_GetTXPinLevel*
		LL_LPUART_SetTXPinLevel
CR3	CTSE	*LL_LPUART_DisableCTSHWFlowCtrl*

Register	Field	Function
		LL_LPUART_EnableCTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	CTSIE	*LL_LPUART_DisableIT_CTS*
		LL_LPUART_EnableIT_CTS
		LL_LPUART_IsEnabledIT_CTS
	DDRE	*LL_LPUART_DisableDMADeactOnRxErr*
		LL_LPUART_EnableDMADeactOnRxErr
		LL_LPUART_IsEnabledDMADeactOnRxErr
	DEM	*LL_LPUART_DisableDEMode*
		LL_LPUART_EnableDEMode
		LL_LPUART_IsEnabledDEMode
	DEP	*LL_LPUART_GetDESignalPolarity*
		LL_LPUART_SetDESignalPolarity
	DMAR	*LL_LPUART_DisableDMAReq_RX*
		LL_LPUART_EnableDMAReq_RX
		LL_LPUART_IsEnabledDMAReq_RX
	DMAT	*LL_LPUART_DisableDMAReq_TX*
		LL_LPUART_EnableDMAReq_TX
		LL_LPUART_IsEnabledDMAReq_TX
	EIE	*LL_LPUART_DisableIT_ERROR*
		LL_LPUART_EnableIT_ERROR
		LL_LPUART_IsEnabledIT_ERROR
	HDSEL	*LL_LPUART_DisableHalfDuplex*
		LL_LPUART_EnableHalfDuplex
		LL_LPUART_IsEnabledHalfDuplex
	OVRDIS	*LL_LPUART_DisableOverrunDetect*
		LL_LPUART_EnableOverrunDetect
		LL_LPUART_IsEnabledOverrunDetect
	RTSE	*LL_LPUART_DisableRTSHWFlowCtrl*
		LL_LPUART_EnableRTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	WUFIE	*LL_LPUART_DisableIT_WKUP*
		LL_LPUART_EnableIT_WKUP
		LL_LPUART_IsEnabledIT_WKUP

Register	Field	Function
	WUS	LL_LPUART_GetWKUPType
		LL_LPUART_SetWKUPType
ICR	CMCF	LL_LPUART_ClearFlag_CM
	CTSCF	LL_LPUART_ClearFlag_nCTS
	FECF	LL_LPUART_ClearFlag_FE
	IDLECF	LL_LPUART_ClearFlag_IDLE
	NCF	LL_LPUART_ClearFlag_NE
	ORECF	LL_LPUART_ClearFlag_ORE
	PECF	LL_LPUART_ClearFlag_PE
	TCCF	LL_LPUART_ClearFlag_TC
	WUCF	LL_LPUART_ClearFlag_WKUP
ISR	BUSY	LL_LPUART_IsActiveFlag_BUSY
	CMF	LL_LPUART_IsActiveFlag_CM
	CTS	LL_LPUART_IsActiveFlag_CTS
	CTSIF	LL_LPUART_IsActiveFlag_nCTS
	FE	LL_LPUART_IsActiveFlag_FE
	IDLE	LL_LPUART_IsActiveFlag_IDLE
	NE	LL_LPUART_IsActiveFlag_NE
	ORE	LL_LPUART_IsActiveFlag_ORE
	PE	LL_LPUART_IsActiveFlag_PE
	REACK	LL_LPUART_IsActiveFlag_REACK
	RWU	LL_LPUART_IsActiveFlag_RWU
	RXNE	LL_LPUART_IsActiveFlag_RXNE
	SBKF	LL_LPUART_IsActiveFlag_SBK
	TC	LL_LPUART_IsActiveFlag_TC
	TEACK	LL_LPUART_IsActiveFlag_TEACK
	TXE	LL_LPUART_IsActiveFlag_TXE
	WUF	LL_LPUART_IsActiveFlag_WKUP
RDR	RDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_ReceiveData8
		LL_LPUART_ReceiveData9
RQR	MMRQ	LL_LPUART_RequestEnterMuteMode
	RXFRQ	LL_LPUART_RequestRxDataFlush
	SBKRQ	LL_LPUART_RequestBreakSending
TDR	TDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_TransmitData8

Register	Field	Function
		LL_LPUART_TransmitData9

78.16 PWR

Table 40: 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
	DS_EE_KOFF	LL_PWR_DisableNVMKeptOff
		LL_PWR_EnableNVMKeptOff
		LL_PWR_IsEnabledNVMKeptOff
	FWU	LL_PWR_DisableFastWakeUp
		LL_PWR_EnableFastWakeUp
		LL_PWR_IsEnabledFastWakeUp
	LPRUN	LL_PWR_DisableLowPowerRunMode
		LL_PWR_EnableLowPowerRunMode
		LL_PWR_EnterLowPowerRunMode
		LL_PWR_ExitLowPowerRunMode
		LL_PWR_IsEnabledLowPowerRunMode
	LPSDSR	LL_PWR_EnterLowPowerRunMode
		LL_PWR_ExitLowPowerRunMode
		LL_PWR_GetRegulModeLP
		LL_PWR_SetRegulModeLP
	PDDS	LL_PWR_GetPowerMode
		LL_PWR_SetPowerMode
	PLS	LL_PWR_GetPVDLevel
		LL_PWR_SetPVDLevel
	PVDE	LL_PWR_DisablePVD
		LL_PWR_EnablePVD
		LL_PWR_IsEnabledPVD
	ULP	LL_PWR_DisableUltraLowPower
		LL_PWR_EnableUltraLowPower
		LL_PWR_IsEnabledUltraLowPower
	VOS	LL_PWR_GetRegulVoltageScaling

Register	Field	Function
		LL_PWR_SetRegulVoltageScaling
CSR	EWUP1	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP2	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP3	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	PVDO	*LL_PWR_IsActiveFlag_PVDO*
	REGLPF	*LL_PWR_IsActiveFlag_REGLPF*
	SBF	*LL_PWR_IsActiveFlag_SB*
	VOSF	*LL_PWR_IsActiveFlag_VOSF*
	VREFINTRDYF	*LL_PWR_IsActiveFlag_VREFINTRDY*
	WUF	*LL_PWR_IsActiveFlag_WU*

78.17 RCC

Table 41: Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
CCIPR	CLK48SEL	*LL_RCC_GetRNGClockSource*
		LL_RCC_GetUSBClockSource
	HSI48SEL	*LL_RCC_SetRNGClockSource*
		LL_RCC_SetUSBClockSource
	I2CxSEL	*LL_RCC_GetI2CClockSource*
		LL_RCC_SetI2CClockSource
	LPTIMxSEL	*LL_RCC_GetLPTIMClockSource*
		LL_RCC_SetLPTIMClockSource
	LPUART1SEL	*LL_RCC_GetLPUARTClockSource*
		LL_RCC_SetLPUARTClockSource
	USARTxSEL	*LL_RCC_GetUSARTClockSource*
		LL_RCC_SetUSARTClockSource
CFGR	HPRE	*LL_RCC_GetAHBPrescaler*
		LL_RCC_SetAHBPrescaler
	MCOPRE	*LL_RCC_ConfigMCO*
	MCOSEL	*LL_RCC_ConfigMCO*

Register	Field	Function
	PLLDIV	LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetDivider
	PLLMUL	LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetMultiplicator
	PLLSRC	LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetMainSource
	PPRE1	LL_RCC_GetAPB1Prescaler
		LL_RCC_SetAPB1Prescaler
	PPRE2	LL_RCC_GetAPB2Prescaler
		LL_RCC_SetAPB2Prescaler
	STOPWUCK	LL_RCC_GetClkAfterWakeFromStop
		LL_RCC_SetClkAfterWakeFromStop
	SW	LL_RCC_SetSysClkSource
	SWS	LL_RCC_GetSysClkSource
CICR	CSSC	LL_RCC_ClearFlag_HSECSS
	HSERDYC	LL_RCC_ClearFlag_HSERDY
	HSI48RDYC	LL_RCC_ClearFlag_HSI48RDY
	HSIRDYC	LL_RCC_ClearFlag_HSIRDY
	LSECSSC	LL_RCC_ClearFlag_LSECSS
	LSERDYC	LL_RCC_ClearFlag_LSERDY
	LSIRDYC	LL_RCC_ClearFlag_LSIRDY
	MSIRDYC	LL_RCC_ClearFlag_MSIRDY
	PLLRDYC	LL_RCC_ClearFlag_PLLRDY
CIER	HSERDYIE	LL_RCC_DisableIT_HSERDY
		LL_RCC_EnableIT_HSERDY
		LL_RCC_IsEnabledIT_HSERDY
	HSI48RDYIE	LL_RCC_DisableIT_HSI48RDY
		LL_RCC_EnableIT_HSI48RDY
		LL_RCC_IsEnabledIT_HSI48RDY
	HSIRDYIE	LL_RCC_DisableIT_HSIRDY
		LL_RCC_EnableIT_HSIRDY
		LL_RCC_IsEnabledIT_HSIRDY
	LSECSSIE	LL_RCC_DisableIT_LSECSS
		LL_RCC_EnableIT_LSECSS
		LL_RCC_IsEnabledIT_LSECSS
	LSERDYIE	LL_RCC_DisableIT_LSERDY

Register	Field	Function
		LL_RCC_EnableIT_LSERDY
		LL_RCC_IsEnabledIT_LSERDY
	LSIRDYIE	LL_RCC_DisableIT_LSIRDY
		LL_RCC_EnableIT_LSIRDY
		LL_RCC_IsEnabledIT_LSIRDY
	MSIRDYIE	LL_RCC_DisableIT_MSIRDY
		LL_RCC_EnableIT_MSIRDY
		LL_RCC_IsEnabledIT_MSIRDY
	PLLRDYIE	LL_RCC_DisableIT_PLLRDY
		LL_RCC_EnableIT_PLLRDY
		LL_RCC_IsEnabledIT_PLLRDY
CIFR	CSSF	LL_RCC_IsActiveFlag_HSECSS
	HSERDYF	LL_RCC_IsActiveFlag_HSERDY
	HSI48RDYF	LL_RCC_IsActiveFlag_HSI48RDY
	HSIRDYF	LL_RCC_IsActiveFlag_HSIRDY
	LSECSSF	LL_RCC_IsActiveFlag_LSECSS
	LSERDYF	LL_RCC_IsActiveFlag_LSERDY
	LSIRDYF	LL_RCC_IsActiveFlag_LSIRDY
	MSIRDYF	LL_RCC_IsActiveFlag_MSIRDY
	PLLRDYF	LL_RCC_IsActiveFlag_PLLRDY
CR	CSSHSEON	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
	HSIDIVEN	LL_RCC_HSI_DisableDivider
		LL_RCC_HSI_EnableDivider
	HSIDIVF	LL_RCC_IsActiveFlag_HSIDIV
	HSIKERON	LL_RCC_HSI_DisableInStopMode
		LL_RCC_HSI_EnableInStopMode
	HSION	LL_RCC_HSI_Disable
		LL_RCC_HSI_Enable
	HSIOUTEN	LL_RCC_HSI_DisableOutput
		LL_RCC_HSI_EnableOutput
	HSIRDY	LL_RCC_HSI_IsReady

Register	Field	Function
	MSION	LL_RCC_MSI_Disable
		LL_RCC_MSI_Enable
	MSIRDY	LL_RCC_MSI_IsReady
	PLLON	LL_RCC_PLL_Disable
		LL_RCC_PLL_Enable
	PLLRDY	LL_RCC_PLL_IsReady
	RTCPRE	LL_RCC_GetRTC_HSEPrescaler
		LL_RCC_SetRTC_HSEPrescaler
CRRCR	HSI48CAL	LL_RCC_HSI48_GetCalibration
	HSI48DIV6OUTEN	LL_RCC_HSI48_DisableDivider
		LL_RCC_HSI48_EnableDivider
		LL_RCC_HSI48_IsDivided
	HSI48ON	LL_RCC_HSI48_Disable
		LL_RCC_HSI48_Enable
	HSI48RDY	LL_RCC_HSI48_IsReady
CSR	FWRSTF	LL_RCC_IsActiveFlag_FWRST
	IWDGRSTF	LL_RCC_IsActiveFlag_IWDGRST
	LPWRRSTF	LL_RCC_IsActiveFlag_LPWRRST
	LSEBYP	LL_RCC_LSE_DisableBypass
		LL_RCC_LSE_EnableBypass
	LSECSSD	LL_RCC_LSE_IsCSSDetected
	LSECSSON	LL_RCC_LSE_DisableCSS
		LL_RCC_LSE_EnableCSS
	LSEDRV	LL_RCC_LSE_GetDriveCapability
		LL_RCC_LSE_SetDriveCapability
	LSEON	LL_RCC_LSE_Disable
		LL_RCC_LSE_Enable
	LSERDY	LL_RCC_LSE_IsReady
	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
	RTCEN	LL_RCC_DisableRTC

Register	Field	Function
		LL_RCC_EnableRTC
		LL_RCC_IsEnabledRTC
	RTCRST	*LL_RCC_ForceBackupDomainReset*
		LL_RCC_ReleaseBackupDomainReset
	RTCSEL	*LL_RCC_GetRTCClockSource*
		LL_RCC_SetRTCClockSource
	SFTRSTF	*LL_RCC_IsActiveFlag_SFTRST*
	WWDGRSTF	*LL_RCC_IsActiveFlag_WWDGRST*
ICSCR	HSICAL	*LL_RCC_HSI_GetCalibration*
	HSITRIM	*LL_RCC_HSI_GetCalibTrimming*
		LL_RCC_HSI_SetCalibTrimming
	MSICAL	*LL_RCC_MSI_GetCalibration*
	MSIRANGE	*LL_RCC_MSI_GetRange*
		LL_RCC_MSI_SetRange
	MSITRIM	*LL_RCC_MSI_GetCalibTrimming*
		LL_RCC_MSI_SetCalibTrimming

78.18 RNG

Table 42: Correspondence between RNG registers and RNG low-layer driver functions

Register	Field	Function
CR	IE	*LL_RNG_DisableIT*
		LL_RNG_EnableIT
		LL_RNG_IsEnabledIT
	RNGEN	*LL_RNG_Disable*
		LL_RNG_Enable
		LL_RNG_IsEnabled
DR	RNDATA	*LL_RNG_ReadRandData32*
SR	CECS	*LL_RNG_IsActiveFlag_CECS*
	CEIS	*LL_RNG_ClearFlag_CEIS*
		LL_RNG_IsActiveFlag_CEIS
	DRDY	*LL_RNG_IsActiveFlag_DRDY*
	SECS	*LL_RNG_IsActiveFlag_SECS*
	SEIS	*LL_RNG_ClearFlag_SEIS*
		LL_RNG_IsActiveFlag_SEIS

78.19 RTC

Table 43: Correspondence between RTC registers and RTC low-layer driver functions

Register	Field	Function
ALRMAR	DT	*LL_RTC_ALMA_GetDay*
		LL_RTC_ALMA_SetDay
	DU	*LL_RTC_ALMA_GetDay*
		LL_RTC_ALMA_GetWeekDay
		LL_RTC_ALMA_SetDay
		LL_RTC_ALMA_SetWeekDay
	HT	*LL_RTC_ALMA_ConfigTime*
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	HU	*LL_RTC_ALMA_ConfigTime*
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	MNT	*LL_RTC_ALMA_ConfigTime*
		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*

Register	Field	Function
		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
ALRMBR	DT	LL_RTC_ALMB_GetDay
		LL_RTC_ALMB_SetDay
	DU	LL_RTC_ALMB_GetDay
		LL_RTC_ALMB_GetWeekDay
		LL_RTC_ALMB_SetDay
		LL_RTC_ALMB_SetWeekDay
	HT	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	HU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	MNT	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetMinute
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MNU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetMinute
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MSK1	LL_RTC_ALMB_GetMask

Register	Field	Function
		LL_RTC_ALMB_SetMask
	MSK2	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK3	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK4	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	PM	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetTimeFormat
		LL_RTC_ALMB_SetTimeFormat
	ST	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	SU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	WDSEL	LL_RTC_ALMB_DisableWeekday
		LL_RTC_ALMB_EnableWeekday
ALRMBSSR	MASKSS	LL_RTC_ALMB_GetSubSecondMask
		LL_RTC_ALMB_SetSubSecondMask
	SS	LL_RTC_ALMB_GetSubSecond
		LL_RTC_ALMB_SetSubSecond
BKPxR	BKP	LL_RTC_BAK_GetRegister
		LL_RTC_BAK_SetRegister
CALR	CALM	LL_RTC_CAL_GetMinus
		LL_RTC_CAL_SetMinus
	CALP	LL_RTC_CAL_IsPulseInserted
		LL_RTC_CAL_SetPulse
	CALW16	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	CALW8	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
CR	ADD1H	LL_RTC_TIME_IncHour
	ALRAE	LL_RTC_ALMA_Disable

Register	Field	Function
		LL_RTC_ALMA_Enable
	ALRAIE	LL_RTC_DisableIT_ALRA
		LL_RTC_EnableIT_ALRA
		LL_RTC_IsEnabledIT_ALRA
	ALRBE	LL_RTC_ALMB_Disable
		LL_RTC_ALMB_Enable
	ALRBIE	LL_RTC_DisableIT_ALRB
		LL_RTC_EnableIT_ALRB
		LL_RTC_IsEnabledIT_ALRB
	BCK	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

Register	Field	Function
		LL_RTC_WAKEUP_SetClock
	WUTE	*LL_RTC_WAKEUP_Disable*
		LL_RTC_WAKEUP_Enable
		LL_RTC_WAKEUP_IsEnabled
	WUTIE	*LL_RTC_DisableIT_WUT*
		LL_RTC_EnableIT_WUT
		LL_RTC_IsEnabledIT_WUT
DR	DT	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	DU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	MT	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	MU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	WDU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetWeekDay
		LL_RTC_DATE_SetWeekDay
	YT	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
	YU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
ISR	ALRAF	*LL_RTC_ClearFlag_ALRA*

Register	Field	Function
		LL_RTC_IsActiveFlag_ALRA
	ALRAWF	LL_RTC_IsActiveFlag_ALRAW
	ALRBF	LL_RTC_ClearFlag_ALRB
		LL_RTC_IsActiveFlag_ALRB
	ALRBWF	LL_RTC_IsActiveFlag_ALRBW
	INIT	LL_RTC_DisableInitMode
		LL_RTC_EnableInitMode
	INITF	LL_RTC_IsActiveFlag_INIT
	INITS	LL_RTC_IsActiveFlag_INITS
	RECALPF	LL_RTC_IsActiveFlag_RECALP
	RSF	LL_RTC_ClearFlag_RS
		LL_RTC_IsActiveFlag_RS
	SHPF	LL_RTC_IsActiveFlag_SHP
	TAMP1F	LL_RTC_ClearFlag_TAMP1
		LL_RTC_IsActiveFlag_TAMP1
	TAMP2F	LL_RTC_ClearFlag_TAMP2
		LL_RTC_IsActiveFlag_TAMP2
	TAMP3F	LL_RTC_ClearFlag_TAMP3
		LL_RTC_IsActiveFlag_TAMP3
	TSF	LL_RTC_ClearFlag_TS
		LL_RTC_IsActiveFlag_TS
	TSOVF	LL_RTC_ClearFlag_TSOV
		LL_RTC_IsActiveFlag_TSOV
	WUTF	LL_RTC_ClearFlag_WUT
		LL_RTC_IsActiveFlag_WUT
	WUTWF	LL_RTC_IsActiveFlag_WUTW
OR	ALARMOUTTYPE	LL_RTC_GetAlarmOutputType
		LL_RTC_SetAlarmOutputType
	OUT_RMP	LL_RTC_DisableOutRemap
		LL_RTC_EnableOutRemap
PRER	PREDIV_A	LL_RTC_GetAsynchPrescaler
		LL_RTC_SetAsynchPrescaler
	PREDIV_S	LL_RTC_GetSynchPrescaler
		LL_RTC_SetSynchPrescaler
SHIFTR	ADD1S	LL_RTC_TIME_Synchronize
	SUBFS	LL_RTC_TIME_Synchronize

Register	Field	Function
SSR	SS	LL_RTC_TIME_GetSubSecond
TAMPCR	TAMP1E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP1IE	LL_RTC_DisableIT_TAMP1
		LL_RTC_EnableIT_TAMP1
		LL_RTC_IsEnabledIT_TAMP1
	TAMP1MF	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP1NOERASE	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP1TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMP2E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP2IE	LL_RTC_DisableIT_TAMP2
		LL_RTC_EnableIT_TAMP2
		LL_RTC_IsEnabledIT_TAMP2
	TAMP2MF	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP2NOERASE	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP2TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMP3E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP3IE	LL_RTC_DisableIT_TAMP3
		LL_RTC_EnableIT_TAMP3
		LL_RTC_IsEnabledIT_TAMP3
	TAMP3MF	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP3NOERASE	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP3TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMPFLT	LL_RTC_TAMPER_GetFilterCount
		LL_RTC_TAMPER_SetFilterCount

Register	Field	Function
	TAMPFREQ	LL_RTC_TAMPER_GetSamplingFreq
		LL_RTC_TAMPER_SetSamplingFreq
	TAMPIE	LL_RTC_DisableIT_TAMP
		LL_RTC_EnableIT_TAMP
		LL_RTC_IsEnabledIT_TAMP
	TAMPPRCH	LL_RTC_TAMPER_GetPrecharge
		LL_RTC_TAMPER_SetPrecharge
	TAMPPUDIS	LL_RTC_TAMPER_DisablePullUp
		LL_RTC_TAMPER_EnablePullUp
	TAMPTS	LL_RTC_TS_DisableOnTamper
		LL_RTC_TS_EnableOnTamper
TR	HT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	HU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	MNT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	MNU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	PM	LL_RTC_TIME_Config
		LL_RTC_TIME_GetFormat
		LL_RTC_TIME_SetFormat
	ST	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
	SU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get

Register	Field	Function
		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
	SU	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

78.20 SPI

Table 44: Correspondence between SPI registers and SPI low-layer driver functions

Register	Field	Function
CR1	BIDIMODE	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection

Register	Field	Function
	BIDIOE	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	BR	LL_SPI_GetBaudRatePrescaler
		LL_SPI_SetBaudRatePrescaler
	CPHA	LL_SPI_GetClockPhase
		LL_SPI_SetClockPhase
	CPOL	LL_SPI_GetClockPolarity
		LL_SPI_SetClockPolarity
	CRCEN	LL_SPI_DisableCRC
		LL_SPI_EnableCRC
		LL_SPI_IsEnabledCRC
	CRCNEXT	LL_SPI_SetCRCNext
	DFF	LL_SPI_GetDataWidth
		LL_SPI_SetDataWidth
	LSBFIRST	LL_SPI_GetTransferBitOrder
		LL_SPI_SetTransferBitOrder
	MSTR	LL_SPI_GetMode
		LL_SPI_SetMode
	RXONLY	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	SPE	LL_SPI_Disable
		LL_SPI_Enable
		LL_SPI_IsEnabled
	SSI	LL_SPI_GetMode
		LL_SPI_SetMode
	SSM	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode
CR2	ERRIE	LL_SPI_DisableIT_ERR
		LL_SPI_EnableIT_ERR
		LL_SPI_IsEnabledIT_ERR
	FRF	LL_SPI_GetStandard
		LL_SPI_SetStandard
	RXDMAEN	LL_SPI_DisableDMAReq_RX
		LL_SPI_EnableDMAReq_RX
		LL_SPI_IsEnabledDMAReq_RX
	RXNEIE	LL_SPI_DisableIT_RXNE

Register	Field	Function
		LL_SPI_EnableIT_RXNE
		LL_SPI_IsEnabledIT_RXNE
	SSOE	*LL_SPI_GetNSSMode*
		LL_SPI_SetNSSMode
	TXDMAEN	*LL_SPI_DisableDMAReq_TX*
		LL_SPI_EnableDMAReq_TX
		LL_SPI_IsEnabledDMAReq_TX
	TXEIE	*LL_SPI_DisableIT_TXE*
		LL_SPI_EnableIT_TXE
		LL_SPI_IsEnabledIT_TXE
CRCPR	CRCPOLY	*LL_SPI_GetCRCPolynomial*
		LL_SPI_SetCRCPolynomial
DR	DR	*LL_SPI_DMA_GetRegAddr*
		LL_SPI_ReceiveData16
		LL_SPI_ReceiveData8
		LL_SPI_TransmitData16
		LL_SPI_TransmitData8
RXCRCR	RXCRC	*LL_SPI_GetRxCRC*
SR	BSY	*LL_SPI_IsActiveFlag_BSY*
	CRCERR	*LL_SPI_ClearFlag_CRCERR*
		LL_SPI_IsActiveFlag_CRCERR
	FRE	*LL_SPI_ClearFlag_FRE*
		LL_SPI_IsActiveFlag_FRE
	MODF	*LL_SPI_ClearFlag_MODF*
		LL_SPI_IsActiveFlag_MODF
	OVR	*LL_SPI_ClearFlag_OVR*
		LL_SPI_IsActiveFlag_OVR
	RXNE	*LL_SPI_IsActiveFlag_RXNE*
	TXE	*LL_SPI_IsActiveFlag_TXE*
TXCRCR	TXCRC	*LL_SPI_GetTxCRC*

78.21 SYSTEM

Table 45: Correspondence between SYSTEM registers and SYSTEM low-layer driver functions

Register	Field	Function
APB1FZ	DBG_I2C1_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Register	Field	Function
	DBG_I2C2_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_I2C3_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_IWDG_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_LPTIMER_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_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
APB2FZ	DBG_TIM21_STOP	LL_DBGMCU_APB2_GRP1_FreezePeriph
		LL_DBGMCU_APB2_GRP1_UnFreezePeriph
	DBG_TIM22_STOP	LL_DBGMCU_APB2_GRP1_FreezePeriph
		LL_DBGMCU_APB2_GRP1_UnFreezePeriph
DBGMCU_CR	DBG_SLEEP	LL_DBGMCU_DisableDBGSleepMode
		LL_DBGMCU_EnableDBGSleepMode
	DBG_STANDBY	LL_DBGMCU_DisableDBGStandbyMode
		LL_DBGMCU_EnableDBGStandbyMode
	DBG_STOP	LL_DBGMCU_DisableDBGStopMode
		LL_DBGMCU_EnableDBGStopMode
DBGMCU_IDCODE	DEV_ID	LL_DBGMCU_GetDeviceID
	REV_ID	LL_DBGMCU_GetRevisionID
FLASH_ACR	DISAB_BUF	LL_FLASH_DisableBuffers
		LL_FLASH_EnableBuffers
	LATENCY	LL_FLASH_GetLatency
		LL_FLASH_SetLatency

Register	Field	Function
	PRE_READ	LL_FLASH_DisablePreRead
		LL_FLASH_EnablePreRead
	PRFTEN	LL_FLASH_DisablePrefetch
		LL_FLASH_EnablePrefetch
		LL_FLASH_IsPrefetchEnabled
	RUN_PD	LL_FLASH_DisableRunPowerDown
		LL_FLASH_EnableRunPowerDown
	SLEEP_PD	LL_FLASH_DisableSleepPowerDown
		LL_FLASH_EnableSleepPowerDown
FLASH_PDKEYR	PDKEY1	LL_FLASH_DisableRunPowerDown
		LL_FLASH_EnableRunPowerDown
	PDKEY2	LL_FLASH_DisableRunPowerDown
		LL_FLASH_EnableRunPowerDown
SYSCFG_CFGR1	BOOT_MODE	LL_SYSCFG_GetBootMode
	MEM_MODE	LL_SYSCFG_GetRemapMemory
		LL_SYSCFG_SetRemapMemory
	UFB	LL_SYSCFG_GetFlashBankMode
		LL_SYSCFG_SetFlashBankMode
SYSCFG_CFGR2	CAPA	LL_SYSCFG_GetVLCDRailConnection
		LL_SYSCFG_SetVLCDRailConnection
	FWDIS	LL_SYSCFG_EnableFirewall
		LL_SYSCFG_IsEnabledFirewall
	I2C_PBx_FMP	LL_SYSCFG_DisableFastModePlus
		LL_SYSCFG_EnableFastModePlus
	I2Cx_FMP	LL_SYSCFG_DisableFastModePlus
		LL_SYSCFG_EnableFastModePlus
SYSCFG_CFGR3	ENBUF_SENSOR_ADC	LL_SYSCFG_TEMPSENSOR_Disable
		LL_SYSCFG_TEMPSENSOR_Enable
	ENBUF_VREFINT_ADC	LL_SYSCFG_VREFINT_DisableADC
		LL_SYSCFG_VREFINT_EnableADC
	ENBUF_VREFINT_COMP	LL_SYSCFG_VREFINT_DisableCOMP
		LL_SYSCFG_VREFINT_EnableCOMP
	ENREF_HSI48	LL_SYSCFG_VREFINT_DisableHSI48
		LL_SYSCFG_VREFINT_EnableHSI48
	REF_LOCK	LL_SYSCFG_VREFINT_IsLocked
		LL_SYSCFG_VREFINT_Lock

Register	Field	Function
	SEL_VREF_OUT	*LL_SYSCFG_VREFINT_GetConnection*
		LL_SYSCFG_VREFINT_SetConnection
	VREFINT_RDYF	*LL_SYSCFG_VREFINT_IsReady*
SYSCFG_EXTICR1	EXTI0	*LL_SYSCFG_SetEXTISource*
	EXTI1	*LL_SYSCFG_SetEXTISource*
	EXTI2	*LL_SYSCFG_SetEXTISource*
	EXTI3	*LL_SYSCFG_SetEXTISource*
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*
	EXTI8	*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*

78.22 TIM

Table 46: Correspondence between TIM registers and TIM low-layer driver functions

Register	Field	Function
ARR	ARR	*LL_TIM_GetAutoReload*
		LL_TIM_SetAutoReload
CCER	CC1E	*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
	CC1P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity

Register	Field	Function
		LL_TIM_OC_SetPolarity
	CC2E	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
	CC2P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_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
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity

Register	Field	Function
		LL_TIM_OC_SetPolarity
CCMR1	CC1S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC2S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC1F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC1PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC2F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC2PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	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
		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*

Register	Field	Function
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC4CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC4FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC4M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC4PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCR1	CCR1	*LL_TIM_IC_GetCaptureCH1*
		LL_TIM_OC_GetCompareCH1
		LL_TIM_OC_SetCompareCH1
CCR2	CCR2	*LL_TIM_IC_GetCaptureCH2*
		LL_TIM_OC_GetCompareCH2
		LL_TIM_OC_SetCompareCH2
CCR3	CCR3	*LL_TIM_IC_GetCaptureCH3*
		LL_TIM_OC_GetCompareCH3
		LL_TIM_OC_SetCompareCH3
CCR4	CCR4	*LL_TIM_IC_GetCaptureCH4*
		LL_TIM_OC_GetCompareCH4
		LL_TIM_OC_SetCompareCH4
CNT	CNT	*LL_TIM_GetCounter*
		LL_TIM_SetCounter
CR1	ARPE	*LL_TIM_DisableARRPreload*
		LL_TIM_EnableARRPreload
		LL_TIM_IsEnabledARRPreload
	CEN	*LL_TIM_DisableCounter*
		LL_TIM_EnableCounter
		LL_TIM_IsEnabledCounter
	CKD	*LL_TIM_GetClockDivision*
		LL_TIM_SetClockDivision
	CMS	*LL_TIM_GetCounterMode*

Register	Field	Function
		LL_TIM_SetCounterMode
	DIR	LL_TIM_GetCounterMode
		LL_TIM_GetDirection
		LL_TIM_SetCounterMode
	OPM	LL_TIM_GetOnePulseMode
		LL_TIM_SetOnePulseMode
	UDIS	LL_TIM_DisableUpdateEvent
		LL_TIM_EnableUpdateEvent
		LL_TIM_IsEnabledUpdateEvent
	URS	LL_TIM_GetUpdateSource
		LL_TIM_SetUpdateSource
CR2	CCDS	LL_TIM_CC_GetDMAReqTrigger
		LL_TIM_CC_SetDMAReqTrigger
	MMS	LL_TIM_SetTriggerOutput
	TI1S	LL_TIM_IC_DisableXORCombination
		LL_TIM_IC_EnableXORCombination
		LL_TIM_IC_IsEnabledXORCombination
DCR	DBA	LL_TIM_ConfigDMABurst
	DBL	LL_TIM_ConfigDMABurst
DIER	CC1DE	LL_TIM_DisableDMAReq_CC1
		LL_TIM_EnableDMAReq_CC1
		LL_TIM_IsEnabledDMAReq_CC1
	CC1IE	LL_TIM_DisableIT_CC1
		LL_TIM_EnableIT_CC1
		LL_TIM_IsEnabledIT_CC1
	CC2DE	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

Register	Field	Function
		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
	TDE	LL_TIM_DisableDMAReq_TRIG
		LL_TIM_EnableDMAReq_TRIG
		LL_TIM_IsEnabledDMAReq_TRIG
	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	CC1G	LL_TIM_GenerateEvent_CC1
	CC2G	LL_TIM_GenerateEvent_CC2
	CC3G	LL_TIM_GenerateEvent_CC3
	CC4G	LL_TIM_GenerateEvent_CC4
	TG	LL_TIM_GenerateEvent_TRIG
	UG	LL_TIM_GenerateEvent_UPDATE
PSC	PSC	LL_TIM_GetPrescaler
		LL_TIM_SetPrescaler
SMCR	ECE	LL_TIM_DisableExternalClock
		LL_TIM_EnableExternalClock
		LL_TIM_IsEnabledExternalClock
		LL_TIM_SetClockSource
	ETF	LL_TIM_ConfigETR
	ETP	LL_TIM_ConfigETR
	ETPS	LL_TIM_ConfigETR
	MSM	LL_TIM_DisableMasterSlaveMode
		LL_TIM_EnableMasterSlaveMode

Register	Field	Function
		LL_TIM_IsEnabledMasterSlaveMode
	OCCS	LL_TIM_SetOCRefClearInputSource
	SMS	LL_TIM_SetClockSource
		LL_TIM_SetEncoderMode
		LL_TIM_SetSlaveMode
	TS	LL_TIM_SetTriggerInput
SR	CC1IF	LL_TIM_ClearFlag_CC1
		LL_TIM_IsActiveFlag_CC1
	CC1OF	LL_TIM_ClearFlag_CC1OVR
		LL_TIM_IsActiveFlag_CC1OVR
	CC2IF	LL_TIM_ClearFlag_CC2
		LL_TIM_IsActiveFlag_CC2
	CC2OF	LL_TIM_ClearFlag_CC2OVR
		LL_TIM_IsActiveFlag_CC2OVR
	CC3IF	LL_TIM_ClearFlag_CC3
		LL_TIM_IsActiveFlag_CC3
	CC3OF	LL_TIM_ClearFlag_CC3OVR
		LL_TIM_IsActiveFlag_CC3OVR
	CC4IF	LL_TIM_ClearFlag_CC4
		LL_TIM_IsActiveFlag_CC4
	CC4OF	LL_TIM_ClearFlag_CC4OVR
		LL_TIM_IsActiveFlag_CC4OVR
	TIF	LL_TIM_ClearFlag_TRIG
		LL_TIM_IsActiveFlag_TRIG
	UIF	LL_TIM_ClearFlag_UPDATE
		LL_TIM_IsActiveFlag_UPDATE
TIM21_OR	ETR_RMP	LL_TIM_SetRemap
	TI1_RMP	LL_TIM_SetRemap
	TI2_RMP	LL_TIM_SetRemap
TIM22_OR	ETR_RMP	LL_TIM_SetRemap
	TI1_RMP	LL_TIM_SetRemap
TIM2_OR	ETR_RMP	LL_TIM_SetRemap
	TI4_RMP	LL_TIM_SetRemap
TIM3_OR	ETR_RMP	LL_TIM_SetRemap
	TI1_RMP	LL_TIM_SetRemap
	TI2_RMP	LL_TIM_SetRemap

Register	Field	Function
	TI4_RMP	*LL_TIM_SetRemap*

78.23 USART

Table 47: Correspondence between USART registers and USART low-layer driver functions

Register	Field	Function
BRR	BRR	*LL_USART_GetBaudRate*
		LL_USART_SetBaudRate
CR1	CMIE	*LL_USART_DisableIT_CM*
		LL_USART_EnableIT_CM
		LL_USART_IsEnabledIT_CM
	DEAT	*LL_USART_GetDEAssertionTime*
		LL_USART_SetDEAssertionTime
	DEDT	*LL_USART_GetDEDeassertionTime*
		LL_USART_SetDEDeassertionTime
	EOBIE	*LL_USART_DisableIT_EOB*
		LL_USART_EnableIT_EOB
		LL_USART_IsEnabledIT_EOB
	IDLEIE	*LL_USART_DisableIT_IDLE*
		LL_USART_EnableIT_IDLE
		LL_USART_IsEnabledIT_IDLE
	M0	*LL_USART_ConfigCharacter*
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	M1	*LL_USART_ConfigCharacter*
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	MME	*LL_USART_DisableMuteMode*
		LL_USART_EnableMuteMode
		LL_USART_IsEnabledMuteMode
	OVER8	*LL_USART_GetOverSampling*
		LL_USART_SetOverSampling
	PCE	*LL_USART_ConfigCharacter*
		LL_USART_GetParity
		LL_USART_SetParity
	PEIE	*LL_USART_DisableIT_PE*
		LL_USART_EnableIT_PE

Register	Field	Function
		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
	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
	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
	ABRMODE	LL_USART_GetAutoBaudRateMode

Register	Field	Function
		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
	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

Register	Field	Function
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableLIN
		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

Register	Field	Function
		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
	EIE	LL_USART_DisableIT_ERROR
		LL_USART_EnableIT_ERROR
		LL_USART_IsEnabledIT_ERROR
	HDSEL	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableHalfDuplex
		LL_USART_EnableHalfDuplex
		LL_USART_IsEnabledHalfDuplex
	IREN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableIrda
		LL_USART_EnableIrda
		LL_USART_IsEnabledIrda
	IRLP	LL_USART_GetIrdaPowerMode
		LL_USART_SetIrdaPowerMode
	NACK	LL_USART_DisableSmartcardNACK
		LL_USART_EnableSmartcardNACK
		LL_USART_IsEnabledSmartcardNACK
	ONEBIT	LL_USART_DisableOneBitSamp
		LL_USART_EnableOneBitSamp

Register	Field	Function
		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
		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

Register	Field	Function
	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*
	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*

Register	Field	Function
	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

78.24 WWDG

Table 48: Correspondence between WWDG registers and WWDG low-layer driver functions

Register	Field	Function
CFR	EWI	*LL_WWDG_EnableIT_EWKUP*
		LL_WWDG_IsEnabledIT_EWKUP
	W	*LL_WWDG_GetWindow*
		LL_WWDG_SetWindow
	WDGTB	*LL_WWDG_GetPrescaler*
		LL_WWDG_SetPrescaler
CR	T	*LL_WWDG_GetCounter*
		LL_WWDG_SetCounter
	WDGA	*LL_WWDG_Enable*
		LL_WWDG_IsEnabled
SR	EWIF	*LL_WWDG_ClearFlag_EWKUP*
		LL_WWDG_IsActiveFlag_EWKUP

79 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture that speeds 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 STM32L0 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32L0 devices. To ensure compatibility between all devices as well as portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *Section 2.4: "Devices supported by 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: stm32l0xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration, Ethernet parameters configuration…)

A template is provided in the HAL drivers folders (stm32l0xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32l0xx_hal.h file has to be included.

What is the difference between stm32l0xx_hal_ppp.c/.h and stm32l0xx_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 (stm32l0xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32l0xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Is it possible to use the APIs available in stm32l0xx_ll_ppp.c?

These APIs cannot be used directly because they are internal and offer services to upper layer drivers. As an example stm32l0xx_ll_fmc.c/h driver is used by stm32l0xx_hal_sram.c, stm32l0xx_hal_nor.c, stm32l0xx_hal_nand.c and stm32l0xx_hal_sdram.c drivers.

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_stm32l0xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef *pHandle* structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32l0xx_hal_msp.c. A template is provided in the HAL driver folders (stm32l0xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute __*weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).
2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() in stm32l0xx_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 stm32l0xx_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 stm32l0xx_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.

80 Revision history

Table 49: Document revision history

Date	Revision	Changes
03-Jun-2014	1	Initial release.
30-Jul-2015	2	*Section 2.4: "Devices supported by HAL drivers"* added support for STM32L07xxx and STM32L08xxx and updated stm32l0xx_hal_dac_ex.c and stm32l0xx_hal_lcd.c support for STM32L05/6xx devices. Removed note related to STM32Cube V1.0 in *Section 2.12.2.1: "HAL global initialization"*. Updated common macros in *Section 2.9: "HAL common resources"* Updated GPIO_Typedef type (AF0) in *Section 22: "HAL GPIO Generic Driver"*. Added new HAL_NVIC_GetPriority(IRQn_Type IRQn function in *Section 10: "HAL CORTEX Generic Driver"*.
20-Nov-2015	3	Added stm32l0xx_flash.ld in *Section 2.1.2: "User-application files"*. Added STM32L011xx, STM32L021xx, STM32L031xx and STM32L041xx in *Section 2.4: "Devices supported by HAL drivers"*. Updated example of peripheral structure in *Section 2.2.1: "Peripheral handle structures"*. Changed HAL_ADCEx_CalibrationStart() into HAL_ADCEx_Calibration_Start() in *Table 10: "HAL extension APIs"*. Replaced GPIO_SPEED_LOW by GPIO_SPEED_FREQ_LOW, GPIO_SPEED_MEDIUM by GPIO_SPEED_FREQ_MEDIUM, GPIO_SPEED_FAST by GPIO_SPEED_FREQ_FREQ_HIGH and GPIO_SPEED_HIGH by GPIO_SPEED_FREQ_VERY_HIGH) speed in *Table 12: "Description of GPIO_InitTypeDef structure"*. Updated example of system clock configuration code in *Section 2.12.2.2: "System clock initialization"*. Add new API to control the MPU (see and). Renamed some HAL macros and driver define statements to ensure consistency all over STM32 families: • Added new define statements to specify the GPIO Speed (GPIO_SPEED_LOW replaced by GPIO_SPEED_FREQ_LOW for example). • Added new macro to check the Comparator inputs (IS_COMP1_LPTIMCONNECTION and IS_COMP2_LPTIMCONNECTION).
05-Apr-2016	4	Updated . Updated *Figure 7: "HAL driver model"*. Added *Section 3: "Overview of Low Layer drivers"*. Added *Section 4: "Cohabiting of HAL and LL "*. *Section 10.2: "CORTEX Firmware driver API description"*: remove __HAL_CORTEX_SYSTICKCLK_CONFIG(..) macro since this service is already covered by HAL_SYSTICK_CLKSourceConfig() function. *Section 17.3: "DMA Firmware driver defines"*: Added HAL_DMA_GET_COUNTER macrro. Added descriptions of LL drivers for the following peripherals: ADC, COMP, CRC, CRS, DAC, DMA, EXTI, GPIO, I2C, I2S, IWDG, LPTIM, LPUART, PWR, RCC, RNG, RTC, SPI, TIM, USART, WWDG, as well as additional Low Level Bus, System, Cortex and Utilities APIs.

Date	Revision	Changes
06-Jun-2016	5	**HAL/LL COMP** • Added missing definition for COMP_INPUT_PLUS_IO6 and LL_COMP_INPUT_PLUS_IO6, supported by STM32L0 Category1 (STM32L011xx, STM32L021xx). • Removed COMP_INVERTINGINPUT_IO3 definition. • Renamed COMP_INVERTINGINPUT_IO2 to COMP_INPUT_MINUS_DAC1_CH2. • The EXTI set-up is now managed by HAL_COMP_Init() function, using updated definitions of COMP_TRIGGERMODE_xxx. The functions HAL_COMP_Start_IT() and HAH_COMP_Stop_IT() have been removed. **HAL LCD** • Corrected SYSCFG LCD External Capacitors definitions. • Added new __HAL_SYSCFG_VLCD_CAPA_CONFIG() macro to configure the VLCD Decoupling capacitance connection. • Added new __HAL_SYSCFG_GET_VLCD_CAPA_CONFIG() macro to return the decoupling of LCD capacitance configured by the user. • Added LCD Voltage output buffer enable macro definitions.
19-Dec-2016	6	Update for release V1.8.0 of the HAL and LL drivers. Refer to the release note provided within the firmware package for details.