UM1576
User manual

Description of STM32W108xx Standard Peripheral Library

Introduction

The STM32W108xx Standard Peripheral Library covers 3 abstraction levels, and includes:

- A complete register address mapping with all bits, bit fields and registers declared in C. This avoids a cumbersome task and more important, it brings the benefits of a bug free reference mapping file, speeding up the early project phase.
- A collection of routines and data structures covering all peripheral functions (drivers with common API). It can directly be used as a reference framework, since it also includes macros for supporting core-related intrinsic features, common constants, and definition of data types.
- A set of examples covering all available peripherals with template projects for the most common development tools. With the appropriate hardware evaluation board, this allows to get started with a brand-new micro within few hours.

Each driver consists of a set of functions covering all peripheral features. The development of each driver is driven by a common API (application programming interface) which standardizes the driver structure, the functions and the parameter names. The driver source code is developed in 'Strict ANSI-C' (relaxed ANSI-C for projects and example files). It is fully documented and is MISRA-C 2004 compliant. Writing the whole library in 'Strict ANSI-C' makes it independent from the development tools. Only the start-up files depend on the development tools.Thanks to the Standard Peripheral Library, low-level implementation details are transparent so that reusing code on a different MCU requires only to reconfigure the compiler. As a result, developers can easily migrate designs across the STM32 series to quickly bring product line extensions to market without any redesign. In addition, the library is built around a modular architecture that makes it easy to tailor and run it on the same MCU using hardware platforms different from ST evaluation boards.

The Standard Peripheral Library implements run-time failure detection by checking the input values for all library functions. Such dynamic checking contributes towards enhancing the robustness of the software. Run-time detection is suitable for user application development and debugging. It adds an overhead which can be removed from the final application code to minimize code size and execution speed. For more details refer to *Section 1.1.5: "Run-time checking"*. Since the Standard Peripheral Library is generic and covers all peripheral features, the size and/or execution speed of the application code may not be optimized. For many applications, the library may be used as is.

The firmware library user manual is structured as follows:

- Document conventions, rules, architecture and overview of the Library package.
- How to use and customize the Library (step by step).
- Detailed description of each peripheral driver: configuration structure, functions and how to use the provided API to build your application.

The STM32W108xx Standard Peripheral Library will be referred to as Library throughout the document, unless otherwise specified.

Table 1: Applicable products

Type	Part numbers
Microcontrollers	STM32W108xx

Contents

List of tables

List of figures

1 STM32W108xx Standard Peripheral Library

1.1 Coding rules and conventions

The conventions used in the present user manual and in the library are described in the sections below.

1.1.1 Acronyms

Table 1: "List of abbreviations" describes the acronyms used in this document.

Table 1: List of abbreviations

Acronym	Peripheral / unit
ADC	Analog-to-digital
DMA	DMA controller
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/O
I^2C	Inter-integrated circuit
NVIC	Nested vectored interrupt controller
PWR	Power control
CLK	Clock controller
SPI	Serial peripheral interface
SysTick	System tick timer
TIM	General-purpose timer
UART	Universal asynchronous receiver transmitter
WDG	Watchdog

1.1.2 Naming conventions

The following naming conventions are used in the library:

- **PPP** refers to any peripheral acronym, for example **ADC**. See *Section 1.1: "Coding rules and conventions"* for more information.
- System and source/header file names are preceded by ‘stm32w108xx_’, for example stm32w108xx_conf.h.
- Constants used in one file are defined within this file. A constant used in more than one file is defined in a header file. All constants are written in upper case, 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 upper case and uses the same acronyms as in the STM32W108xx reference manual document.
- Peripheral registers are declared in the **PPP_TypeDef** structure (e.g. **ADC_TypeDef**) in stm32w108xx.h file.

- Almost all peripheral function names are preceded by the corresponding peripheral acronym in upper case followed by an underscore. The first letter in each word is in upper case, for example **UART_SendData**. 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 initialization parameters for the PPP peripheral are named **PPP_InitTypeDef** (e.g. **ADC_InitTypeDef**).
- The functions used to initialize the PPP peripheral according to parameters specified in **PPP_InitTypeDef** are named **PPP_Init**, e.g. **ADC_Init**.
- The functions used to reset the PPP peripheral registers to their default values are named **PPP_DeInit**, e.g. **ADC_DeInit**.
- The functions used to fill the **PPP_InitTypeDef** structure with the reset values of each member are named **PPP_StructInit**, e.g. **UART_StructInit**.
- The functions used to enable or disable the specified PPP peripheral are named **PPP_Cmd**, for example **UART_Cmd**.
- The functions used to enable or disable an interrupt source for the specified PPP peripheral are named **PPP_ITConfig**, e.g. **UART_ITConfig**.
- The functions used to check whether the specified PPP flag is set or reset are named **PPP_GetFlagStatus**, e.g. **I2C_GetFlagStatus**.
- The functions used to clear a PPP flag are named **PPP_ClearFlag**, for example **FLASH_ClearFlag**.
- The functions used to check whether the specified PPP interrupt has occurred or not are named **PPP_GetITStatus**, e.g. **TIM_GetITStatus**.
- The functions used to clear a PPP interrupt pending bit are named **PPP_ClearITPendingBit**, e.g. **I2C_ClearITPendingBit**.

1.1.3 Coding rules

This section describes the coding rules used in the library.

General

- All codes should comply with ANSI C standard and should compile without warning under at least its main compiler. Any warnings that cannot be eliminated should be commented in the code.
- The library uses ANSI standard data types defined in the ANSI C header file <stdint.h>.
- The library has no blocking code and all required waiting loops (polling loops) are controlled by an expiry programmed timeout.

Variable types

Specific variable types are already defined with a fixed type and size. These types are defined in the file stm32w108xx.h

```
typedef enum {
  RESET = 0,
  SET = !RESET
}
FlagStatus, ITStatus;

typedef enum {
  DISABLE = 0,
  ENABLE = !DISABLE
}
FunctionalState;

typedef enum {
```

```
  ERROR = 0,
  SUCCESS = !ERROR
}
ErrorStatus;

typedef enum{
 Bit_RESET = 0,
  Bit_SET
}
BitAction;
```

Peripherals

Pointers to peripherals are used to access the peripheral control registers. They point to data structures that represent the mapping of the peripheral control registers.

Peripheral registers structure

stm32w108xx.h contains the definition of all peripheral register structures. The example below illustrates the TIM register structure declaration:

```
"/*------------------ Timer ---------------*/
typedef struct
{
  __IO uint32_t CR1;              /*!< TIM control register 1,
Address offset 0x00 */
  __IO uint32_t CR2;              /*!< TIM control register 2,
Address offset 0x04 */
  __IO uint32_t SMCR;             /*!< TIM slave Mode Control
register,         Address offset 0x08 */
       uint32_t RESERVED0[2];     /*!< Reserved */
  __IO uint32_t EGR;              /*!< TIM event generation
register            Address offset 0x14 */
  __IO uint32_t CCMR1;            /*!< TIM capture/compare mode
register 1,     Address offset 0x18 */
  __IO uint32_t CCMR2;            /*!< TIM capture/compare mode
register 2,     Address offset 0x1C */
  __IO uint32_t CCER;             /*!< TIM capture/compare enable
register,     Address offset 0x20 */
  __IO uint32_t CNT;              /*!< TIM counter register,
Address offset 0x24 */
  __IO uint32_t PSC;              /*!< TIM prescaler register,
Address offset 0x28 */
  __IO uint32_t ARR;              /*!< TIM auto-reload register,
Address offset 0x2C */
       uint32_t RESERVED1;        /*!< Reserved */
  __IO uint32_t CCR1;           /*!< TIM capture/compare register
1,Address offset 0x34 */
  __IO uint32_t CCR2;           /*!< TIM capture/compare register
2,Address offset 0x38 */
  __IO uint32_t CCR3;           /*!< TIM capture/compare register
3,Address offset 0x3C */
  __IO uint32_t CCR4;           /*!< TIM capture/compare register
4,Address offset 0x40 */
       uint32_t RESERVED2[3];     /*!< Reserved */
  __IO uint32_t OR;               /*!< TIM option register,
Address offset 0x50 */
```

```
} TIM_TypeDef;
```

The register names are the register acronyms written in upper case for each peripheral. RESERVEDi (I being an integer that indexes the reserved field) indicates a reserved field.

Each peripheral has several dedicated registers which contain different flags. Registers are defined within a dedicated structure for each peripheral. Flags are defined as acronyms written in upper case and preceded by '**PPP_FLAG**_'. The flag definition is adapted to each peripheral case and defined in stm3w108xx_ppp.h.

Peripheral declaration

All peripherals are declared in stm32w108xx.h. The following example shows the declaration of the TIM peripheral:

```
...
#define PERIPH_BASE                  ((uint32_t)0x40000000)
 /*!< Peripheral base address in the alias region */

#define TIM1_BASE                    (PERIPH_BASE + 0xE000)
#define TIM2_BASE                    (PERIPH_BASE + 0xF000)

#define TIM1_IT_BASE                 (PERIPH_BASE + 0xA800)
#define TIM2_IT_BASE                 (PERIPH_BASE + 0xA804)

#define TIM1                         ((TIM_TypeDef *) TIM1_BASE)
#define TIM2                         ((TIM_TypeDef *) TIM2_BASE)

#define TIM1_IT                    ((TIM_IT_TypeDef *) TIM1_IT_BASE)
#define TIM2_IT                    ((TIM_IT_TypeDef *) TIM2_IT_BASE)
```

TIMx_BASE is the base address of a specific TIM and TIMx is a pointer to a register structure that refers to a specific TIM.

The peripheral registers are accessed as follows:

TIMx->CR1 = 0x0001;

Peripheral registers bits

All the peripheral registers bits are defined as constants in the stm32w108xx.h file. They are defined as acronyms written in upper-case into the form:

PPP_<register_name>_<bit_name>

Example:

```
/********** Bit definition for TIM_CR1 register  ***************/
#define  TIM_CR1_CEN ((uint32_t)0x00000001)        /*!< Counter
enable */
#define  TIM_CR1_UDIS         ((uint32_t)0x00000002)        /*!<
Update disable */
#define  TIM_CR1_URS          ((uint32_t)0x00000004)        /*!<
Update request source */
#define  TIM_CR1_OPM          ((uint32_t)0x00000008)        /*!<
One pulse mode */
#define  TIM_CR1_DIR          ((uint32_t)0x00000010)        /*!<
Direction */

#define  TIM_CR1_CMS           ((uint32_t)0x00000060)        /*!<
```

```
CMS[1:0] bits  (Center-aligned mode selection) */
#define  TIM_CR1_CMS_0          ((uint32_t)0x00000020)        /*!<
Bit 0 */
#define  TIM_CR1_CMS_1          ((uint32_t)0x00000040)        /*!<
Bit 1 */

#define  TIM_CR1_ARPE           ((uint32_t)0x00000080)        /*!<
Auto-reload Buffer enable */

#define  TIM_CR1_CKD            ((uint32_t)0x00000300)        /*!<
CKD[1:0] bits (clock division) */
#define  TIM_CR1_CKD_0          ((uint32_t)0x00000100)        /*!<
Bit 0 */
#define  TIM_CR1_CKD_1          ((uint32_t)0x00000200)        /*!<
Bit 1 */
```

1.1.4 Bit-Banding

The Cortex-M3 memory map includes two bit-band memory regions. These regions map each word in an alias region of memory to a bit in a bit-band region of memory. Writing to a word in the alias region has the same effect as a read/modify/write operation on the targeted bit in the bit-band region.

All the STM32W108xx peripheral registers are mapped in a bit-band region. This feature is consequently intensively used in functions which perform single bit set/reset in order to reduce and optimize code size.

The sections below describe how the bit-band access is used in the Library.

Mapping formula

The mapping formula shows how to link each word in the alias region to a corresponding target bit in the bit-band region. The mapping formula is given below:

bit_word_offset = (byte_offset x 32) + (bit_number × 4)
bit_word_addr = bit_band_base + bit_word_offset

where:

- bit_word_offset is the position of the target bit in the bit-band memory region
- bit_word_addr is the address of the word in the alias memory region that maps to the targeted bit.
- bit_band_base is the starting address of the alias region
- byte_offset is the number of the byte in the bit-band region that contains the targeted bit
- bit_number is the bit position (0-7) of the targeted bit.

1.1.5 Run-time checking

The library implements run-time failure detection by checking the input values of all library functions. The run-time checking is achieved by using an **assert_param** macro. This macro is used in all the library functions which have an input parameter. It allows checking 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 stm32w108xx_conf.h file.

Example:EXTI_ClearFlag function

stm32w108xx_exti.c:

```
void EXTI_ClearITPendingBit(uint32_t EXTI_IRQn)
{
  /* Check the parameters */
  assert_param(IS_EXTI_IRQ(EXTI_IRQn));
  EXTI->PR = (uint32_t)(1<< ((EXTI_IRQn & 0x000000F0) >>4));
}
```

stm32w108xx_exti.h:

```
/** @defgroup EXTI_IRQ_Sources
  * @{
  */
#define EXTI_IRQA       ((uint32_t)0x00000000)
#define EXTI_IRQB       ((uint32_t)0x00000010)
#define EXTI_IRQC       ((uint32_t)0x00000020)
#define EXTI_IRQD       ((uint32_t)0x00000031)

#define IS_EXTI_IRQ(EXTI_IRQ) (((EXTI_IRQ) == EXTI_IRQA) || \
                               ((EXTI_IRQ) == EXTI_IRQB) || \
                               ((EXTI_IRQ) == EXTI_IRQC) || \
                               ((EXTI_IRQ) == EXTI_IRQD))
/**
  * @}
```

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 **stm32w108xx_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 */
#endif /*
```

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 */
  printf("\n\r Wrong parameter value detected on\r\n");
  printf("       file  %s\r\n", file);
  printf("       line  %d\r\n", line);
  /* Infinite loop */
  while (1)
  {
  }
}
#endif /* USE_FULL_ASSERT */
```

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

However if you want to keep this functionality in your final application, reuse the **assert_param** macro defined within the library to test the parameter values before calling the library functions.

1.1.6 MISRA-C 2004 compliance

The C programming language is growing in importance for embedded systems. However, when it comes to developing code for safety-critical applications, this language has many drawbacks. There are several unspecified, implementation-defined, and undefined aspects of the C language that make it unsuited for developing safety-critical systems.

The Motor Industry Software Reliability Association's Guidelines for the use of the C language in critical systems (MISRA-C 2004 [1]) describe a subset of the C language well suited for developing safety-critical systems.

The STM32W108xx standard peripheral drivers (STM32W108xx_StdPeriph_Driver) have been developed to be MISRA-C 2004 compliant.

The following section describes how the StdPeriph_Driver complies with MISRA-C 2004 (as described in section 4.4 Claiming compliance of the standard [1]):

- A compliance matrix has been completed which shows how compliance has been enforced.
- The whole STM32W108xx_StdPeriph_Driver C code is compliant with MISRA-C 2004 rules. Deviations are documented.
- A list of all instances of rules not being followed is being maintained, and for each instance there is an appropriately signed-off deviation.
- All the issues listed in section 4.2 "The programming language and coding context of the standard" [1], that need to be checked during the firmware development phase, have been addressed during the development of the STM32W108xx standard peripherals driver and appropriate measures have been taken.

Compliance matrix

The compliance of the STM32W108xx standard peripherals driver (STM32W108xx_StdPeriph_Driver) with MISRA-C 2004 has been checked using the IAR

C/C++ Compiler for ARM. MISRA compliance applies only to STM32W108xx standard peripherals driver source file. Examples and project files are not MISRA compliant.

Two options are available for checking MISRA compliance:

- The compiler: IAR C/C++ Compiler for ARM V6.20
- Manual checking (code review)

The following table lists the MISCRA-C 2004 rules that are frequently violated in the code.

Table 2: MSIRA-C 2004 compliance matrix

MISRA-C 2004 rule number	Required/Advisory	Summary	Reason
1.1	Required	Compiler is configured to allow extensions - all code shall conform to ISO 9899 standard C, with no extensions permitted	IAR compiler extensions are enabled. This was allowed to support new CMSIS types.
5.1	Required	Identifiers (internal and external) shall not rely on significance of more than 31 characters	Some long parameters names are defined for code readability.
8.1	Required	No prototype seen - functions shall always have prototype declarations and the prototype shall be visible at both the function definition	This rule is violated as there is no function prototype for __WFI and __WFE macros in the CMSIS layer.
10.1	Required	The value of an expression of integer type shall not be implicitly converted to a different underlying type.	Complexity
10.6	Required	A 'U' suffix shall be applied to all constants of 'unsigned' type	The "stdint.h" defined types are used to be CMSIS compliant.
11.2	Required	Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.	Needed when addressing memory mapped registers
11.3	Advisory	A cast should not be performed between a pointer type and an integral type.	Needed when addressing memory mapped registers
16.7	Advisory	A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.	
19.1	Advisory	#include statements in a file shall only be preceded by other preprocessor directives or comments	This rule was violated to be in line with the CMSIS architecture.

How to check that your code is MISRA-C 2004 compliant

The default IAR project template provided with the STM32W108xx Standard Peripheral Library is already pre-configured for MISRA-C 2004 compliance. Then, the user has to enable the MISRA-C 2004 checker if needed.

To enable the IAR MISRA-C 2004 checker, go to Project->Options (ALT+F7) and then in "General Options" Category select the "MISRA-C:2004" tab and check the "Enable MISRA-C" box. With the default EWARM template project, all violated rules described above are unchecked.

To use the IAR MISRA-C Checker to verify that your code is MISRA-C 2004 compliant, please follow the following steps:

1. Enable the IAR MISRA-C 2004 Checker
2. Inside the core_cm3.h file add the following directive "#pragma system_include" to prevent the MISRA-C checker to check this file.
3. Uncomment the "USE_FULL_ASSERT" inside the STM32w108xx_conf.h file

Only the STM32W108xx standard peripherals driver is MISRA-C 2004 Compliant.

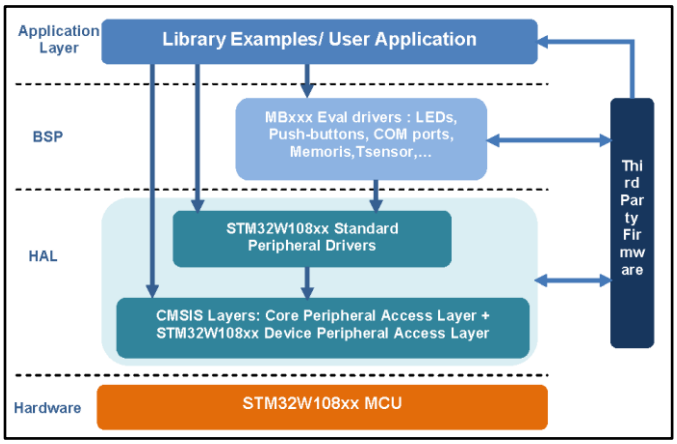
[1] MISRA-C 2004 Guidelines for the use of the C language in critical systems, Motor Industry Software Reliability Association, October 2004

1.2 Architecture

The library is built around a modular programming model ensuring the independencies between the several components building the main application and allowing an easy porting on a large product range, evaluation boards and even the use of some integrated firmware components for other application with the minimum changes on the code of the common parts.

The following figure provides a global view of the STM32W108xx Standard Peripheral Library usage and interaction with other firmware components.

Figure 1: Library architecture



HAL

HAL is a Hardware Abstraction Layer (HAL) that allows controlling the different STM32W108xx device's registers and features.

- CMSIS layer
  - Core Peripheral Access Layer: contains name definitions, address definitions and helper functions to access core registers and peripherals.
  - STM32W108xx Device Peripheral Access Layer: provides definitions for all the peripheral register's definitions, bits definitions and memory mapping for STM32W108xx devices.
- STM32W108xx standard peripheral driver that provides drivers and header files for all the peripherals. It uses CMSIS layer to access STM32W108xx registers.

BSP

BSP is a board specific package (BSP) that implements an abstraction layer to interact with the Human Interface resources; buttons, LEDs and COM ports (UARTs) available on STMicroelectronics evaluation boards (MBxxx). A common API is provided to manage these different resources, and can be easily tailored to support any other development board, by just adapting the initialization routine.

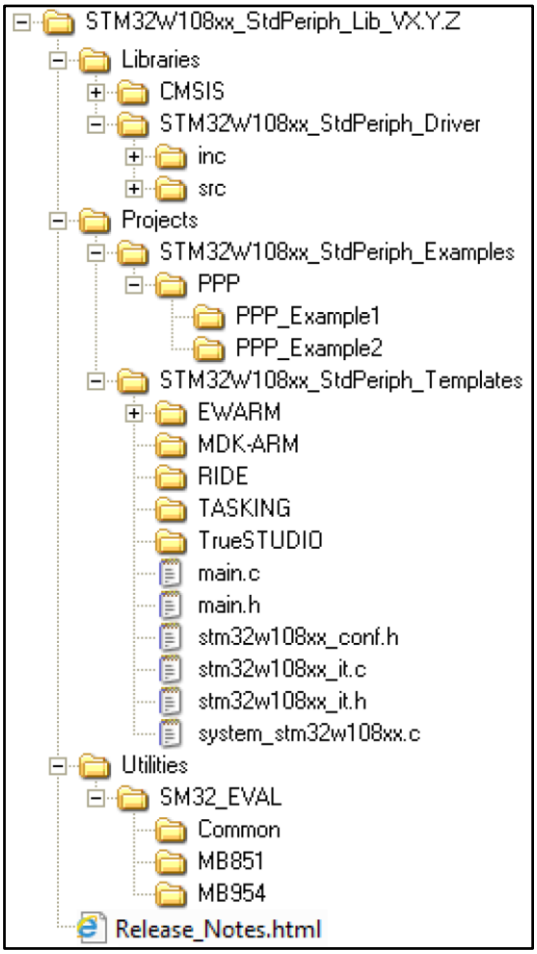
Application layer

The application layer consists of a set of examples covering all available peripherals with template projects for the most common development Tools. With the appropriate hardware evaluation board, this allows to get started with a brand new micro within few hours.

1.3 Package description

The Library is supplied in one single zip file. The extraction of the zip file generates one folder, STM32W108xx_StdPeriph_Lib_VX.Y.Z, which contains the following subfolders:

Figure 2: Library package structure



1. VX.Y.Z refer to the library version, ex. V1.0.0

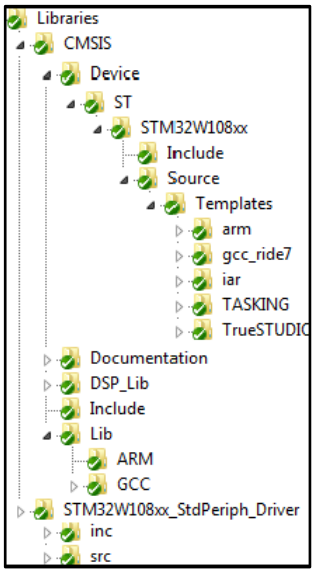
The library package consists of three main folders, described in *Section 1.3.1: "Library folder structure"*

1.3.1 Library folder structure

This folder contains all CMSIS files and STM32W108xx Standard Peripheral Drivers.

The library folder structure is shown in the figure below:

Figure 3: Library folder structure



CMSIS subfolder

This subfolder contains the STM32W108xx and Cortex-M3 CMSIS files:

- Cortex-M CMSIS files containing name definitions, address definitions and helper functions to access Cortex-M3 core registers and peripherals.
- STM32W108xx CMSIS files consist of:
  - stm32w108xx.h: this file contains the definitions of all peripheral registers, bits, and memory mapping for STM32W108xx devices.
  - system_stm32w108xx.c/.h: this file contains the system clock configuration for STM32W108xx devices. It exports SystemInit() function which sets up the system clock source. This function is called at startup just after reset and before connecting to the main program. The call is made inside the startup_stm32w108xx.s file.
  - startup_stm32w108xx.s: this file contains the Cortex-M3 startup code and interrupt vectors for all STM32W108xx device interrupt handlers.

STM32W108xx_StdPeriph_Driver subfolder

This subfolder contains all the subdirectories and files that make up the core of the library. They do not need to be modified by the user:

- inc subfolder contains the peripheral drivers header files.
- src subfolder contains the peripheral drivers source files.

Each peripheral has a source code file, stm32w108xx_ppp.c, and a header file, stm32w108xx_ppp.h. The stm32w108xx_ppp.c file contains all the firmware functions required to use the PPP peripheral.

The library files are listed and described in details in the following tables.

Table 3: Description of CMSIS files

File name	Description
core_cm3.c	Defines several help functions to access Cortex-M3 core registers.
core_cm3.h	Describes the data structures for the Cortex-M3 core peripherals and performs the address mapping of these structures. It also provides basic access to the Cortex-M3 core registers and core peripherals using efficient functions defined as static inline.
stm32w108xx.h	CMSIS Cortex-M3 STM32W108xx peripheral access layer header file. This file contains the definitions of all peripheral registers, bits, and memory mapping for STM32W108xx devices. It contains: • Configuration section allowing: – To select the device used in the target application – To use or not the peripheral drivers in your application code (meaning that the code is based on direct access to peripheral registers rather than drivers API). This option is controlled by #define USE_STDPERIPH_DRIVER – To change few application-specific parameters such as the HSE crystal frequency • Data structures and address mapping for all peripherals • Peripheral registers declarations and bits definition • Macros to access peripheral registers hardware This file also contains the library release number defined by the define statement __STM32W108XX_STDPERIPH_VERSION
system_stm32w108xx.c	This file contains the system clock configuration for STM32W108xx devices. This file includes two functions and one global variable to be called from the user application: • SystemInit(): this function setups the system clock source. This function is called at startup just after reset and before branch to the main program. The call is made inside the startup_stm32w108xx.s file. • SystemCoreClock: this variable contains the core clock. It can be used by the application code to set up the SysTick timer or configure other parameters. • SystemCoreClockUpdate(): this function updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.
system_stm32w108xx.h	Header file for system_stm32w108xx.c
startup_stm32w108xx.s	Provides the Cortex-M3 startup code and interrupt vectors for all STM32W108xx device interrupt handlers. This module performs the following functions:

File name	Description
	• It sets the initial SP • It sets the initial PC == Reset_Handler • It sets the vector table entries with the exceptions ISR address • It branches to __main in the C library (which eventually calls main()). A file is provided for each compiler.

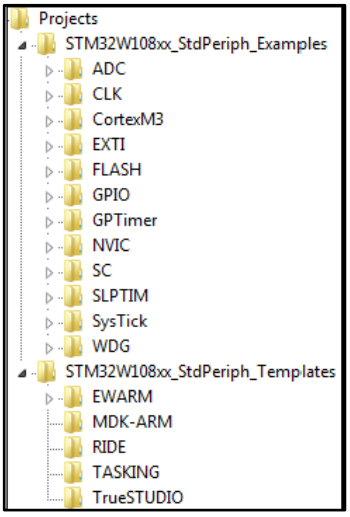
Table 4: STM32W108xx_StdPeriph_Driver files description

File name	Description
stm32w108xx_ppp.c	Driver source code file of PPP peripheral coded in Strict ANSI-C, and independent from the development Tools.
stm32w108xx_ppp.h	Provides functions prototypes and variable definitions used within for stm32w108xx_ppp.c file
misc.c	Provides all the miscellaneous firmware functions (add-on to CMSIS functions)
misc.h	Header for misc.c file

1.3.2 Project folder

This folder contains template projects and peripheral examples. Its structure is shown in the figure below.

Figure 4: Project folder structure



STM32W108xx_StdPeriph_Template subfolder

This subfolder contains standard template projects for the supported development tools that compile the needed STM32W108xx standard peripheral drivers plus all the user-modifiable files that are necessary to create a new project.

The files are listed and described in details in the following table.

Table 5: STM32W108xx_StdPeriph_Template files description

File name	Description
main.c	Template source file allowing starting a development from scratch using the library drivers.
main.h	header file for main.c
stm32w108xx_conf.h	Header file allowing to enable/disable the peripheral drivers header files inclusion. This file can also be used to enable or disable the library run-time failure detection before compiling the firmware library drivers, through the preprocessor define USE_FULL_ASSERT
system_stm32w108xx.c	This file contains the system clock configuration for STM32W108xx devices. This file provides two functions and one global variable to be called from user application: • SystemInit(): this function sets up the system clock source. This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32w108xx.s" file. • SystemCoreClock: this variable contains the core clock. It can be used by the user application to set up the SysTick timer or configure other parameters. • SystemCoreClockUpdate(): this function updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.
stm32w108xx_it.c	Template source file containing the interrupt service routine (ISR) for Cortex-M3 exceptions. You can add additional ISR(s) for the used peripheral(s) (for the available peripheral interrupt handler name, please refer to the startup file startup_stm32w108xx.s).
stm32w108xx_it.h	Header file for stm32w108xx_it.c

STM32W108xx_StdPeriph_Examples sub folder

This subfolder contains, for each peripheral, the minimum set of files needed to run a typical example on this peripheral. In addition to the user files described in the section above, each subfolder contains a readme.txt file describing the example and how to make it work.

For more details about the available examples within the library please refer to Library_Examples.html file located in the root of this folder.

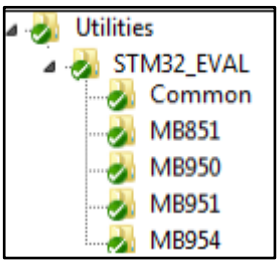
1.3.3 Utilities folder

This folder contains the abstraction layer allowing interacting with the human interface resources (buttons, LEDs and COM ports (UARTs)) available on STMicroelectronics evaluation boards. A common API is provided to manage these different resources. It can be easily tailored to support any other development board, by adapting the initialization routine.

Additional drivers are provided to manage the different memories and storage media available on these boards.

The Utilities folder structure is shown below.

Figure 5: Utilities folder structure



It contains common files and folder, plus a folder for STM32W108xx evaluation board files.

Table 6: Utilities/STM32_EVAL files description

File name	Description
mbxx.c	This file provides a set of firmware functions to manage LEDs, pushbuttons, and COM ports.
mbxx.h	Header file for mbxx.c.
mbxxx_i2c_ee.c	This file provides a set of functions needed to manage an I2C M24LR64 mounted on an MBxxx board.
mbxxx_i2c_ee.h	Header file for mbxxx_i2c_ee.c.
mbxxx_i2c_lis302dl.c	This file provides a set of functions needed to manage the LIS302DL MEMS accelerometer available on an MBxxx board.
mbxxx_i2c_lis302dl.h	Header file for mbxxx_i2c_lis302dl.c.
mbxxx_adc_tsensor.c	This file provides a set of functions needed to manage an ADC STLM20.
mbxxx_adc_tsensor.h	Header file for mbxxx_adc_tsensor.c.

1.4 Supported devices and development tools

1.4.1 Supported_devices_and_development_Tools.dita

Supported devices

The library supports all STM32W108xx microcontroller memory and peripherals. By using this library moving the application firmware from one STM32W108xx device to another becomes straightforward.

The device part number is defined as follows in stm32w108xx.h file:

```
/** Uncomment the line below according to the target STM32 device
used in your application
  */
#if !defined (STM32W108C8) && !defined (STM32W108CB) && !defined
(STM32W108CC) && !defined (STM32W108CZ) && !defined (STM32W108HB)
  /* #define STM32W108C8 */     /*!< STM32W108C8: 64k FLASH */
  /* #define STM32W108CB */     /*!< STM32W108CB: 128k FLASH */
  /* #define STM32W108CC */     /*!< STM32W108CC: 256k FLASH */
  /* #define STM32W108CZ */     /*!< STM32W108CZ: 192k FLASH */
  /* #define STM32W108HB */     /*!< STM32W108HB: 128k FLASH */
#endif
```

This define statement can be used at application level to configure the application firmware for STM32W108xx devices.

1.4.2 Supported development tools and compilers

STM32W108xx devices are supported by a full range of development solutions from lead suppliers that deliver start-to-finish control of application development from a single integrated development environment.

A template project is available for each development tool:

- **IAR Embedded Workbench for ARM (EWARM)** development tool
  - Compiler: IAR's C/C++
- **Keil (MDK-ARM)** development tool
  - Compiler: ARM C/C++ compiler
- **TASKING VX-toolset for ARM Cortex-M3** development tool
  - Compiler: Tasking VX C/C++
- **Raisonance IDE RIDE7 (RIDE)** development tool
  - Compiler: GNU C/C++
- **Atollic TrueSTUDIO STM32 (TrueSTUDIO)** development tool
  - Compiler: GNU C/C++ .

Refer to the library release notes to know about the supported development tool.

2 How to use and customize the library

The following sections explain all the steps required to configure, customize, run your first example, and develop your application based on the library.

2.1 Library configuration parameters

The configuration interface allows customizing the library for your application. It is not mandatory to modify this configuration and you can use the default configuration without any modification.

To configure these parameters, you 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 7: Library configuration parameters

Parameter	File	Description
STM32W108XX(1)	stm3w108x x.h	Default status: enabled Defines the root number of STM3W108xx devices. This define statement can be used at application level to configure the application firmware for STM3W108xx.
USE_STDPERIPH_DRIVER(1)	stm3w108x x.h	Default status: disabled When disabled, the peripheral drivers are not included and the application code is based on direct access to peripherals registers.
HSE_VALUE	stm3w108x x.h	Default value: 24 MHz Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.
HSE_STARTUP_TIMEOUT	stm3w108x x.h	Default value: 0x0500 Defines the maximum external oscillator (HSE) startup timeout value. The user must adjust this define statement when using a different statement startup time.
HSI_VALUE	stm3w108x x.h	Default value: 12 MHz Defines the value of the internal oscillator (HSI) expressed in Hz.

__MPU_PRESENT
__NVIC_PRIO_BITS
__Vendor_SysTickConfig
__FPU_PRESENT
	stm3w108x x.h	These define statements are used by Cortex-M3 CMSIS layer to inform about the options supported by STM3W108xx devices:
		`#define __CM3_REV 0x0101 /*!< Core revision r1p1`
		`#define __MPU_PRESENT 1 /*!< STM32W108XX provides an MPU`
		`#define __NVIC_PRIO_BITS 4 /*!< STM32W108XX uses 4 Bits for the Priority Levels */`
		`#define __Vendor_SysTickConfig 0 /*!< Set to 1 if different SysTick Config is used */`

Parameter	File	Description
		`#define __FPU_PRESENT 0 /*!< FPU is not present` They should not be modified by the user.
USE_FULL_ASSERT	stm3w108xx_conf.h	Default status: disabled This define statement is used to enable or disable the library run-time failure detection before compiling the firmware library drivers. When enabled, the "assert_param" macro is expanded in the library drivers code. Run-time detection can be used for user application development and debugging. It adds an overhead which can be removed from the final application code to minimize code size and maximize execution speed.

Peripheral header file inclusion	stm3w108xx_conf.h	This file allows to enable/disable the inclusion of the peripheral driver header files. By default all header files are included.
		
		`#include "stm32w108xx_adc.h"`
		`#include "stm32w108xx_rst.h"`
		`#include "stm32w108xx_gpio.h"`
		`#include "stm32w108xx_exti.h"`
		`#include "stm32w108xx_sc.h"`
		`#include "stm32w108xx_flash.h"`
		`#include "stm32w108xx_tim.h"`
		`#include "stm32w108xx_wdg.h"`
		`#include "stm32w108xx_slptim.h"`
		`#include "stm32w108xx_pwr.h"`
		`#include "stm32w108xx_clk.h"`
		`#include "stm32w108xx_misc.h"`

VECT_TAB_OFFSET	system_stm32w108xx.c	Default value: 0x00 Defines the vector table base offset. It must be a multiple of 0x200. Use this define statement to build an application that will be loaded at an address different from the Flash memory base address (for example, when building an application to be loaded through in-application programming (IAP) program).

Notes:

(1)These define statements are declared in the compiler preprocessor section of the template projects provided within the library. As a consequence, you do not need to enable them in the corresponding header file.

2.2 Library programming model

Direct register Access

This model is based on direct register access using the CMSIS layer. This layer provides the definition of all STM3W108xx peripheral registers and bits, as well as memory mapping.

The advantage of this approach if that the code produced is compact and efficient. The drawback is that the developer should know in details the peripheral operation, registers and bits meaning, and the configuration procedure. This task is time consuming, and might lead to programming errors, which may slow down the project development phase.

To use this model, proceed as follows:

1. Comment the line *#define USE_STDPERIPH_DRIVER* in *stm32w108xx.h* file. Make sure that this define statement is not defined in the compiler preprocessor section.
2. Use peripheral registers structure and bits definition available within *stm32w108xx.h* to build the application

Peripheral driver access

In this model the application code uses the peripheral driver API to control the peripheral configuration and operation. It allows any device to be used in the user application without the need for in-depth study of each peripheral specification. As a result, using the peripheral drivers saves significant time that would otherwise be spent in coding, while reducing the application development and integration cost.

However, since the drivers are generic and cover all peripherals functionalities, the size and/or execution speed of the application code may not be optimized.

To use this model, proceed as follows:

1. Add the line *#define USE_STDPERIPH_DRIVER* in the compiler preprocessor section or uncomment the line *#define USE_STDPERIPH_DRIVER* in *stm32w108xx.h*.
2. In *stm32w108xx_conf.h* file, select the peripherals to include their header file (by default all header files are included in the template file)
3. Use the peripheral drivers API provided by stm32w108xx_ppp.h/.c files under Libraries\STM3W108xx_StdPeriph_Driver to build your application. For more information, refer to the detailed description of each peripheral driver.
4. In addition to the peripheral drivers, you can reuse/adapt the rich set of examples available within the library. This reduces your application development time and allows you to start within few hours.

For many applications, the peripheral drivers can be used as is. However, for applications having tough constraints in terms of code size and/or execution speed, these drivers should be used as reference on how to configure the peripherals and tailor them to specific application requirements, in combination with peripheral direct register access.

The application code performance in terms of size and/or speed depends also on the C compiler optimization settings. To help you make the application code smaller, faster or balanced between size and speed, fine tune the optimizations according to your application needs. For more information please refer to your C compiler documentation.

2.3 Peripheral initialization and configuration

This section describes step by step how to initialize and configure a peripheral. The peripheral is referred to as PPP.

1. In the main application file, declare a **PPP_InitTypeDef** structure, for example:
   ```
   PPP_InitTypeDef  PPP_InitStructure;
   ```
   The PPP_InitStructure is a working variable located in data memory area. It allows to initialize one or more PPP instances.
2. Fill the PPP_InitStructure variable with the allowed values of the structure member. Two solutions are possible:
   a. Configure the whole structure by following the procedure described below:
      ```
      PPP_InitStructure.member1 = val1;
      PPP_InitStructure.member2 = val2;
      PPP_InitStructure.memberN = valN;
      /* where N is the number of the structure members */
      ```
      The previous initialization step can be merged in one single line to optimize the code size:
      ```
      PPP_InitTypeDef  PPP_InitStructure = { val1, val2,.., valN}
      ```
   b. Configure only a few members of the structure: in this case modify the PPP_InitStructure variable that has been already filled by a call to the **PPP_StructInit(..)** function. This ensures that the other members of the PPP_InitStructure variable are initialized to the appropriate values (in most cases their default values).
      ```
      PPP_StructInit(&PPP_InitStructure);
      PPP_InitStructure.memberX = valX;
      PPP_InitStructure.memberY = valY;
      /*where X and Y are the members the user wants to
      configure*/
      ```
3. Initialize the PPP peripheral by calling the **PPP_Init(..)** function.
   ```
   PPP_Init(PPP, &PPP_InitStructure);
   ```
4. At this stage the PPP peripheral is initialized and can be enabled by making a call to **PPP_Cmd(..)** function.
   ```
   PPP_Cmd(PPP, ENABLE);
   ```
   The PPP peripheral can then be used through a set of dedicated functions. These functions are specific to the peripheral. For more details refer to the peripheral driver chapter.

PPP_DeInit(..) function can be used to set all PPP peripheral registers to their default values (only for debug purpose):

```
PPP_DeInit(PPP);
```

To modify the peripheral settings after configuring it, you have to proceed as follows:

```
PPP_InitStucture.memberX = valX;
PPP_InitStructure.memberY = valY;
PPP_Init(PPP, &PPP_InitStructure);
/* where X and Y are the only members that user wants to modify*/
```

2.4 How to run your first example

The library provides a rich set of examples covering the main features of each peripheral. All the examples are independent from the development tools. These examples run on STMicroelectronics MBxxx evaluation board and can be easily tailored to any other supported device and development board. Only source files are provided for each example and user can tailor the provided project template to run the selected example with his preferred development Tool.

2.4.1 Prerequisites

1. Latest release of documents and library.
   You can download the latest version of STM32W108xx related documents and library from STMicroelectronics web site: www.st.com/stm32
2. Hardware: to run the examples, you need an MBxxx evaluation board from STMicroelectronics or any other compatible hardware.
3. To use your own hardware, simply adapt the example hardware configuration to your platform.
4. Development tools
   Use your preferred development tool, MDK-ARM (Keil), EWARM (IAR), RIDE (Raisonance), TASKING or TrueSTUDIO (Atollic). Just check that the version you are using supports STM32W108xx devices (see section *Section 1.4.2: "Supported development tools and compilers"*

2.4.2 Run your first example

This section describes how to load and execute the template example (empty main.c) provided within the library.

To achieve this goal you have to proceed as described below:

1. Download and unzip the STM32W108xx_StdPeriph_Lib_VX.Y.Z.zip in the folder of your choice.
2. Power-up the MBxxx evaluation board.
3. Connect your JTAG probe to the JTAG connector of the EVAL board and to the USB port of your PC. The MBxxx evaluation board features a build-in J-LINK debugger and programmer which makes the external hardware debuggers useless to load and debug your program. Simply select J-LINK as your debugger in your Development Tool configuration menu and connect the J2 to your host PC through an USB cable. Refer to your development tool documentation to know if it supports the J-LINK debugger.
4. Run the template example: go to STM32W108xx_StdPeriph_Lib_VX.Y.Z\Projects\STM32W108xx_StdPeriph_Template folder, and proceed as follows depending on the development tool you are using:
   a. EWARM
      a. Open the EWARM\Project.eww workspace
      b. Rebuild all files: Project->Rebuild all
      c. Load project image: Project->Debug
      d. Run program: Debug->Go(F5)
   b. MDK-ARM
      a. Open the MDK-ARM\Project.uvproj project
      b. Rebuild all files: Project->Rebuild all target files
      c. Load project image: Debug->Start/Stop Debug Session
      d. Run program: Debug->Run (F5)
   c. TrueSTUDIO
      a. Open the TrueSTUDIO development tool.

b. Click File->Switch Workspace->Other and browse to TrueSTUDIO workspace directory.
c. Click File->Import, select General->Existing Projects into Workspace and then click Next.
d. Browse to the TrueSTUDIO workspace directory and select the STM32W108xx evaluation project
e. Rebuild all project files: Select the project in the "Project explorer" window then click on Project->build project menu.
f. Run program: Select the project in the "Project explorer" window then click Run->Debug (F11)

d. RIDE
a. Open the Project.rprj project
b. Rebuild all files: Project->build project
c. Load project image: Debug->start(ctrl+D)
d. Run program: Debug->Run(ctrl+F9)

e. TASKING
a. Open the TASKING toolchain.
b. Click on File->Import, select General->'Existing Projects into Workspace' and click Next
c. Browse to TASKING workspace directory and select the STM32W108xx evaluation project to configure the project for STM32W108xx devices
d. Rebuild all project files by selecting the project in the "Project explorer" window and clicking on Project->build project menu
e. Run the program by selecting the project in the "Project explorer" window and clicking Run->Debug (F11).

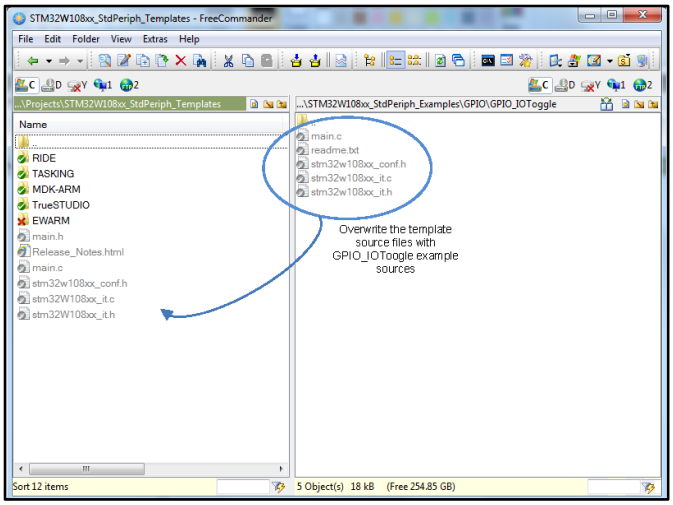
2.4.3 Run a peripheral example

Only the source files of the library peripheral examples are provided. You can tailor the project template provided to run the selected example with your development tool.

As an example, the following sequence is required to run the GPIO_IOToggle example:

1. Copy all source files from Project\STM32W108xx_StdPeriph_Examples\GPIO\GPIO_IOToggle to the template folder under Project\STM32W108xx_StdPeriph_Templates (see *Figure 6: "How to run a peripheral example "*).
2. Open your preferred development tool, and proceed as described in *Section 2.4.2: "Run your first example"*
3. If the example uses additional source files which are not included in the template project, add manually the files to the project source list. Refer to the readme.txt file of your example for more details.

Figure 6: How to run a peripheral example



2.5 How to develop your first application

This section describes all steps required for using and customizing the library to build an application from scratch. It gives a real example based on the requirements described below:

- MBxxx evaluation board used as reference hardware
- System clock configured to 24 MHz, with 1 Flash wait state
- EXTI_IRQC (mapped to PB3) configured to generate an interrupt on each rising or falling edge
- In the interrupt routine, the 2 LEDs connected to specific GPIO pins are toggled
- When EXTI has been configured, a hardware interrupt (pushbutton S1 on the MB851 board) is generated on EXTI_IRQC and makes LED1 and LED3 toggle.

2.5.1 Starting point

The example to start with is EXTI_PushButton provided within the library package (\Projects\STM32W108xx_StdPeriph_Examples\EXTI\).

The template folder contains all the required template files as well as the project files for different development tools.

Reuse the template files as follow:

- main.c: first move the template main.c file to another location (to backup the template for future use), then create a new empty C file and rename it to main.c. This file will be used to implement the program code as described in the section below.
- stm32w108xx_it.c: use this template file to add the code required to manage the EXTI Line interrupt.
- stm32w108xx_it.h: use this template file to add the EXTI Line interrupt prototype.
- stm32w108_conf.h: use this template file without any change
- system_stm32w108xx.c: use the template file without any change

2.5.2 Library configuration parameters

To configure the library for your application, use the library default parameters as defined in *Table 7: "Library configuration parameters"*

2.5.3 system_stm32w108xx.c

This file contains the SystemInit() function that configures the system clock source. This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32w108xx.s" file.

2.5.4 main.c

The main.c file calls the library driver functions to configure the EXTI, GPIO and NIVC peripherals.

Include the library and MBxxx evaluation board resources:

```
/* Includes ------------------------------------------------------*/
#if defined (USE_MB851_REVA_REVB) || defined (USE_MB851_REVC) ||
defined (USE_MB851_REVD)
 #include "mb851.h"
#elif defined (USE_MB954_REVA) || defined (USE_MB954_REVB) ||
defined (USE_MB954_REVC)
 #include "mb954.h"
#elif defined (USE_MB951)
 #include "mb951.h"
#elif defined (USE_MB950)
 #include "mb950.h"
#else
 #error "Please select first the STM32W board to be used."
#endif /* USE_MB851_REVA_REVB or USE_MB851_REVC or USE_MB851_REVD*/
```

2.5.5 mbxxx.c

Declare three structure variables, used to initialize the EXTI, GPIO and NIVC peripherals

```
/* Private typedef -----------------------------------------------*/
EXTI_InitTypeDef   EXTI_InitStructure;
GPIO_InitTypeDef   GPIO_InitStructure;
NVIC_InitTypeDef   NVIC_InitStructure;
```

Configure Button pin as input

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_PUD;
  GPIO_InitStructure.GPIO_Pin = BUTTON_PIN[Button];
  GPIO_Init(BUTTON_PORT[Button], &GPIO_InitStructure);
```

Set pull-up on button I/O

```
GPIO_SetBits(BUTTON_PORT[Button], BUTTON_PIN[Button]);
```

Configure Button EXTI line

```
EXTI_InitStructure.EXTI_Source = BUTTON_EXTI_SOURCE[Button];
  EXTI_InitStructure.EXTI_IRQn = BUTTON_EXTI_IRQn[Button];
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling_Edge;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_InitStructure.EXTI_DigitalFilterCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
```

Enable and set Button EXTI Interrupt to the lowest priority

```
NVIC_InitStructure.NVIC_IRQChannel = BUTTON_IRQn[Button];
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
```

Configure LED1 and LED3 in output push-pull mode (main.c)

```
STM_EVAL_LEDInit(LED1);
  STM_EVAL_LEDInit(LED3);
```

The stm32w108xx_it.c file can be used to implement the EXTI Line interrupt service routine (ISR).

1. In "STM32W108xx Peripherals Interrupt Handlers" section, add the following code

```
/*******************************************************************/
/*          stm32w108xx Peripherals Interrupt Handlers             */
/*  Add here the Interrupt Handler for the used peripheral(s)
*/    /* (PPP), for the available peripheral interrupt
handler's     */
/* name please refer to the startup
file(startup_stm32w108xx.s)*/
/*******************************************************************/
/**
  * @brief  This function handles IRQC interrupt request.
  * @param  None
  * @retval None
  */
void EXTIC_IRQHandler(void)
{
  while(STM_EVAL_PBGetState(BUTTON_S1) != SET)
  {}
  if(EXTI_GetITStatus(EXTI_IRQC) == SET)
  {
    /* Toggle LED1 and LED3 I/O */
    GPIO_PORT[0]->ODR ^= BSRR_VAL1;
    GPIO_PORT[1]->ODR ^= BSRR_VAL2;
    EXTI_ClearITPendingBit(EXTI_IRQC);
  }
}
```

2. In stm32w108xx _it.h file add the EXTI Line ISR prototype as follows (just after the line void SysTick_Handler(void);)

```
void EXTIC_IRQHandler(void)
```

3 Analog-to-digital converter (ADC)

3.1 ADC Firmware driver registers structures

3.1.1 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32w108xx_adc.h

Data Fields

- ***uint32_t ADC_Resolution***
- ***uint32_t ADC_VoltageP***
- ***uint32_t ADC_VoltageN***
- ***uint32_t ADC_Input***
- ***uint32_t ADC_Clock***
- ***uint32_t ADC_DMAMode***
- ***int32_t ADC_Offset***
- ***uint32_t ADC_Gain***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ADC_Resolution***
  - Selects the resolution of the conversion. This parameter can be a value of ***ADC_Resolution***
- ***uint32_t ADC_InitTypeDef::ADC_VoltageP***
  - Selects the voltage range for the P input channel. This parameter can be a Low voltage range or High voltage range
- ***uint32_t ADC_InitTypeDef::ADC_VoltageN***
  - Selects the voltage range for the N input channel. This parameter can be a Low voltage range or High voltage range
- ***uint32_t ADC_InitTypeDef::ADC_Input***
  - Selects the channels. This parameter must range from 0x0 to 0x5 and from 0x9 to 0xB. The other values are reserved
- ***uint32_t ADC_InitTypeDef::ADC_Clock***
  - Selects the ADC clock. This parameter must be 0 or 1
- ***uint32_t ADC_InitTypeDef::ADC_DMAMode***
  - Selects the ADC DMA mode. This parameter must be linear or auto wrap
- ***int32_t ADC_InitTypeDef::ADC_Offset***
  - Specifies the offset added to the basic ADC conversion result. This parameter must be 16 bits signed offset
- ***uint32_t ADC_InitTypeDef::ADC_Gain***
  - Specifies the gain factor that is multiplied by the offset-corrected ADC result to produce the output value. This parameter must be 16 bits unsigned gain

3.1.2 ADC_TypeDef

ADC_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t ISR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t IER***
- ***uint32_t RESERVED1***
- ***__IO uint32_t CR***
- ***__IO uint32_t OFFSETR***
- ***__IO uint32_t GAINR***
- ***__IO uint32_t DMACR***
- ***__IO uint32_t DMASR***
- ***__IO uint32_t DMAMSAR***
- ***__IO uint32_t DMANDTR***
- ***__IO uint32_t DMAMNAR***
- ***__IO uint32_t DMACNDTR***

Field Documentation

- ***__IO uint32_t ADC_TypeDef::ISR***
  - ADC Interrupt Status Register, Address offset: 0xA810
- ***uint32_t ADC_TypeDef::RESERVED0[15]***
  - Reserved
- ***__IO uint32_t ADC_TypeDef::IER***
  - ADC Interrupt Enable Register, Address offset: 0xA850
- ***uint32_t ADC_TypeDef::RESERVED1[2540]***
  - Reserved.
- ***__IO uint32_t ADC_TypeDef::CR***
  - ADC Configuration Register, Address offset: 0xD004
- ***__IO uint32_t ADC_TypeDef::OFFSETR***
  - ADC Offset Register, Address offset: 0xD008
- ***__IO uint32_t ADC_TypeDef::GAINR***
  - ADC Gain Register, Address offset: 0xD00C
- ***__IO uint32_t ADC_TypeDef::DMACR***
  - ADC DMA Configuration Register, Address offset: 0xD010
- ***__IO uint32_t ADC_TypeDef::DMASR***
  - ADC DMA Status Register, Address offset: 0xD014
- ***__IO uint32_t ADC_TypeDef::DMAMSAR***
  - ADC DMA Memory Start Address Register, Address offset: 0xD018
- ***__IO uint32_t ADC_TypeDef::DMANDTR***
  - ADC DMA Number of Data Register, Address offset: 0xD01C
- ***__IO uint32_t ADC_TypeDef::DMAMNAR***
  - ADC DMA Memory Next Address Register, Address offset: 0xD020
- ***__IO uint32_t ADC_TypeDef::DMACNDTR***
  - ADC DMA Channel Number of Data Register, Address offset: 0xD024

3.2 ADC Firmware driver API description

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

3.2.1 How to use this driver

1. Configure GPIO pins to be used by the ADC in analog mode.
2. Configure the voltage reference (internal or external).
3. Set the offset and gain values.
4. Reset the ADC DMA, define the DMA buffer, and start the DMA in the proper transfer mode.
5. Write the ADC configuration register to define the inputs, voltage range, sample time and start the conversions.

3.2.2 Initialization and Control functions

This section provides functions allowing to:

- Initialize and configure the ADC
- ADC Conversion Resolution (5bits --> 12bits)
- Enable or disable the ADC peripheral
- ***ADC_DeInit()***
- ***ADC_Init()***
- ***ADC_StructInit()***
- ***ADC_Cmd()***

3.2.3 ADC channel Configuration functions

This section provides function allowing to configure the ADC channels

- ***ADC_ChannelConfig()***

3.2.4 DMA Configuration functions

This section provides functions allowing to:

- initialize and configure the DMA
- reset and enable the DMA
- ***ADC_DMA_Config()***
- ***ADC_DMA_ChannelLoadEnable()***
- ***ADC_DMA_ChannelReset()***
- ***ADC_DMA_GetNextAddress()***
- ***ADC_DMA_GetCounter()***

3.2.5 Interrupts and flags management functions

This section provides functions allowing to :

- configure the ADC Interrupts and to get the status and clear flags and Interrupts pending bits.
- get the status of DMA and clear flags

Flags and Interrupts for ADC

- Flags :
  a. ADC_IT_DMABHF: DMA buffer half full interrupt pending
  b. ADC_IT_DMABF: DMA buffer full interrupt pending
  c. ADC_IT_SAT: Gain correction saturation interrupt pending
  d. ADC_IT_DMAOVF: DMA buffer overflow interrupt pending
- Interrupts :
  a. ADC_IT_DMABHF: DMA buffer half full interrupt enable
  b. ADC_IT_DMABF: DMA buffer full interrupt enable
  c. ADC_IT_SAT: Gain correction saturation interrupt enable
  d. ADC_IT_DMAOVF: DMA buffer overflow interrupt enable

Flags for ADC_DMA

- Flags :
  a. ADC_FLAG_ACT: DMA active
  b. ADC_FLAG_OVF: DMA over flow
- ***ADC_ITConfig()***
- ***ADC_DMA_GetFlagStatus()***
- ***ADC_GetITStatus()***
- ***ADC_ClearITPendingBit()***

3.2.6 Initialization and configuration functions

3.2.6.1 ADC_DeInit

Function Name	**void ADC_DeInit (void)**
Function Description	Deinitializes ADC peripheral registers to their default reset values.
Parameters	• None.
Return values	• None.
Notes	• None.

3.2.6.2 ADC_Init

Function Name	**void ADC_Init (*ADC_InitTypeDef** * **ADC_InitStruct)**
Function Description	Initializes the ADC peripheral according to the specified parameters in the ADC_InitStruct.

Parameters	• **ADC_InitStruct :** pointer to ADC_InitTypeDef structure that contains the configuration information for the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.6.3 ADC_StructInit

Function Name	**void ADC_StructInit (*ADC_InitTypeDef** * ADC_InitStruct)**
Function Description	Fills each ADC_InitStruct member with its default value.
Parameters	• **ADC_InitStruct :** pointer to a ADC_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

3.2.6.4 ADC_Cmd

Function Name	**void ADC_Cmd (FunctionalState NewState)**
Function Description	Enables or disables the specified ADC peripheral.
Parameters	• **NewState :** new state of the ADC peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.7 ADC channel configuration function

3.2.7.1 ADC_ChannelConfig

Function Name	**void ADC_ChannelConfig (uint32_t ADC_Channels)**

Function Description	Selects the ADC channel.
Parameters	• **ADC_Channels :** specifies the ADC channel This parameter can be one of the following values: – ***ADC_SOURCE_ADC0_VREF2 :*** – ***ADC_SOURCE_ADC0_GND :*** – ***ADC_SOURCE_ADC1_VREF2 :*** – ***ADC_SOURCE_ADC1_GND :*** – ***ADC_SOURCE_ADC2_VREF2 :*** – ***ADC_SOURCE_ADC2_GND :*** – ***ADC_SOURCE_ADC3_VREF2 :*** – ***ADC_SOURCE_ADC3_GND :*** – ***ADC_SOURCE_ADC4_VREF2 :*** – ***ADC_SOURCE_ADC5_VREF2 :*** – ***ADC_SOURCE_ADC1_ADC0 :*** – ***ADC_SOURCE_ADC0_ADC1 :*** – ***ADC_SOURCE_ADC3_ADC2 :*** – ***ADC_SOURCE_ADC2_ADC3 :*** – ***ADC_SOURCE_ADC5_ADC4 :*** – ***ADC_SOURCE_GND_VREF2 :*** – ***ADC_SOURCE_VGND :*** – ***ADC_SOURCE_VREF_VREF2 :*** – ***ADC_SOURCE_VREF :***
Return values	• None.
Notes	• None.

3.2.8 Channel DMA configuration functions

3.2.8.1 ADC_DMA_Config

Function Name	**void ADC_DMA_Config (uint32_t ADC_DMABeg, uint32_t ADC_DMASize)**
Function Description	Selects the specified DMA configuration.
Parameters	• **ADC_DMABeg :** specifies the ADC buffer start address • **ADC_DMASize :** specifies the ADC buffer size
Return values	• None.
Notes	• None.

3.2.8.2 ADC_DMA_ChannelLoadEnable

Function Name	**void ADC_DMA_ChannelLoadEnable (void)**
Function Description	Enables specified ADC DMA Channel.
Parameters	• None.
Return values	• None.
Notes	• None.

3.2.8.3 ADC_DMA_ChannelReset

Function Name	**void ADC_DMA_ChannelReset (void)**
Function Description	Reset specified ADC DMA.
Parameters	• None.
Return values	• None.
Notes	• None.

3.2.8.4 ADC_DMA_GetNextAddress

Function Name	**uint32_t ADC_DMA_GetNextAddress (void)**
Function Description	Gets the the location that will be written next by the DMA.
Parameters	• None.
Return values	• **the current DMA address**
Notes	• None.

3.2.8.5 ADC_DMA_GetCounter

Function Name	**uint32_t ADC_DMA_GetCounter (void)**
Function Description	Gets the number of 16-bit conversion results that have been written to the buffer.
Parameters	• None.
Return values	• **The number of conversions**
Notes	• None.

3.2.9 Interrupts and flags management functions

3.2.9.1 ADC_ITConfig

Function Name	**void ADC_ITConfig (uint32_t ADC_IT, FunctionalState NewState)**
Function Description	Enables or disables the specified ADC interrupts.
Parameters	• **ADC_IT :** specifies the ADC interrupt sources to be enabled or disabled This parameter can be one of the following values: – ***ADC_IT_DMABHF :*** DMA buffer half full interrupt enable – ***ADC_IT_DMABF :*** DMA buffer full interrupt enable – ***ADC_IT_SAT :*** Gain correction saturation interrupt enable – ***ADC_IT_DMAOVF :*** DMA buffer overflow interrupt enable • **NewState :** new state of the specified ADC interrupts This parameter can be: ENABLE or DISABLE
Return values	• None.
Notes	• None.

3.2.9.2 ADC_DMA_GetFlagStatus

Function Name	**FlagStatus ADC_DMA_GetFlagStatus (uint32_t ADC_DMA_FLAG)**
Function Description	Checks whether the specified ADC interrupt has occurred or not.
Parameters	• **ADC_DMA_FLAG :** specifies the flag to check This parameter can be one of the following values: – ***ADC_FLAG_ACT :*** DMA active – ***ADC_FLAG_OVF :*** DMA over flow
Return values	• **The new state of ADC_DMA_FLAG (SET or RESET).**
Notes	• None.

3.2.9.3 ADC_GetITStatus

Function Name	**ITStatus ADC_GetITStatus (uint32_t ADC_IT)**
Function Description	Checks whether the specified ADC pending interrupt has occurred or not.
Parameters	• **ADC_IT :** specifies the flag to check This parameter can be one of the following values: – ***ADC_IT_DMABHF :*** DMA buffer half full interrupt pending – ***ADC_IT_DMABF :*** DMA buffer full interrupt pending – ***ADC_IT_SAT :*** Gain correction saturation interrupt pending – ***ADC_IT_DMAOVF :*** DMA buffer overflow interrupt pending
Return values	• **The new state of ADC_IT (SET or RESET)**
Notes	• None.

3.2.9.4 ADC_ClearITPendingBit

Function Name	**void ADC_ClearITPendingBit (uint32_t ADC_IT)**
Function Description	Clears the ADC interrupt pending bits.

Parameters

- **ADC_IT :** specifies the flag to clear This parameter can be one of the following values:
  - ***ADC_IT_DMABHF :*** DMA buffer half full interrupt pending
  - ***ADC_IT_DMABF :*** DMA buffer full interrupt pending
  - ***ADC_IT_SAT :*** Gain correction saturation interrupt pending
  - ***ADC_IT_DMAOVF :*** DMA buffer overflow interrupt pending

Return values

- None.

Notes

- None.

3.3 ADC Firmware driver defines

3.3.1 ADC

ADC

ADC_Channels

- #define: ***ADC_MUX_ADC0 ((uint32_t)0x00000000)***
- #define: ***ADC_MUX_ADC1 ((uint32_t)0x00000001)***
- #define: ***ADC_MUX_ADC2 ((uint32_t)0x00000002)***
- #define: ***ADC_MUX_ADC3 ((uint32_t)0x00000003)***
- #define: ***ADC_MUX_ADC4 ((uint32_t)0x00000004)***
- #define: ***ADC_MUX_ADC5 ((uint32_t)0x00000005)***
- #define: ***ADC_MUX_GND ((uint32_t)0x00000008)***

- #define: ***ADC_MUX_VREF2 ((uint32_t)0x00000009)***

- #define: ***ADC_MUX_VREF ((uint32_t)0x0000000A)***

- #define: ***ADC_MUX_VREG2 ((uint32_t)0x0000000B)***

- #define: ***ADC_MUXN_BITS ((uint32_t)0x00000004)***

- #define: ***ADC_SOURCE_ADC0_VREF2 ((ADC_MUX_ADC0 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_ADC0_GND ((ADC_MUX_ADC0 << ADC_MUXN_BITS) | ADC_MUX_GND)***

- #define: ***ADC_SOURCE_ADC1_VREF2 ((ADC_MUX_ADC1 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_ADC1_GND ((ADC_MUX_ADC1 << ADC_MUXN_BITS) | ADC_MUX_GND)***

- #define: ***ADC_SOURCE_ADC2_VREF2 ((ADC_MUX_ADC2 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_ADC2_GND ((ADC_MUX_ADC2 << ADC_MUXN_BITS) | ADC_MUX_GND)***

- #define: ***ADC_SOURCE_ADC3_VREF2 ((ADC_MUX_ADC3 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_ADC3_GND ((ADC_MUX_ADC3 << ADC_MUXN_BITS) | ADC_MUX_GND)***

- #define: ***ADC_SOURCE_ADC4_VREF2 ((ADC_MUX_ADC4 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_ADC5_VREF2 ((ADC_MUX_ADC5 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_ADC1_ADC0 ((ADC_MUX_ADC1 << ADC_MUXN_BITS) | ADC_MUX_ADC0)***

- #define: ***ADC_SOURCE_ADC0_ADC1 ((ADC_MUX_ADC1 << ADC_MUXN_BITS) | ADC_MUX_ADC0)***

- #define: ***ADC_SOURCE_ADC3_ADC2 ((ADC_MUX_ADC3 << ADC_MUXN_BITS) | ADC_MUX_ADC2)***

- #define: ***ADC_SOURCE_ADC2_ADC3 ((ADC_MUX_ADC3 << ADC_MUXN_BITS) | ADC_MUX_ADC2)***

- #define: ***ADC_SOURCE_ADC5_ADC4 ((ADC_MUX_ADC5 << ADC_MUXN_BITS) | ADC_MUX_ADC4)***

- #define: ***ADC_SOURCE_GND_VREF2 ((ADC_MUX_GND << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_GND ((ADC_MUX_GND << ADC_MUXN_BITS) | ADC_MUX_GND)***

- #define: ***ADC_SOURCE_VREF_VREF2 ((ADC_MUX_VREF << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_VREF ((ADC_MUX_VREF << ADC_MUXN_BITS) | ADC_MUX_GND)***

- #define: ***ADC_SOURCE_VREF2_VREF2 ((ADC_MUX_VREF2 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_VREF2 ((ADC_MUX_VREF2 << ADC_MUXN_BITS) | ADC_MUX_GND)***

- #define: ***ADC_SOURCE_VREG2_VREF2 ((ADC_MUX_VREG2 << ADC_MUXN_BITS) | ADC_MUX_VREF2)***

- #define: ***ADC_SOURCE_VDD_GND ((ADC_MUX_VREG2 << ADC_MUXN_BITS) | ADC_MUX_GND)***

ADC_clock

- #define: ***ADC_Clock_1MHz ADC_CR_CLK***

- #define: ***ADC_Clock_6MHz 0x00000000***

ADC_DMAMode

- #define: ***ADC_DMAMode_Linear ((uint32_t)0x00000000)***

- #define: ***ADC_DMAMode_AutoWrap ((uint32_t)0x00000002)***

ADC_DMA_flags_definition

- #define: ***ADC_FLAG_ACT ((uint32_t)0x00000001)***

- #define: ***ADC_FLAG_OVF ((uint32_t)0x00000002)***

ADC_interrupts_definition

- #define: ***ADC_IT_DMABHF ((uint32_t)0x00000002)***

- #define: ***ADC_IT_DMABF ((uint32_t)0x00000004)***

- #define: ***ADC_IT_SAT ((uint32_t)0x00000008)***

- #define: ***ADC_IT_DMAOVF ((uint32_t)0x00000010)***

ADC_resolution

- #define: ***ADC_Resolution_12b ((uint32_t)0x0000E000)***

- #define: ***ADC_Resolution_11b ((uint32_t)0x0000C000)***

- #define: ***ADC_Resolution_10b ((uint32_t)0x0000A000)***

- #define: ***ADC_Resolution_9b ((uint32_t)0x00008000)***

- #define: ***ADC_Resolution_8b ((uint32_t)0x00006000)***

- #define: ***ADC_Resolution_7b ((uint32_t)0x00004000)***

- #define: ***ADC_Resolution_6b ((uint32_t)0x00002000)***

- #define: ***ADC_Resolution_5b ((uint32_t)0x00000000)***

ADC_VoltageN

- #define: ***ADC_VoltageN_Low ((uint32_t)0x00000000)***

- #define: ***ADC_VoltageN_High ((uint32_t)0x00000800)***

ADC_VoltageP

- #define: ***ADC_VoltageP_Low ((uint32_t)0x00000000)***

- #define: ***ADC_VoltageP_High ((uint32_t)0x00001000)***

4 Clock (CLK)

4.1 CLK Firmware driver registers structures

4.1.1 CLK_TypeDef

CLK_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t SLEEPCR***
- ***__IO uint32_t LSI10KCR***
- ***__IO uint32_t LSI1KCR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t HSECR1***
- ***__IO uint32_t HSICR***
- ***__IO uint32_t HSECOMPR***
- ***__IO uint32_t PERIODCR***
- ***__IO uint32_t PERIODSR***
- ***__IO uint32_t DITHERCR***
- ***__IO uint32_t HSECR2***
- ***__IO uint32_t CPUCR***

Field Documentation

- ***__IO uint32_t CLK_TypeDef::SLEEPCR***
  - Sleep timer control register, Address offset: 0x0008
- ***__IO uint32_t CLK_TypeDef::LSI10KCR***
  - LSI Clock (10KHz) control register, Address offset: 0x000C
- ***__IO uint32_t CLK_TypeDef::LSI1KCR***
  - LSI Clock (1KHz) control register, Address offset: 0x0010
- ***uint32_t CLK_TypeDef::RESERVED0[4092]***
  - Reserved
- ***__IO uint32_t CLK_TypeDef::HSECR1***
  - HSE Clock (24MHz) control register 1, Address offset: 0x4004
- ***__IO uint32_t CLK_TypeDef::HSICR***
  - HSI Clock (12MHz) trim register, Address offset: 0x4008
- ***__IO uint32_t CLK_TypeDef::HSECOMPR***
  - HSE comparator Output, Address offset: 0x400C
- ***__IO uint32_t CLK_TypeDef::PERIODCR***
  - Clock period control register, Address offset: 0x4010
- ***__IO uint32_t CLK_TypeDef::PERIODSR***
  - Clock period status register, Address offset: 0x4014
- ***__IO uint32_t CLK_TypeDef::DITHERCR***
  - Clock dither control register, Address offset: 0x4018
- ***__IO uint32_t CLK_TypeDef::HSECR2***
  - HSE Clock (24MHz) control register 2, Address offset: 0x401C
- ***__IO uint32_t CLK_TypeDef::CPUCR***
  - Clock source select register, Address offset: 0x4020

4.2 CLK Firmware driver API description

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

4.2.1 CLK specific features

After reset the device is running from OSCHF (12 MHz)

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

1. Configure the clock source to be used to drive the System clock
2. Configure the System clock frequency: 24MHz/12MHz
3. Configure the Flash clock frequency: 24MHz/12Mhz/6MHz

4.2.2 Internal-external clocks configuration functions

This section describes the functions allowing configuring the internal/external clocks,

1. HSI (high-frequency RC oscillator (OSCHF)), is used as the default system clock source when power is applied to the core domain. The nominal frequency coming out of reset is 12 MHz.
2. HSE (high-frequency crystal oscillator), 24 MHz crystal oscillator

- ***CLK_DeInit()***
- ***CLK_InternalCalibrateLSI()***
- ***CLK_InternalCalibrateHSI()***
- ***CLK_Config()***
- ***CLK_HSECmd()***
- ***CLK_SLPTIMClockConfig()***
- ***CLK_1KClockCalibration()***
- ***CLK_RCTuneConfig()***
- ***CLK_MeasurePeriod()***
- ***CLK_GetMeasurePeriod()***
- ***CLK_GetClocksFreq()***

4.2.3 Internal and external clocks

4.2.3.1 CLK_DeInit

Function Name	**void CLK_DeInit (void)**
Function Description	Resets the CLOCK configuration to the default reset state.
Parameters	• None.
Return values	• None.
Notes	• The default reset state of the clock configuration is given below: HSI ON and used as system clock source

4.2.3.2 CLK_InternalCalibrateLSI

Function Name	**void CLK_InternalCalibrateLSI (void)**
Function Description	Calibrate the low speed internal clock (LSI) to be close to 10KHZ in order to generate 1KHZ clock.
Parameters	• None.
Return values	• None.
Notes	• None.

4.2.3.3 CLK_InternalCalibrateHSI

Function Name	**void CLK_InternalCalibrateHSI (void)**
Function Description	Calibrate the high speed internal clock (HSI) to be close to 12MHZ.
Parameters	• None.
Return values	• None.
Notes	• To calibrate the HSI, the high speed external clock (HSE) must be the system clock.

4.2.3.4 CLK_Config

Function Name	**void CLK_Config (uint8_t MODE)**
Function Description	Configures the clock mode to use:
Parameters	• **MODE :** specifies the frequency mode to use. This parameter can be one of the following values: – ***MODE0 :*** Normal CPU, SCLK =12MHZ, PCLK=6MHZ, Flash Program/Erase Inactive =6Mhz, FlashProgram/Erase Active = 12Mhz. – ***MODE1 :*** Fast CPU, SCLK =12MHZ, PCLK=6MHZ, Flash Program/Erase Inactive =12Mhz, FlashProgram/Erase Active = 12Mhz.

- ***MODE2 :*** Normal CPU, SCLK =24MHZ, PCLK=12MHZ, Flash Program/Erase Inactive =12Mhz, FlashProgram/Erase Active = 12Mhz.
- ***MODE3 :*** Fast CPU, SCLK =24MHZ, PCLK=12MHZ, Flash Program/Erase Inactive =24Mhz, FlashProgram/Erase Active = 12Mhz.

Return values	• None.
Notes	• None.

4.2.3.5 CLK_HSECmd

Function Name	**void CLK_HSECmd (FunctionalState NewState)**
Function Description	Enables or disables the External High Speed oscillator (HSE).
Parameters	• **NewState :** new state of the HSE. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• After enabling HSE the user should wait for HSE_STARTUP_TIMEOUT • to be sure that the clok is stabilized.

4.2.3.6 CLK_SLPTIMClockConfig

Function Name	**void CLK_SLPTIMClockConfig (uint32_t CLK_SLPTIM,FunctionalState NewState)**
Function Description	Enables or disables the specified SLPTIM clock.
Parameters	• **CLK_SLPTIM :** specifies the SLPTIM clock to be enabled or disabled. This parameter can be any combination of the following values: – ***SLPTIM_CLK_32KH :*** 32kHz external XTAL – ***SLPTIM_CLK_10KH :*** 10kHz internal RC (during deep sleep) • **NewState :** new state of the SLPTIM clock. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

4.2.3.7 CLK_1KClockCalibration

Function Name	**void CLK_1KClockCalibration (uint32_t CALINT, uint32_t CALFRAC)**
Function Description	Calibration of CLK1K clock.
Parameters	• **CALINT :** specifies the divider value integer portion. This parameter can be a value between 0x0 and 0x1F. • **CALFRAC :** specifies the divider value fractional portion. This parameter can be a value between 0x0 and 0x7FF.
Return values	• **None.**
Notes	• None.

4.2.3.8 CLK_RCTuneConfig

Function Name	**void CLK_RCTuneConfig (uint32_t TUNE_VALUE)**
Function Description	Set tune value for CLKRC clock.
Parameters	• **TUNE_VALUE :** specifies the tune value for CLKRC clock. This parameter can be a value between 0x0 and 0xF.
Return values	• **None.**
Notes	• None.

4.2.3.9 CLK_MeasurePeriod

Function Name	**void CLK_MeasurePeriod (uint32_t CLK_MEASURED)**
Function Description	Select the clock period to be measured.
Parameters	• **CLK_MEASURED :** specifies the clock for which the period will be measured. This parameter can be : – ***MEASURE_CLKRC :*** Measure CLKRC. – ***MEASURE_OSCHF :*** Measure OSCHF. – ***MEASURE_TUNEFILT :*** Measure

TUNE_FILTER_RESULT.

Return values	• **None.**
Notes	• None.

4.2.3.10 CLK_GetMeasurePeriod

Function Name	**uint32_t CLK_GetMeasurePeriod (void)**
Function Description	Returns the clock period measured depend on clock selected.
Parameters	• **None. :**
Return values	• **None.**
Notes	• measured period is equal to: 16 x Clock period in clk12m cycles (CLKRC/TUNE_FILTER_RESULT modes) 256 x clock period in clk12m cycles (OSCHF mode)

4.2.3.11 CLK_GetClocksFreq

Function Name	**uint32_t CLK_GetClocksFreq (void)**
Function Description	Returns the clock source used as system clock.
Parameters	• None.
Return values	• **The clock source used as system clock. The returned value can be one of the following:** – ***0x00,0x01,0x10: HSI used as system clock*** – ***0x03: HSE used as system clock***
Notes	• None.

4.3 CLK Firmware driver defines

4.3.1 CLK

CLK_HSE_configuration

- #define: ***CLK_MODE0 ((uint32_t)0x00000000)***

- #define: ***CLK_MODE1 ((uint32_t)0x00000001)***

- #define: ***CLK_MODE2 ((uint32_t)0x00000010)***

- #define: ***CLK_MODE3 ((uint32_t)0x00000011)***

CLK_PC_Trace_Select

- #define: ***GPIO_BBDEBUG ((uint32_t)0x00000000)***

- #define: ***GPIO_PCTRACE ((uint32_t)0x00000001)***

5 External interrupt (EXTI)

5.1 EXTI Firmware driver registers structures

5.1.1 EXTI_InitTypeDef

EXTI_InitTypeDef is defined in the stm32w108xx_exti.h

Data Fields

- ***uint32_t EXTI_Source***
- ***uint8_t EXTI_IRQn***
- ***EXTITrigger_TypeDef EXTI_Trigger***
- ***FunctionalState EXTI_LineCmd***
- ***FunctionalState EXTI_DigitalFilterCmd***

Field Documentation

- ***uint32_t EXTI_InitTypeDef::EXTI_Source***
  - Specifies the EXTI source to be configured. This parameter can be GPIO_SourcePxy where x can be (A, B or C) and y can be (0..7).
- ***uint8_t EXTI_InitTypeDef::EXTI_IRQn***
  - Specifies the GPIO IRQ handler for the EXTI source. This parameter can be EXTI_IRQn where n can be (A, B, C or D).
- ***EXTITrigger_TypeDef EXTI_InitTypeDef::EXTI_Trigger***
  - Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of **EXTI_InitTypeDef**
- ***FunctionalState EXTI_InitTypeDef::EXTI_LineCmd***
  - Specifies the new state of the selected EXTI line. This parameter can be set either to ENABLE or DISABLE
- ***FunctionalState EXTI_InitTypeDef::EXTI_DigitalFilterCmd***
  - Specifies the new state of the digital filter. This parameter can be set either to ENABLE or DISABLE

5.1.2 EXTI_TypeDef

EXTI_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t PR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t TSR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t CR***

Field Documentation

- ***__IO uint32_t EXTI_TypeDef::PR***
  - EXTI pending register, Address offset 0xA814
- ***uint32_t EXTI_TypeDef::RESERVED0[18]***
  - Reserved
- ***__IO uint32_t EXTI_TypeDef::TSR[4]***
  - EXTI trigger source register, Address offset 0xA860
- ***uint32_t EXTI_TypeDef::RESERVED1[1257]***
  - Reserved
- ***__IO uint32_t EXTI_TypeDef::CR[2]***
  - EXTI configuration register, Address offset 0xBC14

5.2 EXTI Firmware driver API description

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

5.2.1 EXTI features

External interrupt/event lines are mapped as following:

1. All available GPIO pins are connected to the 4 external interrupt/event lines from EXTIA to EXTID.
2. EXTIA and EXTIB have fixed pins assignment (PB0 and PB6).
3. EXTIC and EXTID can use any GPIO pin.

5.2.2 How to use this driver

In order to use an I/O pin as an external interrupt source, follow the steps below:

1. Configure the I/O in input mode using GPIO_Init()
2. Select the mode(interrupt, event) and configure the trigger selection using EXTI_Init().
3. Configure NVIC IRQ channel mapped to the EXTI line using NVIC_Init().

5.2.3 EXTI initialization and configuration functions

- ***EXTI_DeInit()***
- ***EXTI_Init()***
- ***EXTI_StructInit()***

5.2.4 EXTI interrupts and flags management functions

- ***EXTI_GetITStatus()***
- ***EXTI_ClearITPendingBit()***

5.2.5 EXTI Initialization and Configuration

5.2.5.1 EXTI_DeInit

Function Name	**void EXTI_DeInit (void)**
Function Description	Deinitializes the EXTI_IRQn line registers to their default reset values.
Parameters	• None.
Return values	• None.
Notes	• None.

5.2.5.2 EXTI_Init

Function Name	**void EXTI_Init (*EXTI_InitTypeDef** * **EXTI_InitStruct)**
Function Description	Initializes the EXTI peripheral according to the specified parameters in the EXTI_InitStruct.
Parameters	• **EXTI_InitStruct :** pointer to a EXTI_InitTypeDef structure that contains the configuration information for the EXTI peripheral.
Return values	• None.
Notes	• None.

5.2.5.3 EXTI_StructInit

Function Name	**void EXTI_StructInit (*EXTI_InitTypeDef** * **EXTI_InitStruct)**
Function Description	Fills each EXTI_InitStruct member with its reset value.
Parameters	• **EXTI_InitStruct :** pointer to a EXTI_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

5.2.6 Interrupts and flags management functions

5.2.6.1 EXTI_GetITStatus

Function Name	**ITStatus EXTI_GetITStatus (uint32_t EXTI_IRQn)**
Function Description	Checks whether the specified EXTI line is asserted or not.
Parameters	• **EXTI_IRQn :** specifies the EXTI line to check. This parameter can be: EXTI_IRQn: External interrupt line n where x(A, B, C or D).
Return values	• **The new state of EXTI_IRQn (SET or RESET).**
Notes	• None.

5.2.6.2 EXTI_ClearITPendingBit

Function Name	**void EXTI_ClearITPendingBit (uint32_t EXTI_IRQn)**
Function Description	Clears the EXTI's line pending bits.
Parameters	• **EXTI_IRQn :** specifies the EXTI lines to clear. This parameter can be any combination of EXTI_IRQn where n can be (A, B, C or D).
Return values	• None.
Notes	• None.

5.3 EXTI Firmware driver defines

5.3.1 EXTI

EXTI

EXTI_IRQ_Sources

- #define: ***EXTI_IRQA ((uint32_t)0x00000000)***

- #define: ***EXTI_IRQB ((uint32_t)0x00000010)***

- #define: ***EXTI_IRQC ((uint32_t)0x00000020)***

- #define: ***EXTI_IRQD ((uint32_t)0x00000031)***

EXTI_Pin_sources

- #define: ***EXTI_SourcePA0 ((uint8_t)0x00)***

- #define: ***EXTI_SourcePA1 ((uint8_t)0x01)***

- #define: ***EXTI_SourcePA2 ((uint8_t)0x02)***

- #define: ***EXTI_SourcePA3 ((uint8_t)0x03)***

- #define: ***EXTI_SourcePA4 ((uint8_t)0x04)***

- #define: ***EXTI_SourcePA5 ((uint8_t)0x05)***

- #define: ***EXTI_SourcePA6 ((uint8_t)0x06)***

- #define: ***EXTI_SourcePA7 ((uint8_t)0x07)***

- #define: ***EXTI_SourcePB0 ((uint8_t)0x08)***
- #define: ***EXTI_SourcePB1 ((uint8_t)0x09)***
- #define: ***EXTI_SourcePB2 ((uint8_t)0x0A)***
- #define: ***EXTI_SourcePB3 ((uint8_t)0x0B)***
- #define: ***EXTI_SourcePB4 ((uint8_t)0x0C)***
- #define: ***EXTI_SourcePB5 ((uint8_t)0x0D)***
- #define: ***EXTI_SourcePB6 ((uint8_t)0x0E)***
- #define: ***EXTI_SourcePB7 ((uint8_t)0x0F)***
- #define: ***EXTI_SourcePC0 ((uint8_t)0x10)***
- #define: ***EXTI_SourcePC1 ((uint8_t)0x11)***
- #define: ***EXTI_SourcePC2 ((uint8_t)0x12)***
- #define: ***EXTI_SourcePC3 ((uint8_t)0x13)***

- #define: ***EXTI_SourcePC4 ((uint8_t)0x14)***

- #define: ***EXTI_SourcePC5 ((uint8_t)0x15)***

- #define: ***EXTI_SourcePC6 ((uint8_t)0x16)***

- #define: ***EXTI_SourcePC7 ((uint8_t)0x17)***

6 FLASH Memory (FLASH)

6.1 FLASH Firmware driver registers structures

6.1.1 FLASH_TypeDef

FLASH_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t CLKER***
- ***__IO uint32_t CLKSR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t ACR***
- ***__IO uint32_t KEYR***
- ***__IO uint32_t OPTKEYR***
- ***__IO uint32_t SR***
- ***__IO uint32_t CR***
- ***__IO uint32_t AR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t OBR***
- ***__IO uint32_t WRPR***

Field Documentation

- ***__IO uint32_t FLASH_TypeDef::CLKER***
  - FLASH controller clock enable register Address offset: 0x402C
- ***__IO uint32_t FLASH_TypeDef::CLKSR***
  - FLASH controller clock status register Address offset: 0x4030
- ***uint32_t FLASH_TypeDef::RESERVED0[4083]***
  - Reserved
- ***__IO uint32_t FLASH_TypeDef::ACR***
  - FLASH access control register, Address offset: 0x8000
- ***__IO uint32_t FLASH_TypeDef::KEYR***
  - FLASH key register, Address offset: 0x8004
- ***__IO uint32_t FLASH_TypeDef::OPTKEYR***
  - FLASH OPT key register, Address offset: 0x8008
- ***__IO uint32_t FLASH_TypeDef::SR***
  - FLASH status register, Address offset: 0x800C
- ***__IO uint32_t FLASH_TypeDef::CR***
  - FLASH control register, Address offset: 0x8010
- ***__IO uint32_t FLASH_TypeDef::AR***
  - FLASH address register, Address offset: 0x8014
- ***uint32_t FLASH_TypeDef::RESERVED1***
  - Reserved
- ***__IO uint32_t FLASH_TypeDef::OBR***
  - FLASH option bytes register, Address offset: 0x801C
- ***__IO uint32_t FLASH_TypeDef::WRPR***
  - FLASH option bytes register, Address offset: 0x8020

6.1.2 OB_TypeDef

OB_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint16_t RDP***
- ***__IO uint16_t Rsvd0***
- ***__IO uint16_t Rsvd1***
- ***__IO uint16_t Rsvd2***
- ***__IO uint16_t WRP0***
- ***__IO uint16_t WRP1***
- ***__IO uint16_t WRP2***
- ***__IO uint16_t WRP3***

Field Documentation

- ***__IO uint16_t OB_TypeDef::RDP***
  - FLASH option byte Read protection, Address 0x00
- ***__IO uint16_t OB_TypeDef::Rsvd0***
  - Reserved
- ***__IO uint16_t OB_TypeDef::Rsvd1***
  - Reserved
- ***__IO uint16_t OB_TypeDef::Rsvd2***
  - Reserved
- ***__IO uint16_t OB_TypeDef::WRP0***
  - FLASH option byte write protection 0, Address offset: 0x08
- ***__IO uint16_t OB_TypeDef::WRP1***
  - FLASH option byte write protection 1, Address offset: 0x0A
- ***__IO uint16_t OB_TypeDef::WRP2***
  - FLASH option byte write protection 2, Address offset: 0x0C
- ***__IO uint16_t OB_TypeDef::WRP3***
  - FLASH option byte write protection 0, Address offset: 0x0E

6.2 FLASH Firmware driver API description

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

6.2.1 How to use this driver

This driver describes the functions allowing configuring and programming the Flash memory of all STM32W108xx devices. These functions are split in 4 groups

1. FLASH Interface configuration functions: this group includes the management of following features:
   - Set the latency
   - Enable/Disable the prefetch buffer

2. FLASH Memory Programming functions: this group includes all needed functions to erase and program the main memory:
   - Lock and Unlock the Flash interface.
   - Erase function: Erase Page, erase all pages.
   - Program functions: Half Word and Word write.
3. FLASH Option Bytes Programming functions: this group includes all needed functions to:
   - Set/Reset the write protection
   - Set the Read protection Level
   - Get the Write protection
   - Get the read protection status
4. FLASH Interrupts and flag management functions: this group includes all needed functions to:
   - Enable/Disable the flash interrupt sources
   - Get flags status
   - Clear flags
   - Get Flash operation status
   - Wait for last flash operation

6.2.2 FLASH Interface configuration functions

FLASH_Interface configuration_Functions, includes the following functions:

- void FLASH_SetLatency(uint32_t FLASH_Latency):

To correctly read data from Flash memory, the number of wait states (LATENCY) must be correctly programmed according to the frequency of the CPU clock (SCLK)

- FlagStatus FLASH_GetPrefetchBufferStatus(void);
- void FLASH_PrefetchBufferCmd(FunctionalState NewState);

All these functions don't need the unlock sequence.

- ***FLASH_SetLatency()***
- ***FLASH_GetPrefetchBufferStatus()***
- ***FLASH_PrefetchBufferCmd()***

6.2.3 FLASH Memory Programming functions

The FLASH Memory Programming functions, includes the following functions:

- void FPEC_ClockCmd(FunctionalState NewState)
- void FLASH_Unlock(void);
- void FLASH_Lock(void);
- void FLASH_HalfCycleAccessCmd(uint32_t FLASH_HalfCycleAccess)
- FLASH_Status FLASH_ErasePage(uint32_t Page_Address);
- FLASH_Status FLASH_EraseAllPages(void);
- FLASH_Status FLASH_ProgramWord(uint32_t Address, uint32_t Data);
- FLASH_Status FLASH_ProgramHalfWord(uint32_t Address, uint16_t Data);

Any erase or program operation should follow these steps:

1. Call the FPEC_ClockCmd() function to enable/disable the flash control register and enable/disable flash clock.
2. Call the desired function to erase page or program data.

- ***FPEC_ClockCmd()***

- ***FLASH_Unlock()***
- ***FLASH_Lock()***
- ***FLASH_HalfCycleAccessCmd()***
- ***FLASH_ErasePage()***
- ***FLASH_EraseAllPages()***
- ***FLASH_ProgramWord()***
- ***FLASH_ProgramHalfWord()***

6.2.4 Option Bytes Programming functions

The FLASH_Option Bytes Programming_functions, includes the following functions:

- FLASH_Status FLASH_EraseOptionBytes(void);
- FLASH_Status FLASH_EnableWriteProtection(uint32_t FLASH_Pages);
- FLASH_Status FLASH_ReadOutProtection(FunctionalState NewState);
- uint32_t FLASH_GetWriteProtectionOptionByte(void);
- FlagStatus FLASH_GetReadOutProtectionStatus(void);
- ***FLASH_EraseOptionBytes()***
- ***FLASH_EnableWriteProtection()***
- ***FLASH_ReadOutProtection()***
- ***FLASH_GetWriteProtectionOptionByte()***
- ***FLASH_GetReadOutProtectionStatus()***

6.2.5 Interrupts and flags management functions

- ***FLASH_ITConfig()***
- ***FLASH_GetFlagStatus()***
- ***FLASH_ClearFlag()***
- ***FLASH_GetStatus()***
- ***FLASH_WaitForLastOperation()***

6.2.6 FLASH Interface configuration functions

6.2.6.1 FLASH_SetLatency

Function Name	**void FLASH_SetLatency (uint32_t FLASH_Latency)**
Function Description	Sets the code latency value.
Parameters	• **FLASH_Latency :** specifies the FLASH Latency value. This parameter can be one of the following values: – ***FLASH_Latency_0 :*** FLASH Zero Latency cycle – ***FLASH_Latency_1 :*** FLASH One Latency cycle – ***FLASH_Latency_2 :*** FLASH Two Latency cycles
Return values	• None.
Notes	• None.

6.2.6.2 FLASH_GetPrefetchBufferStatus

Function Name	**FlagStatus FLASH_GetPrefetchBufferStatus (void)**
Function Description	Checks whether the FLASH Prefetch Buffer status is set or not.
Parameters	• None.
Return values	• **FLASH Prefetch Buffer Status (SET or RESET).**
Notes	• None.

6.2.6.3 FLASH_PrefetchBufferCmd

Function Name	**void FLASH_PrefetchBufferCmd (uint32_t FLASH_PrefetchBuffer)**
Function Description	Enables or disables the Prefetch Buffer.
Parameters	• **FLASH_PrefetchBuffer :** specifies the Prefetch buffer status. This parameter can be one of the following values: – ***FLASH_PrefetchBuffer_Enable :*** FLASH Prefetch Buffer Enable – ***FLASH_PrefetchBuffer_Disable :*** FLASH Prefetch Buffer Disable
Return values	• None.
Notes	• None.

6.2.7 FLASH Memory Programming functions

6.2.7.1 FPEC_ClockCmd

Function Name	**void FPEC_ClockCmd (FunctionalState NewState)**
Function Description	Enables or disables the FPEC clock.

Parameters	• **NewState :** new state of the FPEC clock. This parameter can be: ENABLE or DISABLE.
Return values	• **None.**
Notes	• None.

6.2.7.2 FLASH_Unlock

Function Name	**void FLASH_Unlock (void)**
Function Description	Unlocks the FLASH Program Erase Controller.
Parameters	• None.
Return values	• None.
Notes	• None.

6.2.7.3 FLASH_Lock

Function Name	**void FLASH_Lock (void)**
Function Description	Locks the FLASH Program Erase Controller.
Parameters	• None.
Return values	• None.
Notes	• None.

6.2.7.4 FLASH_HalfCycleAccessCmd

Function Name	**void FLASH_HalfCycleAccessCmd (uint32_t FLASH_HalfCycleAccess)**
Function Description	Enables or disables the Half cycle flash access.
Parameters	• **FLASH_HalfCycleAccess :** specifies the FLASH Half cycle

	Access mode. This parameter can be one of the following values: – ***FLASH_HalfCycleAccess_Enable :*** FLASH Half Cycle Enable – ***FLASH_HalfCycleAccess_Disable :*** FLASH Half Cycle Disable
Return values	• None.
Notes	• None.

6.2.7.5 FLASH_ErasePage

Function Name	**FLASH_Status FLASH_ErasePage (uint32_t Page_Address)**
Function Description	Erases a specified FLASH page.
Parameters	• **Page_Address :** The page address to be erased.
Return values	• **FLASH Status: The returned value can be: FLASH_BUSY, FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

6.2.7.6 FLASH_EraseAllPages

Function Name	**FLASH_Status FLASH_EraseAllPages (void)**
Function Description	Erases all FLASH pages.
Parameters	• None.
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

6.2.7.7 FLASH_ProgramWord

Function Name	**FLASH_Status FLASH_ProgramWord (uint32_t Address, uint32_t Data)**
Function Description	Programs a word at a specified address.
Parameters	• **Address :** specifies the address to be programmed. • **Data :** specifies the data to be programmed.
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

6.2.7.8 FLASH_ProgramHalfWord

Function Name	**FLASH_Status FLASH_ProgramHalfWord (uint32_t Address, uint16_t Data)**
Function Description	Programs a half word at a specified address.
Parameters	• **Address :** specifies the address to be programmed. • **Data :** specifies the data to be programmed.
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

6.2.8 Option Bytes Programming functions

6.2.8.1 FLASH_EraseOptionBytes

Function Name	**FLASH_Status FLASH_EraseOptionBytes (void)**
Function Description	Erases the FLASH option bytes.
Parameters	• None.

Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• This functions erases all option bytes except the Read protection (RDP).

6.2.8.2 FLASH_EnableWriteProtection

Function Name	**FLASH_Status FLASH_EnableWriteProtection (uint32_t FLASH_Pages)**
Function Description	Write protects the desired pages.
Parameters	• **FLASH_Pages :** specifies the address of the pages to be write protected. This parameter can be: – ***For STM32W_Low-density_devices :*** value between FLASH_WRProt_Pages0to3 and FLASH_WRProt_Pages60to63. – ***For STM32W_Medium-density_devices :*** value between FLASH_WRProt_Pages0to3 and FLASH_WRProt_Pages124to127. – ***For STM32W_High-density_devices :*** value between FLASH_WRProt_Pages0to1 and FLASH_WRProt_Pages126to127. – ***For STM32W_Connectivity_line_devices :*** value between FLASH_WRProt_Pages0to1 and FLASH_WRProt_Pages94to95. – ***FLASH_WRProt_AllPages :***
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

6.2.8.3 FLASH_ReadOutProtection

Function Name	**FLASH_Status FLASH_ReadOutProtection (FunctionalState NewState)**
Function Description	Enables or disables the read out protection.
Parameters	• **NewState :** : new state of the ReadOut Protection. This

	parameter can be: ENABLE or DISABLE.
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

6.2.8.4 FLASH_GetWriteProtectionOptionByte

Function Name	**uint32_t FLASH_GetWriteProtectionOptionByte (void)**
Function Description	Returns the FLASH Write Protection Option Bytes Register value.
Parameters	• None.
Return values	• **The FLASH Write Protection Option Bytes Register value**
Notes	• None.

6.2.8.5 FLASH_GetReadOutProtectionStatus

Function Name	**FlagStatus FLASH_GetReadOutProtectionStatus (void)**
Function Description	Checks whether the FLASH Read Out Protection Status is set or not.
Parameters	• None.
Return values	• **FLASH ReadOut Protection Status(SET or RESET)**
Notes	• None.

6.2.9 Interrupts and flags management functions

6.2.9.1 FLASH_ITConfig

Function Name	**void FLASH_ITConfig (uint32_t FLASH_IT,FunctionalState NewState)**
Function Description	Enables or disables the specified FLASH interrupts.
Parameters	• **FLASH_IT :** specifies the FLASH interrupt sources to be enabled or disabled. This parameter can be any combination of the following values: – ***FLASH_IT_ERROR :*** FLASH Error Interrupt – ***FLASH_IT_EOP :*** FLASH end of operation Interrupt • **NewState :** new state of the specified Flash interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

6.2.9.2 FLASH_GetFlagStatus

Function Name	**FlagStatus FLASH_GetFlagStatus (uint32_t FLASH_FLAG)**
Function Description	Checks whether the specified FLASH flag is set or not.
Parameters	• **FLASH_FLAG :** specifies the FLASH flag to check. This parameter can be one of the following values: – ***FLASH_FLAG_BSY :*** FLASH Busy flag – ***FLASH_FLAG_PGERR :*** FLASH Program error flag – ***FLASH_FLAG_WRPRTERR :*** FLASH Write protected error flag – ***FLASH_FLAG_EOP :*** FLASH End of Operation flag – ***FLASH_FLAG_OPTERR :*** FLASH Option Byte error flag
Return values	• **The new state of FLASH_FLAG (SET or RESET).**
Notes	• None.

6.2.9.3 FLASH_ClearFlag

Function Name	**void FLASH_ClearFlag (uint32_t FLASH_FLAG)**
Function Description	Clears the FLASH's pending flags.
Parameters	• **FLASH_FLAG :** specifies the FLASH flags to clear. This parameter can be any combination of the following values:

- ***FLASH_FLAG_PGERR :*** FLASH Program error flag
- ***FLASH_FLAG_WRPRTERR :*** FLASH Write protected error flag
- ***FLASH_FLAG_EOP :*** FLASH End of Operation flag

Return values	• None.
Notes	• None.

6.2.9.4 FLASH_GetStatus

Function Name	**FLASH_Status FLASH_GetStatus (void)**
Function Description	Returns the FLASH Status.
Parameters	• None.
Return values	• **FLASH Status: The returned value can be: FLASH_BUSY, FLASH_ERROR_PG, FLASH_ERROR_WRP or FLASH_COMPLETE**
Notes	• None.

6.2.9.5 FLASH_WaitForLastOperation

Function Name	**FLASH_Status FLASH_WaitForLastOperation (uint32_t Timeout)**
Function Description	Waits for a Flash operation to complete or a TIMEOUT to occur.
Parameters	• **Timeout :** FLASH programming Timeout
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

6.3 FLASH Firmware driver defines

6.3.1 FLASH

FLASH

FLASH_Flags

- #define: ***FLASH_FLAG_BSY ((uint32_t)0x00000001)***

FLASH Busy flag

- #define: ***FLASH_FLAG_EOP ((uint32_t)0x00000020)***

FLASH End of Operation flag

- #define: ***FLASH_FLAG_PGERR ((uint32_t)0x00000004)***

FLASH Program error flag

- #define: ***FLASH_FLAG_WRPRTERR ((uint32_t)0x00000010)***

FLASH Write protected error flag

- #define: ***FLASH_FLAG_OPTERR ((uint32_t)0x00000001)***

FLASH Option Byte error flag

FLASH_Interrupts

- #define: ***FLASH_IT_ERROR ((uint32_t)0x00000400)***

FPEC error interrupt source

- #define: ***FLASH_IT_EOP ((uint32_t)0x00001000)***

End of FLASH Operation Interrupt source

Flash_Latency

- #define: ***FLASH_Latency_0 ((uint32_t)0x00000000)***

FLASH Zero Latency cycle

- #define: ***FLASH_Latency_1 ((uint32_t)0x00000001)***

FLASH One Latency cycle

- #define: ***FLASH_Latency_2 ((uint32_t)0x00000002)***

FLASH Two Latency cycles

7 General-purpose I/Os (GPIO)

7.1 GPIO Firmware driver registers structures

7.1.1 GPIO_TypeDef

GPIO_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t CRL***
- ***__IO uint32_t CRH***
- ***__IO uint32_t IDR***
- ***__IO uint32_t ODR***
- ***__IO uint32_t BSR***
- ***__IO uint32_t BRR***

Field Documentation

- ***__IO uint32_t GPIO_TypeDef::CRL***
  - GPIO port configuration low register, Address offset: 0x00
- ***__IO uint32_t GPIO_TypeDef::CRH***
  - GPIO port configuration high register, Address offset: 0x04
- ***__IO uint32_t GPIO_TypeDef::IDR***
  - GPIO port input data register, Address offset: 0x08
- ***__IO uint32_t GPIO_TypeDef::ODR***
  - GPIO port output data register, Address offset: 0x0C
- ***__IO uint32_t GPIO_TypeDef::BSR***
  - GPIO port bit set registerBSR, Address offset: 0x10
- ***__IO uint32_t GPIO_TypeDef::BRR***
  - GPIO port bit reset registerBRR, Address offset: 0x14

7.1.2 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32w108xx_gpio.h

Data Fields

- ***uint32_t GPIO_Pin***
- ***GPIOMode_TypeDef GPIO_Mode***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::GPIO_Pin***
  - Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_pins_define***

- ***GPIOMode_TypeDef GPIO_InitTypeDef::GPIO_Mode***
  - Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIOMode_TypeDef***

7.1.3 GPIO_DBG_TypeDef

GPIO_DBG_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t PCTRACECR***
- ***uint32_t RESERVED***
- ***__IO uint32_t DBGCR***
- ***__IO uint32_t DBGSR***

Field Documentation

- ***__IO uint32_t GPIO_DBG_TypeDef::PCTRACECR***
  - Clock PC trace register, Address offset 0x4028
- ***uint32_t GPIO_DBG_TypeDef::RESERVED[7925]***
- ***__IO uint32_t GPIO_DBG_TypeDef::DBGCR***
  - GPIO debug configuration register, Address offset 0xBC00
- ***__IO uint32_t GPIO_DBG_TypeDef::DBGSR***
  - GPIO debug status register, Address offset 0xBC04

7.2 GPIO Firmware driver API description

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

7.2.1 How to use this driver

1. Configure the GPIO pin(s) using GPIO_Init(). Four possible configuration are available for each pin:
   - Input: Floating, Pull-up, Pull-down.
   - Output: Push-Pull (Pull-up, Pull-down or no Pull) Open Drain (Pull-up, Pull-down or no Pull).
   - Alternate Function: Push-Pull (PP or SPI mode) Open Drain (Pull-up, Pull-down or no Pull).
   - Analog
2. To get the level of a pin configured in input mode use GPIO_ReadInputDataBit()
3. To set/reset the level of a pin configured in output mode use GPIO_SetBits()/GPIO_ResetBits()
4. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
5. A full chip reset affects the GPIO configuration as follows:
   - All pins are configured as floating inputs.
   - The GPIO_EXTREGEN bit is set which overrides the normal configuration for PA7.

– The GPIO_DBGDIS bit is cleared, allowing Serial Wire/JTAG access to override the normal configuration of PC0, PC2, PC3, and PC4.

7.2.2 Initialization and Configuration

- ***GPIO_DeInit()***
- ***GPIO_Init()***
- ***GPIO_StructInit()***

7.2.3 GPIO Read and Write

- ***GPIO_ReadInputDataBit()***
- ***GPIO_ReadInputData()***
- ***GPIO_ReadOutputDataBit()***
- ***GPIO_ReadOutputData()***
- ***GPIO_SetBits()***
- ***GPIO_ResetBits()***
- ***GPIO_WriteBit()***
- ***GPIO_Write()***

7.2.4 GPIO wake and debug configuration functions

- ***GPIO_PCTraceConfig()***
- ***GPIO_DebugInterfaceCmd()***
- ***GPIO_ExternalOverrideCmd()***
- ***GPIO_GetDebugFlagStatus()***

7.2.5 Initialization and Configuration

7.2.5.1 GPIO_DeInit

Function Name	**void GPIO_DeInit (*GPIO_TypeDef** * GPIOx)**
Function Description	Deinitializes the GPIOx peripheral registers to their default reset values.
Parameters	• **GPIOx :** where x can be (A..C) to select the GPIO peripheral.
Return values	• None.
Notes	• None.

7.2.5.2 GPIO_Init

Function Name	**void GPIO_Init (*GPIO_TypeDef** * GPIOx, *GPIO_InitTypeDef** * GPIO_InitStruct)**
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **GPIO_InitStruct :** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral. GPIO_Pin: selects the pin to be configured: GPIO_Pin_0 -> GPIO_Pin_7 GPIO_Mode: selects the mode of the pin: GPIO Analog Mode: GPIO_Mode_ANGPIO Output Mode PP: GPIO_Mode_OUT_PPGPIO Input Mode NOPULL: GPIO_Mode_INGPIO Output Mode OD: GPIO_Mode_OUT_ODGPIO Input Mode PuPd: GPIO_Mode_IN_PUDGPIO Alternate function Mode PP: GPIO_Mode_AF_PPGPIO Alternate function Mode SPI SCLK PP: GPIO_Mode_AF_PP_SPIGPIO Alternate function Mode OD: GPIO_Mode_AF_OD – ***GPIO Analog Mode :*** GPIO_Mode_AN – ***GPIO Output Mode PP :*** GPIO_Mode_OUT_PP – ***GPIO Input Mode NOPULL :*** GPIO_Mode_IN – ***GPIO Output Mode OD :*** GPIO_Mode_OUT_OD – ***GPIO Input Mode PuPd :*** GPIO_Mode_IN_PUD – ***GPIO Alternate function Mode PP :*** GPIO_Mode_AF_PP – ***GPIO Alternate function Mode SPI SCLK PP :*** GPIO_Mode_AF_PP_SPI – ***GPIO Alternate function Mode OD :*** GPIO_Mode_AF_OD
Return values	• None.
Notes	• None.

7.2.5.3 GPIO_StructInit

Function Name	**void GPIO_StructInit (*GPIO_InitTypeDef** * GPIO_InitStruct)**
Function Description	Fills each GPIO_InitStruct member with its default value.
Parameters	• **GPIO_InitStruct :** : pointer to a GPIO_InitTypeDef structure which will be initialized.
Return values	• None.

Notes	• None.

7.2.6 GPIO Read and Write functions

7.2.6.1 GPIO_ReadInputDataBit

Function Name	**uint32_t GPIO_ReadInputDataBit (*GPIO_TypeDef** * **GPIOx, uint32_t GPIO_Pin)**
Function Description	Reads the specified input port pin.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bit to read. This parameter can be GPIO_Pin_x where x can be (0..7).
Return values	• **The input port pin value.**
Notes	• None.

7.2.6.2 GPIO_ReadInputData

Function Name	**uint32_t GPIO_ReadInputData (*GPIO_TypeDef** * **GPIOx)**
Function Description	Reads the specified GPIO input data port.
Parameters	• **GPIOx :** where x can be (A..C) to select the GPIO peripheral.
Return values	• **GPIO input data port value.**
Notes	• None.

7.2.6.3 GPIO_ReadOutputDataBit

Function Name	**uint32_t GPIO_ReadOutputDataBit (*GPIO_TypeDef** * **GPIOx, uint32_t GPIO_Pin)**

Function Description	Reads the specified output data port bit.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **GPIO_Pin :** Specifies the port bit to read. This parameter can be GPIO_Pin_x where x can be (0..7).
Return values	• **The output port pin value.**
Notes	• None.

7.2.6.4 GPIO_ReadOutputData

Function Name	**uint32_t GPIO_ReadOutputData (*GPIO_TypeDef** * **GPIOx)**
Function Description	Reads the specified GPIO output data port.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral.
Return values	• **GPIO output data port value.**
Notes	• None.

7.2.6.5 GPIO_SetBits

Function Name	**void GPIO_SetBits (*GPIO_TypeDef** * **GPIOx, uint32_t GPIO_Pin)**
Function Description	Sets the selected data port bits.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..7).
Return values	• None.
Notes	• This functions uses GPIOx_SET 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.

7.2.6.6 GPIO_ResetBits

Function Name	**void GPIO_ResetBits (*GPIO_TypeDef** * **GPIOx, uint32_t GPIO_Pin)**
Function Description	Clears the selected data port bits.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..7).
Return values	• None.
Notes	• This functions uses GPIOx_CLR 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.

7.2.6.7 GPIO_WriteBit

Function Name	**void GPIO_WriteBit (*GPIO_TypeDef** * **GPIOx, uint32_t GPIO_Pin, BitAction BitVal)**
Function Description	Sets or clears the selected data port bit.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bit to be written. This parameter can be one of GPIO_Pin_x where x can be (0..7). • **BitVal :** specifies the value to be written to the selected bit. This parameter can be one of the BitAction enum values: – ***Bit_RESET :*** to clear the port pin – ***Bit_SET :*** to set the port pin
Return values	• None.
Notes	• None.

7.2.6.8 GPIO_Write

Function Name	**void GPIO_Write (*GPIO_TypeDef** * GPIOx, uint16_t PortVal)**
Function Description	Writes data to the specified GPIO data port.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **PortVal :** specifies the value to be written to the port output data register.
Return values	• None.
Notes	• None.

7.2.7 GPIO Wake and Debug Configuration functions

7.2.7.1 GPIO_PCTraceConfig

Function Name	**void GPIO_PCTraceConfig (uint32_t PCTRACE_SEL)**
Function Description	Selects PC_TRACE source on bb_debug GPIO pins.
Parameters	• **PCTRACE_SEL :** specifies the PC_TRACE source on bb_debug GPIO pins. This parameter can be : – ***GPIO_BBDEBUG :*** bb debug. – ***GPIO_PCTRACE :*** pc trace.
Return values	• **None.**
Notes	• None.

7.2.7.2 GPIO_DebugInterfaceCmd

Function Name	**void GPIO_DebugInterfaceCmd (FunctionalState NewState)**
Function Description	Enables or disables the debug interface.
Parameters	• **NewState :** new state of the debug interface. This parameter can be: ENABLE or DISABLE.
Return values	• None.

Notes	• None.

7.2.7.3 GPIO_ExternalOverrideCmd

Function Name	**void GPIO_ExternalOverrideCmd (FunctionalState NewState)**
Function Description	Enables or Disable REG_EN override of PA7's normal GPIO configuration.
Parameters	• **NewState :** new state of the REG_EN. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

7.2.7.4 GPIO_GetDebugFlagStatus

Function Name	**FlagStatus GPIO_GetDebugFlagStatus (uint16_t GPIO_DBGFLAG)**
Function Description	Checks whether the specified GPIO debug flag is set or not.
Parameters	• **GPIO_DBGFLAG :** specifies the flag to check. This parameter can be one of the following values: – ***GPIO_DBGSR_SWEN :*** Serial Wire interface flag – ***GPIO_DBGSR_FORCEDBG :*** Debugger interface flag – ***GPIO_DBGSR_BOOTMODE :*** nBOOTMODE signal sampled at the end of reset flag
Return values	• **The new state of GPIO_DBGFLAG (SET or RESET).**
Notes	• None.

7.3 GPIO Firmware driver defines

7.3.1 GPIO

GPIO

GPIO_pins_define

- #define: ***GPIO_Pin_0 ((uint32_t)0x00000001)***

Pin 0 selected

- #define: ***GPIO_Pin_1 ((uint32_t)0x00000002)***

Pin 1 selected

- #define: ***GPIO_Pin_2 ((uint32_t)0x00000004)***

Pin 2 selected

- #define: ***GPIO_Pin_3 ((uint32_t)0x00000008)***

Pin 3 selected

- #define: ***GPIO_Pin_4 ((uint32_t)0x00000010)***

Pin 4 selected

- #define: ***GPIO_Pin_5 ((uint32_t)0x00000020)***

Pin 5 selected

- #define: ***GPIO_Pin_6 ((uint32_t)0x00000040)***

Pin 6 selected

- #define: ***GPIO_Pin_7 ((uint32_t)0x00000080)***

Pin 7 selected

- #define: ***GPIO_Pin_All ((uint32_t)0x000000FF)***

All pins selected

8 Power control (PWR)

8.1 PWR Firmware driver registers structures

8.2 PWR_VREG_InitTypeDef

PWR_VREG_InitTypeDef is defined in the stm32w108xx_pwr.h

Data Fields

- ***uint32_t PWR_VREFCmd***
- ***uint32_t PWR_1V8Cmd***
- ***uint32_t PWR_1V8TRIM***
- ***uint32_t PWR_1V2Cmd***
- ***uint32_t PWR_1V2TRIM***

Field Documentation

- ***uint32_t PWR_VREG_InitTypeDef::PWR_VREFCmd***
  - Specifies the new state of the selected VREF. This parameter can be set either to ENABLE or DISABLE
- ***uint32_t PWR_VREG_InitTypeDef::PWR_1V8Cmd***
  - Specifies the new state of the selected 1V8. This parameter can be set either to ENABLE or DISABLE
- ***uint32_t PWR_VREG_InitTypeDef::PWR_1V8TRIM***
  - Specifies wether the 1V8 regulator trim value.
- ***uint32_t PWR_VREG_InitTypeDef::PWR_1V2Cmd***
  - Specifies the new state of the selected 1V2. This parameter can be set either to ENABLE or DISABLE
- ***uint32_t PWR_VREG_InitTypeDef::PWR_1V2TRIM***
  - Specifies wether the 1V2 regulator trim value.

8.3 PWR Firmware driver API description

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

8.3.1 How to use this driver

This driver provides the Low level functions to manage the low level power registers. These functions are split in 4 groups:

1. Voltage Regulator control functions: this group includes the management of following features using PWR_VREGInit() function:
   - Configure the regulator Trim values
   - Enable/Disable VREF, V1.8 and V1.2 voltage regulators
2. WakeUp Pin/Source Configuration functions: this group includes all needed to configure an interrupt as WakeUp source:

- To control the GPIO pin to WakeUp the system from low power mode use the PWR_GPIOWakeUpPinCmd() function.
- To configure the WakeUp method to wake the system from low power mode use the PWR_WakeUpSourceConfig() function.
- To command the WakeUp source filter use PWR_WakeUpFilterConfig() function.

3. DeepSleep mode functions: this group includes the deep sleep feature configuration:
   - To freeze the GPIO state before entering in low power mode use the PWR_FreezestateLVoutput() function.
   - To control the deep sleep mode 0 when debugger is attached use the PWR_DeepSleepMode0Cmd() function.
   - To Wake the core from deep sleep 0 the WakeUp source filter use PWR_CoreWake() function.
   - To Disable the system access to the ACK bit in the CSYSPWRUPACKSR use PWR_InhibitCSYSPWRUPACK() function.
4. WakeUp Status functions: this group includes the required functions to manage the WakeUp interrupt status:
   - When the system wake up from low power mode use PWR_GetFlagStatus() to check witch interrupt is the source for WakeUp.
   - After check the user should clear the WakeUp source in the low power status register using PWR_ClearFlag() function.

8.3.2 Voltage Regulator control function

- ***PWR_DeInit()***
- ***PWR_VREGStructInit()***
- ***PWR_VREGInit()***

8.3.3 WakeUp Pin-Source Configuration function

- ***PWR_GPIOWakeUpPinCmd()***
- ***PWR_WakeUpFilterConfig()***
- ***PWR_WakeUpSourceConfig()***

8.3.4 DeepSleep mode function

- ***PWR_FreezestateLVoutput()***
- ***PWR_DeepSleepMode0Cmd()***
- ***PWR_CoreWake()***
- ***PWR_InhibitCSYSPWRUPACK()***

8.3.5 WakeUp status function

- ***PWR_GetFlagStatus()***
- ***PWR_ClearFlag()***

8.3.6 Voltage Regulator control

8.3.6.1 PWR_DeInit

Function Name	**void PWR_DeInit (void)**
Function Description	Deinitializes the PWR peripheral registers to their default reset values.
Parameters	• None.
Return values	• None.
Notes	• None.

8.3.6.2 PWR_VREGStructInit

Function Name	**void PWR_VREGStructInit (*PWR_VREG_InitTypeDef** * VREG_InitStruct)**
Function Description	Fills each VREG_InitStruct member with its default value.
Parameters	• **VREG_InitStruct :** pointer to a PWR_VREG_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

8.3.6.3 PWR_VREGInit

Function Name	**void PWR_VREGInit (*PWR_VREG_InitTypeDef** * VREG_InitStruct)**
Function Description	Initializes the VREG peripheral according to the specified parameters in the VREG_InitStruct.
Parameters	• **VREG_InitStruct :** pointer to a PWR_VREG_InitTypeDef structure that contains the configuration information for the specified VREG.
Return values	• None.
Notes	• None.

8.3.7 WakeUp Pin_Source Configuration

8.3.7.1 PWR_GPIOWakeUpPinCmd

Function Name	**void PWR_GPIOWakeUpPinCmd (*GPIO_TypeDef* * GPIOx, uint32_t GPIO_Pin,FunctionalState NewState)**
Function Description	Enables or disables the GPIO WakeUp pin.
Parameters	• **GPIOx :** where x can be (A, B or C) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bit to be written. This parameter can be one of GPIO_Pin_x where x can be (0..7). • **NewState :** new state of the GPIO WakeUp pin source. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• The GPIO WakeUp monitoring should be enabled before enabling the GPIO WakeUp pin. To enable the GPIO WakeUp monitoring use PWR_WakeUpSourceConfig() function.

8.3.7.2 PWR_WakeUpFilterConfig

Function Name	**void PWR_WakeUpFilterConfig (uint32_t PWR_WakeUpSource,FunctionalState NewState)**
Function Description	Enables or disables the WakeUp source filter.
Parameters	• **PWR_WakeUpSource :** specifies the selected PWR WakeUp source. This parameter can be one of the following values: – ***PWR_WAKEFILTER_GPIO :*** filter active on GPIO monitoring. – ***PWR_WAKEFILTER_SC1 :*** filter active on SC1. – ***PWR_WAKEFILTER_SC2 :*** filter active on SC2. – ***PWR_WAKEFILTER_IRQD :*** filter active on IRQD. • **NewState :** new state of the WakeUp source. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

8.3.7.3 PWR_WakeUpSourceConfig

Function Name	**void PWR_WakeUpSourceConfig (uint32_t PWR_WakeUpSource,FunctionalState NewState)**
Function Description	Enables or disables the WakeUp method form low power mode.
Parameters	• **PWR_WakeUpSource :** specifies the selected PWR wakeup method. This parameter can be one of the following values: – ***PWR_WAKEUP_CSYSPWRRUPREQ :*** Wake up active on CSYSPWRUPREQ event. – ***PWR_WAKEUP_CPWRRUPREQ :*** Wake up active on CPWRRUPREQ event. – ***PWR_WAKEUP_CORE :*** Wake up active on COREWAKE event. – ***PWR_WAKEUP_WRAP :*** Wake up active on sleep timer compare wrap/overflow event. – ***PWR_WAKEUP_COMPB :*** Wake up active on sleep timer compare B event. – ***PWR_WAKEUP_COMPA :*** Wake up active on sleep timer compare A event. – ***PWR_WAKEUP_IRQD :*** Wake up active on falling/rising edge of pin PC0. – ***PWR_WAKEUP_SC2 :*** Wake up active on falling/rising edge of pin PA2 for SC2. – ***PWR_WAKEUP_SC1 :*** Wake up active on falling/rising edge of pin PB2 for SC12. – ***PWR_WAKEUP_MON :*** Wake up active on GPIO monitoring. • **NewState :** new state of the WakeUp source. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

8.3.8 DeepSleep mode

8.3.8.1 PWR_FreezestateLVoutput

Function Name	**void PWR_FreezestateLVoutput (FunctionalState NewState)**
Function Description	Enables or disables the freeze GPIO state LV output.

Parameters	• **NewState :** new freeze state of the GPIO state LV output. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

8.3.8.2 PWR_DeepSleepMode0Cmd

Function Name	**void PWR_DeepSleepMode0Cmd (FunctionalState NewState)**
Function Description	Enables or disables the deep sleep mode 0 when debugger is attached.
Parameters	• **NewState :** new freeze state of the GPIO state LV output. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

8.3.8.3 PWR_CoreWake

Function Name	**void PWR_CoreWake (void)**
Function Description	Wake core form a deep sleep 0.
Parameters	• None.
Return values	• None.
Notes	• None.

8.3.8.4 PWR_InhibitCSYSPWRUPACK

Function Name	**void PWR_InhibitCSYSPWRUPACK (void)**
Function Description	Disables the cortex-M3 system access to the ACK bit in the

CSYSPWRUPACKSR register.

Parameters	• None.
Return values	• None.
Notes	• None.

8.3.9 WakeUp status

8.3.9.1 PWR_GetFlagStatus

Function Name	**FlagStatus PWR_GetFlagStatus (uint32_t PWR_FLAG)**
Function Description	Checks whether the specified PWR flag is set or not.
Parameters	• **PWR_FLAG :** specifies the low power wake up flag to check. This parameter can be one of the following values: – ***PWR_FLAG_CSYSPWRRUPREQ :*** Wake up done using the DAP access to SYS registers flag – ***PWR_FLAG_CPWRRUPREQ :*** Wake up done using the DAP access to DBG registers flag – ***PWR_FLAG_CORE :*** Wake up done using debug port activity flag – ***PWR_FLAG_WRAP :*** Wake up done using sleep timer wrap flag – ***PWR_FLAG_COMPB :*** Wake up done using sleep timer compare B flag – ***PWR_FLAG_COMPA :*** Wake up done using sleep timer compare A flag – ***PWR_FLAG_IRQD :*** Wake up done using external interrupt IRQD flag – ***PWR_FLAG_SC2 :*** Wake up done using serial controller 2 (PA2) flag – ***PWR_FLAG_SC1 :*** Wake up done using serial controller 1 (PB2) flag – ***PWR_FLAG_MON :*** Wake up done using GPIO monitoring flag – ***PWR_FLAG_CPWRUPREQ :*** REQ flag in the CPWRUPREQSR register – ***PWR_FLAG_CSYSPWRUPREQ :*** REQ flag in the CSYSPWRUPREQSR register – ***PWR_FLAG_CSYSPWRUPREQ :*** ACK flag in the CSYSPWRUPREQSR register
Return values	• **The new state of PWR_FLAG (SET or RESET).**
Notes	• None.

8.3.9.2 PWR_ClearFlag

Function Name	**void PWR_ClearFlag (uint32_t PWR_FLAG)**
Function Description	Clears the PWR pending flags.
Parameters	• **PWR_FLAG :** specifies the low power wake up flag to clear. This parameter can be one of the following values: – ***PWR_FLAG_CSYSPWRRUPREQ :*** Wake up done using the DAP access to SYS registers flag – ***PWR_FLAG_CPWRRUPREQ :*** Wake up done using the DAP access to DBG registers flag – ***PWR_FLAG_CORE :*** Wake up done using debug port activity flag – ***PWR_FLAG_WRAP :*** Wake up done using sleep timer wrap flag – ***PWR_FLAG_COMPB :*** Wake up done using sleep timer compare B flag – ***PWR_FLAG_COMPA :*** Wake up done using sleep timer compare A flag – ***PWR_FLAG_IRQD :*** Wake up done using external interrupt IRQD flag – ***PWR_FLAG_SC2 :*** Wake up done using serial controller 2 (PA2) flag – ***PWR_FLAG_SC1 :*** Wake up done using serial controller 1 (PB2) flag – ***PWR_FLAG_MON :*** Wake up done using GPIO monitoring flag
Return values	• **The new state of PWR_FLAG (SET or RESET).**
Notes	• None.

8.4 PWR Firmware driver defines

8.4.1 PWR

PWR

9 Reset (RST)

9.1 RST Firmware driver registers structures

9.1.1 RST_TypeDef

RST_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t SR***

Field Documentation

- ***__IO uint32_t RST_TypeDef::SR***
  - Reset Status Register Address offset: 0x002C

9.2 RST Firmware driver API description

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

9.2.1 RST specific features

This driver provides the information concerning the reset sources.

The reset can be due to:

1. Core lockup
2. Option byte load failure (may be set with other bits
3. Wake-up from Deep Sleep
4. Software reset
5. Watchdog expiration
6. External reset pin signal
7. The application of a Core power supply (or previously failed)
8. Normal power applied

9.2.2 RST_Group1

This section decsribes the function allowing to get the reset event source:

- ***PWR_GetFlagStatus()***

9.2.3 RST_Group1

9.2.3.1 RST_GetFlagStatus

Function Name	**FlagStatus RST_GetFlagStatus (uint32_t RST_FLAG)**
Function Description	Checks whether the specified RST flag is set or not.
Parameters	• **RST_FLAG :** specifies the RST_FLAG flag to check. This parameter can be one of the following values: – ***RST_FLAG_PWRHV :*** Normal power applied – ***RST_FLAG_PWRLV :*** The application of a Core power supply (or previously failed) – ***RST_FLAG_PIN :*** External reset pin signal. – ***RST_FLAG_WDG :*** Watchdog expiration – ***RST_FLAG_SWRST :*** Software reset. – ***RST_FLAG_WKUP :*** Wake-up from Deep Sleep – ***RST_FLAG_OBFAIL :*** Option byte load failure (may be set with other bits) – ***RST_FLAG_LKUP :*** Core lockup
Return values	• **The new state of RST_FLAG (SET or RESET)**
Notes	• None.

9.3 RST Firmware driver defines

9.3.1 RST

RST

10 Serial controller (SC)

10.1 SC Firmware driver registers structures

10.1.1 SC_DMA_Channel_TypeDef

SC_DMA_Channel_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t DMABEGADDAR***
- ***__IO uint32_t DMAENDADDAR***
- ***__IO uint32_t DMABEGADDBR***
- ***__IO uint32_t DMAENDADDBR***

Field Documentation

- ***__IO uint32_t SC_DMA_Channel_TypeDef::DMABEGADDAR***
  - DMA begin address A register Address offset 0x00
- ***__IO uint32_t SC_DMA_Channel_TypeDef::DMAENDADDAR***
  - DMA end address A register Address offset 0x04
- ***__IO uint32_t SC_DMA_Channel_TypeDef::DMABEGADDBR***
  - DMA begin address B register Address offset 0x08
- ***__IO uint32_t SC_DMA_Channel_TypeDef::DMAENDADDBR***
  - DMA end address B register Address offset 0x0C

10.1.2 SC_DMA_InitTypeDef

SC_DMA_InitTypeDef is defined in the stm32w108xx_sc.h

Data Fields

- ***uint32_t DMA_BeginAddrA***
- ***uint32_t DMA_EndAddrA***
- ***uint32_t DMA_BeginAddrB***
- ***uint32_t DMA_EndAddrB***

Field Documentation

- ***uint32_t SC_DMA_InitTypeDef::DMA_BeginAddrA***
  - Specifies the peripheral begin address A for the selected DMA_channel
- ***uint32_t SC_DMA_InitTypeDef::DMA_EndAddrA***
  - Specifies the peripheral end address A for the selected DMA_channel
- ***uint32_t SC_DMA_InitTypeDef::DMA_BeginAddrB***
  - Specifies the peripheral begin address B for the selected DMA_channel
- ***uint32_t SC_DMA_InitTypeDef::DMA_EndAddrB***

– Specifies the peripheral end address B for the selected DMA_channel

10.1.3 SC_DMA_TypeDef

SC_DMA_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t DMARXCNTAR***
- ***__IO uint32_t DMARXCNTBR***
- ***__IO uint32_t DMATXCNTR***
- ***__IO uint32_t DMASR***
- ***__IO uint32_t DMACR***
- ***__IO uint32_t DMARXERRAR***
- ***__IO uint32_t DMARXERRBR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t DMARXCNTSAVEDR***

Field Documentation

- ***__IO uint32_t SC_DMA_TypeDef::DMARXCNTAR***
  – DMA Rx counter A register Address offset 0x20
- ***__IO uint32_t SC_DMA_TypeDef::DMARXCNTBR***
  – DMA Rx counter B register Address offset 0x24
- ***__IO uint32_t SC_DMA_TypeDef::DMATXCNTR***
  – DMA Tx counter register Address offset 0x28
- ***__IO uint32_t SC_DMA_TypeDef::DMASR***
  – DMA status register Address offset 0x2C
- ***__IO uint32_t SC_DMA_TypeDef::DMACR***
  – DMA control register Address offset 0x30
- ***__IO uint32_t SC_DMA_TypeDef::DMARXERRAR***
  – DMA Rx error A register Address offset 0x34
- ***__IO uint32_t SC_DMA_TypeDef::DMARXERRBR***
  – DMA Rx error B register Address offset 0x38
- ***uint32_t SC_DMA_TypeDef::RESERVED0[13]***
  – Reserved
- ***__IO uint32_t SC_DMA_TypeDef::DMARXCNTSAVEDR***
  – DMA Rx Counter saved register Address offset 0x70

10.1.4 SC_I2C_TypeDef

SC_I2C_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t DR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t I2CSR***

- ***uint32_t RESERVED1***
- ***__IO uint32_t I2CCR1***
- ***__IO uint32_t I2CCR2***
- ***__IO uint32_t CR***
- ***uint32_t RESERVED2***
- ***__IO uint32_t CRR1***
- ***__IO uint32_t CRR2***

Field Documentation

- ***__IO uint32_t SC_I2C_TypeDef::DR***
  - Serial control Data register Address offset 0x00
- ***uint32_t SC_I2C_TypeDef::RESERVED0[1]***
  - Reserved
- ***__IO uint32_t SC_I2C_TypeDef::I2CSR***
  - I2C status register Address offset 0x08
- ***uint32_t SC_I2C_TypeDef::RESERVED1[1]***
  - Reserved
- ***__IO uint32_t SC_I2C_TypeDef::I2CCR1***
  - I2C control register 1 Address offset 0x10
- ***__IO uint32_t SC_I2C_TypeDef::I2CCR2***
  - I2C control register 2 Address offset 0x14
- ***__IO uint32_t SC_I2C_TypeDef::CR***
  - Serial control control register Address offset 0x18
- ***uint32_t SC_I2C_TypeDef::RESERVED2[2]***
  - Reserved
- ***__IO uint32_t SC_I2C_TypeDef::CRR1***
  - Serial control clock rate register 1 Address offset 0x24
- ***__IO uint32_t SC_I2C_TypeDef::CRR2***
  - Serial control clock rate register 2 Address offset 0x28

10.1.5 SC_IT_TypeDef

SC_IT_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t ISR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t IER***
- ***uint32_t RESERVED1***
- ***__IO uint32_t ICR***

Field Documentation

- ***__IO uint32_t SC_IT_TypeDef::ISR***
  - Interrupt Status register Address offset 0x08
- ***uint32_t SC_IT_TypeDef::RESERVED0[15]***

  - Reserved
- ***__IO uint32_t SC_IT_TypeDef::IER***
  - Interrupt Enable register Address offset 0x48
- ***uint32_t SC_IT_TypeDef::RESERVED1[2]***
  - Reserved
- ***__IO uint32_t SC_IT_TypeDef::ICR***
  - Interrupt Control register Address offset 0x54

10.1.6 SC_SPI_TypeDef

SC_SPI_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t DR***
- ***__IO uint32_t SPISR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t CR***
- ***__IO uint32_t SPICR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t CRR1***
- ***__IO uint32_t CRR2***

Field Documentation

- ***__IO uint32_t SC_SPI_TypeDef::DR***
  - Serial control Data register Address offset 0x00
- ***__IO uint32_t SC_SPI_TypeDef::SPISR***
  - SPI status register Address offset 0x04
- ***uint32_t SC_SPI_TypeDef::RESERVED0[4]***
  - Reserved
- ***__IO uint32_t SC_SPI_TypeDef::CR***
  - Serial control control register Address offset 0x18
- ***__IO uint32_t SC_SPI_TypeDef::SPICR***
  - SPI status register Address offset 0x1C
- ***uint32_t SC_SPI_TypeDef::RESERVED1[1]***
  - Reserved
- ***__IO uint32_t SC_SPI_TypeDef::CRR1***
  - Serial control clock rate register 1 Address offset 0x24
- ***__IO uint32_t SC_SPI_TypeDef::CRR2***
  - Serial control clock rate register 2 Address offset 0x28

10.1.7 SC_UART_TypeDef

SC_UART_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t DR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t UARTSR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t CR***
- ***uint32_t RESERVED2***
- ***__IO uint32_t UARTCR***
- ***uint32_t RESERVED3***
- ***__IO uint32_t UARTBRR1***
- ***__IO uint32_t UARTBRR2***

Field Documentation

- ***__IO uint32_t SC_UART_TypeDef::DR***
  - Serial control Data register Address offset 0x00
- ***uint32_t SC_UART_TypeDef::RESERVED0[2]***
  - Reserved
- ***__IO uint32_t SC_UART_TypeDef::UARTSR***
  - UART control register Address offset 0x08
- ***uint32_t SC_UART_TypeDef::RESERVED1[2]***
  - Reserved
- ***__IO uint32_t SC_UART_TypeDef::CR***
  - Serial control control register Address offset 0x14
- ***uint32_t SC_UART_TypeDef::RESERVED2[1]***
  - Reserved
- ***__IO uint32_t SC_UART_TypeDef::UARTCR***
  - UART control register Address offset 0x1C
- ***uint32_t SC_UART_TypeDef::RESERVED3[2]***
  - Reserved
- ***__IO uint32_t SC_UART_TypeDef::UARTBRR1***
  - UART Baud rate register 1 Address offset 0x28
- ***__IO uint32_t SC_UART_TypeDef::UARTBRR2***
  - UART Baud rate register 2 Address offset 0x2C

10.1.8 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32w108xx_sc.h

Data Fields

- ***uint16_t SPI_Mode***
- ***uint16_t SPI_CPOL***
- ***uint16_t SPI_CPHA***
- ***uint32_t SPI_ClockRate***
- ***uint16_t SPI_FirstBit***

Field Documentation

- ***uint16_t SPI_InitTypeDef::SPI_Mode***
  - Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_mode***
- ***uint16_t SPI_InitTypeDef::SPI_CPOL***
  - Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint16_t SPI_InitTypeDef::SPI_CPHA***
  - Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::SPI_ClockRate***
  - This member configures the SPI communication clock rate. The clock rate is computed using the following formula: clock rate = 12MHz/((LIN+1)*(2^EXP)
- ***uint16_t SPI_InitTypeDef::SPI_FirstBit***
  - Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_transmission***

10.1.9 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32w108xx_sc.h

Data Fields

- ***uint32_t UART_BaudRate***
- ***uint32_t UART_WordLength***
- ***uint32_t UART_StopBits***
- ***uint32_t UART_Parity***
- ***uint32_t UART_HardwareFlowControl***

Field Documentation

- ***uint32_t UART_InitTypeDef::UART_BaudRate***
  - This member configures the UART communication baud rate. The baud rate is computed using the following formula: Baudrate = 24MHz/(2*N+F)
- ***uint32_t UART_InitTypeDef::UART_WordLength***
  - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UART_Word_Length***
- ***uint32_t UART_InitTypeDef::UART_StopBits***
  - Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***
- ***uint32_t UART_InitTypeDef::UART_Parity***
  - Specifies the parity mode. 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) This parameter can be a value of ***UART_Parity***
- ***uint32_t UART_InitTypeDef::UART_HardwareFlowControl***
  - Specifies wether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***

10.1.10 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32w108xx_sc.h

Data Fields

- ***uint32_t I2C_ClockRate***

Field Documentation

- ***uint32_t I2C_InitTypeDef::I2C_ClockRate***
  - This member configures the I2C communication clock rate. The clock rate is computed using the following formula: clock rate = 12MHz/((LIN+1)*(2^EXP) This parameter must be set to a value lower than 400kHz

10.2 SC Firmware driver API description

10.2.1 How to use this driver

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

- Peripherals GPIO Configuration:
  a. Select the desired pin GPIO_InitStruct->GPIO_Pin according to the defined Initialization and Configuration Tables for each serial control modes (UART, SPI master, SPI Salve and I2C).
  b. Refer to the Initialization and Configuration Tables to configure the GPIO_InitStruct->GPIO_Mode
  c. Call GPIO_Init() function.
- For the I2C mode, program the clock rate using the I2C_Init() function.
- For the SPI mode, program the Polarity, Phase, First Data, Clock rate and the Peripheral Mode rate using the SPI_Init() function.
- For the UART mode, program the Baud Rate, Word Length , Stop Bit, Parity and Hardware flow control using the UART_Init() function.
- Enable the PPP using the PPP_Cmd() function.
- For UART Mode set pull-up resistors on Tx and Rx pins using GPIO_SetBits() function.
- Enable the NVIC and the corresponding interrupt using the function. PPP_ITConfig() if you need to use interrupt mode.
- When using the DMA mode
  a. Configure the DMA using SC_DMA_Init() function.
  b. Active the needed channel Request using SC_DMA_ChannelLoadEnable() function.

PPP can be UART, SPI or I2C.

The DMA is not support for I2C mode.

10.2.2 Universal Asynchronous Receiver-Transmitter functions

This section provides a set of functions allowing handling Universal Asynchronous Receiver Transmitter communications.

Only SC1 includes an universal asynchronous receiver transmitter (UART) controller.

Initialization and Configuration

The GPIO pins that can be assigned to UART interface are listed in the following table:

Parameter	Direction	GPIO confoiguration	SC1 pin
TXD	Out	Alternate output (push-pull)	PB1
RDX	In	Input	PB2
nCTS	In	Input	PB3
nRTS	Out	Alternate output (push-pull)	PB4

For the asynchronous mode 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. Depending on the frame length defined by the M bit (7-bits or 8-bits), the possible UART frame formats are as listed in the following table: M bit PCE bit UART frame 0 0 SB | 7-bit data | STB 0 1 SB | 7-bit data | PB | STB 1 0 SB | 8-bit data | STB 1 1 SB | 8-bit data | PB | STB
- Hardware flow control.

The UART_Init() function follows the UART asynchronous configuration procedure (details for the procedure is available in datasheet.

Data transfers

In reception, data are received and then stored into an internal Rx buffer while In transmission, data are first stored into an internal Tx buffer before being transmitted.

The read access of the SCx_DR register can be done using UART_ReceiveData() function and returns the Rx buffered value. Whereas a write access to the SCx_DR can be done using UART_SendData() function and stores the written data into Tx buffer.

Interrupts and flags management

This subsection provides also a set of functions allowing configuring the UART Interrupts sources, Requests and check or clear the flags or pending bits status. The user should

identify which mode will be used in his application to manage the communication: Polling mode, Interrupt mode or DMA mode(refer SC_Group4).

In Polling mode, the UART communication can be managed by these flags:

1. UART_FLAG_TXE: to indicate the status of the transmit buffer register.
2. UART_FLAG_RXNE: to indicate the status of the receive buffer register.
3. UART_FLAG_IDLE: to indicate the status of the Idle Line.
4. UART_FLAG_CTS: to indicate the status of the nCTS line.
5. UART_FLAG_FE: to indicate if a frame error occurs.
6. UART_FLAG_PE: to indicate if a parity error occurs.
7. UART_FLAG_OVR: to indicate if an Overrun error occurs.

In this mode it is advised to use the following functions:

- FlagStatus UART_GetFlagStatus(SC_UART_TypeDef* SCx_UART, uint32_t UART_FLAG).

In this mode all the UART flags are cleared by hardware.

In Interrupt mode, the UART communication can be managed by 7 interrupt sources and 7 pending bits:

- Pending bits:
  a. UART_IT_PE: to indicate the status of Parity Error interrupt.
  b. UART_IT_FE: to indicate the status of Framing Error interrupt.
  c. UART_IT_UND: to indicate the status of UnderRun Error interrupt.
  d. UART_IT_OVR: to indicate the status of OverRun Error interrupt.
  e. UART_IT_IDLE: to indicate the status of IDLE line detected interrupt.
  f. UART_IT_TXE: to indicate the status of the Transmit data register empty interrupt.
  g. UART_IT_RXNE: to indicate the status of the Data Register not empty interrupt.
- Interrupt source:
  a. UART_IT_PE: specifies the interrupt source for Parity Error pending interrupt.
  b. UART_IT_FE: specifies the interrupt source for Framing Error pending interrupt.
  c. UART_IT_UND: specifies the interrupt source for UnderRun Error pending interrupt.
  d. UART_IT_OVR: specifies the interrupt source for OverRun Error pending interrupt.
  e. UART_IT_IDLE: specifies the interrupt source for IDLE line detected pending interrupt.
  f. UART_IT_TXE: specifies the interrupt source for the Transmit data register empty pending interrupt.
  g. UART_IT_RXNE: specifies the interrupt source for the Data Register not empty pending interrupt. These parameters are coded in order to use them as interrupt source or as pending bits.

In this mode it is advised to use the following functions:

- void UART_ITConfig(SC_IT_TypeDef* SCx_IT, uint32_t UART_IT, FunctionalState NewState).
- ITStatus UART_GetITStatus(SC_IT_TypeDef* SCx_IT, uint32_t UART_IT).
- void UART_ClearITPendingBit(SC_IT_TypeDef* SCx_IT, uint32_t UART_IT).
- ***UART_DeInit()***
- ***UART_Init()***
- ***UART_StructInit()***
- ***UART_RTSAssertionCmd()***
- ***UART_Cmd()***
- ***UART_ITConfig()***

- ***UART_TriggerEventConfig()***
- ***UART_SendData()***
- ***UART_ReceiveData()***
- ***UART_GetFlagStatus()***
- ***UART_GetITStatus()***
- ***UART_ClearITPendingBit()***

10.2.3 Serial Peripheral Interface functions

This section describes a set of functions allowing handling serial peripheral interface communications.

The SC1 and SC2 include an Serial Peripheral Interface (SPI) master/slave controller.

Initialization and Configuration

The GPIO pins that can be assigned to SPI interface are listed in the following tables:

Table 8: SPI master mode

Parameter	Direction	GPIO confoiguration	SC1 pin	SC2 pin
MOSI	Out	Alternate output (push-pull)	PB1	PA0
MISO	In	Input	PB2	PA1
SCLK	Out	Alternate output (push-pull) Special SCLK mode	PB3	PA2

Table 9: SPI slave mode

Parameter	Direction	GPIO confoiguration	SC1 pin	SC2 pin
MOSI	In	Input	PB2	PA0
MISO	Out	Alternate output (push-pull)	PB1	PA1
SCLK	In	Input	PB3	PA2
nSSEL	In	Input	PB4	PA3

In Serial Peripheral Interface mode, configure the following parameters:

- mode.
- Data Size.
- Polarity.
- Phase.
- Baud Rate
- First Bit Transmission

The SPI_Init() function follows the SPI configuration procedures for Master mode and Slave mode (details for these procedures are available in datasheet).

Data transfers

In reception, data are received and then stored into an internal Rx buffer while In transmission, data are first stored into an internal Tx buffer before being transmitted.

The read access of the SCx_DR register can be done using SPI_ReceiveData() function and returns the Rx buffered value, whereas a write access to the SCx_DR can be done using SPI_SendData() function and stores the written data into Tx buffer.

Interrupts and flags management

This subsection describes the set of functions allowing configuring the SPI Interrupts sources, requesting, checking or clearing the flags or pending bits status. The user should identify which mode will be used in his application to manage the communications: Polling mode, Interrupt mode or DMA mode(refer SC_Group4).

In Polling mode, the SPI communications can be managed by these flags:

1. SPI_FLAG_TXE: to indicate the status of the transmit buffer register.
2. SPI_FLAG_RXNE: to indicate the status of the receive buffer register.
3. SPI_FLAG_IDLE: to indicate the status of the Idle Line.
4. SPI_FLAG_OVR: to indicate if an Overrun error occurs.

In this mode it is advised to use the following functions:

- FlagStatus SPI_GetFlagStatus(SC_SPI_TypeDef* SCx_SPI, uint32_t SPI_FLAG).

In this mode all the SPI flags are cleared by hardware.

In Interrupt mode, the SPI communications can be managed by 5 interrupt sources and 5 pending bits:

- Pending bits:
  a. SPI_IT_UND: to indicate the status of UnderRun Error interrupt.
  b. SPI_IT_OVR: to indicate the status of OverRun Error interrupt.
  c. SPI_IT_IDLE: to indicate the status of IDLE line detected interrupt.
  d. SPI_IT_TXE: to indicate the status of the Transmit data register empty interrupt.
  e. SPI_IT_RXNE: to indicate the status of the Data Register not empty interrupt.
- Interrupt source:
  a. SPI_IT_UND: specifies the interrupt source for UnderRun Error pending interrupt.
  b. SPI_IT_OVR: specifies the interrupt source for OverRun Error pending interrupt.
  c. SPI_IT_IDLE: specifies the interrupt source for IDLE line detected pending interrupt.
  d. SPI_IT_TXE: specifies the interrupt source for the Transmit data register empty pending interrupt.
  e. SPI_IT_RXNE: specifies the interrupt source for the Data Register not empty pending interrupt. These parameters are coded in order to use them as interrupt source or as pending bits.

In this mode it is advized to use the following functions:

- void SPI_ITConfig(SC_IT_TypeDef* SCx_IT, uint32_t SPI_IT, FunctionalState NewState).
- ITStatus SPI_GetITStatus(SC_IT_TypeDef* SCx_IT, uint32_t SPI_IT).
- void SPI_ClearITPendingBit(SC_IT_TypeDef* SCx_IT, uint32_t SPI_IT).
- ***SPI_DeInit()***
- ***SPI_Init()***
- ***SPI_StructInit()***
- ***SPI_ReceivermodeConfig()***

- ***SPI_LastByteRepeatCmd()***
- ***SPI_Cmd()***
- ***SPI_ITConfig()***
- ***SPI_TriggerEventConfig()***
- ***SPI_SendData()***
- ***SPI_ReceiveData()***
- ***SPI_GetFlagStatus()***
- ***SPI_GetITStatus()***
- ***SPI_ClearITPendingBit()***

10.2.4 Inter-Integrated Circuit functions

This section describes a set of functions allowing handling the Inter-Integrated Circuit communications.

The SC1 and SC2 include an Inter-integrated circuit interface (I2C) master controller.

Initialization and Configuration

The GPIO pins that can be assigned to I2C interface are listed in the following table:

Parameter	Direction	GPIO confoiguration	SC1 pin	SC2 pin
SDA	In/Out	Alternate output (open-drain)	PB1	PA1
SCL	In/Out	Alternate output (open-drain)	PB2	PA2

For the Inter-Integrated Circuit mode only the Baud Rate parameter can be configured:

The I2C_Init() function follows the I2C configuration procedure (this procedure is available in datasheet).

The generate START and STOP can be done respectively using I2C_GenerateSTART() and I2C_GenerateSTOP() functions.

The command for the ACK generation can be done I2C_AcknowledgeConfig() function.

Data transfers

To initiate a transmit segment, write the data to the SCx_DR data register, then set the BTE bit in the SCx_I2CCR1 register, and finally wait until the BTE bit is clear and the BTF bit in the SCx_I2CSR register. These steps can be done using I2C_SendData() function.

Alternatively to initiate the reception set the BRE bit in the SCx_I2CCR1 register and keep waiting until the BRE bit is cleared and the BTF bit in the SCx_I2CSR register is set.Then read the Rx buffered value, these steps can be done using I2C_ReceiveData() function.

The transmission of the address byte from master to slave to select the slave device in transmitter or in receiver mode can be done using the I2C_Send7bitAddress() function.

Interrupts and flags management

This subsection describes also a set of functions allowing configuring the I2C Interrupts sources, requesting, checking or clearing the flags or pending bits status.The user should

identify which mode will be used in his application to manage communications: Polling mode, Interrupt mode.

In Polling mode, the I2C communication can be managed by 4 flags:

1. I2C_FLAG_NACK: to indicate the status of the not acknowledge flag.
2. I2C_FLAG_BTF: to indicate the status of the byte transfer finished flag.
3. I2C_FLAG_BRF: to indicate the status of the byte receive finished flag.
4. I2C_FLAG_CMDFIN: to indicate the status of the command finished flag.

In this mode it is advised to use the following functions:

- FlagStatus I2C_GetFlagStatus(SC_I2C_TypeDef* SCx_I2C, uint32_t I2C_FLAG).

In this mode all the I2C flags are cleared by hardware.

Interrupt mode

In Interrupt mode, the I2C communication can be managed by 4 interrupt sources and 4 pending bits:

Interrupt source:

1. I2C_IT_NACK: specifies the interrupt source for the not acknowledge interrupt.
2. I2C_IT_CMDFIN: specifies the interrupt source for the command finished interrupt.
3. I2C_IT_BTF: specifies the interrupt source for the byte transfer finished interrupt.
4. I2C_IT_BRF: specifies the interrupt source for the byte receive finished interrupt.

Pending bits:

1. I2C_IT_NACK: to indicate the status of not acknowledge pending interrupt.
2. I2C_IT_CMDFIN: to indicate the status of command finished pending interrupt.
3. I2C_IT_BTF: to indicate the status of byte transfer finished pending interrupt.
4. I2C_IT_BRF: to indicate the status of byte receive finished pending interrupt.

In this mode it is advized to use the following functions:

- void I2C_ClearITPendingBit(SC_IT_TypeDef* SCx_IT, uint32_t I2C_IT).
- ITStatus I2C_GetITStatus(SC_IT_TypeDef* SCx_IT, uint32_t I2C_IT).
- ***I2C_DeInit()***
- ***I2C_Init()***
- ***I2C_StructInit()***
- ***I2C_GenerateSTART()***
- ***I2C_GenerateSTOP()***
- ***I2C_AcknowledgeConfig()***
- ***I2C_Send7bitAddress()***
- ***I2C_SendData()***
- ***I2C_ReceiveData()***
- ***I2C_Cmd()***
- ***I2C_ITConfig()***
- ***I2C_GetFlagStatus()***
- ***I2C_GetITStatus()***
- ***I2C_ClearITPendingBit()***

10.2.5 DMA transfers management functions

This section describes a set of functions that can be used only in DMA mode.

The SC1 and SC2 include a DMA controller that can be used to manage the UART and SPI communications.

Initialization and Configuration

For the DMA mode the following parameters can be configured:

- Begin address buffer A.
- End address buffer A.
- Begin address buffer B.
- End address buffer B.

The DMA_Init() function follows the DMA configuration procedure.

Data transfers

In DMA mode, the UART and SPI communications can be managed by 4 DMA Channel requests:

1. DMA_ChannelLoad_BTx: specifies the DMA transmit channel buffer B transfer request.
2. DMA_ChannelLoad_ATx: specifies the DMA transmit channel buffer A transfer request.
3. DMA_ChannelLoad_BRx: specifies the DMA receive channel buffer B transfer request.
4. DMA_ChannelLoad_ARx: specifies the DMA receive channel buffer A transfer request.

In this mode it is advised to use the following function to load and enables the specified DMA channel:

- void SC_DMA_ChannelLoadEnable(SC_DMA_TypeDef* SCx_DMA, uint32_t Channelxy).

Interrupts and flags management

This subsection described also a set of functions allowing configuring the DMA Interrupts sources, requesting, checking or clearing the flags or pending bits status. The user should identify which mode will be used in his application to manage communications: Polling mode, Interrupt mode.

In Polling mode, the DMA communications can be managed by 4 flags:

1. DMA_FLAG_RXAACK: to indicate the status of the DMA receive buffer A acknowledge flag.
2. DMA_FLAG_RXBACK: to indicate the status of the DMA receive buffer B acknowledge flag.
3. DMA_FLAG_TXAACK: to indicate the status of the DMA transmit buffer A acknowledge flag.
4. DMA_FLAG_TXBACK: to indicate the status of the DMA transmit buffer B acknowledge flag.
5. DMA_FLAG_OVRA: to indicate the status of the DMA buffer B overrun flag.
6. DMA_FLAG_OVRB: to indicate the status of the DMA buffer B overrun flag.
7. DMA_FLAG_PEA: to indicate the status of the DMA Parity error A flag.
8. DMA_FLAG_PEB: to indicate the status of the DMA Parity error B flag.
9. DMA_FLAG_FEA: to indicate the status of the DMA Frame error A flag.
10. DMA_FLAG_FEB: to indicate the status of the DMA Frame error B flag.

11. DMA_FLAG_NSSS: to indicate the status of the status of the receive count flag.

In this mode it is advised to use the following functions:

- FlagStatus SC_DMA_GetFlagStatus(SC_DMA_TypeDef* SCx_DMA, uint32_t DMA_FLAG).

In this mode all the DMA flags are cleared by hardware.

Interrupt mode

In Interrupt mode, the DMA communications can be managed by 4 interrupt sources and 4 pending bits:

Interrupt source:

1. DMA_IT_TXULODB: specifies the interrupt source for the transmit buffer B unloaded interrupt.
2. DMA_IT_TXULODA: specifies the interrupt source for the transmit buffer A unloaded interrupt.
3. DMA_IT_RXULODB: specifies the interrupt source for the receive buffer B unloaded interrupt.
4. DMA_IT_RXULODA: specifies the interrupt source for the receive buffer A unloaded interrupt.

Pending bits:

1. DMA_IT_TXULODB: to indicate the status of transmit buffer B unloaded pending interrupt.
2. DMA_IT_TXULODA: to indicate the status of transmit buffer A unloaded pending interrupt.
3. DMA_IT_RXULODB: to indicate the status of receive buffer B unloaded pending interrupt.
4. DMA_IT_RXULODA: to indicate the status of receive buffer A unloaded pending interrupt.

In this mode it is advised to use the following functions:

- ITStatus SC_DMA_GetITStatus(SC_IT_TypeDef* SCx_IT, uint32_t DMA_IT).
- SC_DMA_ClearITPendingBit(SC_IT_TypeDef* SCx_IT, uint32_t DMA_IT).
- ***SC_DMA_ChannelReset()***
- ***SC_DMA_Init()***
- ***SC_DMA_StructInit()***
- ***SC_DMA_ITConfig()***
- ***SC_DMA_ChannelLoadEnable()***
- ***SC_DMA_GetCounter()***
- ***SC_DMA_GetReceiverErrorOffset()***
- ***SC_DMA_GetFlagStatus()***
- ***SC_DMA_GetITStatus()***
- ***SC_DMA_ClearITPendingBit()***

10.2.6 Universal Asynchronous Receiver_Transmitter communication

10.2.6.1 UART_DeInit

Function Name **void UART_DeInit (*SC_UART_TypeDef** SCx_UART)**

Function Description	Deinitializes the SCx_UART peripheral registers to their default reset values.
Parameters	• **SCx_UART :** where x can be 1 to select the Serial controller peripheral.
Return values	• None.
Notes	• None.

10.2.6.2 UART_Init

Function Name	**void UART_Init (*SC_UART_TypeDef ** SCx_UART, *UART_InitTypeDef ** UART_InitStruct)**
Function Description	Initializes the SC1_UART peripheral according to the specified parameters in the UART_InitStruct .
Parameters	• **SCx_UART :** where x can be 1 to select the Serial controller peripheral. • **UART_InitStruct :** pointer to a UART_InitTypeDef structure that contains the configuration information for the specified SC1_UART peripheral.
Return values	• None.
Notes	• None.

10.2.6.3 UART_StructInit

Function Name	**void UART_StructInit (*UART_InitTypeDef ** UART_InitStruct)**
Function Description	Fills each UART_InitStruct member with its default value.
Parameters	• **UART_InitStruct :** pointer to a UART_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

10.2.6.4 UART_RTSAssertionCmd

Function Name	**void UART_RTSAssertionCmd (*SC_UART_TypeDef** * **SCx_UART,FunctionalState NewState)**
Function Description	Enables or disables the RTS assertion for the specified SC1_UART peripheral.
Parameters	• **SCx_UART :** where x can be 1 to select the Serial controller peripheral. • **NewState :** new state of the SC1_UART peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.6.5 UART_Cmd

Function Name	**void UART_Cmd (*SC_UART_TypeDef** * **SCx_UART,FunctionalState NewState)**
Function Description	Enables or disables the specified SC1_UART peripheral.
Parameters	• **SCx_UART :** where x can be 1 to select the Serial controller peripheral. • **NewState :** new state of the SC1_UART peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.6.6 UART_ITConfig

Function Name	**void UART_ITConfig (*SC_IT_TypeDef** * **SCx_IT, uint32_t UART_IT,FunctionalState NewState)**
Function Description	Enables or disables the specified SCx_UART interrupts.
Parameters	• **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral. • **UART_IT :** specifies the SCx_UART interrupt source to be

enabled or disabled. This parameter can be one of the following values:
  - ***UART_IT_PE :*** Parity error interrupt mask
  - ***UART_IT_FE :*** Frame error interrupt mask
  - ***UART_IT_UND :*** Underrun interrupt mask (to be checked)
  - ***UART_IT_OVR :*** Overrun interrupt mask
  - ***UART_IT_IDLE :*** Idle line detected interrupt mask
  - ***UART_IT_TXE :*** Transmit data register empty interrupt mask
  - ***UART_IT_RXNE :*** Data Register not empty interrupt mask
- **NewState :** new state of the specified SCx_UART interrupt source. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

10.2.6.7 UART_TriggerEventConfig

Function Name
void UART_TriggerEventConfig (*SC_IT_TypeDef SCx_IT, uint32_t UART_IT, uint32_t TriggerEvent)**

Function Description
Trigger event configuration to handle the specified SCx_UART interrupt.

Parameters
- **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral.
- **UART_IT :** specifies the SCx_UART interrupt to be configured. This parameter can be one of the following values:
  - ***UART_IT_IDLE :*** Idle line detected interrupt
  - ***UART_IT_TXE :*** Transmit data register empty interrupt
  - ***UART_IT_RXNE :*** Data Register not empty interrupt
- **TriggerEvent :** Trigger event configuration of the specified SCx_UART interrupt. This parameter can be one of the following values:
  - ***SC_TriggerEvent_Edge :*** The specified SCx_UART interrupt will be generated on edge
  - ***SC_TriggerEvent_Level :*** The specified SCx_UART interrupt will be generated on level

Return values
- None.

Notes
- None.

10.2.6.8 UART_SendData

Function Name	**void UART_SendData (*SC_UART_TypeDef* * SCx_UART, uint8_t Data)**
Function Description	Transmits a Data through the SC1_UART peripheral.
Parameters	• **SCx_UART :** where x can be 1 to select the Serial controller peripheral. • **Data :** Data to be transmitted.
Return values	• None.
Notes	• None.

10.2.6.9 UART_ReceiveData

Function Name	**uint8_t UART_ReceiveData (*SC_UART_TypeDef* * SCx_UART)**
Function Description	Returns the most recent received data by the SC1_UART peripheral.
Parameters	• **SCx_UART :** where x can be 1 to select the Serial controller peripheral.
Return values	• **The value of the received data.**
Notes	• None.

10.2.6.10 UART_GetFlagStatus

Function Name	**FlagStatus UART_GetFlagStatus (*SC_UART_TypeDef* * SCx_UART, uint32_t UART_FLAG)**
Function Description	Checks whether the specified SC1_UART flag is set or not.
Parameters	• **SCx_UART :** where x can be 1 to select the Serial controller peripheral. • **UART_FLAG :** specifies the SCx_UART flag to check. This parameter can be one of the following values: – ***UART_FLAG_CTS :*** Clear to send flag. – ***UART_FLAG_RXNE :*** Receive data register not empty

flag.
- *UART_FLAG_TXE :* Transmit data register empty flag.
- *UART_FLAG_ORE :* OverRun Error flag.
- *UART_FLAG_FE :* Framing Error flag.
- *UART_FLAG_PE :* Parity Error flag.
- *UART_FLAG_IDLE :* Idle Line detection flag.

Return values
- **The new state of UART_FLAG (SET or RESET).**

Notes
- None.

10.2.6.11 UART_GetITStatus

Function Name **ITStatus UART_GetITStatus (*SC_IT_TypeDef** SCx_IT, uint32_t UART_IT)**

Function Description Checks whether the specified SC1_UART pending interrupt is set or not.

Parameters
- **SCx_IT :** where x can be 1 to select the Serial controller peripheral.
- **UART_IT :** specifies the pending interrupt to check. This parameter can be one of the following values:
  - *UART_IT_PE :* Parity error interrupt pending
  - *UART_IT_FE :* Frame error interrupt pending
  - *UART_IT_UND :* Underrun interrupt pending (to be checked)
  - *UART_IT_OVR :* Overrun interrupt pending
  - *UART_IT_IDLE :* Idle line detected interrupt pending
  - *UART_IT_TXE :* Transmit data register empty interrupt pending
  - *UART_IT_RXNE :* Data Register not empty interrupt pending

Return values
- **The new state of UART_IT (SET or RESET).**

Notes
- None.

10.2.6.12 UART_ClearITPendingBit

Function Name **void UART_ClearITPendingBit (*SC_IT_TypeDef** SCx_IT, uint32_t UART_IT)**

Function Description	Clears the SC1_UART interrupt pending bits.
Parameters	• **SCx_IT :** where x can be 1 to select the Serial controller peripheral. • **UART_IT :** specifies the pending interrupt to check. This parameter can be one of the following values: – ***UART_IT_PE :*** Parity error interrupt pending – ***UART_IT_FE :*** Frame error interrupt pending – ***UART_IT_UND :*** Underrun interrupt pending (to be checked) – ***UART_IT_OVR :*** Overrun interrupt pending – ***UART_IT_IDLE :*** Idle line detected interrupt pending – ***UART_IT_TXE :*** Transmit data register empty interrupt pending – ***UART_IT_RXNE :*** Data Register not empty interrupt pending
Return values	• None.
Notes	• None.

10.2.7 Serial peripheral interface communication

10.2.7.1 SPI_DeInit

Function Name	**void SPI_DeInit (*SC_SPI_TypeDef ** SCx_SPI)**
Function Description	Deinitializes the SCx_SPI peripheral registers to their default reset values.
Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral.
Return values	• None.
Notes	• None.

10.2.7.2 SPI_Init

Function Name	**void SPI_Init (*SC_SPI_TypeDef ** SCx_SPI, *SPI_InitTypeDef ** SPI_InitStruct)**
Function Description	Initializes the SCx_SPI peripheral according to the specified parameters in the SPI_InitStruct.

Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral. • **SPI_InitStruct :** pointer to a SPI_InitTypeDef structure that contains the configuration information for the specified SPI peripheral.
Return values	• None.
Notes	• None.

10.2.7.3 SPI_StructInit

Function Name	**void SPI_StructInit (*SPI_InitTypeDef ** SPI_InitStruct)**
Function Description	Fills each SPI_InitStruct member with its default value.
Parameters	• **SPI_InitStruct :** pointer to a SPI_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

10.2.7.4 SPI_ReceiverModeConfig

Function Name	**void SPI_ReceiverModeConfig (*SC_SPI_TypeDef ** SCx_SPI, uint32_t SPI_ReceiverMode)**
Function Description	Configures the Receiver driven mode for the selected SCx_SPI (Master mode only).
Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral. • **SPI_ReceiverMode :** specifies the Receiver driven mode to be configured. This parameter can be one of the following values: – ***SPI_ReceiverMode_TxDataReady :*** Initiate transactions when transmit data is available – ***SPI_ReceiverMode_RxFIFOFree :*** Initiate transactions when receive buffer has space
Return values	• None.
Notes	• None.

10.2.7.5 SPI_LastByteRepeatCmd

Function Name	**void SPI_LastByteRepeatCmd (*SC_SPI_TypeDef** * **SCx_SPI,FunctionalState NewState)**
Function Description	Enables or disables the last byte repeat transmission feature for the specified SCx_SPI peripheral (Slave mode only).
Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral. • **NewState :** new state of the SCx_SPI peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.7.6 SPI_Cmd

Function Name	**void SPI_Cmd (*SC_SPI_TypeDef** * **SCx_SPI,FunctionalState NewState)**
Function Description	Enables or disables the specified SCx_SPI peripheral.
Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral. • **NewState :** new state of the SCx_SPI peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.7.7 SPI_ITConfig

Function Name	**void SPI_ITConfig (*SC_IT_TypeDef** * **SCx_IT, uint32_t SPI_IT,FunctionalState NewState)**
Function Description	Enables or disables the specified SCx_SPI interrupts.

Parameters

- **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral.
- **SPI_IT :** specifies the SCx_SPI interrupt source to be enabled or disabled. This parameter can be one of the following values:
  - ***SPI_IT_UND :*** Underrun interrupt mask
  - ***SPI_IT_OVR :*** Overrun interrupt mask
  - ***SPI_IT_IDLE :*** Idle line detected interrupt mask
  - ***SPI_IT_TXE :*** Transmit data register empty interrupt mask
  - ***SPI_IT_RXNE :*** Data Register not empty interrupt mask
- **NewState :** new state of the specified SCx_SPI interrupt source. This parameter can be: ENABLE or DISABLE.

Return values

- None.

Notes

- None.

10.2.7.8 SPI_TriggerEventConfig

Function Name

void SPI_TriggerEventConfig (*SC_IT_TypeDef SCx_IT, uint32_t SPI_IT, uint32_t TriggerEvent)**

Function Description

Trigger event configuration to handle the specified SCx_SPI interrupt.

Parameters

- **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral.
- **SPI_IT :** specifies the SCx_SPI interrupt to be configured. This parameter can be one of the following values:
  - ***SPI_IT_IDLE :*** Idle line detected interrupt
  - ***SPI_IT_TXE :*** Transmit data register empty interrupt
  - ***SPI_IT_RXNE :*** Data Register not empty interrupt
- **TriggerEvent :** Trigger event configuration of the specified SCx_SPI interrupt. This parameter can be one of the following values:
  - ***SC_TriggeSPI_ITConfig rEvent_Edge :*** The specified SCx_SPI interrupt will be generated on edge
  - ***SC_TriggerEvent_Level :*** The specified SCx_SPI interrupt will be generated on level

Return values

- None.

Notes

- None.

10.2.7.9 SPI_SendData

Function Name	**void SPI_SendData (*SC_SPI_TypeDef* * SCx_SPI, uint8_t Data)**
Function Description	Transmits a Data through the SCx_SPI peripheral.
Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral. • **Data :** Data to be transmitted.
Return values	• None.
Notes	• None.

10.2.7.10 SPI_ReceiveData

Function Name	**uint8_t SPI_ReceiveData (*SC_SPI_TypeDef* * SCx_SPI)**
Function Description	Returns the most recent received data by the SCx_SPI peripheral.
Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral.
Return values	• **The value of the received data.**
Notes	• None.

10.2.7.11 SPI_GetFlagStatus

Function Name	**FlagStatus SPI_GetFlagStatus (*SC_SPI_TypeDef* * SCx_SPI, uint32_t SPI_FLAG)**
Function Description	Checks whether the specified SCx_SPI flag is set or not.
Parameters	• **SCx_SPI :** where x can be 1 or 2 to select the Serial controller peripheral. • **SPI_FLAG :** specifies the SCx_SPI flag to check. This parameter can be one of the following values: – ***SPI_FLAG_OVR :*** OverRun Error flag. – ***SPI_FLAG_TXE :*** Transmit data register empty flag. – ***SPI_FLAG_RXNE :*** Receive data register not empty flag.

– ***SPI_FLAG_IDLE :*** IDLE line flag.

Return values
- **The new state of SPI_FLAG (SET or RESET).**

Notes
- None.

10.2.7.12 SPI_GetITStatus

Function Name
ITStatus SPI_GetITStatus (*SC_IT_TypeDef SCx_IT, uint32_t SPI_IT)**

Function Description
Checks whether the specified SCx_SPI pending interrupt is set or not.

Parameters
- **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral.
- **SPI_IT :** specifies the SCx_SPI pending interrupt to check. This parameter can be one of the following values:
  – ***SPI_IT_UND :*** Underrun interrupt pending
  – ***SPI_IT_OVR :*** Overrun interrupt pending
  – ***SPI_IT_IDLE :*** Idle line detected interrupt pending
  – ***SPI_IT_TXE :*** Transmit data register empty interrupt pending
  – ***SPI_IT_RXNE :*** Data Register not empty interrupt pending

Return values
- **The new state of SPI_IT (SET or RESET).**

Notes
- None.

10.2.7.13 SPI_ClearITPendingBit

Function Name
void SPI_ClearITPendingBit (*SC_IT_TypeDef SCx_IT, uint32_t SPI_IT)**

Function Description
Clears the SCx_SPI interrupt pending bits.

Parameters
- **SCx_IT :** where x can be 1 to select the Serial controller peripheral.
- **SPI_IT :** specifies the SCx_SPI pending interrupt to check. This parameter can be one of the following values:
  – ***SPI_IT_UND :*** Underrun interrupt pending
  – ***SPI_IT_OVR :*** Overrun interrupt pending
  – ***SPI_IT_IDLE :*** Idle line detected interrupt pending

- ***SPI_IT_TXE :*** Transmit data register empty interrupt pending
- ***SPI_IT_RXNE :*** Data Register not empty interrupt pending

Return values	• None.
Notes	• None.

10.2.8 Inter-Integrated Circuit communication

10.2.8.1 I2C_DeInit

Function Name	**void I2C_DeInit (*SC_I2C_TypeDef ** SCx_I2C)**
Function Description	Deinitializes the SCx_I2C peripheral registers to their default reset values.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral.
Return values	• None.
Notes	• None.

10.2.8.2 I2C_Init

Function Name	**void I2C_Init (*SC_I2C_TypeDef ** SCx_I2C, *I2C_InitTypeDef ** I2C_InitStruct)**
Function Description	Initializes the SCx_I2C peripheral according to the specified parameters in the I2C_InitStruct.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral. • **I2C_InitStruct :** pointer to a I2C_InitTypeDef structure that contains the configuration information for the specified SCx_I2C peripheral.
Return values	• None.
Notes	• None.

10.2.8.3 I2C_StructInit

Function Name	**void I2C_StructInit (*I2C_InitTypeDef** * **I2C_InitStruct)**
Function Description	Fills each I2C_InitStruct member with its default value.
Parameters	• **I2C_InitStruct :** pointer to a I2C_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

10.2.8.4 I2C_GenerateSTART

Function Name	**void I2C_GenerateSTART (*SC_I2C_TypeDef** * **SCx_I2C)**
Function Description	Generates SCx_I2C communication START condition.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral.
Return values	• None.
Notes	• None.

10.2.8.5 I2C_GenerateSTOP

Function Name	**void I2C_GenerateSTOP (*SC_I2C_TypeDef** * **SCx_I2C)**
Function Description	Generates SCx_I2C communication STOP condition.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral.
Return values	• None.
Notes	• None.

10.2.8.6 I2C_AcknowledgeConfig

Function Name	**void I2C_AcknowledgeConfig (*SC_I2C_TypeDef** * SCx_I2C,FunctionalState NewState)**
Function Description	Generates SCx_I2C communication Acknowledge.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral. • **NewState :** new state of the Acknowledge. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.8.7 I2C_Send7bitAddress

Function Name	**void I2C_Send7bitAddress (*SC_I2C_TypeDef** * SCx_I2C, uint8_t Address, uint8_t I2C_Direction)**
Function Description	Transmits the address byte to select the slave device.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral. • **Address :** specifies the slave address which will be transmitted • **I2C_Direction :** specifies whether the SCx_I2C device will be a Transmitter or a Receiver. This parameter can be one of the following values – ***I2C_Direction_Transmitter :*** Transmitter mode – ***I2C_Direction_Receiver :*** Receiver mode
Return values	• **None.**
Notes	• None.

10.2.8.8 I2C_SendData

Function Name	**void I2C_SendData (*SC_I2C_TypeDef** * SCx_I2C, uint8_t Data)**
Function Description	Transmits a Data through the SCx_I2C peripheral.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral. • **Data :** Data to be transmitted.
Return values	• None.
Notes	• None.

10.2.8.9 I2C_ReceiveData

Function Name	**uint8_t I2C_ReceiveData (*SC_I2C_TypeDef** * SCx_I2C)**
Function Description	Returns the most recent received data by the SCx_I2C peripheral.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral.
Return values	• **The value of the received data.**
Notes	• None.

10.2.8.10 I2C_Cmd

Function Name	**void I2C_Cmd (*SC_I2C_TypeDef** * SCx_I2C,FunctionalState NewState)**
Function Description	Enables or disables the specified SCx_I2C peripheral.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral. • **NewState :** new state of the SCx_I2C peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.8.11 I2C_ITConfig

Function Name	**void I2C_ITConfig (*SC_IT_TypeDef** * **SCx_IT, uint32_t I2C_IT,FunctionalState NewState)**
Function Description	Enables or disables the specified SCx_I2C interrupts.
Parameters	• **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral. • **I2C_IT :** specifies the SCx_I2C interrupt source to be enabled or disabled. This parameter can be one of the following values: – ***I2C_IT_NACK :*** Not acknowledge interrupt mask – ***I2C_IT_CMDFIN :*** Command finished interrupt mask – ***I2C_IT_BTF :*** Byte transfer finished interrupt mask – ***I2C_IT_BRF :*** Byte receive finished interrupt mask • **NewState :** new state of the specified SCx_I2C interrupt source. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.8.12 I2C_GetFlagStatus

Function Name	**FlagStatus I2C_GetFlagStatus (*SC_I2C_TypeDef** * **SCx_I2C, uint32_t I2C_FLAG)**
Function Description	Checks whether the specified I2C flag is set or not.
Parameters	• **SCx_I2C :** where x can be 1 or 2 to select the Serial controller peripheral. • **I2C_FLAG :** specifies the SCx_I2C flag to check. This parameter can be one of the following values: – ***I2C_FLAG_NACK :*** Not acknowledge flag – ***I2C_FLAG_BTF :*** Byte transfer finished flag – ***I2C_FLAG_BRF :*** Byte receive finished flag – ***I2C_FLAG_CMDFIN :*** Command finished flag
Return values	• **The new state of I2C_FLAG (SET or RESET).**
Notes	• None.

10.2.8.13 I2C_GetITStatus

Function Name	**ITStatus I2C_GetITStatus (*SC_IT_TypeDef** * **SCx_IT, uint32_t I2C_IT)**
Function Description	Checks whether the specified SCx_I2C pending interrupt is set or not.
Parameters	• **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral. • **I2C_IT :** specifies the SCx_I2C interrupt pending to check. This parameter can be one of the following values: – ***I2C_IT_NACK :*** Not acknowledge interrupt pending – ***I2C_IT_CMDFIN :*** Command finished interrupt pending – ***I2C_IT_BTF :*** Byte transfer finished interrupt pending – ***I2C_IT_BRF :*** Byte receive finished interrupt pending
Return values	• **The new state of I2C_IT (SET or RESET).**
Notes	• None.

10.2.8.14 I2C_ClearITPendingBit

Function Name	**void I2C_ClearITPendingBit (*SC_IT_TypeDef** * **SCx_IT, uint32_t I2C_IT)**
Function Description	Clears the SCx_I2C interrupt pending bits.
Parameters	• **SCx_IT :** where x can be 1 to select the Serial controller peripheral. • **I2C_IT :** specifies the SCx_I2C interrupt pending to check. This parameter can be one of the following values: – ***I2C_IT_NACK :*** Not acknowledge interrupt pending – ***I2C_IT_CMDFIN :*** Command finished interrupt pending – ***I2C_IT_BTF :*** Byte transfer finished interrupt pending – ***I2C_IT_BRF :*** Byte receive finished interrupt pending
Return values	• None.
Notes	• None.

10.2.9 DMA transfers management

10.2.9.1 SC_DMA_ChannelReset

Function Name	**void SC_DMA_ChannelReset (*SC_DMA_TypeDef** * **SCx_DMA, uint32_t Channely)**
Function Description	Reset the specified SCx_DMA Channely buffer addresses.
Parameters	• **SCx_DMA :** where x can be 1 or 2 to select the Serial controller peripheral. • **Channely :** specifies the SCx_DMA channel to be enabled. This parameter can be one of the following values: – ***DMA_ChannelReset_Tx :*** DMA reset transmit channels mask – ***DMA_ChannelReset_Rx :*** DMA reset receive channels mask
Return values	• None.
Notes	• None.

10.2.9.2 SC_DMA_Init

Function Name	**void SC_DMA_Init (*SC_DMA_Channel_TypeDef** * **SCx_DMA_Channely, *SC_DMA_InitTypeDef** * **SC_DMA_InitStruct)**
Function Description	Initializes the SCx_DMA Channely according to the specified parameters in the DMA_InitStruct.
Parameters	• **SCx_DMA_Channely :** where x can be 1 or 2 to select the SCx_DMA and y can be Tx or Rx to select the SCx_DMA Channel. • **SC_DMA_InitStruct :** pointer to a DMA_InitTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• None.
Notes	• None.

10.2.9.3 SC_DMA_StructInit

Function Name	**void SC_DMA_StructInit (*SC_DMA_InitTypeDef** * SC_DMA_InitStruct)**
Function Description	Fills each DMA_InitStruct member with its default value.
Parameters	• **SC_DMA_InitStruct :** pointer to a DMA_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

10.2.9.4 SC_DMA_ITConfig

Function Name	**void SC_DMA_ITConfig (*SC_IT_TypeDef** * SCx_IT, uint32_t DMA_IT,FunctionalState NewState)**
Function Description	Enables or disables the specified SCx_DMA interrupts.
Parameters	• **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral. • **DMA_IT :** specifies the SCx_DMA interrupt source to be enabled or disabled. This parameter can be one of the following values: – ***DMA_IT_TXULODB :*** DMA transmit buffer B unloaded interrupt mask – ***DMA_IT_TXULODA :*** DMA transmit buffer A unloaded interrupt mask – ***DMA_IT_RXULODB :*** DMA receive buffer B unloaded interrupt mask – ***DMA_IT_RXULODA :*** DMA receive buffer A unloaded interrupt mask • **NewState :** new state of the specified SCx_DMA interrupt source. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

10.2.9.5 SC_DMA_ChannelLoadEnable

Function Name	**void SC_DMA_ChannelLoadEnable (*SC_DMA_TypeDef** * SCx_DMA, uint32_t Channelxy)**
Function Description	Load and Enables the specified SCx_DMA Channelxy buffers addresses.
Parameters	• **SCx_DMA :** where x can be 1 or 2 to select the Serial controller peripheral. • **Channelxy :** specifies the SCx_DMA channel to be enabled. This parameter can be one of the following values: – ***DMA_ChannelLoad_BTx :*** DMA transmit channel buffer B mask – ***DMA_ChannelLoad_ATx :*** DMA transmit channel buffer A mask – ***DMA_ChannelLoad_BRx :*** DMA receive channel buffer B mask – ***DMA_ChannelLoad_ARx :*** DMA receive channel buffer A mask
Return values	• None.
Notes	• None.

10.2.9.6 SC_DMA_GetCounter

Function Name	**uint32_t SC_DMA_GetCounter (*SC_DMA_TypeDef** * SCx_DMA, uint32_t Counter)**
Function Description	Returns the most recent value for the specific SCx_DMA counter register.
Parameters	• **SCx_DMA :** where x can be 1 or 2 to select the Serial controller peripheral. • **Counter :** specifies the SCx_DMA counter register to be read. This parameter can be one of the following values: – ***DMA_Counter_RXCNTA :*** DMA receive counter A register – ***DMA_Counter_RXCNTB :*** DMA receive counter B register – ***DMA_Counter_TXCNT :*** DMA transmit counter register – ***DMA_Counter_RXCNTSAVED :*** DMA receive counter saved register
Return values	• **The DMA register counter value.**
Notes	• None.

10.2.9.7 SC_DMA_GetReceiverErrorOffset

Function Name	**uint32_t SC_DMA_GetReceiverErrorOffset (*SC_DMA_TypeDef** * **SCx_DMA, uint32_t RegisterError)**
Function Description	Returns the specified SCx_DMA receive error register.
Parameters	• **SCx_DMA :** where x can be 1 or 2 to select the Serial controller peripheral. • **RegisterError :** specifies the SCx_DMA receiver error register to be read. This parameter can be one of the following values: – ***DMA_ReceiverError_CNTA :*** DMA receive error register A – ***DMA_ReceiverError_CNTB :*** DMA receive error register B
Return values	• **The DMA receive error register value.**
Notes	• None.

10.2.9.8 SC_DMA_GetFlagStatus

Function Name	**FlagStatus SC_DMA_GetFlagStatus (*SC_DMA_TypeDef** * **SCx_DMA, uint32_t DMA_FLAG)**
Function Description	Checks whether the specified DMA flag is set or not.
Parameters	• **SCx_DMA :** where x can be 1 or 2 to select the Serial controller peripheral. • **DMA_FLAG :** specifies the SCx_DMA flag to check. This parameter can be one of the following values: – ***DMA_FLAG_RXAACK :*** DMA receive buffer A acknowledge flag – ***DMA_FLAG_RXBACK :*** DMA receive buffer B acknowledge flag – ***DMA_FLAG_TXAACK :*** DMA transmit buffer A acknowledge flag – ***DMA_FLAG_TXBACK :*** DMA transmit buffer B acknowledge flag – ***DMA_FLAG_OVRA :*** DMA buffer B overrun flag – ***DMA_FLAG_OVRB :*** DMA buffer B overrun flag – ***DMA_FLAG_PEA :*** DMA Parity error A flag

- ***DMA_FLAG_PEB :*** DMA Parity error B flag
- ***DMA_FLAG_FEA :*** DMA Frame error A flag
- ***DMA_FLAG_FEB :*** DMA Frame error B flag
- ***DMA_FLAG_NSSS :*** DMA Status of the receive count flag

Return values
- **The new state of DMA_FLAG (SET or RESET).**

Notes
- None.

10.2.9.9 SC_DMA_GetITStatus

Function Name
ITStatus SC_DMA_GetITStatus (*SC_IT_TypeDef SCx_IT, uint32_t DMA_IT)**

Function Description
Checks whether the specified SCx_DMA pending interrupt is set or not.

Parameters
- **SCx_IT :** where x can be 1 or 2 to select the Serial controller peripheral.
- **DMA_IT :** specifies the SCx_DMA interrupt pending to check. This parameter can be one of the following values:
  - ***DMA_IT_TXULODB :*** DMA transmit buffer B unloaded interrupt pending
  - ***DMA_IT_TXULODA :*** DMA transmit buffer A unloaded interrupt pending
  - ***DMA_IT_RXULODB :*** DMA receive buffer B unloaded interrupt pending
  - ***DMA_IT_RXULODA :*** DMA receive buffer A unloaded interrupt pending

Return values
- **The new state of DMA_IT (SET or RESET).**

Notes
- None.

10.2.9.10 SC_DMA_ClearITPendingBit

Function Name
void SC_DMA_ClearITPendingBit (*SC_IT_TypeDef SCx_IT, uint32_t DMA_IT)**

Function Description
Clears the SCx_DMA interrupt pending bits.

Parameters
- **SCx_IT :** where x can be 1 to select the Serial controller peripheral.

- **DMA_IT :** specifies the SCx_DMA interrupt pending to check. This parameter can be one of the following values:
  - ***DMA_IT_TXULODB :*** DMA transmit buffer B unloaded interrupt pending
  - ***DMA_IT_TXULODA :*** DMA transmit buffer A unloaded interrupt pending
  - ***DMA_IT_RXULODB :*** DMA receive buffer B unloaded interrupt pending
  - ***DMA_IT_RXULODA :*** DMA receive buffer A unloaded interrupt pending

Return values
- None.

Notes
- None.

10.3 SC Firmware driver defines

10.3.1 SC

SC

SC_mode

- #define: ***SC_Mode_Disable ((uint32_t)0x00000000)***
- #define: ***SC_Mode_UART ((uint32_t)0x00000001)***
- #define: ***SC_Mode_SPI ((uint32_t)0x00000002)***
- #define: ***SC_Mode_I2C ((uint32_t)0x00000003)***

11 General-purpose timers (TIM)

11.1 TIM Firmware driver registers structures

11.1.1 TIM_ICInitTypeDef

TIM_ICInitTypeDef is defined in the stm32w108xx_tim.h

Data Fields

- ***uint32_t TIM_Channel***
- ***uint32_t TIM_ICPolarity***
- ***uint32_t TIM_ICSelection***
- ***uint32_t TIM_ICPrescaler***
- ***uint32_t TIM_ICFilter***

Field Documentation

- ***uint32_t TIM_ICInitTypeDef::TIM_Channel***
  - Specifies the TIM channel. This parameter can be a value of ***TIM_Channel***
- ***uint32_t TIM_ICInitTypeDef::TIM_ICPolarity***
  - Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_ICInitTypeDef::TIM_ICSelection***
  - Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_ICInitTypeDef::TIM_ICPrescaler***
  - Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_ICInitTypeDef::TIM_ICFilter***
  - Specifies the input capture filter. This parameter can be a number between 0x0 and 0xF

11.1.2 TIM_IT_TypeDef

TIM_IT_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t ISR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t IMR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t IER***

Field Documentation

- ***__IO uint32_t TIM_IT_TypeDef::ISR***
  - TIM interrupt and status register Address offset 0x00
- ***uint32_t TIM_IT_TypeDef::RESERVED0[5]***
  - Reserved
- ***__IO uint32_t TIM_IT_TypeDef::IMR***
  - TIM interrupt missed register Address offset 0x18
- ***uint32_t TIM_IT_TypeDef::RESERVED1[9]***
  - Reserved
- ***__IO uint32_t TIM_IT_TypeDef::IER***
  - TIM interrupt enable register Address offset 0x40

11.1.3 TIM_OCInitTypeDef

TIM_OCInitTypeDef is defined in the stm32w108xx_tim.h

Data Fields

- ***uint32_t TIM_OCMode***
- ***uint32_t TIM_OutputState***
- ***uint32_t TIM_Pulse***
- ***uint32_t TIM_OCPolarity***

Field Documentation

- ***uint32_t TIM_OCInitTypeDef::TIM_OCMode***
  - Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OCInitTypeDef::TIM_OutputState***
  - Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_Output_Compare_state***
- ***uint32_t TIM_OCInitTypeDef::TIM_Pulse***
  - Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between 0x0000 and 0xFFFF
- ***uint32_t TIM_OCInitTypeDef::TIM_OCPolarity***
  - Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***

11.1.4 TIM_TimeBaseInitTypeDef

TIM_TimeBaseInitTypeDef is defined in the stm32w108xx_tim.h

Data Fields

- ***uint32_t TIM_Prescaler***
- ***uint32_t TIM_CounterMode***
- ***uint32_t TIM_Period***

Field Documentation

- ***uint32_t TIM_TimeBaseInitTypeDef::TIM_Prescaler***
  - Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between 0x0000 and 0x000F
- ***uint32_t TIM_TimeBaseInitTypeDef::TIM_CounterMode***
  - Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_TimeBaseInitTypeDef::TIM_Period***
  - Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between 0x0000 and 0xFFFF.

11.1.5 TIM_TypeDef

TIM_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t SMCR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t EGR***
- ***__IO uint32_t CCMR1***
- ***__IO uint32_t CCMR2***
- ***__IO uint32_t CCER***
- ***__IO uint32_t CNT***
- ***__IO uint32_t PSC***
- ***__IO uint32_t ARR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t CCR1***
- ***__IO uint32_t CCR2***
- ***__IO uint32_t CCR3***
- ***__IO uint32_t CCR4***
- ***uint32_t RESERVED2***
- ***__IO uint32_t OR***

Field Documentation

- ***__IO uint32_t TIM_TypeDef::CR1***
  - TIM control register 1, Address offset 0x00
- ***__IO uint32_t TIM_TypeDef::CR2***
  - TIM control register 2, Address offset 0x04
- ***__IO uint32_t TIM_TypeDef::SMCR***
  - TIM slave Mode Control register, Address offset 0x08
- ***uint32_t TIM_TypeDef::RESERVED0[2]***
  - Reserved
- ***__IO uint32_t TIM_TypeDef::EGR***

  - TIM event generation register Address offset 0x14
- ***__IO uint32_t TIM_TypeDef::CCMR1***
  - TIM capture/compare mode register 1, Address offset 0x18
- ***__IO uint32_t TIM_TypeDef::CCMR2***
  - TIM capture/compare mode register 2, Address offset 0x1C
- ***__IO uint32_t TIM_TypeDef::CCER***
  - TIM capture/compare enable register, Address offset 0x20
- ***__IO uint32_t TIM_TypeDef::CNT***
  - TIM counter register, Address offset 0x24
- ***__IO uint32_t TIM_TypeDef::PSC***
  - TIM prescaler register, Address offset 0x28
- ***__IO uint32_t TIM_TypeDef::ARR***
  - TIM auto-reload register, Address offset 0x2C
- ***uint32_t TIM_TypeDef::RESERVED1***
  - Reserved
- ***__IO uint32_t TIM_TypeDef::CCR1***
  - TIM capture/compare register 1, Address offset 0x34
- ***__IO uint32_t TIM_TypeDef::CCR2***
  - TIM capture/compare register 2, Address offset 0x38
- ***__IO uint32_t TIM_TypeDef::CCR3***
  - TIM capture/compare register 3, Address offset 0x3C
- ***__IO uint32_t TIM_TypeDef::CCR4***
  - TIM capture/compare register 4, Address offset 0x40
- ***uint32_t TIM_TypeDef::RESERVED2[3]***
  - Reserved
- ***__IO uint32_t TIM_TypeDef::OR***
  - TIM option register, Address offset 0x50

11.2 TIM Firmware driver API description

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

11.2.1 How to use this driver

This driver describes the functions allowing configuring and programming the timers (TIM) of all STM32W108xx devices. These functions are split in 8 groups:

1. TIM TimeBase management: this group includes all needed functions to configure the TM Timebase unit:
   - Set/Get Prescaler.
   - Set/Get Autoreload.
   - Counter modes configuration.
   - Set Clock division.
   - Select the One Pulse mode.
   - Update Request Configuration.
   - Update Disable Configuration.
   - Auto-Preload Configuration.
   - Enable/Disable the counter.
2. TIM Output Compare management: this group includes all needed functions to configure the Capture/Compare unit used in Output compare mode:
   - Configure each channel, independently, in Output Compare mode.

- Select the output compare modes.
- Select the Polarities of each channel.
- Set/Get the Capture/Compare register values.
- Select the Output Compare Fast mode.
- Select the Output Compare Forced mode.
- Output Compare-Preload Configuration.
- Clear Output Compare Reference.
- Select the OCREF Clear signal.
- Enable/Disable the Capture/Compare Channels.

3. TIM Input Capture management: this group includes all needed functions to configure the Capture/Compare unit used in Input Capture mode:
   - Configure each channel in input capture mode.
   - Configure Channel1/2 in PWM Input mode.
   - Set the Input Capture Prescaler.
   - Get the Capture/Compare values.
4. Interrupts and flags management functions: this group includes all needed functions to manage interrupts:
   - Enables or disables the specified TIM interrupts.
   - Configures the TIMx event to be generate by software.
   - Checks whether the TIM interrupt has occurred or not.
   - Clears the TIMx's interrupt pending bits.
5. TIM clocks management: this group includes all needed functions to configure the clock controller unit:
   - Select internal/External clock.
   - Select the external clock mode: ETR(mode1/mode2), TIx or ITRx.
6. TIM synchronization management: this group includes all needed. functions to configure the Synchronization unit:
   - Select Input Trigger.
   - Select Output Trigger.
   - Select Master Slave mode.
   - ETR Configuration when used as external trigger.
7. TIM specific interface management, this group includes all needed functions to use the specific TIM interface:
   - Encoder Interface Configuration.
   - Select Hall Sensor.
8. TIM specific remapping management includes the Remapping configuration of specific timers

11.2.2 TimeBase management functions

TIM Driver: how to use it in Timing(Time base) Mode

To use the Timer in Timing(Time base) mode, the following steps are mandatory:

1. Fill the TIM_TimeBaseInitStruct with the desired parameters.
2. Call TIM_TimeBaseInit(TIMx, &TIM_TimeBaseInitStruct) to configure the Time Base unit with the corresponding configuration.
3. Enable the NVIC if you need to generate the update interrupt.
4. Enable the corresponding interrupt using the function TIM_ITConfig(TIMx, TIM_IT_Update).
5. Call the TIM_Cmd(ENABLE) function to enable the TIM counter.

All other functions can be used seperatly to modify, if needed, a specific feature of the Timer.

- ***TIM_DeInit()***
- ***TIM_TimeBaseInit()***
- ***TIM_TimeBaseStructInit()***
- ***TIM_PrescalerConfig()***
- ***TIM_CounterModeConfig()***
- ***TIM_SetCounter()***
- ***TIM_SetAutoreload()***
- ***TIM_GetCounter()***
- ***TIM_GetPrescaler()***
- ***TIM_UpdateDisableConfig()***
- ***TIM_UpdateRequestConfig()***
- ***TIM_ARRPreloadConfig()***
- ***TIM_SelectOnePulseMode()***
- ***TIM_Cmd()***

11.2.3 Output Compare management functions

TIM Driver: how to use it in Output Compare Mode

To use the Timer in Output Compare mode, the following steps are mandatory:

1. Configure the TIM pins by configuring the corresponding GPIO pins
2. Configure the Time base unit as described in the first part of this driver, if needed, else the Timer will run with the default configuration:
   - Autoreload value = 0xFFFF.
   - Prescaler value = 0x0000.
   - Counter mode = Up counting.
3. Fill the TIM_OCInitStruct with the desired parameters including:
   - The TIM Output Compare mode: TIM_OCMode.
   - TIM Output State: TIM_OutputState.
   - TIM Pulse value: TIM_Pulse.
   - TIM Output Compare Polarity : TIM_OCPolarity.
4. Call TIM_OCxInit(TIMx, &TIM_OCInitStruct) to configure the desired channel with the corresponding configuration.
5. Call the TIM_Cmd(ENABLE) function to enable the TIM counter.

All other functions can be used separately to modify, if needed, a specific feature of the Timer.

In case of PWM mode, this function is mandatory: TIM_OCxPreloadConfig(TIMx, TIM_OCPreload_ENABLE).

If the corresponding interrupt are needed, the user should:

1. Enable the NVIC to use the TIM interrupts.
2. Enable the corresponding interrupt using the function TIM_ITConfig(TIMx, TIM_IT_CCx).

- ***TIM_OC1Init()***
- ***TIM_OC2Init()***
- ***TIM_OC3Init()***
- ***TIM_OC4Init()***
- ***TIM_OCStructInit()***
- ***TIM_SelectOCxM()***
- ***TIM_SetCompare1()***
- ***TIM_SetCompare2()***
- ***TIM_SetCompare3()***
- ***TIM_SetCompare4()***
- ***TIM_ForcedOC1Config()***
- ***TIM_ForcedOC2Config()***
- ***TIM_ForcedOC3Config()***
- ***TIM_ForcedOC4Config()***
- ***TIM_OC1PreloadConfig()***
- ***TIM_OC2PreloadConfig()***
- ***TIM_OC3PreloadConfig()***
- ***TIM_OC4PreloadConfig()***
- ***TIM_OC1FastConfig()***
- ***TIM_OC2FastConfig()***
- ***TIM_OC3FastConfig()***
- ***TIM_OC4FastConfig()***
- ***TIM_OC1PolarityConfig()***
- ***TIM_OC2PolarityConfig()***
- ***TIM_OC3PolarityConfig()***
- ***TIM_OC4PolarityConfig()***
- ***TIM_CCxCmd()***

11.2.4 Input Capture management functions

TIM Driver: how to use it in Input Capture Mode

To use the Timer in Input Capture mode, the following steps are mandatory:

1. Configure the TIM pins by configuring the corresponding GPIO pins.
2. Configure the Time base unit as described in the first part of this driver, if needed, else the Timer will run with the default configuration:
   - Autoreload value = 0xFFFF.
   - Prescaler value = 0x0000.
   - Counter mode = Up counting.
3. Fill the TIM_ICInitStruct with the desired parameters including:
   - TIM Channel: TIM_Channel.
   - TIM Input Capture polarity: TIM_ICPolarity.
   - TIM Input Capture selection: TIM_ICSelection.
   - TIM Input Capture Prescaler: TIM_ICPrescaler.
   - TIM Input CApture filter value: TIM_ICFilter.

4. Call TIM_ICInit(TIMx, &TIM_ICInitStruct) to configure the desired channel with the corresponding configuration and to measure only frequency or duty cycle of the input signal,or, Call TIM_PWMIConfig(TIMx, &TIM_ICInitStruct) to configure the desired channels with the corresponding configuration and to measure the frequency and the duty cycle of the input signal.
5. Enable the NVIC to read the measured frequency.
6. Enable the corresponding interrupt to read the Captured value, using the function TIM_ITConfig(TIMx, TIM_IT_CCx).
7. Call the TIM_Cmd(ENABLE) function to enable the TIM counter.
8. Use TIM_GetCapturex(TIMx); to read the captured value.

All other functions can be used separately to modify, if needed, a specific feature of the Timer.

- ***TIM_ICInit()***
- ***TIM_ICStructInit()***
- ***TIM_PWMIConfig()***
- ***TIM_GetCapture1()***
- ***TIM_GetCapture2()***
- ***TIM_GetCapture3()***
- ***TIM_GetCapture4()***
- ***TIM_SetIC1Prescaler()***
- ***TIM_SetIC2Prescaler()***
- ***TIM_SetIC3Prescaler()***
- ***TIM_SetIC4Prescaler()***

11.2.5 Interrupts and flags management functions

- ***TIM_ITConfig()***
- ***TIM_GenerateEvent()***
- ***TIM_GetITStatus()***
- ***TIM_ClearITPendingBit()***

11.2.6 Clocks management functions

- ***TIM_InternalClockConfig()***
- ***TIM_ITRxExternalClockConfig()***
- ***TIM_TIxExternalClockConfig()***
- ***TIM_ETRClockMode1Config()***
- ***TIM_ETRClockMode2Config()***

11.2.7 Synchronization management functions

TIM Driver: how to use it in synchronization Mode

Case of multiple Timers

1. Configure the Master Timers using the following functions:

– void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, uint32_t TIM_TRGOSource).
– void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, uint32_t TIM_MasterSlaveMode);

2. Configure the Slave Timers using the following functions:
– void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, uint32_t TIM_InputTriggerSource);
– void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint32_t TIM_SlaveMode);

Case of Timers and external trigger(ETR pin)

1. Configure the Etrenal trigger using this function:
– void TIM_ETRConfig(TIM_TypeDef* TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter);
2. Configure the Slave Timers using the following functions:
– void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, uint32_t TIM_InputTriggerSource);
– void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint32_t TIM_SlaveMode);

- ***TIM_SelectInputTrigger()***
- ***TIM_SelectOutputTrigger()***
- ***TIM_SelectSlaveMode()***
- ***TIM_SelectMasterSlaveMode()***
- ***TIM_ETRConfig()***

11.2.8 Specific interface management functions

- ***TIM_EncoderInterfaceConfig()***
- ***TIM_SelectHallSensor()***

11.2.9 Specific remapping management function

- ***TIM_ClockMaskConfig()***
- ***TIM_SelectExternalTriggerClock()***
- ***TIM_RemapCmd()***

11.2.10 TimeBase management functions

11.2.10.1 TIM_DeInit

Function Name	**void TIM_DeInit (*TIM_TypeDef** * TIMx)**
Function Description	Deinitializes the TIMx peripheral registers to their default reset values.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral.
Return values	• None.
Notes	• None.

11.2.10.2 TIM_TimeBaseInit

Function Name	**void TIM_TimeBaseInit (*TIM_TypeDef** * **TIMx, *TIM_TimeBaseInitTypeDef** * **TIM_TimeBaseInitStruct)**
Function Description	Initializes the TIMx Time Base Unit peripheral according to the specified parameters in the TIM_TimeBaseInitStruct.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **TIM_TimeBaseInitStruct :** pointer to a TIM_TimeBaseInitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

11.2.10.3 TIM_TimeBaseStructInit

Function Name	**void TIM_TimeBaseStructInit (*TIM_TimeBaseInitTypeDef** * **TIM_TimeBaseInitStruct)**
Function Description	Fills each TIM_TimeBaseInitStruct member with its default value.
Parameters	• **TIM_TimeBaseInitStruct :** : pointer to a TIM_TimeBaseInitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

11.2.10.4 TIM_PrescalerConfig

Function Name	**void TIM_PrescalerConfig (*TIM_TypeDef** * **TIMx, uint32_t Prescaler, uint32_t TIM_PSCReloadMode)**
Function Description	Configures the TIMx Prescaler.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **Prescaler :** specifies the Prescaler Register value • **TIM_PSCReloadMode :** specifies the TIM Prescaler Reload mode This parameter can be one of the following values:

- *TIM_PSCReloadMode_Update :* The Prescaler is loaded at the update event.
- *TIM_PSCReloadMode_Immediate :* The Prescaler is loaded immediatly.

Return values
- None.

Notes
- None.

11.2.10.5 TIM_CounterModeConfig

Function Name **void TIM_CounterModeConfig (*TIM_TypeDef** TIMx, uint32_t TIM_CounterMode)**

Function Description Specifies the TIMx Counter Mode to be used.

Parameters
- **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
- **TIM_CounterMode :** specifies the Counter Mode to be used This parameter can be one of the following values:
  - *TIM_CounterMode_Up :* TIM Up Counting Mode
  - *TIM_CounterMode_Down :* TIM Down Counting Mode
  - *TIM_CounterMode_CenterAligned1 :* TIM Center Aligned Mode1
  - *TIM_CounterMode_CenterAligned2 :* TIM Center Aligned Mode2
  - *TIM_CounterMode_CenterAligned3 :* TIM Center Aligned Mode3

Return values
- None.

Notes
- None.

11.2.10.6 TIM_SetCounter

Function Name **void TIM_SetCounter (*TIM_TypeDef** TIMx, uint32_t Counter)**

Function Description Sets the TIMx Counter Register value.

Parameters
- **TIMx :** where x can be 1 and 2 to select the TIM peripheral.
- **Counter :** specifies the Counter register new value.

Return values
- None.

Notes
- None.

11.2.10.7 TIM_SetAutoreload

Function Name	**void TIM_SetAutoreload (*TIM_TypeDef** * TIMx, uint32_t Autoreload)**
Function Description	Sets the TIMx Autoreload Register value.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **Autoreload :** specifies the Autoreload register new value.
Return values	• None.
Notes	• None.

11.2.10.8 TIM_GetCounter

Function Name	**uint32_t TIM_GetCounter (*TIM_TypeDef** * TIMx)**
Function Description	Gets the TIMx Counter value.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral.
Return values	• **Counter Register value.**
Notes	• None.

11.2.10.9 TIM_GetPrescaler

Function Name	**uint32_t TIM_GetPrescaler (*TIM_TypeDef** * TIMx)**
Function Description	Gets the TIMx Prescaler value.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral.
Return values	• **Prescaler Register value.**
Notes	• None.

11.2.10.10 TIM_UpdateDisableConfig

Function Name	**void TIM_UpdateDisableConfig (*TIM_TypeDef* * *TIM_TypeDef* * TIMx,FunctionalState NewState)**
Function Description	Enables or Disables the TIMx Update event.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **NewState :** new state of the TIMx UDIS bit This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

11.2.10.11 TIM_UpdateRequestConfig

Function Name	**void TIM_UpdateRequestConfig (*TIM_TypeDef* * TIMx, uint32_t TIM_UpdateSource)**
Function Description	Configures the TIMx Update Request Interrupt source.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **TIM_UpdateSource :** specifies the Update source. This parameter can be one of the following values: – ***TIM_UpdateSource_Regular :*** Source of update is the counter overflow/underflow or the setting of UG bit, or an update generation through the slave mode controller. – ***TIM_UpdateSource_Global :*** Source of update is counter overflow/underflow.
Return values	• None.
Notes	• None.

11.2.10.12 TIM_ARRPreloadConfig

Function Name	**void TIM_ARRPreloadConfig (*TIM_TypeDef** * TIMx,FunctionalState NewState)**
Function Description	Enables or disables TIMx peripheral Preload register on ARR.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **NewState :** new state of the TIMx peripheral Preload register This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

11.2.10.13 TIM_SelectOnePulseMode

Function Name	**void TIM_SelectOnePulseMode (*TIM_TypeDef** * TIMx, uint32_t TIM_OPMode)**
Function Description	Selects the TIMx's One Pulse Mode.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **TIM_OPMode :** specifies the OPM Mode to be used. This parameter can be one of the following values: – ***TIM_OPMode_Single :*** – ***TIM_OPMode_Repetitive :***
Return values	• None.
Notes	• None.

11.2.10.14 TIM_Cmd

Function Name	**void TIM_Cmd (*TIM_TypeDef** * TIMx,FunctionalState NewState)**
Function Description	Enables or disables the specified TIM peripheral.
Parameters	• **TIMx :** where x can be 1 and 2 and 17to select the TIMx peripheral. • **NewState :** new state of the TIMx peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

11.2.11 Output Compare management functions

11.2.11.1 TIM_OC1Init

Function Name	**void TIM_OC1Init (*TIM_TypeDef* * TIMx, *TIM_OCInitTypeDef* * TIM_OCInitStruct)**
Function Description	Initializes the TIMx Channel1 according to the specified parameters in the TIM_OCInitStruct.
Parameters	• **TIMx :** where x can be 1 and 2 to select the TIM peripheral. • **TIM_OCInitStruct :** pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

11.2.11.2 TIM_OC2Init

Function Name	**void TIM_OC2Init (*TIM_TypeDef* * TIMx, *TIM_OCInitTypeDef* * TIM_OCInitStruct)**
Function Description	Initializes the TIMx Channel2 according to the specified parameters in the TIM_OCInitStruct.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCInitStruct :** pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

11.2.11.3 TIM_OC3Init

Function Name	**void TIM_OC3Init (*TIM_TypeDef** * TIMx, *TIM_OCInitTypeDef** * TIM_OCInitStruct)**
Function Description	Initializes the TIMx Channel3 according to the specified parameters in the TIM_OCInitStruct.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCInitStruct :** pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

11.2.11.4 TIM_OC4Init

Function Name	**void TIM_OC4Init (*TIM_TypeDef** * TIMx, *TIM_OCInitTypeDef ** * TIM_OCInitStruct)**
Function Description	Initializes the TIMx Channel4 according to the specified parameters in the TIM_OCInitStruct.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCInitStruct :** pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

11.2.11.5 TIM_OCStructInit

Function Name	**void TIM_OCStructInit (*TIM_OCInitTypeDef** * TIM_OCInitStruct)**
Function Description	Fills each TIM_OCInitStruct member with its default value.
Parameters	• **TIM_OCInitStruct :** : pointer to a TIM_OCInitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

11.2.11.6 TIM_SelectOCxM

Function Name	**void TIM_SelectOCxM (*TIM_TypeDef** * **TIMx, uint32_t TIM_Channel, uint32_t TIM_OCMode)**
Function Description	Selects the TIM Output Compare Mode.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_Channel :** specifies the TIM Channel This parameter can be one of the following values: – ***TIM_Channel_1 :*** TIM Channel 1 – ***TIM_Channel_2 :*** TIM Channel 2 – ***TIM_Channel_3 :*** TIM Channel 3 – ***TIM_Channel_4 :*** TIM Channel 4 • **TIM_OCMode :** specifies the TIM Output Compare Mode. This parameter can be one of the following values: – ***TIM_OCMode_Timing :*** – ***TIM_OCMode_Active :*** – ***TIM_OCMode_Toggle :*** – ***TIM_OCMode_PWM1 :*** – ***TIM_OCMode_PWM2 :*** – ***TIM_ForcedAction_Active :*** – ***TIM_ForcedAction_InActive :***
Return values	• None.
Notes	• This function disables the selected channel before changing the Output Compare Mode. User has to enable this channel using TIM_CCxCmd and TIM_CCxNCmd functions.

11.2.11.7 TIM_SetCompare1

Function Name	**void TIM_SetCompare1 (*TIM_TypeDef** * **TIMx, uint32_t Compare1)**
Function Description	Sets the TIMx Capture Compare1 Register value.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **Compare1 :** specifies the Capture Compare1 register new value.
Return values	• None.
Notes	• None.

11.2.11.8 TIM_SetCompare2

Function Name	**void TIM_SetCompare2 (*TIM_TypeDef** * **TIMx, uint32_t Compare2)**
Function Description	Sets the TIMx Capture Compare2 Register value.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **Compare2 :** specifies the Capture Compare2 register new value.
Return values	• None.
Notes	• None.

11.2.11.9 TIM_SetCompare3

Function Name	**void TIM_SetCompare3 (*TIM_TypeDef** * **TIMx, uint32_t Compare3)**
Function Description	Sets the TIMx Capture Compare3 Register value.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **Compare3 :** specifies the Capture Compare3 register new value.
Return values	• None.
Notes	• None.

11.2.11.10 TIM_SetCompare4

Function Name	**void TIM_SetCompare4 (*TIM_TypeDef** * **TIMx, uint32_t Compare4)**
Function Description	Sets the TIMx Capture Compare4 Register value.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **Compare4 :** specifies the Capture Compare4 register new

value.

Return values	• None.
Notes	• None.

11.2.11.11 TIM_ForcedOC1Config

Function Name	**void TIM_ForcedOC1Config (*TIM_TypeDef** * TIMx, uint32_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 1 waveform to active or inactive level.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ForcedAction :** specifies the forced Action to be set to the output waveform. This parameter can be one of the following values: – ***TIM_ForcedAction_Active :*** Force active level on OC1REF – ***TIM_ForcedAction_InActive :*** Force inactive level on OC1REF.
Return values	• None.
Notes	• None.

11.2.11.12 TIM_ForcedOC2Config

Function Name	**void TIM_ForcedOC2Config (*TIM_TypeDef** * TIMx, uint32_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 2 waveform to active or inactive level.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ForcedAction :** specifies the forced Action to be set to the output waveform. This parameter can be one of the following values: – ***TIM_ForcedAction_Active :*** Force active level on OC2REF – ***TIM_ForcedAction_InActive :*** Force inactive level on OC2REF.
Return values	• None.
Notes	• None.

11.2.11.13 TIM_ForcedOC3Config

Function Name	**void TIM_ForcedOC3Config (*TIM_TypeDef** * **TIMx, uint32_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 3 waveform to active or inactive level.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ForcedAction :** specifies the forced Action to be set to the output waveform. This parameter can be one of the following values: – ***TIM_ForcedAction_Active :*** Force active level on OC3REF – ***TIM_ForcedAction_InActive :*** Force inactive level on OC3REF.
Return values	• None.
Notes	• None.

11.2.11.14 TIM_ForcedOC4Config

Function Name	**void TIM_ForcedOC4Config (*TIM_TypeDef** * **TIMx, uint32_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 4 waveform to active or inactive level.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ForcedAction :** specifies the forced Action to be set to the output waveform. This parameter can be one of the following values: – ***TIM_ForcedAction_Active :*** Force active level on OC4REF – ***TIM_ForcedAction_InActive :*** Force inactive level on OC4REF.
Return values	• None.
Notes	• None.

11.2.11.15 TIM_OC1PreloadConfig

Function Name	**void TIM_OC1PreloadConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on CCR1.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPreload :** new state of the TIMx peripheral Preload register This parameter can be one of the following values: – ***TIM_OCPreload_Enable :*** – ***TIM_OCPreload_Disable :***
Return values	• None.
Notes	• None.

11.2.11.16 TIM_OC2PreloadConfig

Function Name	**void TIM_OC2PreloadConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on CCR2.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPreload :** new state of the TIMx peripheral Preload register This parameter can be one of the following values: – ***TIM_OCPreload_Enable :*** – ***TIM_OCPreload_Disable :***
Return values	• None.
Notes	• None.

11.2.11.17 TIM_OC3PreloadConfig

Function Name	**void TIM_OC3PreloadConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on

CCR3.

Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPreload :** new state of the TIMx peripheral Preload register This parameter can be one of the following values: – ***TIM_OCPreload_Enable :*** – ***TIM_OCPreload_Disable :***
Return values	• None.
Notes	• None.

11.2.11.18 TIM_OC4PreloadConfig

Function Name	**void TIM_OC4PreloadConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on CCR4.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPreload :** new state of the TIMx peripheral Preload register This parameter can be one of the following values: – ***TIM_OCPreload_Enable :*** – ***TIM_OCPreload_Disable :***
Return values	• None.
Notes	• None.

11.2.11.19 TIM_OC1FastConfig

Function Name	**void TIM_OC1FastConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 1 Fast feature.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCFast :** new state of the Output Compare Fast Enable Bit. This parameter can be one of the following values: – ***TIM_OCFast_Enable :*** TIM output compare fast enable – ***TIM_OCFast_Disable :*** TIM output compare fast disable
Return values	• None.

Notes	• None.

11.2.11.20 TIM_OC2FastConfig

Function Name	**void TIM_OC2FastConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 2 Fast feature.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCFast :** new state of the Output Compare Fast Enable Bit. This parameter can be one of the following values: – ***TIM_OCFast_Enable :*** TIM output compare fast enable – ***TIM_OCFast_Disable :*** TIM output compare fast disable
Return values	• None.
Notes	• None.

11.2.11.21 TIM_OC3FastConfig

Function Name	**void TIM_OC3FastConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 3 Fast feature.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCFast :** new state of the Output Compare Fast Enable Bit. This parameter can be one of the following values: – ***TIM_OCFast_Enable :*** TIM output compare fast enable – ***TIM_OCFast_Disable :*** TIM output compare fast disable
Return values	• None.
Notes	• None.

11.2.11.22 TIM_OC4FastConfig

Function Name	**void TIM_OC4FastConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 4 Fast feature.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCFast :** new state of the Output Compare Fast Enable Bit. This parameter can be one of the following values: – ***TIM_OCFast_Enable :*** TIM output compare fast enable – ***TIM_OCFast_Disable :*** TIM output compare fast disable
Return values	• None.
Notes	• None.

11.2.11.23 TIM_OC1PolarityConfig

Function Name	**void TIM_OC1PolarityConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 1 polarity.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC1 Polarity This parmeter can be one of the following values: – ***TIM_OCPolarity_High :*** Output Compare active high – ***TIM_OCPolarity_Low :*** Output Compare active low
Return values	• None.
Notes	• None.

11.2.11.24 TIM_OC2PolarityConfig

Function Name	**void TIM_OC2PolarityConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPolarity)**

Function Description	Configures the TIMx channel 2 polarity.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC2 Polarity This parmeter can be one of the following values: – ***TIM_OCPolarity_High :*** Output Compare active high – ***TIM_OCPolarity_Low :*** Output Compare active low
Return values	• None.
Notes	• None.

11.2.11.25 TIM_OC3PolarityConfig

Function Name	**void TIM_OC3PolarityConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 3 polarity.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC3 Polarity This parmeter can be one of the following values: – ***TIM_OCPolarity_High :*** Output Compare active high – ***TIM_OCPolarity_Low :*** Output Compare active low
Return values	• None.
Notes	• None.

11.2.11.26 TIM_OC4PolarityConfig

Function Name	**void TIM_OC4PolarityConfig (*TIM_TypeDef** * TIMx, uint32_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 4 polarity.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC4 Polarity This parmeter can be one of the following values: – ***TIM_OCPolarity_High :*** Output Compare active high – ***TIM_OCPolarity_Low :*** Output Compare active low
Return values	• None.
Notes	• None.

11.2.11.27 TIM_CCxCmd

Function Name	**void TIM_CCxCmd (*TIM_TypeDef ** TIMx, uint32_t TIM_Channel, uint32_t TIM_CCx)**
Function Description	Enables or disables the TIM Capture Compare Channel x.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_Channel :** specifies the TIM Channel This parameter can be one of the following values: – ***TIM_Channel_1 :*** TIM Channel 1 – ***TIM_Channel_2 :*** TIM Channel 2 – ***TIM_Channel_3 :*** TIM Channel 3 – ***TIM_Channel_4 :*** TIM Channel 4 • **TIM_CCx :** specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_Enable or TIM_CCx_Disable.
Return values	• None.
Notes	• None.

11.2.12 Input Compare management functions

11.2.12.1 TIM_ICInit

Function Name	**void TIM_ICInit (*TIM_TypeDef ** TIMx, *TIM_ICInitTypeDef ** TIM_ICInitStruct)**
Function Description	Initializes the TIM peripheral according to the specified parameters in the TIM_ICInitStruct.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ICInitStruct :** pointer to a TIM_ICInitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

11.2.12.2 TIM_ICStructInit

Function Name	**void TIM_ICStructInit (*TIM_ICInitTypeDef** * **TIM_ICInitStruct)**
Function Description	Fills each TIM_ICInitStruct member with its default value.
Parameters	• **TIM_ICInitStruct :** : pointer to a TIM_ICInitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

11.2.12.3 TIM_PWMIConfig

Function Name	**void TIM_PWMIConfig (*TIM_TypeDef** * **TIMx, *TIM_ICInitTypeDef** * **TIM_ICInitStruct)**
Function Description	Configures the TIM peripheral according to the specified parameters in the TIM_ICInitStruct to measure an external PWM signal.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ICInitStruct :** pointer to a TIM_ICInitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

11.2.12.4 TIM_GetCapture1

Function Name	**uint32_t TIM_GetCapture1 (*TIM_TypeDef** * **TIMx)**
Function Description	Gets the TIMx Input Capture 1 value.

Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
Return values	• **Capture Compare 1 Register value.**
Notes	• None.

11.2.12.5 TIM_GetCapture2

Function Name	**uint32_t TIM_GetCapture2 (*TIM_TypeDef* * TIMx)**
Function Description	Gets the TIMx Input Capture 2 value.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
Return values	• **Capture Compare 2 Register value.**
Notes	• None.

11.2.12.6 TIM_GetCapture3

Function Name	**uint32_t TIM_GetCapture3 (*TIM_TypeDef* * TIMx)**
Function Description	Gets the TIMx Input Capture 3 value.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
Return values	• **Capture Compare 3 Register value.**
Notes	• None.

11.2.12.7 TIM_GetCapture4

Function Name	**uint32_t TIM_GetCapture4 (*TIM_TypeDef* * TIMx)**
Function Description	Gets the TIMx Input Capture 4 value.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral.

Return values	• **Capture Compare 4 Register value.**
Notes	• None.

11.2.12.8 TIM_SetIC1Prescaler

Function Name	**void TIM_SetIC1Prescaler (*TIM_TypeDef** * **TIMx, uint32_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 1 prescaler.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ICPSC :** specifies the Input Capture1 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 values	• None.
Notes	• None.

11.2.12.9 TIM_SetIC2Prescaler

Function Name	**void TIM_SetIC2Prescaler (*TIM_TypeDef** * **TIMx, uint32_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 2 prescaler.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ICPSC :** specifies the Input Capture2 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 values	• None.
Notes	• None.

11.2.12.10 TIM_SetIC3Prescaler

Function Name	**void TIM_SetIC3Prescaler (*TIM_TypeDef* * *TIM_TypeDef* * TIMx, uint32_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 3 prescaler.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ICPSC :** specifies the Input Capture3 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 values	• None.
Notes	• None.

11.2.12.11 TIM_SetIC4Prescaler

Function Name	**void TIM_SetIC4Prescaler (*TIM_TypeDef* * TIMx, uint32_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 4 prescaler.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_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 values	• None.
Notes	• None.

11.2.13 Interrupts and flags management functions

11.2.13.1 TIM_ITConfig

Function Name	**void TIM_ITConfig (*TIM_IT_TypeDef* * TIMx_IT, uint32_t TIM_ITRPT,FunctionalState NewState)**
Function Description	Enables or disables the specified TIM interrupts.
Parameters	• **TIMx_IT :** where x can be 1 or 2 to select the TIMx peripheral. • **TIM_ITRPT :** specifies the TIM interrupts sources to be enabled or disabled. This parameter can be any combination of the following values: – ***TIM_IT_Update :*** TIM update Interrupt source – ***TIM_IT_CC1 :*** TIM Capture Compare 1 Interrupt source – ***TIM_IT_CC2 :*** TIM Capture Compare 2 Interrupt source – ***TIM_IT_CC3 :*** TIM Capture Compare 3 Interrupt source – ***TIM_IT_CC4 :*** TIM Capture Compare 4 Interrupt source – ***TIM_IT_Trigger :*** TIM Trigger Interrupt source • **NewState :** new state of the TIM interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

11.2.13.2 TIM_GenerateEvent

Function Name	**void TIM_GenerateEvent (*TIM_TypeDef* * TIMx, uint32_t TIM_EventSource)**
Function Description	Configures the TIMx event to be generate by software.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral.

- **TIM_EventSource :** specifies the event source. This parameter can be one or more 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
- None.

Notes
- None.

11.2.13.3 TIM_GetITStatus

Function Name **ITStatus TIM_GetITStatus (*TIM_IT_TypeDef* * TIMx_IT, uint32_t TIM_ITRPT)**

Function Description Checks whether the TIM interrupt has occurred or not.

Parameters
- **TIMx_IT :** where x can be 1 or 2 to select the TIM peripheral.
- **TIM_ITRPT :** specifies the TIM interrupt source to check. This parameter can be one of the following values:
  - ***TIM_IT_Update :*** TIM update Interrupt source
  - ***TIM_IT_CC1 :*** TIM Capture Compare 1 Interrupt source
  - ***TIM_IT_CC2 :*** TIM Capture Compare 2 Interrupt source
  - ***TIM_IT_CC3 :*** TIM Capture Compare 3 Interrupt source
  - ***TIM_IT_CC4 :*** TIM Capture Compare 4 Interrupt source
  - ***TIM_IT_Trigger :*** TIM Trigger Interrupt source

Return values
- **The new state of the TIM_IT(SET or RESET).**

Notes
- None.

11.2.13.4 TIM_ClearITPendingBit

Function Name **void TIM_ClearITPendingBit (*TIM_IT_TypeDef* * TIMx_IT,**

uint32_t TIM_ITRPT)

Function Description	Clears the TIMx's interrupt pending bits.
Parameters	• **TIMx_IT :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_ITRPT :** specifies the pending bit to clear. This parameter can be any combination of the following values: – ***TIM_IT_Update :*** TIM1 update Interrupt source – ***TIM_IT_CC1 :*** TIM Capture Compare 1 Interrupt source – ***TIM_IT_CC2 :*** TIM Capture Compare 2 Interrupt source – ***TIM_IT_CC3 :*** TIM Capture Compare 3 Interrupt source – ***TIM_IT_CC4 :*** TIM Capture Compare 4 Interrupt source – ***TIM_IT_Trigger :*** TIM Trigger Interrupt source
Return values	• None.
Notes	• None.

11.2.14 Clock management functions

11.2.14.1 TIM_InternalClockConfig

Function Name	**void TIM_InternalClockConfig (** ***TIM_TypeDef** * **TIMx)**
Function Description	Configures the TIMx internal Clock.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
Return values	• None.
Notes	• None.

11.2.14.2 TIM_ITRxExternalClockConfig

Function Name	**void TIM_ITRxExternalClockConfig (** ***TIM_TypeDef** * ***TIM_TypeDef** * **TIMx, uint32_t TIM_InputTriggerSource)**
Function Description	Configures the TIMx Internal Trigger as External Clock.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_InputTriggerSource :** Trigger source. This parameter

can only be:
  - ***TIM_TS_ITR0 :*** Internal Trigger 0

Return values
- None.

Notes
- None.

11.2.14.3 TIM_TIxExternalClockConfig

Function Name
void TIM_TIxExternalClockConfig (*TIM_TypeDef TIMx, uint32_t TIM_TIxExternalCLKSource, uint32_t TIM_ICPolarity, uint32_t ICFilter)**

Function Description
Configures the TIMx Trigger as External Clock.

Parameters
- **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
- **TIM_TIxExternalCLKSource :** Trigger source. This parameter can be one of the following values:
  - ***TIM_TIxExternalCLK1Source_TI1ED :*** TI1 Edge Detector
  - ***TIM_TIxExternalCLK1Source_TI1 :*** Filtered Timer Input 1
  - ***TIM_TIxExternalCLK1Source_TI2 :*** Filtered Timer Input 2
- **TIM_ICPolarity :** specifies the TIx Polarity. This parameter can be one of the following values:
  - ***TIM_ICPolarity_Rising :***
  - ***TIM_ICPolarity_Falling :***
- **ICFilter :** : specifies the filter value. This parameter must be a value between 0x0 and 0xF.

Return values
- None.

Notes
- None.

11.2.14.4 TIM_ETRClockMode1Config

Function Name
void TIM_ETRClockMode1Config (*TIM_TypeDef TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)**

Function Description
Configures the External clock Mode1.

Parameters
- **TIMx :** where x can be 1 or 2 to select the TIM peripheral.

- **TIM_ExtTRGPrescaler :** The external Trigger Prescaler. This parameter can be one of the following values:
  - ***TIM_ExtTRGPSC_OFF :*** ETRP Prescaler OFF.
  - ***TIM_ExtTRGPSC_DIV2 :*** ETRP frequency divided by 2.
  - ***TIM_ExtTRGPSC_DIV4 :*** ETRP frequency divided by 4.
  - ***TIM_ExtTRGPSC_DIV8 :*** ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity :** The external Trigger Polarity. This parameter can be one of the following values:
  - ***TIM_ExtTRGPolarity_Inverted :*** active low or falling edge active.
  - ***TIM_ExtTRGPolarity_NonInverted :*** active high or rising edge active.
- **ExtTRGFilter :** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values
- None.

Notes
- None.

11.2.14.5 TIM_ETRClockMode2Config

Function Name **void TIM_ETRClockMode2Config (*TIM_TypeDef** TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)**

Function Description Configures the External clock Mode2.

Parameters
- **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
- **TIM_ExtTRGPrescaler :** The external Trigger Prescaler. This parameter can be one of the following values:
  - ***TIM_ExtTRGPSC_OFF :*** ETRP Prescaler OFF.
  - ***TIM_ExtTRGPSC_DIV2 :*** ETRP frequency divided by 2.
  - ***TIM_ExtTRGPSC_DIV4 :*** ETRP frequency divided by 4.
  - ***TIM_ExtTRGPSC_DIV8 :*** ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity :** The external Trigger Polarity. This parameter can be one of the following values:
  - ***TIM_ExtTRGPolarity_Inverted :*** active low or falling edge active.
  - ***TIM_ExtTRGPolarity_NonInverted :*** active high or rising edge active.
- **ExtTRGFilter :** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values
- None.

Notes
- None.

11.2.15 Synchronization management functions

11.2.15.1 TIM_SelectInputTrigger

Function Name	**void TIM_SelectInputTrigger (*TIM_TypeDef** * TIMx, uint32_t TIM_InputTriggerSource)**
Function Description	Selects the Input Trigger source.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_InputTriggerSource :** The Input Trigger source. This parameter can be one of the following values: – ***TIM_TS_ITR0 :*** Internal Trigger 0 – ***TIM_TS_TI1F_ED :*** TI1 Edge Detector – ***TIM_TS_TI1FP1 :*** Filtered Timer Input 1 – ***TIM_TS_TI2FP2 :*** Filtered Timer Input 2 – ***TIM_TS_ETRF :*** External Trigger input
Return values	• None.
Notes	• None.

11.2.15.2 TIM_SelectOutputTrigger

Function Name	**void TIM_SelectOutputTrigger (*TIM_TypeDef** * TIMx, uint32_t TIM_TRGOSource)**
Function Description	Selects the TIMx Trigger Output Mode.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_TRGOSource :** specifies the Trigger Output source. This paramter can be one of the following values:
Return values	• None.
Notes	• None.

11.2.15.3 TIM_SelectSlaveMode

Function Name	**void TIM_SelectSlaveMode (*TIM_TypeDef** * **TIMx, uint32_t TIM_SlaveMode)**
Function Description	Selects the TIMx Slave Mode.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_SlaveMode :** specifies the Timer Slave Mode. This paramter can be one of the following values: – ***TIM_SlaveMode_Reset :*** Rising edge of the selected trigger signal (TRGI) re-initializes the counter and triggers an update of the registers. – ***TIM_SlaveMode_Gated :*** The counter clock is enabled when the trigger signal (TRGI) is high. – ***TIM_SlaveMode_Trigger :*** The counter starts at a rising edge of the trigger TRGI. – ***TIM_SlaveMode_External1 :*** Rising edges of the selected trigger (TRGI) clock the counter.
Return values	• None.
Notes	• None.

11.2.15.4 TIM_SelectMasterSlaveMode

Function Name	**void TIM_SelectMasterSlaveMode (*TIM_TypeDef** * **TIMx, uint32_t TIM_MasterSlaveMode)**
Function Description	Sets or Resets the TIMx Master/Slave Mode.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_MasterSlaveMode :** specifies the Timer Master Slave Mode. This paramter can be one of the following values: – ***TIM_MasterSlaveMode_Enable :*** synchronization between the current timer and its slaves (through TRGO). – ***TIM_MasterSlaveMode_Disable :*** No action
Return values	• None.
Notes	• None.

11.2.15.5 TIM_ETRConfig

Function Name	**void TIM_ETRConfig (*TIM_TypeDef** * **TIMx, uint32_t**

TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)

Function Description: Configures the TIMx External Trigger (ETR).

Parameters:

- **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
- **TIM_ExtTRGPrescaler :** The external Trigger Prescaler. This parameter can be one of the following values:
  - ***TIM_ExtTRGPSC_OFF :*** ETRP Prescaler OFF.
  - ***TIM_ExtTRGPSC_DIV2 :*** ETRP frequency divided by 2.
  - ***TIM_ExtTRGPSC_DIV4 :*** ETRP frequency divided by 4.
  - ***TIM_ExtTRGPSC_DIV8 :*** ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity :** The external Trigger Polarity. This parameter can be one of the following values:
  - ***TIM_ExtTRGPolarity_Inverted :*** active low or falling edge active.
  - ***TIM_ExtTRGPolarity_NonInverted :*** active high or rising edge active.
- **ExtTRGFilter :** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values:

- None.

Notes:

- None.

11.2.16 Specific interface functions

11.2.16.1 TIM_EncoderInterfaceConfig

Function Name: **void TIM_EncoderInterfaceConfig (*TIM_TypeDef * TIM_TypeDef* * TIMx, uint32_t TIM_EncoderMode, uint32_t TIM_IC1Polarity, uint32_t TIM_IC2Polarity)**

Function Description: Configures the TIMx Encoder Interface.

Parameters:

- **TIMx :** where x can be 1 or 2 to select the TIM peripheral.
- **TIM_EncoderMode :** specifies the TIMx Encoder Mode. This parameter can be one of the following values:
  - ***TIM_EncoderMode_TI1 :*** Counter counts on TI1FP1 edge depending on TI2FP2 level.
  - ***TIM_EncoderMode_TI2 :*** Counter counts on TI2FP2 edge depending on TI1FP1 level.
  - ***TIM_EncoderMode_TI12 :*** Counter counts on both TI1FP1 and TI2FP2 edges depending on the level of the other input.
- **TIM_IC1Polarity :** specifies the IC1 Polarity This parmeter can be one of the following values:
  - ***TIM_ICPolarity_Falling :*** IC Falling edge.
  - ***TIM_ICPolarity_Rising :*** IC Rising edge.
- **TIM_IC2Polarity :** specifies the IC2 Polarity This parmeter

can be one of the following values:
- ***TIM_ICPolarity_Falling :*** IC Falling edge.
- ***TIM_ICPolarity_Rising :*** IC Rising edge.

Return values	• None.
Notes	• None.

11.2.16.2 TIM_SelectHallSensor

Function Name	**void TIM_SelectHallSensor (*TIM_TypeDef** TIMx,FunctionalState NewState)**
Function Description	Enables or disables the TIMx's Hall sensor interface.
Parameters	• **TIMx :** where x can be 1, 2 or 3 to select the TIM peripheral. • **NewState :** new state of the TIMx Hall sensor interface. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

11.2.17 Specific remapping management function

11.2.17.1 TIM_ClockMaskConfig

Function Name	**void TIM_ClockMaskConfig (*TIM_TypeDef** TIMx,FunctionalState NewState)**
Function Description	Selects the TIMx Extenal trigger used in external clock mode 2.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **NewState :** new state of the TIMx CLKMSKEN bit This parameter can be: ENABLE or DISABLE.
Return values	• **: None**
Notes	• None.

11.2.17.2 TIM_SelectExternalTriggerClock

Function Name	**void TIM_SelectExternalTriggerClock (*TIM_TypeDef** * **TIMx, uint32_t TIM_EXTRIGCLK)**
Function Description	Selects the TIMx Extenal trigger used in external clock mode 2.
Parameters	• **TIMx :** where x can be 1 or 2 to select the TIM peripheral. • **TIM_EXTRIGCLK :** specifies the TIM input reampping source. This parameter can be one of the following values: – ***TIM_EXTRIGPCLK :*** PCLK. – ***TIM_EXTRIG1KHCLK :*** calibrated 1 kHz clock. – ***TIM_EXTRIG32KHCLK :*** 32 kHz reference clock (if available). – ***TIM_EXTRIGTIMxCLK :*** TIMxCLK pin.
Return values	• **: None**
Notes	• None.

11.2.17.3 TIM_RemapCmd

Function Name	**void TIM_RemapCmd (*TIM_TypeDef** * **TIMx, uint32_t TIM_Remap,FunctionalState NewState)**
Function Description	Configures the TIM2 Remapping input Capabilities.
Parameters	• **TIMx :** where x can be 2 to select the TIM peripheral. • **TIM_Remap :** specifies the TIM input reampping source. This parameter can be one of the following values: – ***TIM_REMAPC1 :*** TIM2 Channel 1 is connected to GPIOA (PA0) or to GPIOB (PB1). – ***TIM_REMAPC2 :*** TIM2 Channel 2 is connected to GPIOA (PA3) or to GPIOB (PB2). – ***TIM_REMAPC3 :*** TIM2 Channel 3 is connected to GPIOA (PA1) or to GPIOB (PB3). – ***TIM_REMAPC4 :*** TIM2 Channel 4 is connected to GPIOA (PA2) or to GPIOB (PB4). • **NewState :** new state of the TIMx TIM2_OR_REMAPCy bit (y can be 1..4). This parameter can be: ENABLE or DISABLE.
Return values	• **: None**
Notes	• None.

11.3 TIM Firmware driver defines

11.3.1 TIM

TIM

TIM_Capture_Compare_state

- #define: ***TIM_CCx_Enable ((uint32_t)0x00000001)***

- #define: ***TIM_CCx_Disable ((uint32_t)0x00000000)***

TIM_Channel

- #define: ***TIM_Channel_1 ((uint32_t)0x00000000)***

- #define: ***TIM_Channel_2 ((uint32_t)0x00000004)***

- #define: ***TIM_Channel_3 ((uint32_t)0x00000008)***

- #define: ***TIM_Channel_4 ((uint32_t)0x0000000C)***

TIM_Counter_Mode

- #define: ***TIM_CounterMode_Up ((uint32_t)0x00000000)***

- #define: ***TIM_CounterMode_Down ((uint32_t)0x00000010)***

- #define: ***TIM_CounterMode_CenterAligned1 ((uint32_t)0x00000020)***

- #define: ***TIM_CounterMode_CenterAligned2 ((uint32_t)0x00000040)***

- #define: ***TIM_CounterMode_CenterAligned3 ((uint32_t)0x00000060)***

TIM_Encoder_Mode

- #define: ***TIM_EncoderMode_TI1 ((uint32_t)0x00000001)***

- #define: ***TIM_EncoderMode_TI2 ((uint32_t)0x00000002)***

- #define: ***TIM_EncoderMode_TI12 ((uint32_t)0x00000003)***

TIM_Event_Source

- #define: ***TIM_EventSource_Update ((uint32_t)0x00000001)***

- #define: ***TIM_EventSource_CC1 ((uint32_t)0x00000002)***

- #define: ***TIM_EventSource_CC2 ((uint32_t)0x00000004)***

- #define: ***TIM_EventSource_CC3 ((uint32_t)0x00000008)***

- #define: ***TIM_EventSource_CC4 ((uint32_t)0x00000010)***

- #define: ***TIM_EventSource_Trigger ((uint32_t)0x00000040)***

TIM_External_Trigger_Polarity

- #define: ***TIM_ExtTRGPolarity_Inverted ((uint32_t)0x00008000)***

- #define: ***TIM_ExtTRGPolarity_NonInverted ((uint32_t)0x00000000)***

TIM_External_Trigger_Prescaler

- #define: ***TIM_ExtTRGPSC_OFF ((uint32_t)0x00000000)***

- #define: ***TIM_ExtTRGPSC_DIV2 ((uint32_t)0x00001000)***

- #define: ***TIM_ExtTRGPSC_DIV4 ((uint32_t)0x00002000)***

- #define: ***TIM_ExtTRGPSC_DIV8 ((uint32_t)0x00003000)***

TIM_External_Trigger_Selection

- #define: ***TIM_EXTRIGPCLK ((uint32_t)0x00000000)***

- #define: ***TIM_EXTRIG1KHCLK ((uint32_t)0x00000001)***

- #define: ***TIM_EXTRIG32KHCLK ((uint32_t)0x00000002)***

- #define: ***TIM_EXTRIGTIMxCLK ((uint32_t)0x00000003)***

TIM_Forced_Action

- #define: ***TIM_ForcedAction_Active ((uint32_t)0x00000050)***

- #define: ***TIM_ForcedAction_InActive ((uint32_t)0x00000040)***

TIM_Input_Capture_Polarity

- #define: ***TIM_ICPolarity_Rising ((uint32_t)0x00000000)***

- #define: ***TIM_ICPolarity_Falling ((uint32_t)0x00000002)***

- #define: ***TIM_ICPolarity_BothEdge ((uint32_t)0x0000000A)***

TIM_Input_Capture_Prescaler

- #define: ***TIM_ICPSC_DIV1 ((uint32_t)0x00000000)***

Capture performed each time an edge is detected on the capture input.

- #define: ***TIM_ICPSC_DIV2 ((uint32_t)0x00000004)***

Capture performed once every 2 events.

- #define: ***TIM_ICPSC_DIV4 ((uint32_t)0x00000008)***

Capture performed once every 4 events.

- #define: ***TIM_ICPSC_DIV8 ((uint32_t)0x0000000C)***

Capture performed once every 8 events.

TIM_Input_Capture_Selection

- #define: ***TIM_ICSelection_DirectTI ((uint32_t)0x00000001)***

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

- #define: ***TIM_ICSelection_IndirectTI ((uint32_t)0x00000002)***

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively.

- #define: ***TIM_ICSelection_TRGI ((uint32_t)0x00000003)***

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC.

TIM_Internal_Trigger_Selection

- #define: ***TIM_TS_ITR0 ((uint32_t)0x00000000)***

- #define: ***TIM_TS_TI1F_ED ((uint32_t)0x00000040)***

- #define: ***TIM_TS_TI1FP1 ((uint32_t)0x00000050)***

- #define: ***TIM_TS_TI2FP2 ((uint32_t)0x00000060)***

- #define: ***TIM_TS_ETRF ((uint32_t)0x00000070)***

TIM_interrupt_sources

- #define: ***TIM_IT_Update ((uint32_t)0x00000001)***

- #define: ***TIM_IT_CC1 ((uint32_t)0x00000002)***

- #define: ***TIM_IT_CC2 ((uint32_t)0x00000004)***

- #define: ***TIM_IT_CC3 ((uint32_t)0x00000008)***

- #define: ***TIM_IT_CC4 ((uint32_t)0x00000010)***

- #define: ***TIM_IT_Trigger ((uint32_t)0x00000040)***

TIM_Master_Slave_Mode

- #define: ***TIM_MasterSlaveMode_Enable ((uint32_t)0x00000080)***

- #define: ***TIM_MasterSlaveMode_Disable ((uint32_t)0x00000000)***

TIM_OCReferenceClear

- #define: ***TIM_OCReferenceClear_ETRF ((uint32_t)0x00000008)***

- #define: ***TIM_OCReferenceClear_OCREFCLR ((uint32_t)0x00000000)***

TIM_One_Pulse_Mode

- #define: ***TIM_OPMode_Single ((uint32_t)0x00000008)***

- #define: ***TIM_OPMode_Repetitive ((uint32_t)0x00000000)***

TIM_Output_Compare_and_PWM_modes

- #define: ***TIM_OCMode_Timing ((uint32_t)0x00000000)***

- #define: ***TIM_OCMode_Active ((uint32_t)0x00000010)***

- #define: ***TIM_OCMode_Inactive ((uint32_t)0x00000020)***

- #define: ***TIM_OCMode_Toggle ((uint32_t)0x00000030)***

- #define: ***TIM_OCMode_PWM1 ((uint32_t)0x00000060)***

- #define: ***TIM_OCMode_PWM2 ((uint32_t)0x00000070)***

TIM_Output_Compare_Fast_State

- #define: ***TIM_OCFast_Enable ((uint32_t)0x00000004)***

- #define: ***TIM_OCFast_Disable ((uint32_t)0x00000000)***

TIM_Output_Compare_Polarity

- #define: ***TIM_OCPolarity_High ((uint32_t)0x00000000)***

- #define: ***TIM_OCPolarity_Low ((uint32_t)0x00000002)***

TIM_Output_Compare_Preload_State

- #define: ***TIM_OCPreload_Enable ((uint32_t)0x00000008)***

- #define: ***TIM_OCPreload_Disable ((uint32_t)0x00000000)***

TIM_Output_Compare_state

- #define: ***TIM_OutputState_Disable ((uint32_t)0x00000000)***

- #define: ***TIM_OutputState_Enable ((uint32_t)0x00000001)***

TIM_Prescaler_Reload_Mode

- #define: ***TIM_PSCReloadMode_Update ((uint32_t)0x00000000)***

- #define: ***TIM_PSCReloadMode_Immediate ((uint32_t)0x00000001)***

TIM_Remap

- #define: ***TIM_REMAPC1 ((uint32_t)0x00000010)***

- #define: ***TIM_REMAPC2 ((uint32_t)0x00000020)***

- #define: ***TIM_REMAPC3 ((uint32_t)0x00000040)***

- #define: ***TIM_REMAPC4 ((uint32_t)0x00000080)***

TIM_Slave_Mode

- #define: ***TIM_SlaveMode_Reset ((uint32_t)0x00000004)***

- #define: ***TIM_SlaveMode_Gated ((uint32_t)0x00000005)***

- #define: ***TIM_SlaveMode_Trigger ((uint32_t)0x00000006)***

- #define: ***TIM_SlaveMode_External1 ((uint32_t)0x00000007)***

TIM_TIx_External_Clock_Source

- #define: ***TIM_TIxExternalCLK1Source_TI1 ((uint32_t)0x00000050)***

- #define: ***TIM_TIxExternalCLK1Source_TI2 ((uint32_t)0x00000060)***

- #define: ***TIM_TIxExternalCLK1Source_TI1ED ((uint32_t)0x00000040)***

TIM_Trigger_Output_Source

- #define: ***TIM_TRGOSource_Reset ((uint32_t)0x00000000)***

- #define: ***TIM_TRGOSource_Enable ((uint32_t)0x00000010)***

- #define: ***TIM_TRGOSource_Update ((uint32_t)0x00000020)***

- #define: ***TIM_TRGOSource_OC1 ((uint32_t)0x00000030)***

- #define: ***TIM_TRGOSource_OC1Ref ((uint32_t)0x00000040)***

- #define: ***TIM_TRGOSource_OC2Ref ((uint32_t)0x00000050)***

- #define: ***TIM_TRGOSource_OC3Ref ((uint32_t)0x00000060)***

- #define: ***TIM_TRGOSource_OC4Ref ((uint32_t)0x00000070)***

TIM_Update_Source

- #define: ***TIM_UpdateSource_Global ((uint32_t)0x00000000)***

Source of update is the counter overflow/underflow or the setting of UG bit, or an update generation through the slave mode controller.

- #define: ***TIM_UpdateSource_Regular ((uint32_t)0x00000001)***

Source of update is counter overflow/underflow.

12 Sleep timer (SLPTIM)

12.1 SLPTIM Firmware driver registers structures

12.1.1 SLPTIM_InitTypeDef

SLPTIM_InitTypeDef is defined in the stm32w108xx_slptim.h

Data Fields

- ***uint32_t SLPTIM_Clock***
- ***uint32_t SLPTIM_Prescaler***
- ***uint32_t SLPTIM_DebugMode***
- ***uint32_t SLPTIM_CounterMode***

Field Documentation

- ***uint32_t SLPTIM_InitTypeDef::SLPTIM_Clock***
  - Specifies the clock to be used. This parameter must be a value of ***SLPTIM_Clocks_Select***
- ***uint32_t SLPTIM_InitTypeDef::SLPTIM_Prescaler***
  - Specifies the prescaler value used to divide the TIM clock. This parameter can be a value of ***SLPTIM_Clock_Division***
- ***uint32_t SLPTIM_InitTypeDef::SLPTIM_DebugMode***
  - Specifies whether the timer is running or paused during debug mode. This parameter must be a value of ***SLPTIM_Debug_Mode***
- ***uint32_t SLPTIM_InitTypeDef::SLPTIM_CounterMode***
  - Specifies the counter mode. This parameter can be a value of ***SLPTIM_Counter_Mode***

12.1.2 SLPTMR_TypeDef

SLPTMR_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t CNTH***
- ***__IO uint32_t CNTL***
- ***__IO uint32_t CMPAH***
- ***__IO uint32_t CMPAL***
- ***__IO uint32_t CMPBH***
- ***__IO uint32_t CMPBL***
- ***uint32_t RESERVED0***
- ***__IO uint32_t ISR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t IFR***
- ***uint32_t RESERVED2***

- ***__IO uint32_t IER***

Field Documentation

- ***__IO uint32_t SLPTMR_TypeDef::CR***
  - SLPTMR configuration register, Address offset 0x600C
- ***__IO uint32_t SLPTMR_TypeDef::CNTH***
  - SLPTMR counter high register, Address offset 0x6010
- ***__IO uint32_t SLPTMR_TypeDef::CNTL***
  - SLPTMR counter high register, Address offset 0x6014
- ***__IO uint32_t SLPTMR_TypeDef::CMPAH***
  - SLPTMR compare A high register, Address offset 0x6018
- ***__IO uint32_t SLPTMR_TypeDef::CMPAL***
  - SLPTMR compare A low register, Address offset 0x601C
- ***__IO uint32_t SLPTMR_TypeDef::CMPBH***
  - SLPTMR compare B high register, Address offset 0x6020
- ***__IO uint32_t SLPTMR_TypeDef::CMPBL***
  - SLPTMR compare B low register, Address offset 0x6024
- ***uint32_t SLPTMR_TypeDef::RESERVED0[4091]***
  - Reserved
- ***__IO uint32_t SLPTMR_TypeDef::ISR***
  - SLPTMR interrupt status register, Address offset 0xA014
- ***uint32_t SLPTMR_TypeDef::RESERVED1[2]***
  - Reserved
- ***__IO uint32_t SLPTMR_TypeDef::IFR***
  - SLPTMR force interrupts register, Address offset 0xA020
- ***uint32_t SLPTMR_TypeDef::RESERVED2[12]***
  - Reserved
- ***__IO uint32_t SLPTMR_TypeDef::IER***
  - SLPTMR interrupt enable register, Address offset 0xA054

12.2 SLPTIM Firmware driver API description

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

12.2.1 SLPTIM features

The sleep timer is dedicated to system timing and waking from sleep at specific times.

The sleep timer can use either the calibrated 1 kHz reference(CLK1K), or the 32 kHz crystal clock (CLK32K). The default clock source is the internal 1 kHz clock.

The sleep timer has a prescaler that allows for very long periods of sleep to be timed.

The timer provides two compare outputs and wrap detection, all of which can be used to generate an interrupt or a wake up event.

The sleep timer is paused when the debugger halts the ARM Cortex-M3.

12.2.2 How to use this driver

This driver describes the functions allowing configuring and programming the Sleep Timer. These functions are split in 2 groups:

1. SLPTIM management functions: this group includes all needed functions to configure the Sleep Timer.
   - Enable/Disable the counter.
   - Get counter.
   - Select clock to be used as reference.
   - Set/Get compare (A or B) values.
2. Interrupts and flags management functions: this group includes all needed functions to manage interrupts:
   - Enables or disables the specified SLPTIM interrupts.
   - Checks whether the specified SLPTIM flag is set or not.
   - Clears the specified SLPTIM flag.

12.2.3 SLPTIM management functions

To use the Sleep Timer:

1. Fill the SLPTIM_InitStruct with the desired parameters. This must be done while the sleep timer is disabled.
2. Call the SLPTIM_Cmd(ENABLE) function to enable the TIM counter.
3. Enable the clock to be used as reference by calling SLPTIM_ClockConfig() function.

All other functions can be used seperatly to set compareA or compareB value, to get counter value...

- ***SLPTIM_DeInit()***
- ***SLPTIM_Init()***
- ***SLPTIM_StructInit()***
- ***SLPTIM_Cmd()***
- ***SLPTIM_SetCompareA()***
- ***SLPTIM_SetCompareB()***
- ***SLPTIM_GetCounter()***
- ***SLPTIM_GetCounterHigh()***
- ***SLPTIM_GetCounterLow()***

12.2.4 Interrupts and flags management functions

- ***SLPTIM_ForceIT()***
- ***SLPTIM_ITConfig()***
- ***SLPTIM_GetFlagStatus()***
- ***SLPTIM_ClearFlag()***
- ***SLPTIM_GetITStatus()***
- ***SLPTIM_ClearITPendingBit()***

12.2.5 SLPTIM management functions

12.2.5.1 SLPTIM_DeInit

Function Name	**void SLPTIM_DeInit (void)**
Function Description	Deinitializes the SLPTIM peripheral registers to their default reset values.
Return values	• None.
Notes	• None.

12.2.5.2 SLPTIM_Init

Function Name	**void SLPTIM_Init (*SLPTIM_InitTypeDef** * **SLPTIM_InitStruct)**
Function Description	Initializes the SLPTIM Time peripheral according to the specified parameters in the SLPTIM_InitStruct.
Parameters	• **SLPTIM_InitStruct :** pointer to a SLPTIM_InitTypeDef structure that contains the configuration information for the specified TIM peripheral.
Return values	• None.
Notes	• None.

12.2.5.3 SLPTIM_StructInit

Function Name	**void SLPTIM_StructInit (*SLPTIM_InitTypeDef** * **SLPTIM_InitStruct)**
Function Description	Fills each SLPTIM_InitStruct member with its default value.
Parameters	• **SLPTIM_InitStruct :** : pointer to a SLPTIM_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

12.2.5.4 SLPTIM_Cmd

Function Name	**void SLPTIM_Cmd (FunctionalState NewState)**
Function Description	Enables or disables the Sleep Timer.
Parameters	• **NewState :** new state of the Sleep Timer. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

12.2.5.5 SLPTIM_SetCompareA

Function Name	**void SLPTIM_SetCompareA (uint32_t CompareA)**
Function Description	Sets the SLPTIM Compare A Register value.
Parameters	• **CompareA :** specifies the Compare A register new value.
Return values	• None.
Notes	• None.

12.2.5.6 SLPTIM_SetCompareB

Function Name	**void SLPTIM_SetCompareB (uint32_t CompareB)**
Function Description	Sets the SLPTIM Compare B Register value.
Parameters	• **CompareB :** specifies the Compare B register new value.
Return values	• None.
Notes	• None.

12.2.5.7 SLPTIM_GetCounter

Function Name	**uint32_t SLPTIM_GetCounter (void)**
Function Description	Gets the SLPTIM Counter value.
Return values	• **Counter Register value.**
Notes	• None.

12.2.5.8 SLPTIM_GetCounterHigh

Function Name	**uint32_t SLPTIM_GetCounterHigh (void)**
Function Description	Gets the SLPTIM Counter high value.
Return values	• **Counter Register high value.**
Notes	• None.

12.2.5.9 SLPTIM_GetCounterLow

Function Name	**uint32_t SLPTIM_GetCounterLow (void)**
Function Description	Gets the SLPTIM Counter low value.
Return values	• **Counter Register low value.**
Notes	• None.

12.2.6 Interrupts and flags management functions

12.2.6.1 SLPTIM_ForceIT

Function Name	**void SLPTIM_ForceIT (uint32_t SLPTIM_IT)**

Function Description	Forces the specified SLPTIM interrupts.
Parameters	• **SLPTIM_IT :** specifies the SLPTIM interrupts sources to be generated. This parameter can be any combination of the following values: – ***SLPTIM_IT_WRAP :*** Sleep timer overflow – ***SLPTIM_IT_CMPA :*** Sleep timer compare A – ***SLPTIM_IT_CMPB :*** Sleep timer compare B
Return values	• None.
Notes	• None.

12.2.6.2 SLPTIM_ITConfig

Function Name	**void SLPTIM_ITConfig (uint32_t SLPTIM_IT,FunctionalState NewState)**
Function Description	Enables or disables the specified SLPTIM interrupts.
Parameters	• **SLPTIM_IT :** specifies the SLPTIM interrupts sources to be enabled or disabled. This parameter can be any combination of the following values: – ***SLPTIM_IT_WRAP :*** Sleep timer overflow – ***SLPTIM_IT_CMPA :*** Sleep timer compare A – ***SLPTIM_IT_CMPB :*** Sleep timer compare B • **NewState :** new state of the SLPTIM interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

12.2.6.3 SLPTIM_GetFlagStatus

Function Name	**FlagStatus SLPTIM_GetFlagStatus (uint32_t SLPTIM_FLAG)**
Function Description	Checks whether the specified SLPTIM flag is set or not.
Parameters	• **SLPTIM_FLAG :** specifies the flag to check. This parameter can be one of the following values: – ***SLPTIM_FLAG_WRAP :*** Sleep timer overflow – ***SLPTIM_FLAG_CMPA :*** Sleep timer compare A – ***SLPTIM_FLAG_CMPB :*** Sleep timer compare B

Return values	• **The new state of SLPTIM_FLAG (SET or RESET).**
Notes	•

12.2.6.4 SLPTIM_ClearFlag

Function Name	**void SLPTIM_ClearFlag (uint32_t SLPTIM_FLAG)**
Function Description	Clears the specified SLPTIM flag.
Parameters	• **SLPTIM_FLAG :** specifies the flag to clear. This parameter can be one of the following values: – ***SLPTIM_FLAG_WRAP :*** Sleep timer overflow – ***SLPTIM_FLAG_CMPA :*** Sleep timer compare A – ***SLPTIM_FLAG_CMPB :*** Sleep timer compare B
Return values	• **The new state of SLPTIM_FLAG (SET or RESET).**
Notes	•

12.2.6.5 SLPTIM_GetITStatus

Function Name	**ITStatus SLPTIM_GetITStatus (uint32_t SLPTIM_IT)**
Function Description	Checks whether the specified SLPTMR pending interrupt has occurred or not.
Parameters	• **SLPTIM_IT :** specifies the flag to check. This parameter can be one of the following values: This parameter can be one of the following values: – ***SLPTIM_FLAG_WRAP :*** Sleep timer overflow – ***SLPTIM_FLAG_CMPA :*** Sleep timer compare A – ***SLPTIM_FLAG_CMPB :*** Sleep timer compare B
Return values	• **The new state of SLPTIM_IT (SET or RESET).**
Notes	• None.

12.2.6.6 SLPTIM_ClearITPendingBit

Function Name	**void SLPTIM_ClearITPendingBit (uint32_t SLPTIM_IT)**
Function Description	Clears the specified SLPTIM flag.
Parameters	• **SLPTIM_IT :** specifies the flag to clear. This parameter can be one of the following values: – ***SLPTIM_FLAG_WRAP :*** Sleep timer overflow – ***SLPTIM_FLAG_CMPA :*** Sleep timer compare A – ***SLPTIM_FLAG_CMPB :*** Sleep timer compare B
Return values	• **The new state of SLPTIM_FLAG (SET or RESET).**
Notes	•

12.3 SLPTIM Firmware driver defines

12.3.1 SLPTIM

SLPTIM

SLPTIM_Clocks

- #define: ***SLPTIM_CLK_32KH ((uint32_t)CLK_SLEEPCR_LSEEN)***

- #define: ***SLPTIM_CLK_10KH ((uint32_t)CLK_SLEEPCR_LSI10KEN)***

SLPTIM_Clocks_Select

- #define: ***SLPTIM_CLK_32KHZ ((uint32_t)SLPTMR_CR_CLKSEL)***

- #define: ***SLPTIM_CLK_1KHZ ((uint32_t)0x00000000)***

SLPTIM_Clock_Division

- #define: ***SLPTIM_CLK_DIV0 ((uint32_t)0x00000000)***

- #define: ***SLPTIM_CLK_DIV1 ((uint32_t)0x00000010)***

- #define: ***SLPTIM_CLK_DIV2 ((uint32_t)0x00000020)***
- #define: ***SLPTIM_CLK_DIV3 ((uint32_t)0x00000030)***
- #define: ***SLPTIM_CLK_DIV4 ((uint32_t)0x00000040)***
- #define: ***SLPTIM_CLK_DIV5 ((uint32_t)0x00000050)***
- #define: ***SLPTIM_CLK_DIV6 ((uint32_t)0x00000060)***
- #define: ***SLPTIM_CLK_DIV7 ((uint32_t)0x00000070)***
- #define: ***SLPTIM_CLK_DIV8 ((uint32_t)0x00000080)***
- #define: ***SLPTIM_CLK_DIV9 ((uint32_t)0x00000090)***
- #define: ***SLPTIM_CLK_DIV10 ((uint32_t)0x000000A0)***
- #define: ***SLPTIM_CLK_DIV11 ((uint32_t)0x000000B0)***
- #define: ***SLPTIM_CLK_DIV12 ((uint32_t)0x000000C0)***
- #define: ***SLPTIM_CLK_DIV13 ((uint32_t)0x000000D0)***

- #define: ***SLPTIM_CLK_DIV14 ((uint32_t)0x000000E0)***

- #define: ***SLPTIM_CLK_DIV15 ((uint32_t)0x000000F0)***

SLPTIM_Counter_Mode

- #define: ***SLPTIM_CountForward ((uint32_t)0x00000000)***

- #define: ***SLPTIM_CountBackward ((uint32_t)SLPTMR_CR_REVERSE)***

SLPTIM_Debug_Mode

- #define: ***SLPTIM_DBGRUN ((uint32_t)0x00000000)***

- #define: ***SLPTIM_DBGPAUSE ((uint32_t)SLPTMR_CR_DBGP)***

SLPTIM_flags

- #define: ***SLPTIM_FLAG_WRAP ((uint32_t)SLPTMR_ISR_WRAP)***

- #define: ***SLPTIM_FLAG_CMPA ((uint32_t)SLPTMR_ISR_CMPA)***

- #define: ***SLPTIM_FLAG_CMPB ((uint32_t)SLPTMR_ISR_CMPB)***

SLPTIM_interrupt_sources

- #define: ***SLPTIM_IT_WRAP ((uint32_t)SLPTMR_IER_WRAP)***

- #define: ***SLPTIM_IT_CMPA ((uint32_t)SLPTMR_IER_CMPA)***

- #define: ***SLPTIM_IT_CMPB ((uint32_t)SLPTMR_IER_CMPB)***

13 Watchdog timer (WDG)

13.1 WDG Firmware driver registers structures

13.1.1 WDG_TypeDef

WDG_TypeDef is defined in the stm32w108xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t KR***
- ***__IO uint32_t KICKSR***

Field Documentation

- ***__IO uint32_t WDG_TypeDef::CR***
  - WDG configuration register, Address offset 0x6000
- ***__IO uint32_t WDG_TypeDef::KR***
  - WDG key register, Address offset 0x6004
- ***__IO uint32_t WDG_TypeDef::KICKSR***
  - WDG kick-start register, Address offset 0x6008

13.2 WDG Firmware driver API description

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

13.2.1 WDG features

The watchdog timer uses the calibrated 1 kHz clock (CLK1K) as its reference and provides a nominal 2.048 s timeout. A low water mark interrupt occurs at 1.760 s and triggers an NMI to the Cortex-M3 NVIC as an early warning. When enabled, periodically reset the watchdog timer before it expires.

By default, the WDG is disabled at power up of the always-on power domain.

The watchdog timer can be paused when the debugger halts the core.

13.2.2 How to use this driver

This driver allows to use WDG peripheral.

Start the WDG using WDG_Cmd() function.

Restart the WDG timer using WDG_ReloadCounter() function.

Specifies the staus of WDG timer during debug mode using WDG_DebugConfig() function.

13.2.3 WDG activation function

- *WDG_DeInit()*
- *WDG_ReloadCounter()*
- *WDG_Cmd()*
- *WDG_DebugConfig()*
- *WDG_GetStatus()*

13.2.4 WDG activation function

13.2.4.1 WDG_DeInit

Function Name	**void WDG_DeInit (void)**
Function Description	Deinitializes the WDG peripheral registers to their default reset values.
Parameters	• None.
Return values	• None.
Notes	• None.

13.2.4.2 WDG_ReloadCounter

Function Name	**void WDG_ReloadCounter (void)**
Function Description	Reloads WDG counter with value defined in the restart register.
Parameters	• None.
Return values	• None.
Notes	• None.

13.2.4.3 WDG_Cmd

Function Name	**void WDG_Cmd (FunctionalState NewState)**
Function Description	Enables/Disables WDG.

Parameters	• **NewState :** new state of the WDG timer. This parameter can be: ENABLE or DISABLE.
Return values	• **None.**
Notes	• None.

13.2.4.4 WDG_DebugConfig

Function Name	**void WDG_DebugConfig (uint32_t DBG_STATUS)**
Function Description	Status of WDG timer while in debug mode.
Parameters	• **DBG_STATUS :** specifies the staus of WDG timer during debug mode. This parameter can be one of the following values: – ***WDG_DBG_RUN :*** The timer continues working in Debug mode. – ***WDG_DBG_PAUSE :*** The timer is paused in Debug mode when the CPU is halted.
Return values	• **None.**
Notes	• None.

13.2.4.5 WDG_GetStatus

Function Name	**FunctionalState WDG_GetStatus (void)**
Function Description	Returns the status of WDG timer.
Parameters	• None.
Return values	• None.
Notes	• None.

13.3 WDG Firmware driver defines

13.3.1 WDG

WDG

WDG_DebugStatus

- #define: ***WDG_DBG_RUN ((uint32_t)0x00000000)***

- #define: ***WDG_DBG_PAUSE ((uint32_t)0x00000400)***

14 Revision history

Table 10: Revision history

Date	Revision	Changes
19-Oct-2012	1	Initial release.

Please Read Carefully

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at anytime, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately voidany warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com