UM1581
User manual

Description of STM32F30xx/31xx Standard Peripheral Library

Introduction

The STM32F30xx and STM32F31xx Standard Peripheral Library covers 3 abstraction levels, and includes:

- A complete register address mapping with all bits, bitfields 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 STM32F30xx and STM32F31xx Standard Peripheral Library will be referred to as STM32F30xx Library throughout the document, unless otherwise specified.

Table 1: Applicable products

Type	Part numbers
Microcontrollers	STM32F30xx and STM32F31xx

Contents

1 STM32F30xx 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

The table below describes the acronyms used in this document.

Table 2: List of abbreviations

Acronym	Peripheral / unit
ADC	Analog-to-digital converter
CAN	Controller area network
COMP	Analog comparators
CRC	CRC calculation unit
DAC	Digital to analog converter
DBGMCU	Debug MCU
DMA	DMA controller
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/O
I^2C	Inter-integrated circuit
I^2S	Inter-integrated sound
IWDG	Independent watchdog
NVIC	Nested vectored interrupt controller
OPAMP	Operational amplifier
PWR	Power control
RCC	Reset and clock controller
RTC	Real-time clock
SPI	Serial peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
USART	Universal synchronous asynchronous receiver transmitter
WWDG	Window 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 ‘stm32f30x_’, for example stm32f30x_conf.h even if they are valid both for STM32F30xx and STM32F31xx microcontrollers.
- 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 STM32F30xx reference manual document.
- Peripheral registers are declared in the **PPP_TypeDef** structure (e.g. **ADC_TypeDef**) in stm32f30x.h file.
- The 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 **USART_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. **TIM_Init**.
- The functions used to reset the PPP peripheral registers to their default values are named **PPP_DeInit**, e.g. **TIM_DeInit**.
- The functions used to fill the **PPP_InitTypeDef** structure with the reset values of each member are named **PPP_StructInit**, e.g. **USART_StructInit**.
- The functions used to enable or disable the specified PPP peripheral are named **PPP_Cmd**, for example **USART_Cmd**.
- The functions used to enable or disable an interrupt source for the specified PPP peripheral are named **PPP_ITConfig**, e.g. **RCC_ITConfig**.
- The functions used to enable or disable the DMA interface for the specified PPP peripheral are named **PPP_DMAConfig**, e.g. **TIM_DMAConfig**.
- The functions used to configure a peripheral function always end with the string ‘Config’, for example **GPIO_PinAFConfig**.
- 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 **I2C_ClearFlag**.
- The functions used to check whether the specified PPP interrupt has occurred or not are named **PPP_GetITStatus**, e.g. **I2C_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 stm32f30x.h

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

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

typedef enum {
  ERROR = 0,
  SUCCESS = !ERROR
}
ErrorStatus;
```

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

stm32f30x.h contains the definition of all peripheral register structures. The example below illustrates the SPI register structure declaration:

```
"/*------------------ Serial Peripheral Interface ---------------*/
typedef struct
{
  __IO uint16_t CR1;      /*!< SPI control register 1 (not used in
I2S mode), Address offset: 0x00 */
  uint16_t      RESERVED0;/*!< Reserved, 0x02
*/
  __IO uint16_t CR2;      /*!< SPI control register 2, Address
offset: 0x04 */
  uint16_t      RESERVED1;/*!< Reserved, 0x06
*/
  __IO uint16_t SR;       /*!< SPI status register, Address offset:
0x08    */
  uint16_t      RESERVED2;/*!< Reserved, 0x0A
*/
  __IO uint16_t DR;       /*!< SPI data register,Address offset:
0x0C       */
  uint16_t      RESERVED3;/*!< Reserved, 0x0E
*/
  __IO uint16_t CRCPR;    /*!< SPI CRC polynomial register (not
used in I2S mode), Address offset: 0x10 */
  uint16_t      RESERVED4;/*!< Reserved, 0x12
*/
  __IO uint16_t RXCRCR;   /*!< SPI RX CRC register (not used in I2S
mode),Address offset: 0x14 */
  uint16_t      RESERVED5;/*!< Reserved, 0x16 */
```

```
  __IO uint16_t TXCRCR;   /*!< SPI TX CRC register (not used in I2S
mode),Address offset: 0x18 */
  uint16_t      RESERVED6;/*!< Reserved, 0x1A
*/
  __IO uint16_t I2SCFGR;  /*!< SPI_I2S configuration register,
Address offset: 0x1C */
  uint16_t      RESERVED7;/*!< Reserved, 0x1E
*/
  __IO uint16_t I2SPR;    /*!< SPI_I2S prescaler register, Address
offset: 0x20 */
  uint16_t      RESERVED8;/*!< Reserved, 0x22
*/
} SPI_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 stm32f30x_ppp.h.

Peripheral declaration

All peripherals are declared in stm32f30x.h. The following example shows the declaration of the SPI peripheral:

```
...
 /*!< Peripheral base address in the alias region */
#define PERIPH_BASE           ((uint32_t)0x40000000)
...
/*!< Peripheral memory map */
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x00010000)
#define AHB1PERIPH_BASE       (PERIPH_BASE + 0x00020000)
#define AHB2PERIPH_BASE       (PERIPH_BASE + 0x08000000)
...
/*!< APB1 peripherals base address */
#define SPI2_BASE             (APB1PERIPH_BASE + 0x3800)
#define SPI3_BASE             (APB1PERIPH_BASE + 0x3C00)
...
/*!< APB2 peripherals base address */
#define SPI1_BASE             (APB2PERIPH_BASE + 0x3000)
...
/*!< Peripheral Declaration */
...
#define SPI2                ((SPI_TypeDef *) SPI2_BASE)
#define SPI3                ((SPI_TypeDef *) SPI3_BASE)
...
#define SPI1                ((SPI_TypeDef *) SPI1_BASE)
```

SPIx_BASE is the base address of a specific SPI and SPIx is a pointer to a register structure that refers to a specific SPI.

The peripheral registers are accessed as follows:

```
SPI1->CR1 = 0x0001;
```

Peripheral registers bits

All the peripheral registers bits are defined as constants in the stm32f30x.h file. They are defined as acronyms written in upper-case into the form:

```
PPP_<register_name>_<bit_name>
```

Example:

```
#define SPI_CR1_CPHA  ((uint16_t)0x0001) /*!< Clock Phase */
#define SPI_CR1_CPOL  ((uint16_t)0x0002) /*!< Clock Polarity */
#define SPI_CR1_MSTR  ((uint16_t)0x0004) /*!< Master Selection */
#define SPI_CR1_BR    ((uint16_t)0x0038) /*!< BR[2:0] bits (Baud
Rate Control) */
#define SPI_CR1_BR_0  ((uint16_t)0x0008) /*!< Bit 0 */
#define SPI_CR1_BR_1  ((uint16_t)0x0010) /*!< Bit 1 */
#define SPI_CR1_BR_2  ((uint16_t)0x0020) /*!< Bit 2 */
```

1.1.4 Bit-Banding

The Cortex-M4 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 STM32F30/F31xx peripheral registers are mapped in a bit-band region. This feature is consequently intensively used functions performing 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.

Example of implementation

The following example shows how to map the PLLON[24] bit of RCC_CR register in the alias region:

```
...
/*!< Peripheral base address in the alias region */
#define PERIPH_BASE           ((uint32_t)0x40000000)
...
/*!< Peripheral base address in the bit-band region */
#define PERIPH_BB_BASE    ((uint32_t)0x42000000)
...
```

```
/* ------------ RCC registers bit address in the alias region -----
------ */
#define RCC_OFFSET                (RCC_BASE - PERIPH_BASE)
...
/* --- CR Register ---*/
/* Alias word address of PLLON bit */
#define CR_OFFSET       (RCC_OFFSET + 0x00)
#define PLLON_BitNumber 0x18
#define CR_PLLON_BB      (PERIPH_BB_BASE + (CR_OFFSET * 32) +
(PLLON_BitNumber * 4))
```

To code a function which enables/disables the PLL, the usual method is the following:

```
...
void RCC_PLLCmd(FunctionalState NewState)
{
  if (NewState != DISABLE)
  { /* Enable PLL */
    RCC->CR |= RCC_CR_PLLON;
  }
  else
  { /* Disable PLL */
    RCC->CR &= ~RCC_CR_PLLON;
  }
}
```

Using bit-band access this function will be coded as follows:

```
void RCC_PLLCmd(FunctionalState NewState)
{
  *(__IO uint32_t *) CR_PLLON_BB = (uint32_t)NewState;
}
```

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 stm32f30x_conf.h file.

Example:PWR_ClearFlag function

stm32f30x_pwr.c:

```
void PWR_ClearFlag(uint32_t PWR_FLAG)
{
  /* Check the parameters */
  assert_param(IS_PWR_CLEAR_FLAG(PWR_FLAG));
  PWR->CR |=  PWR_FLAG << 2;
}
```

stm32f30x_pwr.h:

```
/* PWR Flag */
#define PWR_FLAG_WU               ((uint32_t)0x00000001)
#define PWR_FLAG_SB               ((uint32_t)0x00000002)
#define PWR_FLAG_PVDO             ((uint32_t)0x00000004)
#define PWR_FLAG_VREFINTRDY              ((uint32_t)0x00000008)
```

```
...
#define IS_PWR_CLEAR_FLAG(FLAG) (((FLAG) == PWR_FLAG_WU) || ((FLAG)
== PWR_FLAG_SB))
```

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 **stm32f30x_conf.h**:

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

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

```
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name
and line number */
  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 STM32F30xx standard peripheral drivers (STM32f30xx_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 STM32F30xx_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 STM32F30xx standard peripherals driver and appropriate measures have been taken.

Compliance matrix

The compliance of the STM32F30xx standard peripherals driver (STM32F30xx_StdPeriph_Driver) with MISRA-C 2004 has been checked using the IAR C/C++ Compiler for ARM. MISRA compliance applies only to STM32F30xx 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 V6.40
- Manual checking (code review)

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

Table 3: 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.

MISRA-C 2004 rule number	Required/Advisory	Summary	Reason
10.1	Required	The value of an expression of integer type shall not be implicitly converted to a different underlying type.	Complexity
10.3	Required	The value of a complex expression of integer type shall only be casted to a type that is not wider and of the same signedness as the underlying type of the expression.	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
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 STM32F30xx 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. Uncomment the "USE_FULL_ASSERT" inside the STM32f30x_conf.h file

Only the STM32F30xx standard peripherals driver are MISRA-C 2004 Compliant.

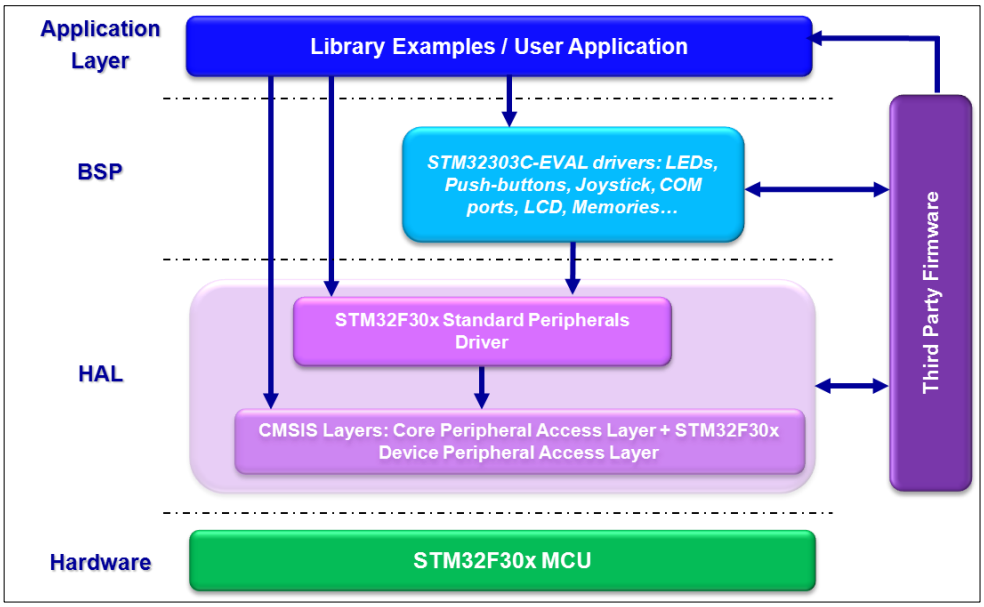
[1] MISRA-C 2004 Guidelines for the use of the C language in critical systems, Motor Industry Software Reliability Association, October 2004

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 STM32F30xx 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 STM32F30/F31xx device registers and features.

- CMSIS layer
  - Core Peripheral Access Layer: contains name definitions, address definitions and helper functions to access core registers and peripherals. It defines also a device independent interface for RTOS Kernels that includes debug channel definitions.
  - STM32F30xx Device Peripheral Access Layer: provides definitions for all the peripheral register's definitions, bits definitions and memory mapping for STM32F30xx and STM32F31xx devices.
- STM32F30xx standard peripheral driver that provides drivers and header files for all the peripherals. It uses CMSIS layer to access STM32F30xx and STM32F31xx registers.

BSP

BSP is a board specific package (BSP) that implements an abstraction layer to interact with the Human Interface resources; buttons, LEDs, LCD and COM ports (USARTs) available on STMicroelectronics evaluation boards. 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.

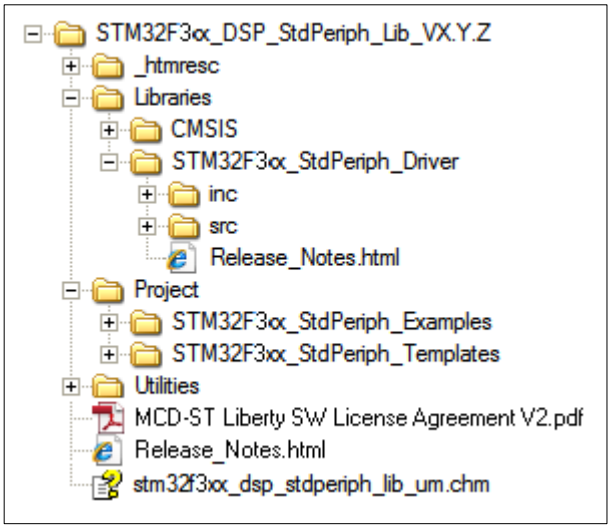
Application layer

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, STM32F30xx_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

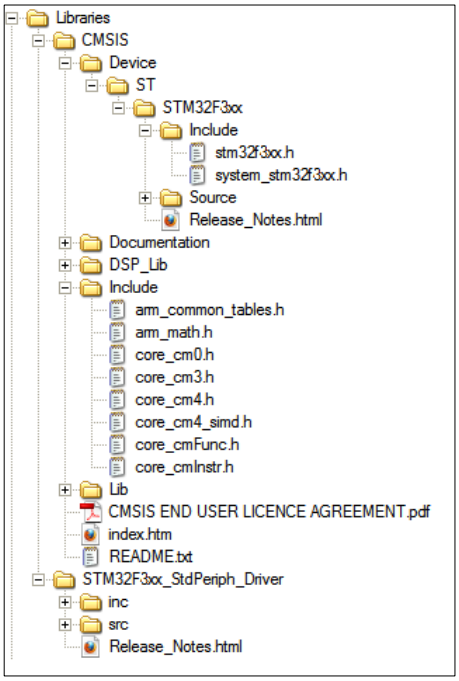
The library package consists of three main folders, described in *Section 1.3.1: "Library folder structure"*

1.3.1 Library folder structure

This folder contains all CMSIS files and STM32F30xx Standard Peripheral Drivers.

The library folder structure is shown in the figure below:

Figure 3: Library folder structure



CMSIS subfolder

This subfolder contains the STM32F30/31xx and Cortex-M4 CMSIS files:

- Cortex-M CMSIS files containing name definitions, address definitions and helper functions to access Cortex-M4 core registers and peripherals. It defines also a device independent interface for RTOS kernels that includes debug channel definitions.
- STM32F30/31xx CMSIS files consist of:
  - stm32f30x.h: this file contains the definitions of all peripheral registers, bits, and memory mapping for STM32F30/31xx devices. It is the unique include file used in the application programmer C source code, usually in the main.c.
  - system_stm32f30x.c/.h: this file contains the system clock configuration for STM32F30/31xx devices. It exports SystemInit() function which sets up the system clock source, PLL multiplier and divider factors, AHB/APBx prescalers and Flash settings. This function is called at startup just after reset and before connecting to the main program. The call is made inside the startup_stm32f30x.s file.
  - startup_stm32f30x.s: this file contains the Cortex-M4 startup code and interrupt vectors for all STM32F30/31xx device interrupt handlers.

STM32F30xx_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, stm32f30x_ppp.c, and a header file, stm32f30x_ppp.h. The stm32f30x_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 4: Description of CMSIS files

File name	Description
core_cm4.h	Describes the data structures for the Cortex-M4 core peripherals and performs the address mapping of these structures. It also provides basic access to the Cortex-M4 core registers and core peripherals using efficient functions defined as static inline.
stm32f30x.h	CMSIS Cortex-M4 STM32F30/31xx peripheral access layer header file. This file contains the definitions of all peripheral registers, bits, and memory mapping for STM32F30/31xx devices. The file is the unique include file used in the application programmer C source code, usually in the main.c. This file 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 __STM32f30XX_STDPERIPH_VERSION
system_stm32f30x.c	This file contains the system clock configuration for STM32F30/31xx 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, PLL multiplier and divider factors, AHB/APBx prescalers and Flash settings. This function is called at startup just after reset and before branch to the main program. The call is made inside the startup_stm32f30x.s file. • SystemCoreClock: this variable contains the core clock (HCLK). 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.  This file is automatically generated by the clock configuration tool "STM32f30x_Clock_Configuration.xls". Using this tool, you can generate a configuration file customized for your application requirements. For more information, please refer to AN4152 available from ST web site.
system_stm32f30x.h	Header file for system_stm32f30x.c
startup_stm32f30x.s	Provides the Cortex-M4 startup code and interrupt vectors for all STM32F30/31xx device interrupt handlers. This module performs the following functions: • It sets the initial SP • It sets the initial PC == Reset_Handler • It sets the vector table entries with the exceptions ISR address

File name	Description
	• It branches to __main in the C library (which eventually calls main()). A file is provided for each compiler.

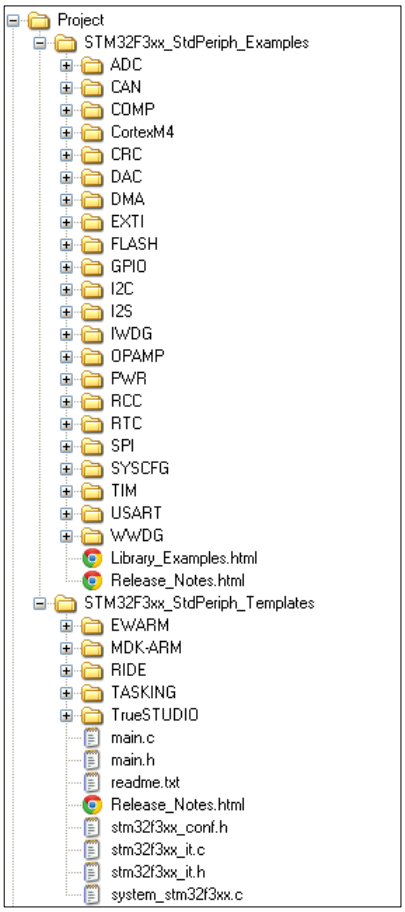
Table 5: STM32f30xx_StdPeriph_Driver files description

File name	Description
stm32f30x_ppp.c	Driver source code file of PPP peripheral coded in Strict ANSI-C, and independent from the development Tools.
stm32f30x_ppp.h	Provides functions prototypes and variable definitions used within for stm32f30x_ppp.c file
stm32f30x_misc.c	Provides all the miscellaneous firmware functions (add-on to CMSIS functions)
stm32f30x_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



STM32F30xx_StdPeriph_Template subfolder

This subfolder contains standard template projects for the supported development tools that compile the needed STM32F30xx 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 6: STM32F30xx_StdPeriph_Templates file 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
stm32f30x_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_stm32f30x.c	This file contains the system clock configuration for STM32F30/31xx 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, PLL multiplier and divider factors, AHB/APBx prescalers and Flash settings. This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f30x.s" file. • SystemCoreClock: this variable contains the core clock (HCLK). 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.  This file is automatically generated by the clock configuration tool "STM32f30x_Clock_Configuration.xls". Using this tool, you can generate a configuration file customized for your application requirements. For more information, please refer to AN4152 available from ST web site.
stm32f30x_it.c	Template source file containing the interrupt service routine (ISR) for Cortex-M4 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_stm32f30x.s).
stm32f30x_it.h	Header file for stm32f30x_it.c

STM32F30xx_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

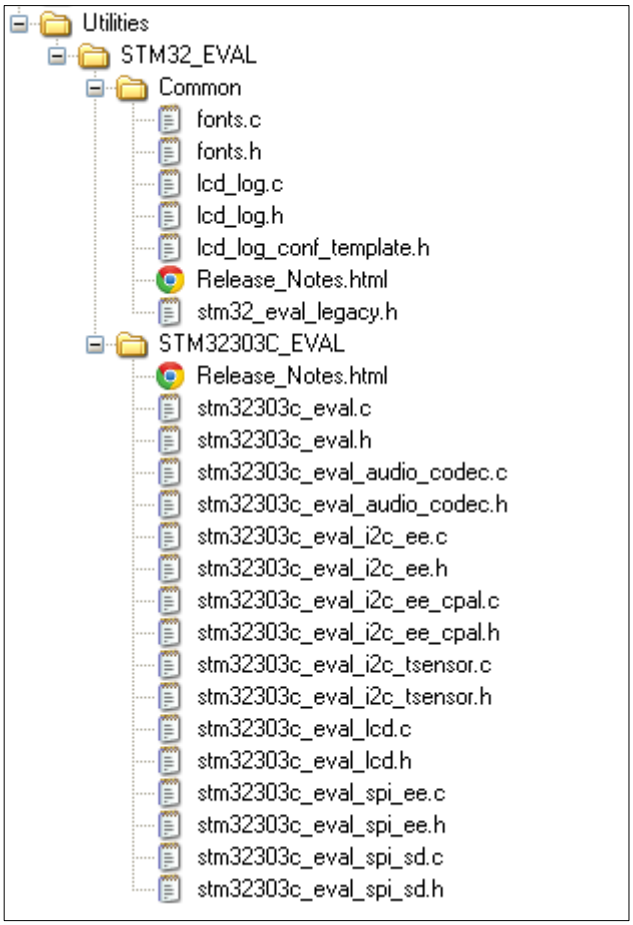
This folder contains the abstraction layer allowing interacting with the human interface resources (buttons, LEDs, LCD and COM ports (USARTs)) available on STMicroelectronics evaluation boards. A common API is provided to manage these different resources. It and can be easily tailored to support any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

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

For each hardware module (e.g. LCD, I2C, EEPROM, external SRAM memory...) the API is fully compatible across all STMicroelectronics evaluation board drivers.

The Utilities folder structure is shown below.

Figure 5: Utilities folder structure



It contains common files and folder, plus a folder for STM32303C_EVAL board files.

Table 7: Utilities/STM32_EVAL files description

File name	Description
stm32303c_eval.c	This file provides: • A set of firmware functions to manage LEDs, pushbuttons, and COM ports • Low level initialization functions for SDcard (on SPI) and serial EEPROM (sEE) available on STM32303C_eval board.
stm32303C_eval.h	Header file for stm32303C_eval.c
stm32303C_eval_audio_codec.c	This file includes the low layer driver for CS42L52 Audio Codec available on STM32303C_eval board.
stm32303C_eval_audio_codec.h	Header file for stm32303C_eval_audio_codec.c
stm32303C_eval_i2c_ee.c	This file provides a set of functions needed to manage the M24M01 I2C EEPROM and the M24LR64 RF EEPROM mounted on STM32303C_eval board.
stm32303C_eval_i2c_ee.h	Header file for stm32303C_eval_i2c_ee.c
stm32303C_eval_lcd.c	This file includes the LCD driver for AM-240320L8TNQW00H (ILI9320), AM-240320LDTNQW00H (SPFD5408B) and AM240320LGTNQW00H (HX8347D) Display Module of STM32303C-EVAL board
stm32303C_eval_lcd.h	Header file for stm32303C_eval_lcd.c
lcd_log.c	Provides all the LCD Log firmware functions. It allows to automatically set a header and footer on any application using the LCD display and to dump user, debug and error messages by using the following macros, LCD_ErrLog(), LCD_UsrLog() and LCD_DbgLog().
fonts.c	Provides text fonts for STM32xx-EVAL LCD driver
stm2303c_eval_i2c_eeprom_cpal.c	This file provides the set of functions needed to manage the M24CXX I2C EEPROM and the M24LR64 RF EEPROM memory mounted on STM32303C-EVAL board using the I2C CPAL drivers.
stm32303c_eval_i2c_eeprom_cpal.c	This file provides the set of functions needed to manage the M24M01 I2C EEPROM and the M24LR64 RF EEPROM.
stm32303c_eval_ i2c_eeprom_cpal.h	Header file for stm32303c_eval_i2c_eeprom_cpal.c.
stm32303c_eval_spi_ee.c	This file provides a set of functions needed to manage the SPI M95xxx EEPROM memory mounted on STM32303C-EVAL board.
stm32303c_eval_spi_ee.h	Header file for stm32303c_eval_spi_ee.c
stm32303c_eval_spi_sd.c	This file provides a set of functions neededd to manage the SPI SD card memory mounted on STM32303C-EVAL board.
stm32303c_eval_spi_sd.h	Header file for stm32303c_eval_spi_sd.c.

1.4 Supported devices and development tools

1.4.1 Supported devices

The library supports the STM32F30xx and STM32F31xx microcontroller memory and peripherals. By using this library moving the application firmware from one STM32F30/31xx device to another becomes straightforward.

The device part number is defined as follows in stm32f30x.h file:

```
#if !defined (STM32F30X)
  #define STM32F30X
#endif
```

This define statement can be used at application level to configure the application firmware for STM32F30/31xx devices.

1.4.2 Supported development tools and compilers

STM32F30/31xx 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.

The library is supported by all major tool providers.

A template project is available for each development tool:

- **IAR Embedded Workbench for ARM (EWARM)** development tool
  - Compiler: IAR's C/C++
- **RealView Microcontroller Development Kit (MDK-ARM)** development tool
  - Compiler: ARM C/C++ compiler
- **TASKING VX-toolset for ARM Cortex-M** 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 version.

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 8: Library configuration parameters

Parameter	File	Description
STM32F30XX(1)	stm32f30x.h	Default status: enabled Defines the root number of STM32F30/31xx devices. This define statement can be used at application level to configure the application firmware for STM32F30/31xx.
USE_STDPERIPH_DRIVER(1)	stm32f30x.h	Default status: enabled When disabled, the peripheral drivers are not included and the application code is based on direct access to peripherals registers.
HSE_VALUE	stm32f30x.h	Default value: 8 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	stm32f30x.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	stm32f30x.h	Default value: 8 MHz Defines the value of the internal oscillator (HSI) expressed in Hz.
__CM4_REV	stm32f30x.h	
__MPU_PRESENT		These define statements are used by Cortex-M4 CMSIS layer to inform about the options supported by STM32F30/31xx devices: `/*!< Configuration of the Cortex-M4 Processor and Core Peripherals`
__NVIC_PRIO_BITS		
__Vendor_SysTickConfig		
__FPU_PRESENT		

Parameter	File	Description
		/ /!<Core revision r0p1>*/ #define __CM4_REV 0x0001 /*!<STM32F30/31x features an MPU>*/ #define __MPU_PRESENT 1 /*!<STM32F30/31x features 4 bits for priority levels>*/ #define __NVIC_PRIO_BITS 4 /*!<Set to 1 if different SysTick configuration is used >*/ #define __Vendor_SysTickConfig 0 /*!<STM32F30/31x features an FPU>*/ #define __FPU_PRESENT 1 They should not be modified by the user.
USE_FULL_ASSERT	stm32f30x_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	stm32f30x_conf.h	This file allows to enable/disable the inclusion of the peripheral driver header files. By default all header files are included. #include "stm32f30x_adc.h" #include "stm32f30x_can.h" #include "stm32f30x_crc.h" #include "stm32f30x_comp.h" #include "stm32f30x_dac.h" #include "stm32f30x_dbgmcu.h" #include "stm32f30x_dma.h" #include "stm32f30x_exti.h" #include "stm32f30x_flash.h" #include "stm32f30x_gpio.h" #include "stm32f30x_syscfg.h"

Parameter	File	Description

		`#include "stm32f30x_i2c.h"`
		`#include "stm32f30x_iwdg.h"`
		`#include "stm32f30x_pwr.h"`
		`#include "stm32f30x_opamp.h"`
		`#include "stm32f30x_rcc.h"`
		`#include "stm32f30x_rtc.h"`
		`#include "stm32f30x_spi.h"`
		`#include "stm32f30x_tim.h"`
		`#include "stm32f30x_usart.h"`
		`#include "stm32f30x_wwdg.h"`
		`#include "misc.h"`

USE_STM32303C_EVAL(1)	stm32303C_eval.h	Default status: disabled This define statement is used to include the driver for STM32303C_EVAL board, when used.
VECT_TAB_SRAM	system_stm32f30x.c	Default status: disabled When enabled, this define statement relocate the vector table in the Internal SRAM
VECT_TAB_OFFSET		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 STM32F30/31xx 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, poceed as follows:

1. Comment the line *#define USE_STDPERIPH_DRIVER* in *stm32f30x.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 *stm32f30x.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 *stm32f30x.h*.
2. In *stm32f30x_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 stm32f30x_ppp.h/.c files under Libraries\STM32F30xx_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.

Before configuring a peripheral, its clock must be enabled by calling one of the following functions:

```
RCC_AHBPeriphClockCmd(RCC_AHB1Periph_PPPx, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx, ENABLE);
```

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 STM32303C-EVAL 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 STM32F30/31xx related documents and library from STMicroelectronics web site: www.st.com/stm32
2. Hardware: to run the examples, you need an STM32303C-EVAL 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 STM32F30/31xx 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 provided within the Library. This example configures the system clock to 72 MHz, initializes the evaluation board LEDs, LCD and USART communication interface, then displays a welcome message on the LCD, and finally toggles four LEDs in an infinite loop.

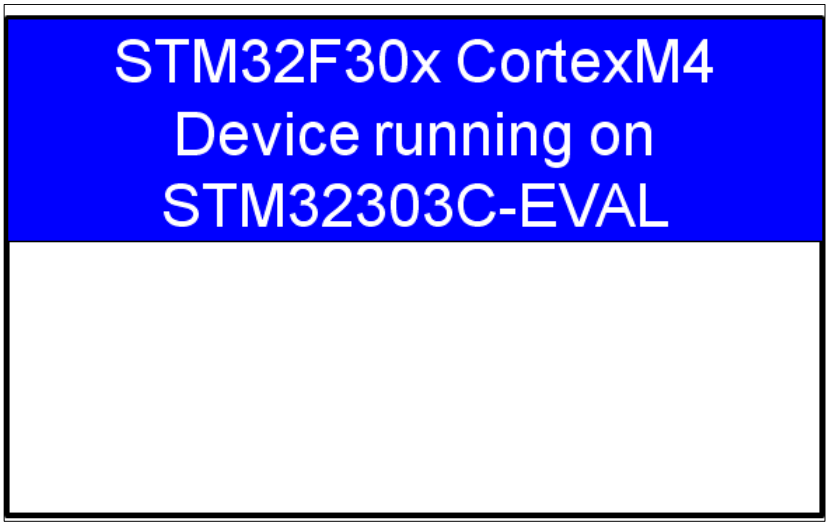
To achieve this goal you have to proceed as described below:

1. Download and unzip the STM32F30xx_dsp_stderiph_Lib_VX.Y.Z.zip in the folder of your choice
2. Power-up the STM32303C-EVAL board
3. Connect your JTAG probe to the JTAG connector (CN10) of the EVAL board and to the USB port of your PC. The STM32303C-EVAL features a build-in ST-Link/V2 debugger and programmer which makes the external hardware debuggers useless to load and debug your program. Simply select ST-Link/V2 as your debugger in your Development Tool configuration menu and connect the CN5 to your host PC through an USB cable. Refer to your development tool documentation to know if it supports the ST-Link/V2 debugger.
4. Run the template example: go to STM32F30xx_StdPeriph_Lib_VX.Y.Z\Project\STM32F30xx_StdPeriph_Templates 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 STM32303C-EVAL 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 STM32303C-EVAL project to configure the project for STM32F30/31xx 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).

If the above sequence has worked correctly, LED1, 2, 3, and 4 should be blinking and the following message is displayed on the LCD screen.

Figure 6: Message displayed on the LCD when running the template example



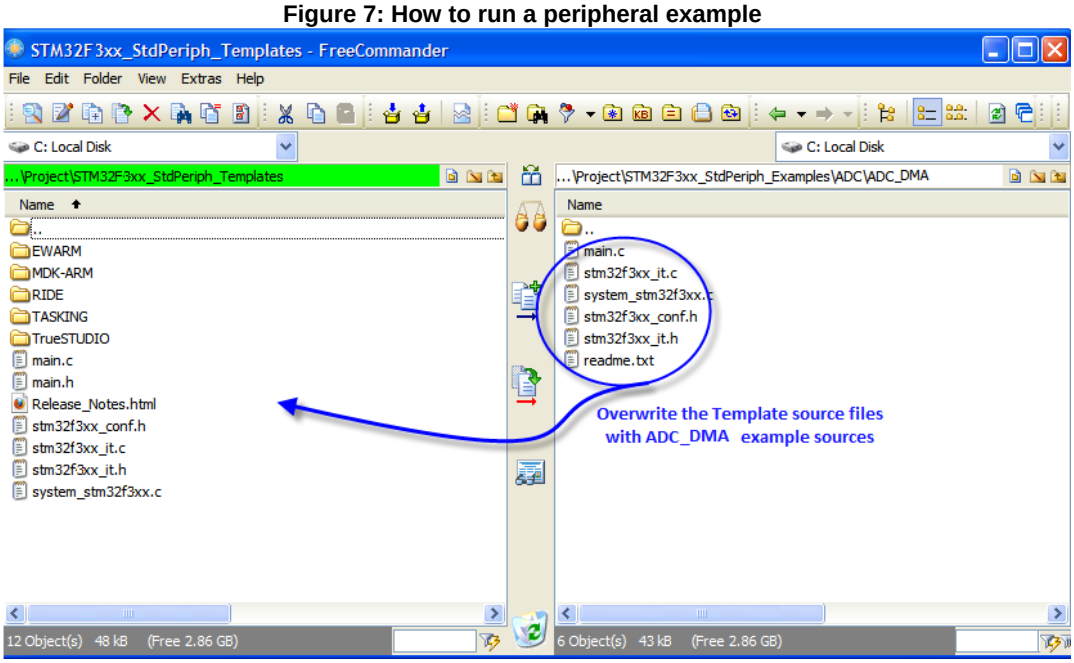
2.4.3 Run a peripheral example

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 ADC_DMA example:

1. Copy all source files from Project\STM32F30xx_StdPeriph_Examples\ADC\ADC_BasicExample to the template folder under Project\STM32F30xx_StdPeriph_Templates, see *Figure 7: "How to run a peripheral example "*
2. Open your preferred development tool, and proceed as described in section *Section 2.4.2: "Run your first example"*
3. If the example use 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 7: How to run a peripheral example



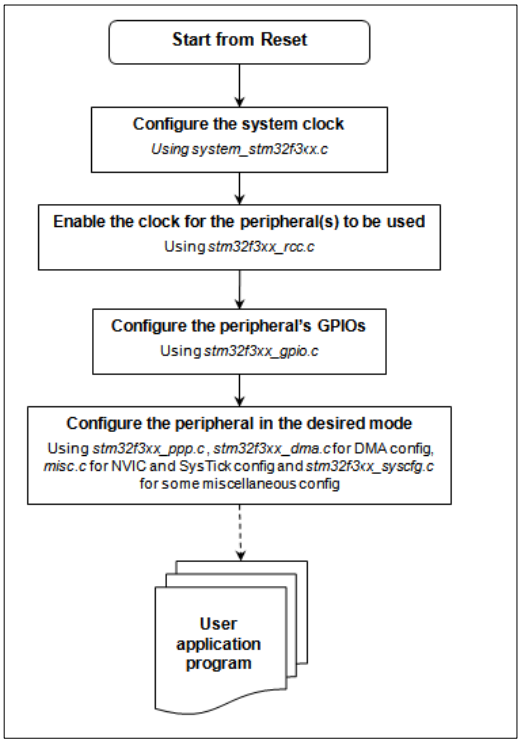
2.5 STM32F30/31xx programming model using the library

This chapter contains useful general information for using the library to develop application based on STM32F30/31xx devices. It describes in details the sequence to use a peripheral, from the configuration of the system to the configuration of the peripheral registers.

After reset the device is running from Internal High Speed oscillator (HSI 8 MHz) with 0 Flash wait state, Flash prefect buffer, D-Cache and I-Cache disabled, and all peripherals off except internal SRAM, Flash and JTAG:

- There is no prescaler on High speed (AHB) and Low speed (APB) buses. All the peripherals mapped on these buses are running at HSI speed.
- The clock for all peripherals is switched off, except for SRAM and FLASH.
- All GPIOs are in input floating state, except for JTAG pins which are assigned to debug. Once the device started from reset, the user application has to configure the system clock and all peripheral hardware resources (GPIO, Interrupt, DMA…) .

Figure 8: STM32F30/31xx programming model using the library



1. **System clock configuration:** the STM32F30/31xx devices can run at frequency up to 72 MHz and feature several prescalers to configure the AHB, APB1 and APB2 frequencies. The maximum frequency of the AHB domain is 72 MHz. The maximum allowed frequency of the high-speed APB2 domain is 72 MHz, while the maximum allowed frequency of the low speed APB1 domain is 36 MHz. If the application requires higher frequency/performance, follow the sequence below to configure the system clock:
   a. Configure the Flash wait state through FLASH_ACR register. For more details refer to *Section 11: "FLASH Memory (FLASH)"*
   b. Select the clock source to be used. Internal (HSI 8MHz) or external (HSE up to 8 MHz).
   c. Configure the PLL (optional), system input clock and AHB, APB1 and APB2 prescaler. For more details, refer to *Section 18: "Reset and clock control (RCC)"*You can use the clock configuration tool (STM32F30xx_Clock_Configuration.xls) to generate a customized system_stm32f30x.c file depending on your application requirements.
2. **Enable the clock for the peripheral(s) to be used**: Before starting to use a peripheral, enable the corresponding interface clock, as well as the clock for the associated GPIOs. This is done by using one of the following functions:
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_PPPx, ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx, ENABLE);
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx, ENABLE);For example, the following function should be used to enable USART1 interface clock :
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);For more details, refer to *Section 19: "Real-time clock (RTC)"*
3. Configure the clock source(s) for peripherals which clocks are not derived from the System clock:
   a. RTC: STM32F30/31xx RTC clock can be derived either from a LSI, LSE or HSE clock divided by 2 to 31. For more details, refer to *Section 19: "Real-time clock (RTC)"*

b. USB FS: in STM32F30/31xx devices, the USB FS requires a frequency equal to 48MHz to work correctly.
c. 12-bit ADC: STM32F30/31xx12-bit ADC features two clock schemes:
   – Clock for the analog circuitry (ADCCLK). This clock is generated from the PLL output. It can reach 72 MHz and can be divided by 1, 2, 4, 6, 8, 10, 12, 16, 32, 64, 128 or 256. The ADCCLK clock can be configured through RCC registers. The AHB clock divided by a programmable prescaler can also be generated. This clock allows the ADC to work at fHCLK/1, /2 or /8. ADCCLK maximum value is 72 MHz when the AHB clock is 72 MHz. ADCCLK is configure through the ADC registers.
   – Clock for the digital interface (used for register read/write access). This clock is equal to the AHB clock. The digital interface clock can be enabled/disabled individually for the ADC through the RCC AHB peripheral clock enable register (RCC_AHBENR).
   – For more details, refer to *Section 3: "Analog-to-digital converter (ADC)"*
d. **Configure the peripheral GPIOs**: Whatever the peripheral mode, the I/Os should be configured as alternate function, before being used as input or output. To configure the I/Os, follow the steps below:
   a. Connect the pin to the desired peripheral alternate function (AF) using GPIO_PinAFConfig() function
   b. Use GPIO_Init() function to configure the I/O pin
   – Configure the desired pin in alternate function mode using GPIO_InitStructure->GPIO_Mode = GPIO_Mode_AF;
   – Select the type, pull-up/pull-down and output speed via GPIO_PuPd, GPIO_OType and GPIO_Speed members For more details, refer to *Section 12: "General-purpose I/Os (GPIO)"*

4. **Configure the peripheral in the desired mode**: refer to the peripheral firmware driver section for details on the initialization procedure and how to use the available API. Other modules need to be configured when using interrupt and DMA:
   a. Using the interrupts: after enabling the interrupt source(s) in the peripheral registers, enable the peripheral interrupt line and configure its priority in the NVIC. For more details, refer to *Section 15: "Miscellaneous add-on to CMSIS functions(misc)"*
   b. Using the DMA: after enabling the DMA source(s) in the peripheral registers, configure and enable the peripheral DMA channel in the DMA controller. For more details, refer to *Section 9: "DMA controller (DMA)"*

2.6 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:

- STM32303C-EVAL board used as reference hardware
- System clock configured to 72 MHz, with 2 Flash wait state, Flash prefetch enabled.
- PE6 pin used as EXTI Line6. This pin is connected externally to a pushbutton.
- PE8 and PE9 pins used in output mode to drive LED1 and LED2, respectively.

2.6.1 Starting point

The typical starting point is the template project provided within the library package (Project\STM32F30x_StdPeriph_Templates). This 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.
- stm32f30x_it.c: use this template file to add the code required to manage the EXTI Line2 interrupt.
- stm32f30x_it.h: use this template file to add the EXTI Line6 interrupt prototype.
- stm32f30x_conf.h: use this template file without any change
- system_stm32f30x.c: use the template file without any change

Follow the steps described in *Section 2.5: "STM32F30/31xx programming model using the library"* to develop your application.

2.6.2 Library configuration parameters

To configure the library for your application, use the library default parameters as defined in *Section 2.1: "Library configuration parameters"*

2.6.3 system_stm32f30x.c

This file contains the SystemInit() function that configures the system clock, system clock source, PLL Multiplier and Divider factors, AHB/APBx prescalers and Flash settings. This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f30x.s" file.

The clock configuration tool "STM32f30x_Clock_Configuration.xls" is used to generate system_stm32f30x.c file that configures the device as follow. The table below shows the default configuration of system_stm32f30xx.c provided within the library:

Table 9: Default clock configuration in system_stm32f30x.c

System Clock source	HSE (System Clock source)
SYSCLK	72000000 Hz
HCLK	72000000 Hz
AHB Prescaler	1
APB1 Prescaler	2
APB2 Prescaler	1
HSE Frequency	8000000 Hz
PLL MUL	9
PREDIV	1
USB clock	ENABLED
Flash latency (number of WS)	2
Prefetch Buffer	ON

2.6.4 main.c

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

Include the library and STM32303C-EVAL-EVAL board resources:

```
/* Includes ------------------------------------------------------*/
#include "stm32f30x.h"  /* The Library entry point */
#include "stm32303C_eval" /* Needed when using STM32303C-EVAL
board*/
```

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;
```

Declare prototype for a local function:

```
/* Private function prototypes -----------------------------------*/
void Delay(__IO uint32_t nCount);
```

The main program will be structured as follow:

```
/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */
int main(void)
{
```

1. System clock configuration:

```
/*!< At this stage the microcontroller clock setting is already
configured,
this is done through SystemInit() function which is called from
startup
file (startup_stm32f30x.s) before to branch to application
main.

To reconfigure the default setting of SystemInit() function,
refer to
system_stm32f30x.c file */
```

2. Enable the clock for the peripheral(s) to be used (EXTI interface clock is always enabled):

```
 /* Enable GPIOA's AHB interface clock */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
  /* Enable SYSCFG's APB interface clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
```

3. Configure the peripheral GPIOs:

```
 /* Connect EXTI6 Line to PE6 pin */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource6);
 /* Configure PE6 pin in input mode */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
```

```
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
```

4. Configure the peripheral in the desired mode:

```
 /* Configure  EXTI line 6 */
  EXTI_InitStructure.EXTI_Line = EXTI_Line6;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set  EXTI line 6 nterrupt to the lowest
priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
```

5. Insert the code below to use the evaluation board HAL to drive the LEDs:

```
 /* Initialize LED1 and LED2 mounted on STM32303C-EVAL board */
  STM_EVAL_LEDInit(LED1);
  STM_EVAL_LEDInit(LED2);
  while (1)
  {
    /* Toggle LD1 */
    STM_EVAL_LEDToggle(LED1);
    /* Insert some delay */
    Delay(0xFFFFF);
  }
}
/**
  * @brief  Inserts a delay time.
  * @param  nCount: specifies the delay time length.
  * @retval None
  */
void Delay(__IO uint32_t nCount)
{
  for(; nCount != 0; nCount--);
}
```

2.6.5 stm32f30x_it.c

The stm32f30x_it.c file can be used to implement the EXTI Line2 interrupt service routine (ISR) in which LED2 toggles each time the ISR is executed.

1. In “STM32F30xx Peripherals Interrupt Handlers“ section, add the following code:

```
/*************************************************************
***/
/*                 STM32F30xx 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_stm32f30x.s).
*/
```

```
/*******************************************************************
************/
/**
 * @brief  This function handles External line 6
 * interrupt request.
 * @param  None
 * @retval None
 */
void EXTI9_5_IRQHandler(void)
{
  if(EXTI_GetITStatus(EXTI_Line6) != RESET)
  {
    /* Toggle LD1 */
    STM_EVAL_LEDToggle(LED1);

    /* Clear the EXTI line 6 pending bit */
    EXTI_ClearITPendingBit(EXTI_Line6);
  }
}
```

2. In stm32f30x_it.h file add the EXTI Line2 ISR prototype as follows (just after the line void SysTick_Handler(void);)

```
void EXTI9_5_IRQHandler(void);
```

3 Analog-to-digital converter (ADC)

3.1 ADC Firmware driver registers structures

3.1.1 ADC_Common_TypeDef

ADC_Common_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CSR***
- ***uint32_t RESERVED***
- ***__IO uint32_t CCR***
- ***__IO uint32_t CDR***

Field Documentation

- ***__IO uint32_t ADC_Common_TypeDef::CSR***
  - ADC Common status register, Address offset: ADC1/3 base address + 0x300
- ***uint32_t ADC_Common_TypeDef::RESERVED***
  - Reserved, ADC1/3 base address + 0x304
- ***__IO uint32_t ADC_Common_TypeDef::CCR***
  - ADC common control register, Address offset: ADC1/3 base address + 0x308
- ***__IO uint32_t ADC_Common_TypeDef::CDR***
  - ADC common regular data register for dual AND triple modes, Address offset: ADC1/3 base address + 0x30C

3.1.2 ADC_CommonInitTypeDef

ADC_CommonInitTypeDef is defined in the stm32f30x_adc.h

Data Fields

- ***uint32_t ADC_Mode***
- ***uint32_t ADC_Clock***
- ***uint32_t ADC_DMAAccessMode***
- ***uint32_t ADC_DMAMode***
- ***uint8_t ADC_TwoSamplingDelay***

Field Documentation

- ***uint32_t ADC_CommonInitTypeDef::ADC_Mode***
  - Configures the ADC to operate in independent or multi mode. This parameter can be a value of ***ADC_mode***
- ***uint32_t ADC_CommonInitTypeDef::ADC_Clock***

  - Select the clock of the ADC. The clock is common for both master and slave ADCs. This parameter can be a value of ***ADC_Clock***
- ***uint32_t ADC_CommonInitTypeDef::ADC_DMAAccessMode***
  - Configures the Direct memory access mode for multi ADC mode. This parameter can be a value of ***ADC_Direct_memory_access_mode_for_multi_mode***
- ***uint32_t ADC_CommonInitTypeDef::ADC_DMAMode***
  - Configures the DMA mode for ADC. This parameter can be a value of ***ADC_DMA_Mode_definition***
- ***uint8_t ADC_CommonInitTypeDef::ADC_TwoSamplingDelay***
  - Configures the Delay between 2 sampling phases. This parameter can be a value between 0x0 and 0xF

3.1.3 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32f30x_adc.h

Data Fields

- ***uint32_t ADC_ContinuousConvMode***
- ***uint32_t ADC_Resolution***
- ***uint32_t ADC_ExternalTrigConvEvent***
- ***uint32_t ADC_ExternalTrigEventEdge***
- ***uint32_t ADC_DataAlign***
- ***uint32_t ADC_OverrunMode***
- ***uint32_t ADC_AutoInjMode***
- ***uint8_t ADC_NbrOfRegChannel***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ADC_ContinuousConvMode***
  - Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::ADC_Resolution***
  - Configures the ADC resolution. This parameter can be a value of ***ADC_resolution***
- ***uint32_t ADC_InitTypeDef::ADC_ExternalTrigConvEvent***
  - Defines the external trigger used to start the analog to digital conversion of regular channels. This parameter can be a value of ***ADC_external_trigger_sources_for_regular_channels_conversion***
- ***uint32_t ADC_InitTypeDef::ADC_ExternalTrigEventEdge***
  - Select the external trigger edge and enable the trigger of a regular group. This parameter can be a value of ***ADC_external_trigger_edge_for_regular_channels_conversion***
- ***uint32_t ADC_InitTypeDef::ADC_DataAlign***
  - Specifies whether the ADC data alignment is left or right. This parameter can be a value of ***ADC_data_align***
- ***uint32_t ADC_InitTypeDef::ADC_OverrunMode***
  - Specifies the way data overrun are managed. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::ADC_AutoInjMode***

  - Enable/disable automatic injected group conversion after regular group conversion. This parameter can be set to ENABLE or DISABLE.
- ***uint8_t ADC_InitTypeDef::ADC_NbrOfRegChannel***
  - Specifies the number of ADC channels that will be converted using the sequencer for regular channel group. This parameter must range from 1 to 16.

3.1.4 ADC_InjectedInitTypeDef

ADC_InjectedInitTypeDef is defined in the stm32f30x_adc.h

Data Fields

- ***uint32_t ADC_ExternalTrigInjecConvEvent***
- ***uint32_t ADC_ExternalTrigInjecEventEdge***
- ***uint8_t ADC_NbrOfInjecChannel***
- ***uint32_t ADC_InjecSequence1***
- ***uint32_t ADC_InjecSequence2***
- ***uint32_t ADC_InjecSequence3***
- ***uint32_t ADC_InjecSequence4***

Field Documentation

- ***uint32_t ADC_InjectedInitTypeDef::ADC_ExternalTrigInjecConvEvent***
  - Defines the external trigger used to start the analog to digital conversion of injected channels. This parameter can be a value of ***ADC_external_trigger_sources_for_Injected_channels_conversion***
- ***uint32_t ADC_InjectedInitTypeDef::ADC_ExternalTrigInjecEventEdge***
  - Select the external trigger edge and enable the trigger of an injected group. This parameter can be a value of ***ADC_external_trigger_edge_for_Injected_channels_conversion***
- ***uint8_t ADC_InjectedInitTypeDef::ADC_NbrOfInjecChannel***
  - Specifies the number of ADC channels that will be converted using the sequencer for injected channel group. This parameter must range from 1 to 4.
- ***uint32_t ADC_InjectedInitTypeDef::ADC_InjecSequence1***
- ***uint32_t ADC_InjectedInitTypeDef::ADC_InjecSequence2***
- ***uint32_t ADC_InjectedInitTypeDef::ADC_InjecSequence3***
- ***uint32_t ADC_InjectedInitTypeDef::ADC_InjecSequence4***

3.1.5 ADC_TypeDef

ADC_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t ISR***
- ***__IO uint32_t IER***
- ***__IO uint32_t CR***
- ***__IO uint32_t CFGR***
- ***uint32_t RESERVED0***

- ***__IO uint32_t SMPR1***
- ***__IO uint32_t SMPR2***
- ***uint32_t RESERVED1***
- ***__IO uint32_t TR1***
- ***__IO uint32_t TR2***
- ***__IO uint32_t TR3***
- ***uint32_t RESERVED2***
- ***__IO uint32_t SQR1***
- ***__IO uint32_t SQR2***
- ***__IO uint32_t SQR3***
- ***__IO uint32_t SQR4***
- ***__IO uint32_t DR***
- ***uint32_t RESERVED3***
- ***uint32_t RESERVED4***
- ***__IO uint32_t JSQR***
- ***uint32_t RESERVED5***
- ***__IO uint32_t OFR1***
- ***__IO uint32_t OFR2***
- ***__IO uint32_t OFR3***
- ***__IO uint32_t OFR4***
- ***uint32_t RESERVED6***
- ***__IO uint32_t JDR1***
- ***__IO uint32_t JDR2***
- ***__IO uint32_t JDR3***
- ***__IO uint32_t JDR4***
- ***uint32_t RESERVED7***
- ***__IO uint32_t AWD2CR***
- ***__IO uint32_t AWD3CR***
- ***uint32_t RESERVED8***
- ***uint32_t RESERVED9***
- ***__IO uint32_t DIFSEL***
- ***__IO uint32_t CALFACT***

Field Documentation

- ***__IO uint32_t ADC_TypeDef::ISR***
  - ADC Interrupt and Status Register, Address offset: 0x00
- ***__IO uint32_t ADC_TypeDef::IER***
  - ADC Interrupt Enable Register, Address offset: 0x04
- ***__IO uint32_t ADC_TypeDef::CR***
  - ADC control register, Address offset: 0x08
- ***__IO uint32_t ADC_TypeDef::CFGR***
  - ADC Configuration register, Address offset: 0x0C
- ***uint32_t ADC_TypeDef::RESERVED0***
  - Reserved, 0x010
- ***__IO uint32_t ADC_TypeDef::SMPR1***
  - ADC sample time register 1, Address offset: 0x14
- ***__IO uint32_t ADC_TypeDef::SMPR2***
  - ADC sample time register 2, Address offset: 0x18
- ***uint32_t ADC_TypeDef::RESERVED1***
  - Reserved, 0x01C

- ***__IO uint32_t ADC_TypeDef::TR1***
  - ADC watchdog threshold register 1, Address offset: 0x20
- ***__IO uint32_t ADC_TypeDef::TR2***
  - ADC watchdog threshold register 2, Address offset: 0x24
- ***__IO uint32_t ADC_TypeDef::TR3***
  - ADC watchdog threshold register 3, Address offset: 0x28
- ***uint32_t ADC_TypeDef::RESERVED2***
  - Reserved, 0x02C
- ***__IO uint32_t ADC_TypeDef::SQR1***
  - ADC regular sequence register 1, Address offset: 0x30
- ***__IO uint32_t ADC_TypeDef::SQR2***
  - ADC regular sequence register 2, Address offset: 0x34
- ***__IO uint32_t ADC_TypeDef::SQR3***
  - ADC regular sequence register 3, Address offset: 0x38
- ***__IO uint32_t ADC_TypeDef::SQR4***
  - ADC regular sequence register 4, Address offset: 0x3C
- ***__IO uint32_t ADC_TypeDef::DR***
  - ADC regular data register, Address offset: 0x40
- ***uint32_t ADC_TypeDef::RESERVED3***
  - Reserved, 0x044
- ***uint32_t ADC_TypeDef::RESERVED4***
  - Reserved, 0x048
- ***__IO uint32_t ADC_TypeDef::JSQR***
  - ADC injected sequence register, Address offset: 0x4C
- ***uint32_t ADC_TypeDef::RESERVED5[4]***
  - Reserved, 0x050 - 0x05C
- ***__IO uint32_t ADC_TypeDef::OFR1***
  - ADC offset register 1, Address offset: 0x60
- ***__IO uint32_t ADC_TypeDef::OFR2***
  - ADC offset register 2, Address offset: 0x64
- ***__IO uint32_t ADC_TypeDef::OFR3***
  - ADC offset register 3, Address offset: 0x68
- ***__IO uint32_t ADC_TypeDef::OFR4***
  - ADC offset register 4, Address offset: 0x6C
- ***uint32_t ADC_TypeDef::RESERVED6[4]***
  - Reserved, 0x070 - 0x07C
- ***__IO uint32_t ADC_TypeDef::JDR1***
  - ADC injected data register 1, Address offset: 0x80
- ***__IO uint32_t ADC_TypeDef::JDR2***
  - ADC injected data register 2, Address offset: 0x84
- ***__IO uint32_t ADC_TypeDef::JDR3***
  - ADC injected data register 3, Address offset: 0x88
- ***__IO uint32_t ADC_TypeDef::JDR4***
  - ADC injected data register 4, Address offset: 0x8C
- ***uint32_t ADC_TypeDef::RESERVED7[4]***
  - Reserved, 0x090 - 0x09C
- ***__IO uint32_t ADC_TypeDef::AWD2CR***
  - ADC Analog Watchdog 2 Configuration Register, Address offset: 0xA0
- ***__IO uint32_t ADC_TypeDef::AWD3CR***
  - ADC Analog Watchdog 3 Configuration Register, Address offset: 0xA4
- ***uint32_t ADC_TypeDef::RESERVED8***
  - Reserved, 0x0A8

- ***uint32_t ADC_TypeDef::RESERVED9***
  - Reserved, 0x0AC
- ***__IO uint32_t ADC_TypeDef::DIFSEL***
  - ADC Differential Mode Selection Register, Address offset: 0xB0
- ***__IO uint32_t ADC_TypeDef::CALFACT***
  - ADC Calibration Factors, Address offset: 0xB4

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. select the ADC clock using the function RCC_ADCCLKConfig()
2. Enable the ADC interface clock using RCC_AHBPeriphClockCmd();
3. ADC pins configuration
   - Enable the clock for the ADC GPIOs using the following function: RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOx, ENABLE);
   - Configure these ADC pins in analog mode using GPIO_Init();
4. Configure the ADC conversion resolution, data alignment, external trigger and edge, sequencer lenght and Enable/Disable the continuous mode using the ADC_Init() function.
5. Activate the ADC peripheral using ADC_Cmd() function.

ADC channels group configuration

- To configure the ADC channels features, use ADC_Init(), ADC_InjectedInit() and ADC_RegularChannelConfig() functions or/and ADC_InjectedChannelConfig()
- To activate the continuous mode, use the ADC_ContinuousModeCmd() function.
- To activate the Discontinuous mode, use the ADC_DiscModeCmd() functions.
- To activate the overrun mode, use the ADC_OverrunModeCmd() functions.
- To activate the calibration mode, use the ADC_StartCalibration() functions.
- To read the ADC converted values, use the ADC_GetConversionValue() function.

DMA for ADC channels features configuration

- To enable the DMA mode for ADC channels group, use the ADC_DMACmd() function.
- To configure the DMA transfer request, use ADC_DMAConfig() function.

3.2.2 Initialization and Configuration functions

This section provides functions allowing to:

1. Initialize and configure the ADC injected and/or regular channels and dual mode.
2. Management of the calibration process
3. ADC Power-on Power-off

4. Single ended or differential mode
5. Enabling the queue of context and the auto delay mode
6. The number of ADC conversions that will be done using the sequencer for regular channel group
7. Enable or disable the ADC peripheral

- ***ADC_DeInit()***
- ***ADC_Init()***
- ***ADC_StructInit()***
- ***ADC_InjectedInit()***
- ***ADC_InjectedStructInit()***
- ***ADC_CommonInit()***
- ***ADC_CommonStructInit()***
- ***ADC_Cmd()***
- ***ADC_StartCalibration()***
- ***ADC_GetCalibrationValue()***
- ***ADC_SetCalibrationValue()***
- ***ADC_SelectCalibrationMode()***
- ***ADC_GetCalibrationStatus()***
- ***ADC_DisableCmd()***
- ***ADC_GetDisableCmdStatus()***
- ***ADC_VoltageRegulatorCmd()***
- ***ADC_SelectDifferentialMode()***
- ***ADC_SelectQueueOfContextMode()***
- ***ADC_AutoDelayCmd()***

3.2.3 Analog Watchdog configuration functions

This section provides functions allowing to configure the 3 Analog Watchdogs (AWDG1, AWDG2 and AWDG3) in the ADC.

A typical configuration Analog Watchdog is done following these steps :

1. The ADC guarded channel(s) is (are) selected using the functions:
   - ADC_AnalogWatchdog1SingleChannelConfig().
   - ADC_AnalogWatchdog2SingleChannelConfig().
   - ADC_AnalogWatchdog3SingleChannelConfig().
2. The Analog watchdog lower and higher threshold are configured using the functions:
   - ADC_AnalogWatchdog1ThresholdsConfig().
   - ADC_AnalogWatchdog2ThresholdsConfig().
   - ADC_AnalogWatchdog3ThresholdsConfig().
3. The Analog watchdog is enabled and configured to enable the check, on one or more channels, using the function:
   - ADC_AnalogWatchdogCmd().

- ***ADC_AnalogWatchdogCmd()***
- ***ADC_AnalogWatchdog1ThresholdsConfig()***
- ***ADC_AnalogWatchdog2ThresholdsConfig()***
- ***ADC_AnalogWatchdog3ThresholdsConfig()***
- ***ADC_AnalogWatchdog1SingleChannelConfig()***
- ***ADC_AnalogWatchdog2SingleChannelConfig()***
- ***ADC_AnalogWatchdog3SingleChannelConfig()***

3.2.4 Temperature Sensor - Vrefint (Internal Reference Voltage) and VBAT management functions

This section provides a function allowing to enable/ disable the internal connections between the ADC and the Vbat/2, Temperature Sensor and the Vrefint source.

A typical configuration to get the Temperature sensor and Vrefint channels voltages is done following these steps :

1. Enable the internal connection of Vbat/2, Temperature sensor and Vrefint sources with the ADC channels using:
   - ADC_TempSensorCmd()
   - ADC_VrefintCmd()
   - ADC_VbatCmd()
2. select the ADC_Channel_TempSensor and/or ADC_Channel_Vrefint and/or ADC_Channel_Vbat using
   - ADC_RegularChannelConfig() or
   - ADC_InjectedChannelConfig() functions
3. Get the voltage values, using:
   - ADC_GetConversionValue() or
   - ADC_GetInjectedConversionValue().

- ***ADC_TempSensorCmd()***
- ***ADC_VrefintCmd()***
- ***ADC_VbatCmd()***

3.2.5 Regular Channels Configuration functions

This section provides functions allowing to manage the ADC regular channels.

To configure a regular sequence of channels use:

1. ADC_RegularChannelConfig() this fuction allows:
   - Configure the rank in the regular group sequencer for each channel
   - Configure the sampling time for each channel
2. ADC_RegularChannelSequencerLengthConfig() to set the length of the regular sequencer

The regular trigger is configured using the following functions:

1. ADC_SelectExternalTrigger()
2. ADC_ExternalTriggerPolarityConfig()

The start and the stop conversion are controlled by:

1. ADC_StartConversion()
2. ADC_StopConversion()

Please Note that the following features for regular channels are configurated using the ADC_Init() function : (++) continuous mode activation (++) Resolution (++) Data Alignement (++) Overrun Mode.

Get the conversion data: This subsection provides an important function in the ADC peripheral since it returns the converted data of the current regular channel. When the Conversion value is read, the EOC Flag is automatically cleared.

To configure the discontinous mode, the following functions should be used:

1. ADC_DiscModeChannelCountConfig() to configure the number of discontinuous channel to be converted.
2. ADC_DiscModeCmd() to enable the discontinuous mode.

To configure and enable/disable the Channel offset use the functions: (++) ADC_SetChannelOffset1() (++) ADC_SetChannelOffset2() (++) ADC_SetChannelOffset3() (++) ADC_SetChannelOffset4() (++) ADC_ChannelOffset1Cmd() (++) ADC_ChannelOffset2Cmd() (++) ADC_ChannelOffset3Cmd() (++) ADC_ChannelOffset4Cmd()

- ***ADC_RegularChannelConfig()***
- ***ADC_RegularChannelSequencerLengthConfig()***
- ***ADC_ExternalTriggerConfig()***
- ***ADC_StartConversion()***
- ***ADC_GetStartConversionStatus()***
- ***ADC_StopConversion()***
- ***ADC_DiscModeChannelCountConfig()***
- ***ADC_DiscModeCmd()***
- ***ADC_GetConversionValue()***
- ***ADC_GetDualModeConversionValue()***
- ***ADC_SetChannelOffset1()***
- ***ADC_SetChannelOffset2()***
- ***ADC_SetChannelOffset3()***
- ***ADC_SetChannelOffset4()***
- ***ADC_ChannelOffset1Cmd()***
- ***ADC_ChannelOffset2Cmd()***
- ***ADC_ChannelOffset3Cmd()***
- ***ADC_ChannelOffset4Cmd()***

3.2.6 Regular Channels DMA Configuration functions

This section provides functions allowing to configure the DMA for ADC regular channels. Since converted regular channel values are stored into a unique data register, it is useful to use DMA for conversion of more than one regular channel. This avoids the loss of the data already stored in the ADC Data register.

1. ADC_DMACmd() function is used to enable the ADC DMA mode, after each conversion of a regular channel, a DMA request is generated.
2. ADC_DMAConfig() function is used to select between the oneshot DMA mode or the circular DMA mode

- ***ADC_DMACmd()***
- ***ADC_DMAConfig()***

3.2.7 Injected channels Configuration functions

This section provide functions allowing to configure the ADC Injected channels, it is composed of 2 sub sections :

1. Configuration functions for Injected channels: This subsection provides functions allowing to configure the ADC injected channels : (+) Configure the rank in the injected group sequencer for each channel (+) Configure the sampling time for each channel (+) Activate the Auto injected Mode (+) Activate the Discontinuous Mode (+) External/software trigger source (+) External trigger edge (+) injected channels sequencer.

2. Get the Specified Injected channel conversion data: This subsection provides an important function in the ADC peripheral since it returns the converted data of the specific injected channel.

This section provide functions allowing to configure the ADC Injected channels, it is composed of 2 sub sections : (#) Configuration functions for Injected channels: This subsection provides functions allowing to configure the ADC injected channels :

- Configure the rank in the injected group sequencer for each channel
- Configure the sampling time for each channel
- Activate the Auto injected Mode
- Activate the Discontinuous Mode
- External/software trigger source
- External trigger edge
- injected channels sequencer. (#) Get the Specified Injected channel conversion data: This subsection provides an important function in the ADC peripheral since it returns the converted data of the specific injected channel.
- ***ADC_StartInjectedConversion()***
- ***ADC_StopInjectedConversion()***
- ***ADC_GetStartInjectedConversionStatus()***
- ***ADC_AutoInjectedConvCmd()***
- ***ADC_InjectedDiscModeCmd()***
- ***ADC_GetInjectedConversionValue()***

3.2.8 Interrupts and flags management functions

This section provides functions allowing to configure the ADC Interrupts, get the status and clear flags and Interrupts pending bits.

The ADC provide 11 Interrupts sources and 11 Flags which can be divided into 3 groups:

1. Flags and Interrupts for ADC regular channels
   a. Flags (+) ADC_FLAG_RDY: ADC Ready flag (+) ADC_FLAG_EOSMP: ADC End of Sampling flag (+) ADC_FLAG_EOC: ADC End of Regular Conversion flag. (+) ADC_FLAG_EOS: ADC End of Regular sequence of Conversions flag (+) ADC_FLAG_OVR: ADC overrun flag
   b. Interrupts (+) ADC_IT_RDY: ADC Ready interrupt source (+) ADC_IT_EOSMP: ADC End of Sampling interrupt source (+) ADC_IT_EOC: ADC End of Regular Conversion interrupt source (+) ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt (+) ADC_IT_OVR: ADC overrun interrupt source
2. Flags and Interrupts for ADC regular channels
   a. Flags (+) ADC_FLAG_JEOC: ADC Ready flag (+) ADC_FLAG_JEOS: ADC End of Sampling flag (+) ADC_FLAG_JQOVF: ADC End of Regular Conversion flag.
   b. Interrupts (+) ADC_IT_JEOC: ADC End of Injected Conversion interrupt source (+) ADC_IT_JEOS: ADC End of Injected sequence of Conversions interrupt source (+) ADC_IT_JQOVF: ADC Injected Context Queue Overflow interrupt source
3. General Flags and Interrupts for the ADC
   a. Flags (+) ADC_FLAG_AWD1: ADC Analog watchdog 1 flag (+) ADC_FLAG_AWD2: ADC Analog watchdog 2 flag (+) ADC_FLAG_AWD3: ADC Analog watchdog 3 flag
   b. Flags (+) ADC_IT_AWD1: ADC Analog watchdog 1 interrupt source (+) ADC_IT_AWD2: ADC Analog watchdog 2 interrupt source (+) ADC_IT_AWD3: ADC Analog watchdog 3 interrupt source
4. Flags for ADC dual mode

a. Flags for Master (+) ADC_FLAG_MSTRDY: ADC master Ready (ADRDY) flag (+) ADC_FLAG_MSTEOSMP: ADC master End of Sampling flag (+) ADC_FLAG_MSTEOC: ADC master End of Regular Conversion flag (+) ADC_FLAG_MSTEOS: ADC master End of Regular sequence of Conversions flag (+) ADC_FLAG_MSTOVR: ADC master overrun flag (+) ADC_FLAG_MSTJEOC: ADC master End of Injected Conversion flag (+) ADC_FLAG_MSTJEOS: ADC master End of Injected sequence of Conversions flag (+) ADC_FLAG_MSTAWD1: ADC master Analog watchdog 1 flag (+) ADC_FLAG_MSTAWD2: ADC master Analog watchdog 2 flag (+) ADC_FLAG_MSTAWD3: ADC master Analog watchdog 3 flag (+) ADC_FLAG_MSTJQOVF: ADC master Injected Context Queue Overflow flag

b. Flags for Slave (+) ADC_FLAG_SLVRDY: ADC slave Ready (ADRDY) flag (+) ADC_FLAG_SLVEOSMP: ADC slave End of Sampling flag (+) ADC_FLAG_SLVEOC: ADC slave End of Regular Conversion flag (+) ADC_FLAG_SLVEOS: ADC slave End of Regular sequence of Conversions flag (+) ADC_FLAG_SLVOVR: ADC slave overrun flag (+) ADC_FLAG_SLVJEOC: ADC slave End of Injected Conversion flag (+) ADC_FLAG_SLVJEOS: ADC slave End of Injected sequence of Conversions flag (+) ADC_FLAG_SLVAWD1: ADC slave Analog watchdog 1 flag (+) ADC_FLAG_SLVAWD2: ADC slave Analog watchdog 2 flag (+) ADC_FLAG_SLVAWD3: ADC slave Analog watchdog 3 flag (+) ADC_FLAG_SLVJQOVF: ADC slave Injected Context Queue Overflow flag The user should identify which mode will be used in his application to manage the ADC controller events: Polling mode or Interrupt mode. In the Polling Mode it is advised to use the following functions: - ADC_GetFlagStatus() : to check if flags events occur. - ADC_ClearFlag() : to clear the flags events. In the Interrupt Mode it is advised to use the following functions: - ADC_ITConfig() : to enable or disable the interrupt source. - ADC_GetITStatus() : to check if Interrupt occurs. - ADC_ClearITPendingBit() : to clear the Interrupt pending Bit (corresponding Flag).

The ADC provide 11 Interrupts sources and 11 Flags which can be divided into 3 groups: (#) Flags and Interrupts for ADC regular channels (##)Flags

- ADC_FLAG_RDY: ADC Ready flag
- ADC_FLAG_EOSMP: ADC End of Sampling flag
- ADC_FLAG_EOC: ADC End of Regular Conversion flag.
- ADC_FLAG_EOS: ADC End of Regular sequence of Conversions flag
- ADC_FLAG_OVR: ADC overrun flag
  a. Interrupts
- ADC_IT_RDY: ADC Ready interrupt source
- ADC_IT_EOSMP: ADC End of Sampling interrupt source
- ADC_IT_EOC: ADC End of Regular Conversion interrupt source
- ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt
- ADC_IT_OVR: ADC overrun interrupt source (#) Flags and Interrupts for ADC regular channels
  a. Flags
- ADC_FLAG_JEOC: ADC Ready flag
- ADC_FLAG_JEOS: ADC End of Sampling flag
- ADC_FLAG_JQOVF: ADC End of Regular Conversion flag.
  a. Interrupts
- ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
- ADC_IT_JEOS: ADC End of Injected sequence of Conversions interrupt source
- ADC_IT_JQOVF: ADC Injected Context Queue Overflow interrupt source (#) General Flags and Interrupts for the ADC
  a. Flags
- ADC_FLAG_AWD1: ADC Analog watchdog 1 flag

- ADC_FLAG_AWD2: ADC Analog watchdog 2 flag
- ADC_FLAG_AWD3: ADC Analog watchdog 3 flag

a. Flags

- ADC_IT_AWD1: ADC Analog watchdog 1 interrupt source
- ADC_IT_AWD2: ADC Analog watchdog 2 interrupt source
- ADC_IT_AWD3: ADC Analog watchdog 3 interrupt source (#) Flags for ADC dual mode

a. Flags for Master

- ADC_FLAG_MSTRDY: ADC master Ready (ADRDY) flag
- ADC_FLAG_MSTEOSMP: ADC master End of Sampling flag
- ADC_FLAG_MSTEOC: ADC master End of Regular Conversion flag
- ADC_FLAG_MSTEOS: ADC master End of Regular sequence of Conversions flag
- ADC_FLAG_MSTOVR: ADC master overrun flag
- ADC_FLAG_MSTJEOC: ADC master End of Injected Conversion flag
- ADC_FLAG_MSTJEOS: ADC master End of Injected sequence of Conversions flag
- ADC_FLAG_MSTAWD1: ADC master Analog watchdog 1 flag
- ADC_FLAG_MSTAWD2: ADC master Analog watchdog 2 flag
- ADC_FLAG_MSTAWD3: ADC master Analog watchdog 3 flag
- ADC_FLAG_MSTJQOVF: ADC master Injected Context Queue Overflow flag

a. Flags for Slave

- ADC_FLAG_SLVRDY: ADC slave Ready (ADRDY) flag
- ADC_FLAG_SLVEOSMP: ADC slave End of Sampling flag
- ADC_FLAG_SLVEOC: ADC slave End of Regular Conversion flag
- ADC_FLAG_SLVEOS: ADC slave End of Regular sequence of Conversions flag
- ADC_FLAG_SLVOVR: ADC slave overrun flag
- ADC_FLAG_SLVJEOC: ADC slave End of Injected Conversion flag
- ADC_FLAG_SLVJEOS: ADC slave End of Injected sequence of Conversions flag
- ADC_FLAG_SLVAWD1: ADC slave Analog watchdog 1 flag
- ADC_FLAG_SLVAWD2: ADC slave Analog watchdog 2 flag
- ADC_FLAG_SLVAWD3: ADC slave Analog watchdog 3 flag
- ADC_FLAG_SLVJQOVF: ADC slave Injected Context Queue Overflow flag The user should identify which mode will be used in his application to manage the ADC controller events: Polling mode or Interrupt mode. In the Polling Mode it is advised to use the following functions: - ADC_GetFlagStatus() : to check if flags events occur. - ADC_ClearFlag() : to clear the flags events. In the Interrupt Mode it is advised to use the following functions: - ADC_ITConfig() : to enable or disable the interrupt source. - ADC_GetITStatus() : to check if Interrupt occurs. - ADC_ClearITPendingBit() : to clear the Interrupt pending Bit (corresponding Flag).
- ***ADC_ITConfig()***
- ***ADC_GetFlagStatus()***
- ***ADC_ClearFlag()***
- ***ADC_GetCommonFlagStatus()***
- ***ADC_ClearCommonFlag()***
- ***ADC_GetITStatus()***
- ***ADC_ClearITPendingBit()***

3.2.9 Initialization and Configuration functions

3.2.9.1 ADC_DeInit

Function Name	**void ADC_DeInit (*ADC_TypeDef** * ADCx)**
Function Description	Deinitializes the ADCx peripheral registers to their default reset values.
Parameters	• **ADCx :** where x can be 1, 2,3 or 4 to select the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.9.2 ADC_Init

Function Name	**void ADC_Init (*ADC_TypeDef** * ADCx, *ADC_InitTypeDef** * ADC_InitStruct)**
Function Description	Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_InitStruct :** pointer to an ADC_InitTypeDef structure that contains the configuration information for the specified ADC peripheral.
Return values	• None.
Notes	• None.

3.2.9.3 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 an ADC_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

3.2.9.4 ADC_InjectedInit

Function Name	**void ADC_InjectedInit (*ADC_TypeDef** * ADCx, *ADC_InjectedInitTypeDef** * ADC_InjectedInitStruct)**
Function Description	Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_InjectInitStruct :** pointer to an ADC_InjecInitTypeDef structure that contains the configuration information for the specified ADC injected channel.
Return values	• None.
Notes	• None.

3.2.9.5 ADC_InjectedStructInit

Function Name	**void ADC_InjectedStructInit (*ADC_InjectedInitTypeDef** * ADC_InjectedInitStruct)**
Function Description	Fills each ADC_InjectedInitStruct member with its default value.
Parameters	• **ADC_InjectedInitStruct :** : pointer to an ADC_InjectedInitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

3.2.9.6 ADC_CommonInit

Function Name	**void ADC_CommonInit (*ADC_TypeDef** * ADCx, *ADC_CommonInitTypeDef** * ADC_CommonInitStruct)**
Function Description	Initializes the ADCs peripherals according to the specified parameters in the ADC_CommonInitStruct.
Parameters	• **ADCx :** where x can be 1 or 4 to select the ADC peripheral. • **ADC_CommonInitStruct :** pointer to an ADC_CommonInitTypeDef structure that contains the

configuration information for All ADCs peripherals.

Return values	• None.
Notes	• None.

3.2.9.7 ADC_CommonStructInit

Function Name	**void ADC_CommonStructInit (*ADC_CommonInitTypeDef** * ADC_CommonInitStruct)**
Function Description	Fills each ADC_CommonInitStruct member with its default value.
Parameters	• **ADC_CommonInitStruct :** pointer to an ADC_CommonInitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

3.2.9.8 ADC_Cmd

Function Name	**void ADC_Cmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified ADC peripheral.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the ADCx peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.9.9 ADC_StartCalibration

Function Name	**void ADC_StartCalibration (*ADC_TypeDef** * ADCx)**
Function Description	Starts the selected ADC calibration process.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.9.10 ADC_GetCalibrationValue

Function Name	**uint32_t ADC_GetCalibrationValue (*ADC_TypeDef** * ADCx)**
Function Description	Returns the ADCx calibration value.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.9.11 ADC_SetCalibrationValue

Function Name	**void ADC_SetCalibrationValue (*ADC_TypeDef** * ADCx, uint32_t ADC_Calibration)**
Function Description	Sets the ADCx calibration register.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.9.12 ADC_SelectCalibrationMode

Function Name	**void ADC_SelectCalibrationMode (*ADC_TypeDef** * ADCx, uint32_t ADC_CalibrationMode)**
Function Description	Select the ADC calibration mode.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_CalibrationMode :** the ADC calibration mode. This parameter can be one of the following values: – ***ADC_CalibrationMode_Single :*** to select the calibration for single channel – ***ADC_CalibrationMode_Differential :*** to select the calibration for differential channel
Return values	• None.
Notes	• None.

3.2.9.13 ADC_GetCalibrationStatus

Function Name	**FlagStatus ADC_GetCalibrationStatus (*ADC_TypeDef** * ADCx)**
Function Description	Gets the selected ADC calibration status.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• **The new state of ADC calibration (SET or RESET).**
Notes	• None.

3.2.9.14 ADC_DisableCmd

Function Name	**void ADC_DisableCmd (*ADC_TypeDef** * ADCx)**
Function Description	ADC Disable Command.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.

Return values	• None.
Notes	• None.

3.2.9.15 ADC_GetDisableCmdStatus

Function Name	**FlagStatus ADC_GetDisableCmdStatus (*ADC_TypeDef** ADCx)**
Function Description	Gets the selected ADC disable command Status.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• **The new state of ADC ADC disable command (SET or RESET).**
Notes	• None.

3.2.9.16 ADC_VoltageRegulatorCmd

Function Name	**void ADC_VoltageRegulatorCmd (*ADC_TypeDef** ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified ADC Voltage Regulator.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the ADCx Voltage Regulator. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.9.17 ADC_SelectDifferentialMode

Function Name	**void ADC_SelectDifferentialMode (*ADC_TypeDef** ADCx,**

uint8_t ADC_Channel, *FunctionalState* NewState)

Function Description	Selectes the differential mode for a specific channel.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_Channel :** the ADC channel to configure for the analog watchdog. This parameter can be one of the following values: – ***ADC_Channel_1 :*** ADC Channel1 selected – ***ADC_Channel_2 :*** ADC Channel2 selected – ***ADC_Channel_3 :*** ADC Channel3 selected – ***ADC_Channel_4 :*** ADC Channel4 selected – ***ADC_Channel_5 :*** ADC Channel5 selected – ***ADC_Channel_6 :*** ADC Channel6 selected – ***ADC_Channel_7 :*** ADC Channel7 selected – ***ADC_Channel_8 :*** ADC Channel8 selected – ***ADC_Channel_9 :*** ADC Channel9 selected – ***ADC_Channel_10 :*** ADC Channel10 selected – ***ADC_Channel_11 :*** ADC Channel11 selected – ***ADC_Channel_12 :*** ADC Channel12 selected – ***ADC_Channel_13 :*** ADC Channel13 selected – ***ADC_Channel_14 :*** ADC Channel14 selected
Return values	• None.
Notes	• : Channel 15, 16 and 17 are fixed to single-ended inputs mode.

3.2.9.18 ADC_SelectQueueOfContextMode

Function Name	**void ADC_SelectQueueOfContextMode (*ADC_TypeDef* * ADCx, *FunctionalState* NewState)**
Function Description	Selects the Queue Of Context Mode for injected channels.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the Queue Of Context Mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.9.19 ADC_AutoDelayCmd

Function Name	**void ADC_AutoDelayCmd (*ADC_TypeDef** * **ADCx, *FunctionalState* NewState)**
Function Description	Selects the ADC Delayed Conversion Mode.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the ADC Delayed Conversion Mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.10 Analog Watchdog configuration functions

3.2.10.1 ADC_AnalogWatchdogCmd

Function Name	**void ADC_AnalogWatchdogCmd (*ADC_TypeDef** * **ADCx, uint32_t ADC_AnalogWatchdog)**
Function Description	Enables or disables the analog watchdog on single/all regular or injected channels.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_AnalogWatchdog :** the ADC analog watchdog configuration. This parameter can be one of the following values: – ***ADC_AnalogWatchdog_SingleRegEnable :*** Analog watchdog on a single regular channel – ***ADC_AnalogWatchdog_SingleInjecEnable :*** Analog watchdog on a single injected channel – ***ADC_AnalogWatchdog_SingleRegOrInjecEnable :*** Analog watchdog on a single regular or injected channel – ***ADC_AnalogWatchdog_AllRegEnable :*** Analog watchdog on all regular channel – ***ADC_AnalogWatchdog_AllInjecEnable :*** Analog watchdog on all injected channel – ***ADC_AnalogWatchdog_AllRegAllInjecEnable :*** Analog watchdog on all regular and injected channels – ***ADC_AnalogWatchdog_None :*** No channel guarded by the analog watchdog
Return values	• None.
Notes	• None.

3.2.10.2 ADC_AnalogWatchdog1ThresholdsConfig

Function Name	**void ADC_AnalogWatchdog1ThresholdsConfig (*ADC_TypeDef** * **ADCx, uint16_t HighThreshold, uint16_t LowThreshold)**
Function Description	Configures the high and low thresholds of the analog watchdog1.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **HighThreshold :** the ADC analog watchdog High threshold value. This parameter must be a 12bit value. • **LowThreshold :** the ADC analog watchdog Low threshold value. This parameter must be a 12bit value.
Return values	• None.
Notes	• None.

3.2.10.3 ADC_AnalogWatchdog2ThresholdsConfig

Function Name	**void ADC_AnalogWatchdog2ThresholdsConfig (*ADC_TypeDef** * **ADCx, uint8_t HighThreshold, uint8_t LowThreshold)**
Function Description	Configures the high and low thresholds of the analog watchdog2.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **HighThreshold :** the ADC analog watchdog High threshold value. This parameter must be a 8bit value. • **LowThreshold :** the ADC analog watchdog Low threshold value. This parameter must be a 8bit value.
Return values	• None.
Notes	• None.

3.2.10.4 ADC_AnalogWatchdog3ThresholdsConfig

Function Name	**void ADC_AnalogWatchdog3ThresholdsConfig (*ADC_TypeDef** * **ADCx, uint8_t HighThreshold, uint8_t LowThreshold)**
Function Description	Configures the high and low thresholds of the analog watchdog3.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **HighThreshold :** the ADC analog watchdog High threshold value. This parameter must be a 8bit value. • **LowThreshold :** the ADC analog watchdog Low threshold value. This parameter must be a 8bit value.
Return values	• None.
Notes	• None.

3.2.10.5 ADC_AnalogWatchdog1SingleChannelConfig

Function Name	**void ADC_AnalogWatchdog1SingleChannelConfig (*ADC_TypeDef** * **ADCx, uint8_t ADC_Channel)**
Function Description	Configures the analog watchdog 2 guarded single channel.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_Channel :** the ADC channel to configure for the analog watchdog. This parameter can be one of the following values: – ***ADC_Channel_1 :*** ADC Channel1 selected – ***ADC_Channel_2 :*** ADC Channel2 selected – ***ADC_Channel_3 :*** ADC Channel3 selected – ***ADC_Channel_4 :*** ADC Channel4 selected – ***ADC_Channel_5 :*** ADC Channel5 selected – ***ADC_Channel_6 :*** ADC Channel6 selected – ***ADC_Channel_7 :*** ADC Channel7 selected – ***ADC_Channel_8 :*** ADC Channel8 selected – ***ADC_Channel_9 :*** ADC Channel9 selected – ***ADC_Channel_10 :*** ADC Channel10 selected – ***ADC_Channel_11 :*** ADC Channel11 selected – ***ADC_Channel_12 :*** ADC Channel12 selected – ***ADC_Channel_13 :*** ADC Channel13 selected – ***ADC_Channel_14 :*** ADC Channel14 selected – ***ADC_Channel_15 :*** ADC Channel15 selected – ***ADC_Channel_16 :*** ADC Channel16 selected – ***ADC_Channel_17 :*** ADC Channel17 selected

– ***ADC_Channel_18 :*** ADC Channel18 selected

Return values
- None.

Notes
- None.

3.2.10.6 ADC_AnalogWatchdog2SingleChannelConfig

Function Name
void ADC_AnalogWatchdog2SingleChannelConfig (*ADC_TypeDef ADCx, uint8_t ADC_Channel)**

Function Description
Configures the analog watchdog 2 guarded single channel.

Parameters
- **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
- **ADC_Channel :** the ADC channel to configure for the analog watchdog. This parameter can be one of the following values:
  – ***ADC_Channel_1 :*** ADC Channel1 selected
  – ***ADC_Channel_2 :*** ADC Channel2 selected
  – ***ADC_Channel_3 :*** ADC Channel3 selected
  – ***ADC_Channel_4 :*** ADC Channel4 selected
  – ***ADC_Channel_5 :*** ADC Channel5 selected
  – ***ADC_Channel_6 :*** ADC Channel6 selected
  – ***ADC_Channel_7 :*** ADC Channel7 selected
  – ***ADC_Channel_8 :*** ADC Channel8 selected
  – ***ADC_Channel_9 :*** ADC Channel9 selected
  – ***ADC_Channel_10 :*** ADC Channel10 selected
  – ***ADC_Channel_11 :*** ADC Channel11 selected
  – ***ADC_Channel_12 :*** ADC Channel12 selected
  – ***ADC_Channel_13 :*** ADC Channel13 selected
  – ***ADC_Channel_14 :*** ADC Channel14 selected
  – ***ADC_Channel_15 :*** ADC Channel15 selected
  – ***ADC_Channel_16 :*** ADC Channel16 selected
  – ***ADC_Channel_17 :*** ADC Channel17 selected
  – ***ADC_Channel_18 :*** ADC Channel18 selected

Return values
- None.

Notes
- None.

3.2.10.7 ADC_AnalogWatchdog3SingleChannelConfig

Function Name	**void ADC_AnalogWatchdog3SingleChannelConfig (*ADC_TypeDef** * **ADCx, uint8_t ADC_Channel)**
Function Description	Configures the analog watchdog 3 guarded single channel.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_Channel :** the ADC channel to configure for the analog watchdog. This parameter can be one of the following values: – ***ADC_Channel_1 :*** ADC Channel1 selected – ***ADC_Channel_2 :*** ADC Channel2 selected – ***ADC_Channel_3 :*** ADC Channel3 selected – ***ADC_Channel_4 :*** ADC Channel4 selected – ***ADC_Channel_5 :*** ADC Channel5 selected – ***ADC_Channel_6 :*** ADC Channel6 selected – ***ADC_Channel_7 :*** ADC Channel7 selected – ***ADC_Channel_8 :*** ADC Channel8 selected – ***ADC_Channel_9 :*** ADC Channel9 selected – ***ADC_Channel_10 :*** ADC Channel10 selected – ***ADC_Channel_11 :*** ADC Channel11 selected – ***ADC_Channel_12 :*** ADC Channel12 selected – ***ADC_Channel_13 :*** ADC Channel13 selected – ***ADC_Channel_14 :*** ADC Channel14 selected – ***ADC_Channel_15 :*** ADC Channel15 selected – ***ADC_Channel_16 :*** ADC Channel16 selected – ***ADC_Channel_17 :*** ADC Channel17 selected – ***ADC_Channel_18 :*** ADC Channel18 selected
Return values	• None.
Notes	• None.

3.2.11 Temperature Sensor- Vrefint (Internal Reference Voltage) and VBAT management function

3.2.11.1 ADC_TempSensorCmd

Function Name	**void ADC_TempSensorCmd (*ADC_TypeDef** * **ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the temperature sensor channel.
Parameters	• **ADCx :** where x can be 1 or 4 to select the ADC peripheral. • **NewState :** new state of the temperature sensor. This

parameter can be: ENABLE or DISABLE.

Return values	• None.
Notes	• None.

3.2.11.2 ADC_VrefintCmd

Function Name	**void ADC_VrefintCmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the Vrefint channel.
Parameters	• **ADCx :** where x can be 1 or 4 to select the ADC peripheral. • **NewState :** new state of the Vrefint. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.11.3 ADC_VbatCmd

Function Name	**void ADC_VbatCmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the Vbat channel.
Parameters	• **ADCx :** where x can be 1 or 4 to select the ADC peripheral. • **NewState :** new state of the Vbat. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.12 Regular Channels Configuration functions

3.2.12.1 ADC_RegularChannelConfig

Function Name	**void ADC_RegularChannelConfig (*ADC_TypeDef** * ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)**
Function Description	Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_Channel :** the ADC channel to configure. This parameter can be one of the following values: – ***ADC_Channel_1 :*** ADC Channel1 selected – ***ADC_Channel_2 :*** ADC Channel2 selected – ***ADC_Channel_3 :*** ADC Channel3 selected – ***ADC_Channel_4 :*** ADC Channel4 selected – ***ADC_Channel_5 :*** ADC Channel5 selected – ***ADC_Channel_6 :*** ADC Channel6 selected – ***ADC_Channel_7 :*** ADC Channel7 selected – ***ADC_Channel_8 :*** ADC Channel8 selected – ***ADC_Channel_9 :*** ADC Channel9 selected – ***ADC_Channel_10 :*** ADC Channel10 selected – ***ADC_Channel_11 :*** ADC Channel11 selected – ***ADC_Channel_12 :*** ADC Channel12 selected – ***ADC_Channel_13 :*** ADC Channel13 selected – ***ADC_Channel_14 :*** ADC Channel14 selected – ***ADC_Channel_15 :*** ADC Channel15 selected – ***ADC_Channel_16 :*** ADC Channel16 selected – ***ADC_Channel_17 :*** ADC Channel17 selected – ***ADC_Channel_18 :*** ADC Channel18 selected • **Rank :** The rank in the regular group sequencer. This parameter must be between 1 to 16. • **ADC_SampleTime :** The sample time value to be set for the selected channel. This parameter can be one of the following values: – ***ADC_SampleTime_1Cycles5 :*** Sample time equal to 1.5 cycles – ***ADC_SampleTime_2Cycles5 :*** Sample time equal to 2.5 cycles – ***ADC_SampleTime_4Cycles5 :*** Sample time equal to 4.5 cycles – ***ADC_SampleTime_7Cycles5 :*** Sample time equal to 7.5 cycles – ***ADC_SampleTime_19Cycles5 :*** Sample time equal to 19.5 cycles – ***ADC_SampleTime_61Cycles5 :*** Sample time equal to 61.5 cycles

- ***ADC_SampleTime_181Cycles5 :*** Sample time equal to 181.5 cycles
- ***ADC_SampleTime_601Cycles5 :*** Sample time equal to 601.5 cycles

Return values
- None.

Notes
- None.

3.2.12.2 ADC_RegularChannelSequencerLengthConfig

Function Name **void ADC_RegularChannelSequencerLengthConfig (*ADC_TypeDef** * ADCx, uint8_t SequencerLength)**

Function Description Sets the ADC regular channel sequence lenght.

Parameters
- **ADCx :** where x can be 1, 2 or 3 to select the ADC peripheral.
- **SequenceLength :** The Regular sequence length. This parameter must be between 1 to 16. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

3.2.12.3 ADC_ExternalTriggerConfig

Function Name **void ADC_ExternalTriggerConfig (*ADC_TypeDef** * ADCx, uint16_t ADC_ExternalTrigConvEvent, uint16_t ADC_ExternalTrigEventEdge)**

Function Description External Trigger Enable and Polarity Selection for regular channels.

Parameters
- **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
- **ADC_ExternalTrigConvEvent :** ADC external Trigger source. This parameter can be one of the following values:
  - ***ADC_ExternalTrigger_Event0 :*** External trigger event 0
  - ***ADC_ExternalTrigger_Event1 :*** External trigger event 1
  - ***ADC_ExternalTrigger_Event2 :*** External trigger event

2
- ***ADC_ExternalTrigger_Event3 :*** External trigger event 3
- ***ADC_ExternalTrigger_Event4 :*** External trigger event 4
- ***ADC_ExternalTrigger_Event5 :*** External trigger event 5
- ***ADC_ExternalTrigger_Event6 :*** External trigger event 6
- ***ADC_ExternalTrigger_Event7 :*** External trigger event 7
- ***ADC_ExternalTrigger_Event8 :*** External trigger event 8
- ***ADC_ExternalTrigger_Event9 :*** External trigger event 9
- ***ADC_ExternalTrigger_Event10 :*** External trigger event 10
- ***ADC_ExternalTrigger_Event11 :*** External trigger event 11
- ***ADC_ExternalTrigger_Event12 :*** External trigger event 12
- ***ADC_ExternalTrigger_Event13 :*** External trigger event 13
- ***ADC_ExternalTrigger_Event14 :*** External trigger event 14
- ***ADC_ExternalTrigger_Event15 :*** External trigger event 15

- **ADC_ExternalTrigEventEdge :** ADC external Trigger Polarity. This parameter can be one of the following values:
  - ***ADC_ExternalTrigEventEdge_OFF :*** Hardware trigger detection disabled (conversions can be launched by software)
  - ***ADC_ExternalTrigEventEdge_RisingEdge :*** Hardware trigger detection on the rising edge
  - ***ADC_ExternalTrigEventEdge_FallingEdge :*** Hardware trigger detection on the falling edge
  - ***ADC_ExternalTrigEventEdge_BothEdge :*** Hardware trigger detection on both the rising and falling edges

Return values
- None.

Notes
- None.

3.2.12.4 ADC_StartConversion

Function Name **void ADC_StartConversion (*ADC_TypeDef ** ADCx)**

Function Description Enables or disables the selected ADC start conversion .

Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.12.5 ADC_GetStartConversionStatus

Function Name	**FlagStatus ADC_GetStartConversionStatus (*ADC_TypeDef** ADCx)**
Function Description	Gets the selected ADC start conversion Status.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• **The new state of ADC start conversion (SET or RESET).**
Notes	• None.

3.2.12.6 ADC_StopConversion

Function Name	**void ADC_StopConversion (*ADC_TypeDef** ADCx)**
Function Description	Stops the selected ADC ongoing conversion.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.12.7 ADC_DiscModeChannelCountConfig

Function Name	**void ADC_DiscModeChannelCountConfig (*ADC_TypeDef** ADCx, uint8_t Number)**

Function Description	Configures the discontinuous mode for the selected ADC regular group channel.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **Number :** specifies the discontinuous mode regular channel count value. This number must be between 1 and 8.
Return values	• None.
Notes	• None.

3.2.12.8 ADC_DiscModeCmd

Function Name	**void ADC_DiscModeCmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the discontinuous mode on regular group channel for the specified ADC.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the selected ADC discontinuous mode on regular group channel. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.12.9 ADC_GetConversionValue

Function Name	**uint16_t ADC_GetConversionValue (*ADC_TypeDef** * ADCx)**
Function Description	Returns the last ADCx conversion result data for regular channel.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• **The Data conversion value.**
Notes	• None.

3.2.12.10 ADC_GetDualModeConversionValue

Function Name	**uint32_t ADC_GetDualModeConversionValue (*ADC_TypeDef* * ADCx)**
Function Description	Returns the last ADC1, ADC2, ADC3 and ADC4 regular conversions results data in the selected dual mode.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• **The Data conversion value.**
Notes	• In dual mode, the value returned by this function is as following Data[15:0] : these bits contain the regular data of the Master ADC. Data[31:16]: these bits contain the regular data of the Slave ADC.

3.2.12.11 ADC_SetChannelOffset1

Function Name	**void ADC_SetChannelOffset1 (*ADC_TypeDef* * ADCx, uint8_t ADC_Channel, uint16_t Offset)**
Function Description	Set the ADC channels conversion value offset1.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_Channel :** the ADC channel to configure. This parameter can be one of the following values: – ***ADC_Channel_1 :*** ADC Channel1 selected – ***ADC_Channel_2 :*** ADC Channel2 selected – ***ADC_Channel_3 :*** ADC Channel3 selected – ***ADC_Channel_4 :*** ADC Channel4 selected – ***ADC_Channel_5 :*** ADC Channel5 selected – ***ADC_Channel_6 :*** ADC Channel6 selected – ***ADC_Channel_7 :*** ADC Channel7 selected – ***ADC_Channel_8 :*** ADC Channel8 selected – ***ADC_Channel_9 :*** ADC Channel9 selected – ***ADC_Channel_10 :*** ADC Channel10 selected – ***ADC_Channel_11 :*** ADC Channel11 selected – ***ADC_Channel_12 :*** ADC Channel12 selected – ***ADC_Channel_13 :*** ADC Channel13 selected – ***ADC_Channel_14 :*** ADC Channel14 selected – ***ADC_Channel_15 :*** ADC Channel15 selected – ***ADC_Channel_16 :*** ADC Channel16 selected

- ***ADC_Channel_17 :*** ADC Channel17 selected
- ***ADC_Channel_18 :*** ADC Channel18 selected

- **Offset :** the offset value for the selected ADC Channel This parameter must be a 12bit value.

Return values
- None.

Notes
- None.

3.2.12.12 ADC_SetChannelOffset2

Function Name **void ADC_SetChannelOffset2 (*ADC_TypeDef* * ADCx, uint8_t ADC_Channel, uint16_t Offset)**

Function Description Set the ADC channels conversion value offset2.

Parameters
- **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
- **ADC_Channel :** the ADC channel to configure. This parameter can be one of the following values:
  - ***ADC_Channel_1 :*** ADC Channel1 selected
  - ***ADC_Channel_2 :*** ADC Channel2 selected
  - ***ADC_Channel_3 :*** ADC Channel3 selected
  - ***ADC_Channel_4 :*** ADC Channel4 selected
  - ***ADC_Channel_5 :*** ADC Channel5 selected
  - ***ADC_Channel_6 :*** ADC Channel6 selected
  - ***ADC_Channel_7 :*** ADC Channel7 selected
  - ***ADC_Channel_8 :*** ADC Channel8 selected
  - ***ADC_Channel_9 :*** ADC Channel9 selected
  - ***ADC_Channel_10 :*** ADC Channel10 selected
  - ***ADC_Channel_11 :*** ADC Channel11 selected
  - ***ADC_Channel_12 :*** ADC Channel12 selected
  - ***ADC_Channel_13 :*** ADC Channel13 selected
  - ***ADC_Channel_14 :*** ADC Channel14 selected
  - ***ADC_Channel_15 :*** ADC Channel15 selected
  - ***ADC_Channel_16 :*** ADC Channel16 selected
  - ***ADC_Channel_17 :*** ADC Channel17 selected
  - ***ADC_Channel_18 :*** ADC Channel18 selected
- **Offset :** the offset value for the selected ADC Channel This parameter must be a 12bit value.

Return values
- None.

Notes
- None.

3.2.12.13 ADC_SetChannelOffset3

Function Name	**void ADC_SetChannelOffset3 (*ADC_TypeDef** * **ADCx, uint8_t ADC_Channel, uint16_t Offset)**
Function Description	Set the ADC channels conversion value offset3.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_Channel :** the ADC channel to configure. This parameter can be one of the following values: – ***ADC_Channel_1 :*** ADC Channel1 selected – ***ADC_Channel_2 :*** ADC Channel2 selected – ***ADC_Channel_3 :*** ADC Channel3 selected – ***ADC_Channel_4 :*** ADC Channel4 selected – ***ADC_Channel_5 :*** ADC Channel5 selected – ***ADC_Channel_6 :*** ADC Channel6 selected – ***ADC_Channel_7 :*** ADC Channel7 selected – ***ADC_Channel_8 :*** ADC Channel8 selected – ***ADC_Channel_9 :*** ADC Channel9 selected – ***ADC_Channel_10 :*** ADC Channel10 selected – ***ADC_Channel_11 :*** ADC Channel11 selected – ***ADC_Channel_12 :*** ADC Channel12 selected – ***ADC_Channel_13 :*** ADC Channel13 selected – ***ADC_Channel_14 :*** ADC Channel14 selected – ***ADC_Channel_15 :*** ADC Channel15 selected – ***ADC_Channel_16 :*** ADC Channel16 selected – ***ADC_Channel_17 :*** ADC Channel17 selected – ***ADC_Channel_18 :*** ADC Channel18 selected • **Offset :** the offset value for the selected ADC Channel This parameter must be a 12bit value.
Return values	• None.
Notes	• None.

3.2.12.14 ADC_SetChannelOffset4

Function Name	**void ADC_SetChannelOffset4 (*ADC_TypeDef** * **ADCx, uint8_t ADC_Channel, uint16_t Offset)**
Function Description	Set the ADC channels conversion value offset4.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_Channel :** the ADC channel to configure. This parameter can be one of the following values:

- ***ADC_Channel_1 :*** ADC Channel1 selected
- ***ADC_Channel_2 :*** ADC Channel2 selected
- ***ADC_Channel_3 :*** ADC Channel3 selected
- ***ADC_Channel_4 :*** ADC Channel4 selected
- ***ADC_Channel_5 :*** ADC Channel5 selected
- ***ADC_Channel_6 :*** ADC Channel6 selected
- ***ADC_Channel_7 :*** ADC Channel7 selected
- ***ADC_Channel_8 :*** ADC Channel8 selected
- ***ADC_Channel_9 :*** ADC Channel9 selected
- ***ADC_Channel_10 :*** ADC Channel10 selected
- ***ADC_Channel_11 :*** ADC Channel11 selected
- ***ADC_Channel_12 :*** ADC Channel12 selected
- ***ADC_Channel_13 :*** ADC Channel13 selected
- ***ADC_Channel_14 :*** ADC Channel14 selected
- ***ADC_Channel_15 :*** ADC Channel15 selected
- ***ADC_Channel_16 :*** ADC Channel16 selected
- ***ADC_Channel_17 :*** ADC Channel17 selected
- ***ADC_Channel_18 :*** ADC Channel18 selected

- **Offset :** the offset value for the selected ADC Channel This parameter must be a 12bit value.

Return values
- None.

Notes
- None.

3.2.12.15 ADC_ChannelOffset1Cmd

Function Name **void ADC_ChannelOffset1Cmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**

Function Description Enables or disables the Offset1.

Parameters
- **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
- **NewState :** new state of the ADCx offset1. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

3.2.12.16 ADC_ChannelOffset2Cmd

Function Name	**void ADC_ChannelOffset2Cmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the Offset2.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the ADCx offset2. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.12.17 ADC_ChannelOffset3Cmd

Function Name	**void ADC_ChannelOffset3Cmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the Offset3.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the ADCx offset3. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.12.18 ADC_ChannelOffset4Cmd

Function Name	**void ADC_ChannelOffset4Cmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the Offset4.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the ADCx offset4. This parameter

can be: ENABLE or DISABLE.

Return values	• None.
Notes	• None.

3.2.13 Regular Channels DMA Configuration functions

3.2.13.1 ADC_DMACmd

Function Name	**void ADC_DMACmd (*ADC_TypeDef** * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified ADC DMA request.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the selected ADC DMA transfer. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.13.2 ADC_DMAConfig

Function Name	**void ADC_DMAConfig (*ADC_TypeDef** * ADCx, uint32_t ADC_DMAMode)**
Function Description	Enables or disables the specified ADC DMA request.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the selected ADC DMA transfer. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.14 Injected channels Configuration functions

3.2.14.1 ADC_StartInjectedConversion

Function Name	**void ADC_StartInjectedConversion (*ADC_TypeDef** * ADCx)**
Function Description	Enables or disables the selected ADC start of the injected channels conversion.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the selected ADC software start injected conversion. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.14.2 ADC_StopInjectedConversion

Function Name	**void ADC_StopInjectedConversion (*ADC_TypeDef** * ADCx)**
Function Description	Stops the selected ADC ongoing injected conversion.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• None.
Notes	• None.

3.2.14.3 ADC_GetStartInjectedConversionStatus

Function Name	**FlagStatus ADC_GetStartInjectedConversionStatus (*ADC_TypeDef** * ADCx)**
Function Description	Gets the selected ADC Software start injected conversion Status.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
Return values	• **The new state of ADC start injected conversion (SET or**

RESET).

Notes	• None.

3.2.14.4 ADC_AutoInjectedConvCmd

Function Name	**void ADC_AutoInjectedConvCmd (*ADC_TypeDef* * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the selected ADC automatic injected group conversion after regular one.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the selected ADC auto injected conversion This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.14.5 ADC_InjectedDiscModeCmd

Function Name	**void ADC_InjectedDiscModeCmd (*ADC_TypeDef* * ADCx, *FunctionalState* NewState)**
Function Description	Enables or disables the discontinuous mode for injected group channel for the specified ADC.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **NewState :** new state of the selected ADC discontinuous mode on injected group channel. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

3.2.14.6 ADC_GetInjectedConversionValue

Function Name	**uint16_t ADC_GetInjectedConversionValue (*ADC_TypeDef** * ADCx, uint8_t ADC_InjectedChannel)**
Function Description	Returns the ADC injected channel conversion result.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_InjectedChannel :** the converted ADC injected channel. This parameter can be one of the following values: – ***ADC_InjectedChannel_1 :*** Injected Channel1 selected – ***ADC_InjectedChannel_2 :*** Injected Channel2 selected – ***ADC_InjectedChannel_3 :*** Injected Channel3 selected – ***ADC_InjectedChannel_4 :*** Injected Channel4 selected
Return values	• **The Data conversion value.**
Notes	• None.

3.2.15 Interrupts and flags management functions

3.2.15.1 ADC_ITConfig

Function Name	**void ADC_ITConfig (*ADC_TypeDef** * ADCx, uint32_t ADC_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified ADC interrupts.
Parameters	• **ADCx :** where x can be 1, 2 or 3 to select the ADC peripheral. • **ADC_IT :** specifies the ADC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values: – ***ADC_IT_RDY :*** ADC Ready (ADRDY) interrupt source – ***ADC_IT_EOSMP :*** ADC End of Sampling interrupt source – ***ADC_IT_EOC :*** ADC End of Regular Conversion interrupt source – ***ADC_IT_EOS :*** ADC End of Regular sequence of Conversions interrupt source – ***ADC_IT_OVR :*** ADC overrun interrupt source – ***ADC_IT_JEOC :*** ADC End of Injected Conversion interrupt source – ***ADC_IT_JEOS :*** ADC End of Injected sequence of Conversions interrupt source – ***ADC_IT_AWD1 :*** ADC Analog watchdog 1 interrupt source

  - ***ADC_IT_AWD2 :*** ADC Analog watchdog 2 interrupt source
  - ***ADC_IT_AWD3 :*** ADC Analog watchdog 3 interrupt source
  - ***ADC_IT_JQOVF :*** ADC Injected Context Queue Overflow interrupt source
- **NewState :** new state of the specified ADC interrupts. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

3.2.15.2 ADC_GetFlagStatus

Function Name
FlagStatus ADC_GetFlagStatus (*ADC_TypeDef ADCx, uint32_t ADC_FLAG)**

Function Description
Checks whether the specified ADC flag is set or not.

Parameters
- **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
- **ADC_FLAG :** specifies the flag to check. This parameter can be one of the following values:
  - ***ADC_FLAG_RDY :*** ADC Ready (ADRDY) flag
  - ***ADC_FLAG_EOSMP :*** ADC End of Sampling flag
  - ***ADC_FLAG_EOC :*** ADC End of Regular Conversion flag
  - ***ADC_FLAG_EOS :*** ADC End of Regular sequence of Conversions flag
  - ***ADC_FLAG_OVR :*** ADC overrun flag
  - ***ADC_FLAG_JEOC :*** ADC End of Injected Conversion flag
  - ***ADC_FLAG_JEOS :*** ADC End of Injected sequence of Conversions flag
  - ***ADC_FLAG_AWD1 :*** ADC Analog watchdog 1 flag
  - ***ADC_FLAG_AWD2 :*** ADC Analog watchdog 2 flag
  - ***ADC_FLAG_AWD3 :*** ADC Analog watchdog 3 flag
  - ***ADC_FLAG_JQOVF :*** ADC Injected Context Queue Overflow flag

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

Notes
- None.

3.2.15.3 ADC_ClearFlag

Function Name	**void ADC_ClearFlag (*ADC_TypeDef** * ADCx, uint32_t ADC_FLAG)**
Function Description	Clears the ADCx's pending flags.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_FLAG :** specifies the flag to clear. This parameter can be any combination of the following values: – ***ADC_FLAG_RDY :*** ADC Ready (ADRDY) flag – ***ADC_FLAG_EOSMP :*** ADC End of Sampling flag – ***ADC_FLAG_EOC :*** ADC End of Regular Conversion flag – ***ADC_FLAG_EOS :*** ADC End of Regular sequence of Conversions flag – ***ADC_FLAG_OVR :*** ADC overrun flag – ***ADC_FLAG_JEOC :*** ADC End of Injected Conversion flag – ***ADC_FLAG_JEOS :*** ADC End of Injected sequence of Conversions flag – ***ADC_FLAG_AWD1 :*** ADC Analog watchdog 1 flag – ***ADC_FLAG_AWD2 :*** ADC Analog watchdog 2 flag – ***ADC_FLAG_AWD3 :*** ADC Analog watchdog 3 flag – ***ADC_FLAG_JQOVF :*** ADC Injected Context Queue Overflow flag
Return values	• None.
Notes	• None.

3.2.15.4 ADC_GetCommonFlagStatus

Function Name	**FlagStatus ADC_GetCommonFlagStatus (*ADC_TypeDef** * ADCx, uint32_t ADC_FLAG)**
Function Description	Checks whether the specified ADC flag is set or not.
Parameters	• **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral. • **ADC_FLAG :** specifies the master or slave flag to check. This parameter can be one of the following values: – ***ADC_FLAG_MSTRDY :*** ADC master Ready (ADRDY) flag – ***ADC_FLAG_MSTEOSMP :*** ADC master End of Sampling flag

- ***ADC_FLAG_MSTEOC :*** ADC master End of Regular Conversion flag
- ***ADC_FLAG_MSTEOS :*** ADC master End of Regular sequence of Conversions flag
- ***ADC_FLAG_MSTOVR :*** ADC master overrun flag
- ***ADC_FLAG_MSTJEOC :*** ADC master End of Injected Conversion flag
- ***ADC_FLAG_MSTJEOS :*** ADC master End of Injected sequence of Conversions flag
- ***ADC_FLAG_MSTAWD1 :*** ADC master Analog watchdog 1 flag
- ***ADC_FLAG_MSTAWD2 :*** ADC master Analog watchdog 2 flag
- ***ADC_FLAG_MSTAWD3 :*** ADC master Analog watchdog 3 flag
- ***ADC_FLAG_MSTJQOVF :*** ADC master Injected Context Queue Overflow flag
- ***ADC_FLAG_SLVRDY :*** ADC slave Ready (ADRDY) flag
- ***ADC_FLAG_SLVEOSMP :*** ADC slave End of Sampling flag
- ***ADC_FLAG_SLVEOC :*** ADC slave End of Regular Conversion flag
- ***ADC_FLAG_SLVEOS :*** ADC slave End of Regular sequence of Conversions flag
- ***ADC_FLAG_SLVOVR :*** ADC slave overrun flag
- ***ADC_FLAG_SLVJEOC :*** ADC slave End of Injected Conversion flag
- ***ADC_FLAG_SLVJEOS :*** ADC slave End of Injected sequence of Conversions flag
- ***ADC_FLAG_SLVAWD1 :*** ADC slave Analog watchdog 1 flag
- ***ADC_FLAG_SLVAWD2 :*** ADC slave Analog watchdog 2 flag
- ***ADC_FLAG_SLVAWD3 :*** ADC slave Analog watchdog 3 flag
- ***ADC_FLAG_SLVJQOVF :*** ADC slave Injected Context Queue Overflow flag

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

Notes
- None.

3.2.15.5 ADC_ClearCommonFlag

Function Name **void ADC_ClearCommonFlag (*ADC_TypeDef* * ADCx, uint32_t ADC_FLAG)**

Function Description

Clears the ADCx's pending flags.

Parameters

- **ADCx :** where x can be 1, 2, 3 or 4 to select the ADC peripheral.
- **ADC_FLAG :** specifies the master or slave flag to clear. This parameter can be one of the following values:
  - ***ADC_FLAG_MSTRDY :*** ADC master Ready (ADRDY) flag
  - ***ADC_FLAG_MSTEOSMP :*** ADC master End of Sampling flag
  - ***ADC_FLAG_MSTEOC :*** ADC master End of Regular Conversion flag
  - ***ADC_FLAG_MSTEOS :*** ADC master End of Regular sequence of Conversions flag
  - ***ADC_FLAG_MSTOVR :*** ADC master overrun flag
  - ***ADC_FLAG_MSTJEOC :*** ADC master End of Injected Conversion flag
  - ***ADC_FLAG_MSTJEOS :*** ADC master End of Injected sequence of Conversions flag
  - ***ADC_FLAG_MSTAWD1 :*** ADC master Analog watchdog 1 flag
  - ***ADC_FLAG_MSTAWD2 :*** ADC master Analog watchdog 2 flag
  - ***ADC_FLAG_MSTAWD3 :*** ADC master Analog watchdog 3 flag
  - ***ADC_FLAG_MSTJQOVF :*** ADC master Injected Context Queue Overflow flag
  - ***ADC_FLAG_SLVRDY :*** ADC slave Ready (ADRDY) flag
  - ***ADC_FLAG_SLVEOSMP :*** ADC slave End of Sampling flag
  - ***ADC_FLAG_SLVEOC :*** ADC slave End of Regular Conversion flag
  - ***ADC_FLAG_SLVEOS :*** ADC slave End of Regular sequence of Conversions flag
  - ***ADC_FLAG_SLVOVR :*** ADC slave overrun flag
  - ***ADC_FLAG_SLVJEOC :*** ADC slave End of Injected Conversion flag
  - ***ADC_FLAG_SLVJEOS :*** ADC slave End of Injected sequence of Conversions flag
  - ***ADC_FLAG_SLVAWD1 :*** ADC slave Analog watchdog 1 flag
  - ***ADC_FLAG_SLVAWD2 :*** ADC slave Analog watchdog 2 flag
  - ***ADC_FLAG_SLVAWD3 :*** ADC slave Analog watchdog 3 flag
  - ***ADC_FLAG_SLVJQOVF :*** ADC slave Injected Context Queue Overflow flag

Return values

- None.

Notes

- None.

3.2.15.6 ADC_GetITStatus

Function Name	**ITStatus ADC_GetITStatus (*ADC_TypeDef** * ADCx, uint32_t ADC_IT)**
Function Description	Checks whether the specified ADC interrupt has occurred or not.
Parameters	• **ADCx :** where x can be 1, 2 or 3 to select the ADC peripheral. • **ADC_IT :** specifies the ADC interrupt source to check. This parameter can be one of the following values: – ***ADC_IT_RDY :*** ADC Ready (ADRDY) interrupt source – ***ADC_IT_EOSMP :*** ADC End of Sampling interrupt source – ***ADC_IT_EOC :*** ADC End of Regular Conversion interrupt source – ***ADC_IT_EOS :*** ADC End of Regular sequence of Conversions interrupt source – ***ADC_IT_OVR :*** ADC overrun interrupt source – ***ADC_IT_JEOC :*** ADC End of Injected Conversion interrupt source – ***ADC_IT_JEOS :*** ADC End of Injected sequence of Conversions interrupt source – ***ADC_IT_AWD1 :*** ADC Analog watchdog 1 interrupt source – ***ADC_IT_AWD2 :*** ADC Analog watchdog 2 interrupt source – ***ADC_IT_AWD3 :*** ADC Analog watchdog 3 interrupt source – ***ADC_IT_JQOVF :*** ADC Injected Context Queue Overflow interrupt source
Return values	• **The new state of ADC_IT (SET or RESET).**
Notes	• None.

3.2.15.7 ADC_ClearITPendingBit

Function Name	**void ADC_ClearITPendingBit (*ADC_TypeDef** * ADCx, uint32_t ADC_IT)**
Function Description	Clears the ADCx's interrupt pending bits.
Parameters	• **ADCx :** where x can be 1, 2 or 3 to select the ADC peripheral. • **ADC_IT :** specifies the ADC interrupt pending bit to clear.

This parameter can be any combination of the following values:

- ***ADC_IT_RDY :*** ADC Ready (ADRDY) interrupt source
- ***ADC_IT_EOSMP :*** ADC End of Sampling interrupt source
- ***ADC_IT_EOC :*** ADC End of Regular Conversion interrupt source
- ***ADC_IT_EOS :*** ADC End of Regular sequence of Conversions interrupt source
- ***ADC_IT_OVR :*** ADC overrun interrupt source
- ***ADC_IT_JEOC :*** ADC End of Injected Conversion interrupt source
- ***ADC_IT_JEOS :*** ADC End of Injected sequence of Conversions interrupt source
- ***ADC_IT_AWD1 :*** ADC Analog watchdog 1 interrupt source
- ***ADC_IT_AWD2 :*** ADC Analog watchdog 2 interrupt source
- ***ADC_IT_AWD3 :*** ADC Analog watchdog 3 interrupt source
- ***ADC_IT_JQOVF :*** ADC Injected Context Queue Overflow interrupt source

Return values
- None.

Notes
- None.

3.3 ADC Firmware driver defines

3.3.1 ADC

ADC

ADC_analog_watchdog_selection

- #define: ***ADC_AnalogWatchdog_SingleRegEnable ((uint32_t)0x00C00000)***

ADC Analog watchdog single regular mode

- #define: ***ADC_AnalogWatchdog_SingleInjecEnable ((uint32_t)0x01400000)***

ADC Analog watchdog single injected mode

- #define: ***ADC_AnalogWatchdog_SingleRegOrInjecEnable ((uint32_t)0x01C00000)***

ADC Analog watchdog single regular or injected mode

- #define: ***ADC_AnalogWatchdog_AllRegEnable ((uint32_t)0x00800000)***

ADC Analog watchdog all regular mode

- #define: ***ADC_AnalogWatchdog_AllInjecEnable ((uint32_t)0x01000000)***

ADC Analog watchdog all injected mode

- #define: ***ADC_AnalogWatchdog_AllRegAllInjecEnable ((uint32_t)0x01800000)***

ADC Analog watchdog all regular and all injected mode

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

ADC Analog watchdog off

ADC_AutoInjecMode

- #define: ***ADC_AutoInjec_Enable ((uint32_t)0x02000000)***

ADC Auto injected Mode enable

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

ADC Auto injected Mode disable

ADC_Calibration_Mode_definition

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

ADC Calibration for single ended channel

- #define: ***ADC_CalibrationMode_Differential ((uint32_t)0x40000000)***

ADC Calibration for differential channel

ADC_channels

- #define: ***ADC_Channel_1 ((uint8_t)0x01)***

ADC Channel 1

- #define: ***ADC_Channel_2 ((uint8_t)0x02)***

ADC Channel 2

- #define: ***ADC_Channel_3 ((uint8_t)0x03)***

ADC Channel 3

- #define: ***ADC_Channel_4 ((uint8_t)0x04)***

ADC Channel 4

- #define: ***ADC_Channel_5 ((uint8_t)0x05)***

ADC Channel 5

- #define: ***ADC_Channel_6 ((uint8_t)0x06)***

ADC Channel 6

- #define: ***ADC_Channel_7 ((uint8_t)0x07)***

ADC Channel 7

- #define: ***ADC_Channel_8 ((uint8_t)0x08)***

ADC Channel 8

- #define: ***ADC_Channel_9 ((uint8_t)0x09)***

ADC Channel 9

- #define: ***ADC_Channel_10 ((uint8_t)0x0A)***

ADC Channel 10

- #define: ***ADC_Channel_11 ((uint8_t)0x0B)***

ADC Channel 11

- #define: ***ADC_Channel_12 ((uint8_t)0x0C)***

ADC Channel 12

- #define: ***ADC_Channel_13 ((uint8_t)0x0D)***

ADC Channel 13

- #define: ***ADC_Channel_14 ((uint8_t)0x0E)***

ADC Channel 14

- #define: ***ADC_Channel_15 ((uint8_t)0x0F)***

ADC Channel 15

- #define: ***ADC_Channel_16 ((uint8_t)0x10)***

ADC Channel 16

- #define: ***ADC_Channel_17 ((uint8_t)0x11)***

ADC Channel 17

- #define: ***ADC_Channel_18 ((uint8_t)0x12)***

ADC Channel 18

- #define: ***ADC_Channel_TempSensor ((uint8_t)ADC_Channel_16)***

- #define: ***ADC_Channel_Vrefint ((uint8_t)ADC_Channel_18)***

- #define: ***ADC_Channel_Vbat ((uint8_t)ADC_Channel_17)***

ADC_Clock

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

ADC Asynchronous clock mode

- #define: ***ADC_Clock_SynClkModeDiv1 ((uint32_t)0x00010000)***

Synchronous clock mode divided by 1

- #define: ***ADC_Clock_SynClkModeDiv2 ((uint32_t)0x00020000)***

Synchronous clock mode divided by 2

- #define: ***ADC_Clock_SynClkModeDiv4 ((uint32_t)0x00030000)***

Synchronous clock mode divided by 4

ADC_Common_flags_definition

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

ADC Master Ready (ADRDY) flag

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

ADC Master End of Sampling flag

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

ADC Master End of Regular Conversion flag

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

ADC Master End of Regular sequence of Conversions flag

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

ADC Master overrun flag

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

ADC Master End of Injected Conversion flag

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

ADC Master End of Injected sequence of Conversions flag

- #define: ***ADC_FLAG_MSTAWD1 ((uint32_t)0x00000080)***

ADC Master Analog watchdog 1 flag

- #define: ***ADC_FLAG_MSTAWD2 ((uint32_t)0x00000100)***

ADC Master Analog watchdog 2 flag

- #define: ***ADC_FLAG_MSTAWD3 ((uint32_t)0x00000200)***

ADC Master Analog watchdog 3 flag

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

ADC Master Injected Context Queue Overflow flag

- #define: ***ADC_FLAG_SLVRDY ((uint32_t)0x00010000)***

ADC Slave Ready (ADRDY) flag

- #define: ***ADC_FLAG_SLVEOSMP ((uint32_t)0x00020000)***

ADC Slave End of Sampling flag

- #define: ***ADC_FLAG_SLVEOC ((uint32_t)0x00040000)***

ADC Slave End of Regular Conversion flag

- #define: ***ADC_FLAG_SLVEOS ((uint32_t)0x00080000)***

ADC Slave End of Regular sequence of Conversions flag

- #define: ***ADC_FLAG_SLVOVR ((uint32_t)0x00100000)***

ADC Slave overrun flag

- #define: ***ADC_FLAG_SLVJEOC ((uint32_t)0x00200000)***

ADC Slave End of Injected Conversion flag

- #define: ***ADC_FLAG_SLVJEOS ((uint32_t)0x00400000)***

ADC Slave End of Injected sequence of Conversions flag

- #define: ***ADC_FLAG_SLVAWD1 ((uint32_t)0x00800000)***

ADC Slave Analog watchdog 1 flag

- #define: ***ADC_FLAG_SLVAWD2 ((uint32_t)0x01000000)***

ADC Slave Analog watchdog 2 flag

- #define: ***ADC_FLAG_SLVAWD3 ((uint32_t)0x02000000)***

ADC Slave Analog watchdog 3 flag

- #define: ***ADC_FLAG_SLVJQOVF ((uint32_t)0x04000000)***

ADC Slave Injected Context Queue Overflow flag

ADC_ContinuousConvMode

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

ADC continuous conversion mode enable

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

ADC continuous conversion mode disable

ADC_data_align

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

ADC Data alignment right

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

ADC Data alignment left

ADC_Direct_memory_access_mode_for_multi_mode

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

DMA mode disabled

- #define: ***ADC_DMAAccessMode_1 ((uint32_t)0x00008000)***

DMA mode enabled for 12 and 10-bit resolution (6 bit)

- #define: ***ADC_DMAAccessMode_2 ((uint32_t)0x0000C000)***

DMA mode enabled for 8 and 6-bit resolution (8bit)

ADC_DMA_Mode_definition

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

ADC DMA Oneshot mode

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

ADC DMA circular mode

ADC_external_trigger_edge_for_Injected_channels_conversion

- #define: ***ADC_ExternalTrigInjecEventEdge_None ((uint16_t)0x0000)***

ADC No external trigger for regular conversion

- #define: ***ADC_ExternalTrigInjecEventEdge_RisingEdge ((uint16_t)0x0040)***

ADC external trigger rising edge for injected conversion

- #define: ***ADC_ExternalTrigInjecEventEdge_FallingEdge ((uint16_t)0x0080)***

ADC external trigger falling edge for injected conversion

- #define: ***ADC_ExternalTrigInjecEventEdge_BothEdge ((uint16_t)0x00C0)***

ADC external trigger both edges for injected conversion

ADC_external_trigger_edge_for_regular_channels_conversion

- #define: ***ADC_ExternalTrigEventEdge_None ((uint16_t)0x0000)***

ADC No external trigger for regular conversion

- #define: ***ADC_ExternalTrigEventEdge_RisingEdge ((uint16_t)0x0400)***

ADC external trigger rising edge for regular conversion

- #define: ***ADC_ExternalTrigEventEdge_FallingEdge ((uint16_t)0x0800)***

ADC ADC external trigger falling edge for regular conversion

- #define: ***ADC_ExternalTrigEventEdge_BothEdge ((uint16_t)0x0C00)***

ADC ADC external trigger both edges for regular conversion

ADC_external_trigger_sources_for_Injected_channels_conversion

- #define: ***ADC_ExternalTrigInjecConvEvent_0 ((uint16_t)0x0000)***

ADC external trigger for injected conversion event 0

- #define: ***ADC_ExternalTrigInjecConvEvent_1 ((uint16_t)0x0004)***

ADC external trigger for injected conversion event 1

- #define: ***ADC_ExternalTrigInjecConvEvent_2 ((uint16_t)0x0008)***

ADC external trigger for injected conversion event 2

- #define: ***ADC_ExternalTrigInjecConvEvent_3 ((uint16_t)0x000C)***

ADC external trigger for injected conversion event 3

- #define: ***ADC_ExternalTrigInjecConvEvent_4 ((uint16_t)0x0010)***

ADC external trigger for injected conversion event 4

- #define: ***ADC_ExternalTrigInjecConvEvent_5 ((uint16_t)0x0014)***

ADC external trigger for injected conversion event 5

- #define: ***ADC_ExternalTrigInjecConvEvent_6 ((uint16_t)0x0018)***

ADC external trigger for injected conversion event 6

- #define: ***ADC_ExternalTrigInjecConvEvent_7 ((uint16_t)0x001C)***

ADC external trigger for injected conversion event 7

- #define: ***ADC_ExternalTrigInjecConvEvent_8 ((uint16_t)0x0020)***

ADC external trigger for injected conversion event 8

- #define: ***ADC_ExternalTrigInjecConvEvent_9 ((uint16_t)0x0024)***

ADC external trigger for injected conversion event 9

- #define: ***ADC_ExternalTrigInjecConvEvent_10 ((uint16_t)0x0028)***

ADC external trigger for injected conversion event 10

- #define: ***ADC_ExternalTrigInjecConvEvent_11 ((uint16_t)0x002C)***

ADC external trigger for injected conversion event 11

- #define: ***ADC_ExternalTrigInjecConvEvent_12 ((uint16_t)0x0030)***

ADC external trigger for injected conversion event 12

- #define: ***ADC_ExternalTrigInjecConvEvent_13 ((uint16_t)0x0034)***

ADC external trigger for injected conversion event 13

- #define: ***ADC_ExternalTrigInjecConvEvent_14 ((uint16_t)0x0038)***

ADC external trigger for injected conversion event 14

- #define: ***ADC_ExternalTrigInjecConvEvent_15 ((uint16_t)0x003C)***

ADC external trigger for injected conversion event 15

ADC_external_trigger_sources_for_regular_channels_conversion

- #define: ***ADC_ExternalTrigConvEvent_0 ((uint16_t)0x0000)***

ADC external trigger event 0

- #define: ***ADC_ExternalTrigConvEvent_1 ((uint16_t)0x0040)***

ADC external trigger event 1

- #define: ***ADC_ExternalTrigConvEvent_2 ((uint16_t)0x0080)***

ADC external trigger event 2

- #define: ***ADC_ExternalTrigConvEvent_3 ((uint16_t)0x00C0)***

ADC external trigger event 3

- #define: ***ADC_ExternalTrigConvEvent_4 ((uint16_t)0x0100)***

ADC external trigger event 4

- #define: ***ADC_ExternalTrigConvEvent_5 ((uint16_t)0x0140)***

ADC external trigger event 5

- #define: ***ADC_ExternalTrigConvEvent_6 ((uint16_t)0x0180)***

ADC external trigger event 6

- #define: ***ADC_ExternalTrigConvEvent_7 ((uint16_t)0x01C0)***

ADC external trigger event 7

- #define: ***ADC_ExternalTrigConvEvent_8 ((uint16_t)0x0200)***

ADC external trigger event 8

- #define: ***ADC_ExternalTrigConvEvent_9 ((uint16_t)0x0240)***

ADC external trigger event 9

- #define: ***ADC_ExternalTrigConvEvent_10 ((uint16_t)0x0280)***

ADC external trigger event 10

- #define: ***ADC_ExternalTrigConvEvent_11 ((uint16_t)0x02C0)***

ADC external trigger event 11

- #define: ***ADC_ExternalTrigConvEvent_12 ((uint16_t)0x0300)***

ADC external trigger event 12

- #define: ***ADC_ExternalTrigConvEvent_13 ((uint16_t)0x0340)***

ADC external trigger event 13

- #define: ***ADC_ExternalTrigConvEvent_14 ((uint16_t)0x0380)***

ADC external trigger event 14

- #define: ***ADC_ExternalTrigConvEvent_15 ((uint16_t)0x03C0)***

ADC external trigger event 15

ADC_flags_definition

- #define: ***ADC_FLAG_RDY ((uint16_t)0x0001)***

ADC Ready (ADRDY) flag

- #define: ***ADC_FLAG_EOSMP ((uint16_t)0x0002)***

ADC End of Sampling flag

- #define: ***ADC_FLAG_EOC ((uint16_t)0x0004)***

ADC End of Regular Conversion flag

- #define: ***ADC_FLAG_EOS ((uint16_t)0x0008)***

ADC End of Regular sequence of Conversions flag

- #define: ***ADC_FLAG_OVR ((uint16_t)0x0010)***

ADC overrun flag

- #define: ***ADC_FLAG_JEOC ((uint16_t)0x0020)***

ADC End of Injected Conversion flag

- #define: ***ADC_FLAG_JEOS ((uint16_t)0x0040)***

ADC End of Injected sequence of Conversions flag

- #define: ***ADC_FLAG_AWD1 ((uint16_t)0x0080)***

ADC Analog watchdog 1 flag

- #define: ***ADC_FLAG_AWD2 ((uint16_t)0x0100)***

ADC Analog watchdog 2 flag

- #define: ***ADC_FLAG_AWD3 ((uint16_t)0x0200)***

ADC Analog watchdog 3 flag

- #define: ***ADC_FLAG_JQOVF ((uint16_t)0x0400)***

ADC Injected Context Queue Overflow flag

ADC_injected_channel_selection

- #define: ***ADC_InjectedChannel_1 ADC_Channel_1***

ADC Injected channel 1

- #define: ***ADC_InjectedChannel_2 ADC_Channel_2***

ADC Injected channel 2

- #define: ***ADC_InjectedChannel_3 ADC_Channel_3***

ADC Injected channel 3

- #define: ***ADC_InjectedChannel_4 ADC_Channel_4***

ADC Injected channel 4

ADC_interrupts_definition

- #define: ***ADC_IT_RDY ((uint16_t)0x0001)***

ADC Ready (ADRDY) interrupt source

- #define: ***ADC_IT_EOSMP ((uint16_t)0x0002)***

ADC End of Sampling interrupt source

- #define: ***ADC_IT_EOC ((uint16_t)0x0004)***

ADC End of Regular Conversion interrupt source

- #define: ***ADC_IT_EOS ((uint16_t)0x0008)***

ADC End of Regular sequence of Conversions interrupt source

- #define: ***ADC_IT_OVR ((uint16_t)0x0010)***

ADC overrun interrupt source

- #define: ***ADC_IT_JEOC ((uint16_t)0x0020)***

ADC End of Injected Conversion interrupt source

- #define: ***ADC_IT_JEOS ((uint16_t)0x0040)***

ADC End of Injected sequence of Conversions interrupt source

- #define: ***ADC_IT_AWD1 ((uint16_t)0x0080)***

ADC Analog watchdog 1 interrupt source

- #define: ***ADC_IT_AWD2 ((uint16_t)0x0100)***

ADC Analog watchdog 2 interrupt source

- #define: ***ADC_IT_AWD3 ((uint16_t)0x0200)***

ADC Analog watchdog 3 interrupt source

- #define: ***ADC_IT_JQOVF ((uint16_t)0x0400)***

ADC Injected Context Queue Overflow interrupt source

ADC_mode

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

ADC independent mode

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

ADC multi ADC mode: Combined Regular simultaneous injected simultaneous mode

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

ADC multi ADC mode: Combined Regular simultaneous Alternate trigger mode

- #define: ***ADC_Mode_InjSimul ((uint32_t)0x00000005)***

ADC multi ADC mode: Injected simultaneous mode

- #define: ***ADC_Mode_RegSimul ((uint32_t)0x00000006)***

ADC multi ADC mode: Regular simultaneous mode

- #define: ***ADC_Mode_Interleave ((uint32_t)0x00000007)***

ADC multi ADC mode: Interleave mode

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

ADC multi ADC mode: Alternate Trigger mode

ADC_OverunMode

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

ADC Overrun Mode enable

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

ADC Overrun Mode disable

ADC_resolution

- #define: ***ADC_Resolution_12b ((uint32_t)0x00000000)***

ADC 12-bit resolution

- #define: ***ADC_Resolution_10b ((uint32_t)0x00000008)***

ADC 10-bit resolution

- #define: ***ADC_Resolution_8b ((uint32_t)0x00000010)***

ADC 8-bit resolution

- #define: ***ADC_Resolution_6b ((uint32_t)0x00000018)***

ADC 6-bit resolution

ADC_sampling_time

- #define: ***ADC_SampleTime_1Cycles5 ((uint8_t)0x00)***

ADC sampling time 1.5 cycle

- #define: ***ADC_SampleTime_2Cycles5 ((uint8_t)0x01)***

ADC sampling time 2.5 cycles

- #define: ***ADC_SampleTime_4Cycles5 ((uint8_t)0x02)***

ADC sampling time 4.5 cycles

- #define: ***ADC_SampleTime_7Cycles5 ((uint8_t)0x03)***

ADC sampling time 7.5 cycles

- #define: ***ADC_SampleTime_19Cycles5 ((uint8_t)0x04)***

ADC sampling time 19.5 cycles

- #define: ***ADC_SampleTime_61Cycles5 ((uint8_t)0x05)***

ADC sampling time 61.5 cycles

- #define: ***ADC_SampleTime_181Cycles5 ((uint8_t)0x06)***

ADC sampling time 181.5 cycles

- #define: ***ADC_SampleTime_601Cycles5 ((uint8_t)0x07)***

ADC sampling time 601.5 cycles

4 Controller area network (bxCAN)

4.1 CAN Firmware driver registers structures

4.1.1 CAN_TypeDef

CAN_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t MCR***
- ***__IO uint32_t MSR***
- ***__IO uint32_t TSR***
- ***__IO uint32_t RF0R***
- ***__IO uint32_t RF1R***
- ***__IO uint32_t IER***
- ***__IO uint32_t ESR***
- ***__IO uint32_t BTR***
- ***uint32_t RESERVED0***
- ***CAN_TxMailBox_TypeDef sTxMailBox***
- ***CAN_FIFOMailBox_TypeDef sFIFOMailBox***
- ***uint32_t RESERVED1***
- ***__IO uint32_t FMR***
- ***__IO uint32_t FM1R***
- ***uint32_t RESERVED2***
- ***__IO uint32_t FS1R***
- ***uint32_t RESERVED3***
- ***__IO uint32_t FFA1R***
- ***uint32_t RESERVED4***
- ***__IO uint32_t FA1R***
- ***uint32_t RESERVED5***
- ***CAN_FilterRegister_TypeDef sFilterRegister***

Field Documentation

- ***__IO uint32_t CAN_TypeDef::MCR***
  - CAN master control register, Address offset: 0x00
- ***__IO uint32_t CAN_TypeDef::MSR***
  - CAN master status register, Address offset: 0x04
- ***__IO uint32_t CAN_TypeDef::TSR***
  - CAN transmit status register, Address offset: 0x08
- ***__IO uint32_t CAN_TypeDef::RF0R***
  - CAN receive FIFO 0 register, Address offset: 0x0C
- ***__IO uint32_t CAN_TypeDef::RF1R***
  - CAN receive FIFO 1 register, Address offset: 0x10
- ***__IO uint32_t CAN_TypeDef::IER***
  - CAN interrupt enable register, Address offset: 0x14
- ***__IO uint32_t CAN_TypeDef::ESR***
  - CAN error status register, Address offset: 0x18

- ***__IO uint32_t CAN_TypeDef::BTR***
  - CAN bit timing register, Address offset: 0x1C
- ***uint32_t CAN_TypeDef::RESERVED0[88]***
  - Reserved, 0x020 - 0x17F
- ***CAN_TxMailBox_TypeDef CAN_TypeDef::sTxMailBox[3]***
  - CAN Tx MailBox, Address offset: 0x180 - 0x1AC
- ***CAN_FIFOMailBox_TypeDef CAN_TypeDef::sFIFOMailBox[2]***
  - CAN FIFO MailBox, Address offset: 0x1B0 - 0x1CC
- ***uint32_t CAN_TypeDef::RESERVED1[12]***
  - Reserved, 0x1D0 - 0x1FF
- ***__IO uint32_t CAN_TypeDef::FMR***
  - CAN filter master register, Address offset: 0x200
- ***__IO uint32_t CAN_TypeDef::FM1R***
  - CAN filter mode register, Address offset: 0x204
- ***uint32_t CAN_TypeDef::RESERVED2***
  - Reserved, 0x208
- ***__IO uint32_t CAN_TypeDef::FS1R***
  - CAN filter scale register, Address offset: 0x20C
- ***uint32_t CAN_TypeDef::RESERVED3***
  - Reserved, 0x210
- ***__IO uint32_t CAN_TypeDef::FFA1R***
  - CAN filter FIFO assignment register, Address offset: 0x214
- ***uint32_t CAN_TypeDef::RESERVED4***
  - Reserved, 0x218
- ***__IO uint32_t CAN_TypeDef::FA1R***
  - CAN filter activation register, Address offset: 0x21C
- ***uint32_t CAN_TypeDef::RESERVED5[8]***
  - Reserved, 0x220-0x23F
- ***CAN_FilterRegister_TypeDef CAN_TypeDef::sFilterRegister[28]***
  - CAN Filter Register, Address offset: 0x240-0x31C

4.1.2 CAN_FIFOMailBox_TypeDef

CAN_FIFOMailBox_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t RIR***
- ***__IO uint32_t RDTR***
- ***__IO uint32_t RDLR***
- ***__IO uint32_t RDHR***

Field Documentation

- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RIR***
  - CAN receive FIFO mailbox identifier register
- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RDTR***
  - CAN receive FIFO mailbox data length control and time stamp register
- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RDLR***

- CAN receive FIFO mailbox data low register

- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RDHR***
  - CAN receive FIFO mailbox data high register

4.1.3 CAN_TxMailBox_TypeDef

CAN_TxMailBox_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t TIR***
- ***__IO uint32_t TDTR***
- ***__IO uint32_t TDLR***
- ***__IO uint32_t TDHR***

Field Documentation

- ***__IO uint32_t CAN_TxMailBox_TypeDef::TIR***
  - CAN TX mailbox identifier register
- ***__IO uint32_t CAN_TxMailBox_TypeDef::TDTR***
  - CAN mailbox data length control and time stamp register
- ***__IO uint32_t CAN_TxMailBox_TypeDef::TDLR***
  - CAN mailbox data low register
- ***__IO uint32_t CAN_TxMailBox_TypeDef::TDHR***
  - CAN mailbox data high register

4.1.4 CAN_FilterRegister_TypeDef

CAN_FilterRegister_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t FR1***
- ***__IO uint32_t FR2***

Field Documentation

- ***__IO uint32_t CAN_FilterRegister_TypeDef::FR1***
  - CAN Filter bank register 1
- ***__IO uint32_t CAN_FilterRegister_TypeDef::FR2***
  - CAN Filter bank register 1

4.1.5 CAN_InitTypeDef

CAN_InitTypeDef is defined in the stm32f30x_can.h

Data Fields

- ***uint16_t CAN_Prescaler***
- ***uint8_t CAN_Mode***
- ***uint8_t CAN_SJW***
- ***uint8_t CAN_BS1***
- ***uint8_t CAN_BS2***
- ***FunctionalState CAN_TTCM***
- ***FunctionalState CAN_ABOM***
- ***FunctionalState CAN_AWUM***
- ***FunctionalState CAN_NART***
- ***FunctionalState CAN_RFLM***
- ***FunctionalState CAN_TXFP***

Field Documentation

- ***uint16_t CAN_InitTypeDef::CAN_Prescaler***
  - Specifies the length of a time quantum. It ranges from 1 to 1024.
- ***uint8_t CAN_InitTypeDef::CAN_Mode***
  - Specifies the CAN operating mode. This parameter can be a value of ***CAN_operating_mode***
- ***uint8_t CAN_InitTypeDef::CAN_SJW***
  - Specifies the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter can be a value of ***CAN_synchronisation_jump_width***
- ***uint8_t CAN_InitTypeDef::CAN_BS1***
  - Specifies the number of time quanta in Bit Segment 1. This parameter can be a value of ***CAN_time_quantum_in_bit_segment_1***
- ***uint8_t CAN_InitTypeDef::CAN_BS2***
  - Specifies the number of time quanta in Bit Segment 2. This parameter can be a value of ***CAN_time_quantum_in_bit_segment_2***
- ***FunctionalState CAN_InitTypeDef::CAN_TTCM***
  - Enable or disable the time triggered communication mode. This parameter can be set either to ENABLE or DISABLE.
- ***FunctionalState CAN_InitTypeDef::CAN_ABOM***
  - Enable or disable the automatic bus-off management. This parameter can be set either to ENABLE or DISABLE.
- ***FunctionalState CAN_InitTypeDef::CAN_AWUM***
  - Enable or disable the automatic wake-up mode. This parameter can be set either to ENABLE or DISABLE.
- ***FunctionalState CAN_InitTypeDef::CAN_NART***
  - Enable or disable the non-automatic retransmission mode. This parameter can be set either to ENABLE or DISABLE.
- ***FunctionalState CAN_InitTypeDef::CAN_RFLM***
  - Enable or disable the Receive FIFO Locked mode. This parameter can be set either to ENABLE or DISABLE.
- ***FunctionalState CAN_InitTypeDef::CAN_TXFP***
  - Enable or disable the transmit FIFO priority. This parameter can be set either to ENABLE or DISABLE.

4.1.6 CAN_FilterInitTypeDef

CAN_FilterInitTypeDef is defined in the stm32f30x_can.h

Data Fields

- ***uint16_t CAN_FilterIdHigh***
- ***uint16_t CAN_FilterIdLow***
- ***uint16_t CAN_FilterMaskIdHigh***
- ***uint16_t CAN_FilterMaskIdLow***
- ***uint16_t CAN_FilterFIFOAssignment***
- ***uint8_t CAN_FilterNumber***
- ***uint8_t CAN_FilterMode***
- ***uint8_t CAN_FilterScale***
- ***FunctionalState CAN_FilterActivation***

Field Documentation

- ***uint16_t CAN_FilterInitTypeDef::CAN_FilterIdHigh***
  - Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter can be a value between 0x0000 and 0xFFFF
- ***uint16_t CAN_FilterInitTypeDef::CAN_FilterIdLow***
  - Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter can be a value between 0x0000 and 0xFFFF
- ***uint16_t CAN_FilterInitTypeDef::CAN_FilterMaskIdHigh***
  - Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter can be a value between 0x0000 and 0xFFFF
- ***uint16_t CAN_FilterInitTypeDef::CAN_FilterMaskIdLow***
  - Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter can be a value between 0x0000 and 0xFFFF
- ***uint16_t CAN_FilterInitTypeDef::CAN_FilterFIFOAssignment***
  - Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of ***CAN_filter_FIFO***
- ***uint8_t CAN_FilterInitTypeDef::CAN_FilterNumber***
  - Specifies the filter which will be initialized. It ranges from 0 to 13.
- ***uint8_t CAN_FilterInitTypeDef::CAN_FilterMode***
  - Specifies the filter mode to be initialized. This parameter can be a value of ***CAN_filter_mode***
- ***uint8_t CAN_FilterInitTypeDef::CAN_FilterScale***
  - Specifies the filter scale. This parameter can be a value of ***CAN_filter_scale***
- ***FunctionalState CAN_FilterInitTypeDef::CAN_FilterActivation***
  - Enable or disable the filter. This parameter can be set either to ENABLE or DISABLE.

4.1.7 CanRxMsg

CanRxMsg is defined in the stm32f30x_can.h

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint8_t IDE***
- ***uint8_t RTR***
- ***uint8_t DLC***
- ***uint8_t Data***
- ***uint8_t FMI***

Field Documentation

- ***uint32_t CanRxMsg::StdId***
  - Specifies the standard identifier. This parameter can be a value between 0 to 0x7FF.
- ***uint32_t CanRxMsg::ExtId***
  - Specifies the extended identifier. This parameter can be a value between 0 to 0x1FFFFFFF.
- ***uint8_t CanRxMsg::IDE***
  - Specifies the type of identifier for the message that will be received. This parameter can be a value of ***CAN_identifier_type***
- ***uint8_t CanRxMsg::RTR***
  - Specifies the type of frame for the received message. This parameter can be a value of ***CAN_remote_transmission_request***
- ***uint8_t CanRxMsg::DLC***
  - Specifies the length of the frame that will be received. This parameter can be a value between 0 to 8
- ***uint8_t CanRxMsg::Data[8]***
  - Contains the data to be received. It ranges from 0 to 0xFF.
- ***uint8_t CanRxMsg::FMI***
  - Specifies the index of the filter the message stored in the mailbox passes through. This parameter can be a value between 0 to 0xFF

4.1.8 CanTxMsg

CanTxMsg is defined in the stm32f30x_can.h

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint8_t IDE***
- ***uint8_t RTR***
- ***uint8_t DLC***
- ***uint8_t Data***

Field Documentation

- ***uint32_t CanTxMsg::StdId***
  - Specifies the standard identifier. This parameter can be a value between 0 to 0x7FF.
- ***uint32_t CanTxMsg::ExtId***
  - Specifies the extended identifier. This parameter can be a value between 0 to 0x1FFFFFFF.
- ***uint8_t CanTxMsg::IDE***
  - Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of ***CAN_identifier_type***
- ***uint8_t CanTxMsg::RTR***
  - Specifies the type of frame for the message that will be transmitted. This parameter can be a value of ***CAN_remote_transmission_request***
- ***uint8_t CanTxMsg::DLC***
  - Specifies the length of the frame that will be transmitted. This parameter can be a value between 0 to 8
- ***uint8_t CanTxMsg::Data[8]***
  - Contains the data to be transmitted. It ranges from 0 to 0xFF.

4.2 CAN Firmware driver API description

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

4.2.1 How to use this driver

1. Enable the CAN controller interface clock using RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN1, ENABLE);
2. CAN pins configuration:
   - Enable the clock for the CAN GPIOs using the following function: RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOx, ENABLE);
   - Connect the involved CAN pins to AF9 using the following function GPIO_PinAFConfig(GPIOx, GPIO_PinSourcex, GPIO_AF_CANx);
   - Configure these CAN pins in alternate function mode by calling the function GPIO_Init();
3. Initialise and configure the CAN using CAN_Init() and CAN_FilterInit() functions.
4. Transmit the desired CAN frame using CAN_Transmit() function.
5. Check the transmission of a CAN frame using CAN_TransmitStatus() function.
6. Cancel the transmission of a CAN frame using CAN_CancelTransmit() function.
7. Receive a CAN frame using CAN_Recieve() function.
8. Release the receive FIFOs using CAN_FIFORelease() function.
9. Return the number of pending received frames using CAN_MessagePending() function.
10. To control CAN events you can use one of the following two methods:
    - Check on CAN flags using the CAN_GetFlagStatus() function.
    - Use CAN interrupts through the function CAN_ITConfig() at initialization phase and CAN_GetITStatus() function into interrupt routines to check if the event has occurred or not. After checking on a flag you should clear it using

CAN_ClearFlag() function. And after checking on an interrupt event you should clear it using CAN_ClearITPendingBit() function.

4.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize the CAN peripherals : Prescaler, operating mode, the maximum number of time quanta to perform resynchronization, the number of time quanta in Bit Segment 1 and 2 and many other modes.
- Configure the CAN reception filter.
- Select the start bank filter for slave CAN.
- Enable or disable the Debug Freeze mode for CAN.
- Enable or disable the CAN Time Trigger Operation communication mode.
- ***CAN_DeInit()***
- ***CAN_Init()***
- ***CAN_FilterInit()***
- ***CAN_StructInit()***
- ***CAN_SlaveStartBank()***
- ***CAN_DBGFreeze()***
- ***CAN_TTComModeCmd()***

4.2.3 CAN Frames Transmission functions

This section provides functions allowing to

- Initiate and transmit a CAN frame message (if there is an empty mailbox).
- Check the transmission status of a CAN Frame.
- Cancel a transmit request.
- ***CAN_Transmit()***
- ***CAN_TransmitStatus()***
- ***CAN_CancelTransmit()***

4.2.4 CAN Frames Reception functions

This section provides functions allowing to

- Receive a correct CAN frame.
- Release a specified receive FIFO (2 FIFOs are available).
- Return the number of the pending received CAN frames.
- ***CAN_Receive()***
- ***CAN_FIFORelease()***
- ***CAN_MessagePending()***

4.2.5 CAN Operation modes functions

This section provides functions allowing to select the CAN Operation modes:

- sleep mode.
- normal mode.
- initialization mode.

- ***CAN_OperatingModeRequest()***
- ***CAN_Sleep()***
- ***CAN_WakeUp()***

4.2.6 CAN Bus Error management functions

This section provides functions allowing to

- Return the CANx's last error code (LEC).
- Return the CANx Receive Error Counter (REC).
- Return the LSB of the 9-bit CANx Transmit Error Counter(TEC).

If TEC is greater than 255, The CAN is in bus-off state.

If REC or TEC are greater than 96, an Error warning flag occurs.

If REC or TEC are greater than 127, an Error Passive Flag occurs.

- ***CAN_GetLastErrorCode()***
- ***CAN_GetReceiveErrorCounter()***
- ***CAN_GetLSBTransmitErrorCounter()***

4.2.7 Interrupts and flags management functions

This section provides functions allowing to configure the CAN Interrupts and to get the status and clear flags and Interrupts pending bits.

The CAN provides 14 Interrupts sources and 15 Flags:

Flags

The 15 flags can be divided on 4 groups:

- Transmit Flags:
  - CAN_FLAG_RQCP0.
  - CAN_FLAG_RQCP1.
  - CAN_FLAG_RQCP2: Request completed MailBoxes 0, 1 and 2 Flags Set when when the last request (transmit or abort) has been performed.
- Receive Flags:
  - CAN_FLAG_FMP0.
  - CAN_FLAG_FMP1: FIFO 0 and 1 Message Pending Flags; Set to signal that messages are pending in the receive FIFO. These Flags are cleared only by hardware.
  - CAN_FLAG_FF0.

  - CAN_FLAG_FF1: FIFO 0 and 1 Full Flags; Set when three messages are stored in the selected FIFO.
  - CAN_FLAG_FOV0.
  - CAN_FLAG_FOV1: FIFO 0 and 1 Overrun Flags; Set when a new message has been received and passed the filter while the FIFO was full.
- Operating Mode Flags:
  - CAN_FLAG_WKU: Wake up Flag; Set to signal that a SOF bit has been detected while the CAN hardware was in Sleep mode.
  - CAN_FLAG_SLAK: Sleep acknowledge Flag; Set to signal that the CAN has entered Sleep Mode.
- Error Flags:
  - CAN_FLAG_EWG: Error Warning Flag; Set when the warning limit has been reached (Receive Error Counter or Transmit Error Counter greater than 96). This Flag is cleared only by hardware.
  - CAN_FLAG_EPV: Error Passive Flag; Set when the Error Passive limit has been reached (Receive Error Counter or Transmit Error Counter greater than 127). This Flag is cleared only by hardware.
  - CAN_FLAG_BOF: Bus-Off Flag; Set when CAN enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255. This Flag is cleared only by hardware.
  - CAN_FLAG_LEC: Last error code Flag; Set If a message has been transferred (reception or transmission) with error, and the error code is hold.

Interrupts

The 14 interrupts can be divided on 4 groups:

- Transmit interrupt:
  - CAN_IT_TME: Transmit mailbox empty Interrupt; If enabled, this interrupt source is pending when no transmit request are pending for Tx mailboxes.
- Receive Interrupts:
  - CAN_IT_FMP0.
  - CAN_IT_FMP1: FIFO 0 and FIFO1 message pending Interrupts; If enabled, these interrupt sources are pending when messages are pending in the receive FIFO. The corresponding interrupt pending bits are cleared only by hardware.
  - CAN_IT_FF0.
  - CAN_IT_FF1: FIFO 0 and FIFO1 full Interrupts; If enabled, these interrupt sources are pending when three messages are stored in the selected FIFO.
  - CAN_IT_FOV0.
  - CAN_IT_FOV1: FIFO 0 and FIFO1 overrun Interrupts; If enabled, these interrupt sources are pending when a new message has been received and passed the filter while the FIFO was full.
- Operating Mode Interrupts:
  - CAN_IT_WKU: Wake-up Interrupt; If enabled, this interrupt source is pending when a SOF bit has been detected while the CAN hardware was in Sleep mode.
  - CAN_IT_SLK: Sleep acknowledge Interrupt: If enabled, this interrupt source is pending when the CAN has entered Sleep Mode.
- Error Interrupts:
  - CAN_IT_EWG: Error warning Interrupt; If enabled, this interrupt source is pending when the warning limit has been reached (Receive Error Counter or Transmit Error Counter=96).
  - CAN_IT_EPV: Error passive Interrupt; If enabled, this interrupt source is pending when the Error Passive limit has been reached (Receive Error Counter or Transmit Error Counter>127).

- CAN_IT_BOF: Bus-off Interrupt; If enabled, this interrupt source is pending when CAN enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255. This Flag is cleared only by hardware.
- CAN_IT_LEC: Last error code Interrupt; If enabled, this interrupt source is pending when a message has been transferred (reception or transmission) with error and the error code is hold.
- CAN_IT_ERR: Error Interrupt; If enabled, this interrupt source is pending when an error condition is pending.

Managing the CAN controller events: The user should identify which mode will be used in his application to manage the CAN controller events: Polling mode or Interrupt mode.

- In the Polling Mode it is advised to use the following functions:
  - CAN_GetFlagStatus() : to check if flags events occur.
  - CAN_ClearFlag() : to clear the flags events.
- In the Interrupt Mode it is advised to use the following functions:
  - CAN_ITConfig() : to enable or disable the interrupt source.
  - CAN_GetITStatus() : to check if Interrupt occurs.
  - CAN_ClearITPendingBit() : to clear the Interrupt pending Bit (corresponding Flag). This function has no impact on CAN_IT_FMP0 and CAN_IT_FMP1 Interrupts pending bits since there are cleared only by hardware.
- ***CAN_ITConfig()***
- ***CAN_GetFlagStatus()***
- ***CAN_ClearFlag()***
- ***CAN_GetITStatus()***
- ***CAN_ClearITPendingBit()***

4.2.8 Initialization and Configuration functions

4.2.8.1 CAN_DeInit

Function Name	**void CAN_DeInit (*CAN_TypeDef** * CANx)**
Function Description	Deinitializes the CAN peripheral registers to their default reset values.
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral.
Return values	• **None.**
Notes	• None.

4.2.8.2 CAN_Init

Function Name	**uint8_t CAN_Init (*CAN_TypeDef** * CANx, *CAN_InitTypeDef** * CAN_InitStruct)**
Function Description	Initializes the CAN peripheral according to the specified

parameters in the CAN_InitStruct.

Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral. • **CAN_InitStruct :** pointer to a CAN_InitTypeDef structure that contains the configuration information for the CAN peripheral.
Return values	• **Constant indicates initialization succeed which will be CAN_InitStatus_Failed or CAN_InitStatus_Success.**
Notes	• None.

4.2.8.3 CAN_FilterInit

Function Name	**void CAN_FilterInit (*CAN_FilterInitTypeDef** * CAN_FilterInitStruct)**
Function Description	Configures the CAN reception filter according to the specified parameters in the CAN_FilterInitStruct.
Parameters	• **CAN_FilterInitStruct :** pointer to a CAN_FilterInitTypeDef structure that contains the configuration information.
Return values	• None.
Notes	• None.

4.2.8.4 CAN_StructInit

Function Name	**void CAN_StructInit (*CAN_InitTypeDef** * CAN_InitStruct)**
Function Description	Fills each CAN_InitStruct member with its default value.
Parameters	• **CAN_InitStruct :** pointer to a CAN_InitTypeDef structure which ill be initialized.
Return values	• None.
Notes	• None.

4.2.8.5 CAN_SlaveStartBank

Function Name	**void CAN_SlaveStartBank (uint8_t CAN_BankNumber)**
Function Description	Select the start bank filter for slave CAN.
Parameters	• **CAN_BankNumber :** Select the start slave bank filter from 1..27.
Return values	• None.
Notes	• None.

4.2.8.6 CAN_DBGFreeze

Function Name	**void CAN_DBGFreeze (*CAN_TypeDef** * **CANx, *FunctionalState* NewState)**
Function Description	Enables or disables the DBG Freeze for CAN.
Parameters	• **CANx :** where x can be 1 or 2 to to select the CAN peripheral. • **NewState :** new state of the CAN peripheral. This parameter can be: ENABLE (CAN reception/transmission is frozen during debug. Reception FIFOs can still be accessed/controlled normally) or DISABLE (CAN is working during debug).
Return values	• None.
Notes	• None.

4.2.8.7 CAN_TTComModeCmd

Function Name	**void CAN_TTComModeCmd (*CAN_TypeDef** * **CANx, *FunctionalState* NewState)**
Function Description	Enables or disables the CAN Time TriggerOperation communication mode.
Parameters	• **CANx :** where x can be 1 or 2 to to select the CAN peripheral. • **NewState :** Mode new state. This parameter can be:

	ENABLE or DISABLE. When enabled, Time stamp (TIME[15:0]) value is sent in the last two data bytes of the 8-byte message: TIME[7:0] in data byte 6 and TIME[15:8] in data byte 7.
Return values	• None.
Notes	• DLC must be programmed as 8 in order Time Stamp (2 bytes) to be sent over the CAN bus.

4.2.9 CAN Frames Transmission functions

4.2.9.1 CAN_Transmit

Function Name	**uint8_t CAN_Transmit (*CAN_TypeDef** * CANx, *CanTxMsg** * TxMessage)**
Function Description	Initiates and transmits a CAN frame message.
Parameters	• **CANx :** where x can be 1 or 2 to to select the CAN peripheral. • **TxMessage :** pointer to a structure which contains CAN Id, CAN DLC and CAN data.
Return values	• **The number of the mailbox that is used for transmission or CAN_TxStatus_NoMailBox if there is no empty mailbox.**
Notes	• None.

4.2.9.2 CAN_TransmitStatus

Function Name	**uint8_t CAN_TransmitStatus (*CAN_TypeDef** * CANx, uint8_t TransmitMailbox)**
Function Description	Checks the transmission status of a CAN Frame.
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral. • **TransmitMailbox :** the number of the mailbox that is used for transmission.
Return values	• **CAN_TxStatus_Ok if the CAN driver transmits the message, CAN_TxStatus_Failed in an other case.**
Notes	• None.

4.2.9.3 CAN_CancelTransmit

Function Name	**void CAN_CancelTransmit (*CAN_TypeDef** * CANx, uint8_t Mailbox)**
Function Description	Cancels a transmit request.
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral. • **Mailbox :** Mailbox number.
Return values	• None.
Notes	• None.

4.2.10 CAN Frames Reception functions

4.2.10.1 CAN_Receive

Function Name	**void CAN_Receive (*CAN_TypeDef** * CANx, uint8_t FIFONumber, *CanRxMsg** * RxMessage)**
Function Description	Receives a correct CAN frame.
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral. • **FIFONumber :** Receive FIFO number, CAN_FIFO0 or CAN_FIFO1. • **RxMessage :** pointer to a structure receive frame which contains CAN Id, CAN DLC, CAN data and FMI number.
Return values	• None.
Notes	• None.

4.2.10.2 CAN_FIFORelease

Function Name	**void CAN_FIFORelease (*CAN_TypeDef** * CANx, uint8_t**

FIFONumber)

Function Description	Releases the specified receive FIFO.
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral. • **FIFONumber :** FIFO to release, CAN_FIFO0 or CAN_FIFO1.
Return values	• None.
Notes	• None.

4.2.10.3 CAN_MessagePending

Function Name	**uint8_t CAN_MessagePending (*CAN_TypeDef** CANx, uint8_t FIFONumber)**
Function Description	Returns the number of pending received messages.
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral. • **FIFONumber :** Receive FIFO number, CAN_FIFO0 or CAN_FIFO1.
Return values	• **NbMessage : which is the number of pending message.**
Notes	• None.

4.2.11 CAN Operating mode functions

4.2.11.1 CAN_OperatingModeRequest

Function Name	**uint8_t CAN_OperatingModeRequest (*CAN_TypeDef** CANx, uint8_t CAN_OperatingMode)**
Function Description	Selects the CAN Operation mode.
Parameters	• **CAN_OperatingMode :** CAN Operating Mode. This parameter can be one of CAN_OperatingMode_TypeDef enumeration.
Return values	• **status of the requested mode which can be:** – ***CAN_ModeStatus_Failed: CAN failed entering the specific mode*** – ***CAN_ModeStatus_Success: CAN Succeed entering the specific mode***
Notes	• None.

4.2.11.2 CAN_Sleep

Function Name	**uint8_t CAN_Sleep (*CAN_TypeDef** * CANx)**
Function Description	Enters the Sleep (low power) mode.
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral.
Return values	• **CAN_Sleep_Ok if sleep entered, CAN_Sleep_Failed otherwise.**
Notes	• None.

4.2.11.3 CAN_WakeUp

Function Name	**uint8_t CAN_WakeUp (*CAN_TypeDef** * CANx)**
Function Description	Wakes up the CAN peripheral from sleep mode .
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral.
Return values	• **CAN_WakeUp_Ok if sleep mode left, CAN_WakeUp_Failed otherwise.**
Notes	• None.

4.2.12 CAN Bus Error management functions

4.2.12.1 CAN_GetLastErrorCode

Function Name	**uint8_t CAN_GetLastErrorCode (*CAN_TypeDef** * CANx)**
Function Description	Returns the CANx's last error code (LEC).
Parameters	• **CANx :** where x can be 1 to select the CAN1 peripheral.
Return values	• **Error code:** – ***CAN_ERRORCODE_NoErr: No Error***

- *CAN_ERRORCODE_StuffErr: Stuff Error*
- *CAN_ERRORCODE_FormErr: Form Error*
- *CAN_ERRORCODE_ACKErr : Acknowledgment Error*
- *CAN_ERRORCODE_BitRecessiveErr: Bit Recessive Error*
- *CAN_ERRORCODE_BitDominantErr: Bit Dominant Error*
- *CAN_ERRORCODE_CRCErr: CRC Error*
- *CAN_ERRORCODE_SoftwareSetErr: Software Set Error*

Notes
- None.

4.2.12.2 CAN_GetReceiveErrorCounter

Function Name **uint8_t CAN_GetReceiveErrorCounter (*CAN_TypeDef** CANx)**

Function Description Returns the CANx Receive Error Counter (REC).

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

Return values
- **CAN Receive Error Counter.**

Notes
- In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception, the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

4.2.12.3 CAN_GetLSBTransmitErrorCounter

Function Name **uint8_t CAN_GetLSBTransmitErrorCounter (*CAN_TypeDef** CANx)**

Function Description Returns the LSB of the 9-bit CANx Transmit Error Counter(TEC).

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

Return values
- **LSB of the 9-bit CAN Transmit Error Counter.**

Notes
- None.

4.2.13 Interrupts and flags management functions

4.2.13.1 CAN_ITConfig

Function Name	**void CAN_ITConfig (*CAN_TypeDef** * CANx, uint32_t CAN_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified CANx interrupts.
Parameters	• **CANx :** where x can be 1 or 2 to to select the CAN peripheral. • **CAN_IT :** specifies the CAN interrupt sources to be enabled or disabled. This parameter can be: – ***CAN_IT_TME :*** Transmit mailbox empty Interrupt – ***CAN_IT_FMP0 :*** FIFO 0 message pending Interrupt – ***CAN_IT_FF0 :*** FIFO 0 full Interrupt – ***CAN_IT_FOV0 :*** FIFO 0 overrun Interrupt – ***CAN_IT_FMP1 :*** FIFO 1 message pending Interrupt – ***CAN_IT_FF1 :*** FIFO 1 full Interrupt – ***CAN_IT_FOV1 :*** FIFO 1 overrun Interrupt – ***CAN_IT_WKU :*** Wake-up Interrupt – ***CAN_IT_SLK :*** Sleep acknowledge Interrupt – ***CAN_IT_EWG :*** Error warning Interrupt – ***CAN_IT_EPV :*** Error passive Interrupt – ***CAN_IT_BOF :*** Bus-off Interrupt – ***CAN_IT_LEC :*** Last error code Interrupt – ***CAN_IT_ERR :*** Error Interrupt • **NewState :** new state of the CAN interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

4.2.13.2 CAN_GetFlagStatus

Function Name	**FlagStatus CAN_GetFlagStatus (*CAN_TypeDef** * CANx, uint32_t CAN_FLAG)**
Function Description	Checks whether the specified CAN flag is set or not.
Parameters	• **CANx :** where x can be 1 or 2 to to select the CAN peripheral. • **CAN_FLAG :** specifies the flag to check. This parameter can be one of the following values:

- ***CAN_FLAG_RQCP0 :*** Request MailBox0 Flag
- ***CAN_FLAG_RQCP1 :*** Request MailBox1 Flag
- ***CAN_FLAG_RQCP2 :*** Request MailBox2 Flag
- ***CAN_FLAG_FMP0 :*** FIFO 0 Message Pending Flag
- ***CAN_FLAG_FF0 :*** FIFO 0 Full Flag
- ***CAN_FLAG_FOV0 :*** FIFO 0 Overrun Flag
- ***CAN_FLAG_FMP1 :*** FIFO 1 Message Pending Flag
- ***CAN_FLAG_FF1 :*** FIFO 1 Full Flag
- ***CAN_FLAG_FOV1 :*** FIFO 1 Overrun Flag
- ***CAN_FLAG_WKU :*** Wake up Flag
- ***CAN_FLAG_SLAK :*** Sleep acknowledge Flag
- ***CAN_FLAG_EWG :*** Error Warning Flag
- ***CAN_FLAG_EPV :*** Error Passive Flag
- ***CAN_FLAG_BOF :*** Bus-Off Flag
- ***CAN_FLAG_LEC :*** Last error code Flag

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

Notes
- None.

4.2.13.3 CAN_ClearFlag

Function Name
void CAN_ClearFlag (*CAN_TypeDef ** CANx, uint32_t CAN_FLAG)

Function Description
Clears the CAN's pending flags.

Parameters
- **CANx :** where x can be 1 or 2 to to select the CAN peripheral.
- **CAN_FLAG :** specifies the flag to clear. This parameter can be one of the following values:
  - ***CAN_FLAG_RQCP0 :*** Request MailBox0 Flag
  - ***CAN_FLAG_RQCP1 :*** Request MailBox1 Flag
  - ***CAN_FLAG_RQCP2 :*** Request MailBox2 Flag
  - ***CAN_FLAG_FF0 :*** FIFO 0 Full Flag
  - ***CAN_FLAG_FOV0 :*** FIFO 0 Overrun Flag
  - ***CAN_FLAG_FF1 :*** FIFO 1 Full Flag
  - ***CAN_FLAG_FOV1 :*** FIFO 1 Overrun Flag
  - ***CAN_FLAG_WKU :*** Wake up Flag
  - ***CAN_FLAG_SLAK :*** Sleep acknowledge Flag
  - ***CAN_FLAG_LEC :*** Last error code Flag

Return values
- None.

Notes
- None.

4.2.13.4 CAN_GetITStatus

Function Name	**ITStatus CAN_GetITStatus (*CAN_TypeDef** * CANx, uint32_t CAN_IT)**
Function Description	Checks whether the specified CANx interrupt has occurred or not.
Parameters	• **CANx :** where x can be 1 or 2 to to select the CAN peripheral. • **CAN_IT :** specifies the CAN interrupt source to check. This parameter can be one of the following values: – ***CAN_IT_TME :*** Transmit mailbox empty Interrupt – ***CAN_IT_FMP0 :*** FIFO 0 message pending Interrupt – ***CAN_IT_FF0 :*** FIFO 0 full Interrupt – ***CAN_IT_FOV0 :*** FIFO 0 overrun Interrupt – ***CAN_IT_FMP1 :*** FIFO 1 message pending Interrupt – ***CAN_IT_FF1 :*** FIFO 1 full Interrupt – ***CAN_IT_FOV1 :*** FIFO 1 overrun Interrupt – ***CAN_IT_WKU :*** Wake-up Interrupt – ***CAN_IT_SLK :*** Sleep acknowledge Interrupt – ***CAN_IT_EWG :*** Error warning Interrupt – ***CAN_IT_EPV :*** Error passive Interrupt – ***CAN_IT_BOF :*** Bus-off Interrupt – ***CAN_IT_LEC :*** Last error code Interrupt – ***CAN_IT_ERR :*** Error Interrupt
Return values	• **The current state of CAN_IT (SET or RESET).**
Notes	• None.

4.2.13.5 CAN_ClearITPendingBit

Function Name	**void CAN_ClearITPendingBit (*CAN_TypeDef** * CANx, uint32_t CAN_IT)**
Function Description	Clears the CANx's interrupt pending bits.
Parameters	• **CANx :** where x can be 1 or 2 to to select the CAN peripheral. • **CAN_IT :** specifies the interrupt pending bit to clear. This parameter can be one of the following values: – ***CAN_IT_TME :*** Transmit mailbox empty Interrupt – ***CAN_IT_FF0 :*** FIFO 0 full Interrupt – ***CAN_IT_FOV0 :*** FIFO 0 overrun Interrupt – ***CAN_IT_FF1 :*** FIFO 1 full Interrupt – ***CAN_IT_FOV1 :*** FIFO 1 overrun Interrupt – ***CAN_IT_WKU :*** Wake-up Interrupt

- ***CAN_IT_SLK :*** Sleep acknowledge Interrupt
- ***CAN_IT_EWG :*** Error warning Interrupt
- ***CAN_IT_EPV :*** Error passive Interrupt
- ***CAN_IT_BOF :*** Bus-off Interrupt
- ***CAN_IT_LEC :*** Last error code Interrupt
- ***CAN_IT_ERR :*** Error Interrupt

Return values
- None.

Notes
- None.

4.3 CAN Firmware driver defines

4.3.1 CAN

CAN

CAN_Error_Code_constants

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

No Error

- #define: ***CAN_ErrorCode_StuffErr ((uint8_t)0x10)***

Stuff Error

- #define: ***CAN_ErrorCode_FormErr ((uint8_t)0x20)***

Form Error

- #define: ***CAN_ErrorCode_ACKErr ((uint8_t)0x30)***

Acknowledgment Error

- #define: ***CAN_ErrorCode_BitRecessiveErr ((uint8_t)0x40)***

Bit Recessive Error

- #define: ***CAN_ErrorCode_BitDominantErr ((uint8_t)0x50)***

Bit Dominant Error

- #define: ***CAN_ErrorCode_CRCErr ((uint8_t)0x60)***

CRC Error

- #define: ***CAN_ErrorCode_SoftwareSetErr ((uint8_t)0x70)***

Software Set Error

CAN_filter_FIFO

- #define: ***CAN_Filter_FIFO0 ((uint8_t)0x00)***

Filter FIFO 0 assignment for filter x

- #define: ***CAN_Filter_FIFO1 ((uint8_t)0x01)***

Filter FIFO 1 assignment for filter x

- #define: ***CAN_FilterFIFO0 CAN_Filter_FIFO0***

- #define: ***CAN_FilterFIFO1 CAN_Filter_FIFO1***

CAN_filter_mode

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

identifier/mask mode

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

identifier list mode

CAN_filter_scale

- #define: ***CAN_FilterScale_16bit ((uint8_t)0x00)***

Two 16-bit filters

- #define: ***CAN_FilterScale_32bit ((uint8_t)0x01)***

One 32-bit filter

CAN_flags

- #define: ***CAN_FLAG_RQCP0 ((uint32_t)0x38000001)***

Request MailBox0 Flag

- #define: ***CAN_FLAG_RQCP1 ((uint32_t)0x38000100)***

Request MailBox1 Flag

- #define: ***CAN_FLAG_RQCP2 ((uint32_t)0x38010000)***

Request MailBox2 Flag

- #define: ***CAN_FLAG_FMP0 ((uint32_t)0x12000003)***

FIFO 0 Message Pending Flag

- #define: ***CAN_FLAG_FF0 ((uint32_t)0x32000008)***

FIFO 0 Full Flag

- #define: ***CAN_FLAG_FOV0 ((uint32_t)0x32000010)***

FIFO 0 Overrun Flag

- #define: ***CAN_FLAG_FMP1 ((uint32_t)0x14000003)***

FIFO 1 Message Pending Flag

- #define: ***CAN_FLAG_FF1 ((uint32_t)0x34000008)***

FIFO 1 Full Flag

- #define: ***CAN_FLAG_FOV1 ((uint32_t)0x34000010)***

FIFO 1 Overrun Flag

- #define: ***CAN_FLAG_WKU ((uint32_t)0x31000008)***

Wake up Flag

- #define: ***CAN_FLAG_SLAK ((uint32_t)0x31000012)***

Sleep acknowledge Flag

- #define: ***CAN_FLAG_EWG ((uint32_t)0x10F00001)***

Error Warning Flag

- #define: ***CAN_FLAG_EPV ((uint32_t)0x10F00002)***

Error Passive Flag

- #define: ***CAN_FLAG_BOF ((uint32_t)0x10F00004)***

Bus-Off Flag

- #define: ***CAN_FLAG_LEC ((uint32_t)0x30F00070)***

Last error code Flag

CAN_identifier_type

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

Standard Id

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

Extended Id

- #define: ***CAN_ID_STD CAN_Id_Standard***

- #define: ***CAN_ID_EXT CAN_Id_Extended***

CAN_InitStatus

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

CAN initialization failed

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

CAN initialization OK

- #define: ***CANINITFAILED CAN_InitStatus_Failed***

- #define: ***CANINITOK CAN_InitStatus_Success***

CAN_interrupts

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

Transmit mailbox empty Interrupt

- #define: ***CAN_IT_FMP0 ((uint32_t)0x00000002)***

FIFO 0 message pending Interrupt

- #define: ***CAN_IT_FF0 ((uint32_t)0x00000004)***

FIFO 0 full Interrupt

- #define: ***CAN_IT_FOV0 ((uint32_t)0x00000008)***

FIFO 0 overrun Interrupt

- #define: ***CAN_IT_FMP1 ((uint32_t)0x00000010)***

FIFO 1 message pending Interrupt

- #define: ***CAN_IT_FF1 ((uint32_t)0x00000020)***

FIFO 1 full Interrupt

- #define: ***CAN_IT_FOV1 ((uint32_t)0x00000040)***

FIFO 1 overrun Interrupt

- #define: ***CAN_IT_WKU ((uint32_t)0x00010000)***

Wake-up Interrupt

- #define: ***CAN_IT_SLK ((uint32_t)0x00020000)***

Sleep acknowledge Interrupt

- #define: ***CAN_IT_EWG ((uint32_t)0x00000100)***

Error warning Interrupt

- #define: ***CAN_IT_EPV ((uint32_t)0x00000200)***

Error passive Interrupt

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

Bus-off Interrupt

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

Last error code Interrupt

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

Error Interrupt

- #define: ***CAN_IT_RQCP0 CAN_IT_TME***

- #define: ***CAN_IT_RQCP1 CAN_IT_TME***

- #define: ***CAN_IT_RQCP2 CAN_IT_TME***

CAN_operating_mode

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

normal mode

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

loopback mode

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

silent mode

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

loopback combined with silent mode

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

Initialization mode

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

Normal mode

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

sleep mode

CAN_operating_mode_status

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

CAN entering the specific mode failed

- #define: ***CAN_ModeStatus_Success ((uint8_t)!CAN_ModeStatus_Failed)***

CAN entering the specific mode Succeed

CAN_receive_FIFO_number_constants

- #define: ***CAN_FIFO0 ((uint8_t)0x00)***

CAN FIFO 0 used to receive

- #define: ***CAN_FIFO1 ((uint8_t)0x01)***

CAN FIFO 1 used to receive

CAN_remote_transmission_request

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

Data frame

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

Remote frame

- #define: ***CAN_RTR_DATA CAN_RTR_Data***

- #define: ***CAN_RTR_REMOTE CAN_RTR_Remote***

CAN_sleep_constants

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

CAN did not enter the sleep mode

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

CAN entered the sleep mode

- #define: ***CANSLEEPFAILED CAN_Sleep_Failed***

- #define: ***CANSLEEPOK CAN_Sleep_Ok***

CAN_synchronisation_jump_width

- #define: ***CAN_SJW_1tq ((uint8_t)0x00)***

1 time quantum

- #define: ***CAN_SJW_2tq ((uint8_t)0x01)***

2 time quantum

- #define: ***CAN_SJW_3tq ((uint8_t)0x02)***

3 time quantum

- #define: ***CAN_SJW_4tq ((uint8_t)0x03)***

4 time quantum

CAN_time_quantum_in_bit_segment_1

- #define: ***CAN_BS1_1tq ((uint8_t)0x00)***

1 time quantum

- #define: ***CAN_BS1_2tq ((uint8_t)0x01)***

2 time quantum

- #define: ***CAN_BS1_3tq ((uint8_t)0x02)***

3 time quantum

- #define: ***CAN_BS1_4tq ((uint8_t)0x03)***

4 time quantum

- #define: ***CAN_BS1_5tq ((uint8_t)0x04)***

5 time quantum

- #define: ***CAN_BS1_6tq ((uint8_t)0x05)***

6 time quantum

- #define: ***CAN_BS1_7tq ((uint8_t)0x06)***

7 time quantum

- #define: ***CAN_BS1_8tq ((uint8_t)0x07)***

8 time quantum

- #define: ***CAN_BS1_9tq ((uint8_t)0x08)***

9 time quantum

- #define: ***CAN_BS1_10tq ((uint8_t)0x09)***

10 time quantum

- #define: ***CAN_BS1_11tq ((uint8_t)0x0A)***

11 time quantum

- #define: ***CAN_BS1_12tq ((uint8_t)0x0B)***

12 time quantum

- #define: ***CAN_BS1_13tq ((uint8_t)0x0C)***

13 time quantum

- #define: ***CAN_BS1_14tq ((uint8_t)0x0D)***

14 time quantum

- #define: ***CAN_BS1_15tq ((uint8_t)0x0E)***

15 time quantum

- #define: ***CAN_BS1_16tq ((uint8_t)0x0F)***

16 time quantum

CAN_time_quantum_in_bit_segment_2

- #define: ***CAN_BS2_1tq ((uint8_t)0x00)***

1 time quantum

- #define: ***CAN_BS2_2tq ((uint8_t)0x01)***

2 time quantum

- #define: ***CAN_BS2_3tq ((uint8_t)0x02)***

3 time quantum

- #define: ***CAN_BS2_4tq ((uint8_t)0x03)***

4 time quantum

- #define: ***CAN_BS2_5tq ((uint8_t)0x04)***

5 time quantum

- #define: ***CAN_BS2_6tq ((uint8_t)0x05)***

6 time quantum

- #define: ***CAN_BS2_7tq ((uint8_t)0x06)***

7 time quantum

- #define: ***CAN_BS2_8tq ((uint8_t)0x07)***

8 time quantum

CAN_transmit_constants

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

CAN transmission failed

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

CAN transmission succeeded

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

CAN transmission pending

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

CAN cell did not provide an empty mailbox

- #define: ***CANTXFAILED CAN_TxStatus_Failed***

- #define: ***CANTXOK CAN_TxStatus_Ok***

- #define: ***CANTXPENDING CAN_TxStatus_Pending***

- #define: ***CAN_NO_MB CAN_TxStatus_NoMailBox***

CAN_wake_up_constants

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

CAN did not leave the sleep mode

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

CAN leaved the sleep mode

- #define: ***CANWAKEUPFAILED CAN_WakeUp_Failed***

- #define: ***CANWAKEUPOK CAN_WakeUp_Ok***

5 Comparators (COMP)

5.1 COMP Firmware driver registers structures

5.1.1 COMP_TypeDef

COMP_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CSR***

Field Documentation

- ***__IO uint32_t COMP_TypeDef::CSR***
  - Comparator control Status register, Address offset: 0x00

5.1.2 COMP_InitTypeDef

COMP_InitTypeDef is defined in the stm32f30x_comp.h

Data Fields

- ***uint32_t COMP_InvertingInput***
- ***uint32_t COMP_NonInvertingInput***
- ***uint32_t COMP_Output***
- ***uint32_t COMP_BlankingSrce***
- ***uint32_t COMP_OutputPol***
- ***uint32_t COMP_Hysteresis***
- ***uint32_t COMP_Mode***

Field Documentation

- ***uint32_t COMP_InitTypeDef::COMP_InvertingInput***
  - Selects the inverting input of the comparator. This parameter can be a value of ***COMP_InvertingInput***
- ***uint32_t COMP_InitTypeDef::COMP_NonInvertingInput***
  - Selects the non inverting input of the comparator. This parameter can be a value of ***COMP_NonInvertingInput***
- ***uint32_t COMP_InitTypeDef::COMP_Output***
  - Selects the output redirection of the comparator. This parameter can be a value of ***COMP_Output***
- ***uint32_t COMP_InitTypeDef::COMP_BlankingSrce***
  - Selects the output blanking source of the comparator. This parameter can be a value of ***COMP_BlankingSrce***
- ***uint32_t COMP_InitTypeDef::COMP_OutputPol***

– Selects the output polarity of the comparator. This parameter can be a value of ***COMP_OutputPoloarity***

- ***uint32_t COMP_InitTypeDef::COMP_Hysteresis***
  – Selects the hysteresis voltage of the comparator. This parameter can be a value of ***COMP_Hysteresis***
- ***uint32_t COMP_InitTypeDef::COMP_Mode***
  – Selects the operating mode of the comparator and allows to adjust the speed/consumption. This parameter can be a value of ***COMP_Mode***

5.2 COMP Firmware driver API description

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

5.2.1 COMP Peripheral features

The device integrates 7 analog comparators COMP1, COMP2...COMP7:

1. The non inverting input and inverting input can be set to GPIO pins as shown in table1. COMP Inputs below.
2. The COMP output is internally is available using COMP_GetOutputLevel() and can be set on GPIO pins. Refer to table 2. COMP Outputs below.
3. The COMP output can be redirected to embedded timers (TIM1, TIM2, TIM3...) Refer to table 3. COMP Outputs redirection to embedded timers below.
4. The comparators COMP1 and COMP2, COMP3 and COMP4, COMP5 and COMP6 can be combined in window mode and only COMP1, COMP3 and COMP5 non inverting input can be used as non-inverting input.
5. The seven comparators have interrupt capability with wake-up from Sleep and Stop modes (through the EXTI controller):
   – COMP1 is internally connected to EXTI Line 21
   – COMP2 is internally connected to EXTI Line 22
   – COMP3 is internally connected to EXTI Line 29
   – COMP4 is internally connected to EXTI Line 30
   – COMP5 is internally connected to EXTI Line 31
   – COMP6 is internally connected to EXTI Line 32
   – COMP7 is internally connected to EXTI Line 33

Table 10: COMP Inputs

		COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
Inverting Input	1/4 VREFINT	OK	OK	OK	OK	OK	OK	OK
	1/2 VREFINT	OK	OK	OK	OK	OK	OK	OK
	3/4 VREFINT	OK	OK	OK	OK	OK	OK	OK
	VREFINT	OK	OK	OK	OK	OK	OK	OK
	DAC1 OUT (PA4)	OK	OK	OK	OK	OK	OK	OK
	DAC2 OUT (PA5)	OK	OK	OK	OK	OK	OK	OK
	IO1	PA0	PA2	PD15	PE8	PD13	PD10	PC0

		COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
Non Inverting	IO1	PA1	PA7	PB14	PB0	PD12	PD11	PA0
Input	IO2	---	PA3	PD14	PE7	PB13	PB11	PC1

Table 11: COMP Outputs

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
PA0	PA2	PB1	PC8	PC7	PA10	PC2
PF4	PA7	---	PA8	PA9	PC6	---
PA6	PA12	---	---	---	---	---
PA11	PB9	---	---	---	---	---
PB8	---	---	---	---	---	---

Table 12: COMP Outputs redirection to embedded timers

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN
TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2
TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN
TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2
TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2
+	+	+	+	+	+	+
TIM8BKIN2	TIM8BKIN2	TIM8BKIN2	TIM8BKIN2	TIM8BKIN2	TIM8BKIN2	TIM8BKIN2
TIM1 OCREFCLR	TIM1 OCREFCLR	TIM1 OCREFCLR	TIM1 OCREFCLR	TIM1 OCREFCLR	TIM1 OCREFCLR	TIM1 OCREFCLR
TIM1 IC1	TIM1 IC1	TIM2 OCREFCLR	TIM3 IC3	TIM2 IC1	TIM2 IC2	TIM8 OCREFCLR
TIM2 IC4	TIM2 IC4	TIM3 IC2	TIM3 OCREFCLR	TIM3 OCREFCLR	TIM2 OCREFCLR	TIM2 IC3
TIM2 OCREFCLR	TIM2 OCREFCLR	TIM4 IC1	TIM4 IC2	TIM4 IC3	TIM16 OCREFCLR	TIM1 IC2
TIM3 IC1	TIM3 IC1	TIM15 IC1	TIM15 OCREFCLR	TIM16 BKIN	TIM16 IC1	TIM17 OCREFCLR
TIM3 OCREFCLR	TIM3 OCREFCLR	TIM15 BKIN	TIM15 IC2	TIM17 IC1	TIM4 IC4	TIM17 BKIN

Table 13: COMP Outputs blanking sources

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
TIM1 OC5	TIM1 OC5	TIM1 OC5	TIM3 OC4	TIM3 OC3	TIM2 OC4	TIM1 OC5
TIM3 OC3	TIM3 OC3	TIM2 OC4	TIM15 OC1	TIM8 BKIN	TIM15 OC2	TIM15 OC2
TIM2 OC3	TIM2 OC3	--------	TIM8 OC5	TIM8 OC5	TIM8 OC5	TIM8 OC5

5.2.2 How to use this driver

This driver provides functions to configure and program the Comparators of all STM32F30x devices. To use the comparator, perform the following steps:

1. Enable the SYSCFG APB clock to get write access to comparator register using RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
2. Configure the comparator input in analog mode using GPIO_Init()
3. Configure the comparator output in alternate function mode using GPIO_Init() and use GPIO_PinAFConfig() function to map the comparator output to the GPIO pin
4. Configure the comparator using COMP_Init() function:
   - Select the inverting input
   - Select the non-inverting input
   - Select the output polarity
   - Select the output redirection
   - Select the hysteresis level
   - Select the power mode
5. Enable the comparator using COMP_Cmd() function
6. If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using EXTI_Init() function. After that enable the comparator interrupt vector using NVIC_Init() function.

5.2.3 Initialization and Configuration functions

- ***COMP_DeInit()***
- ***COMP_Init()***
- ***COMP_StructInit()***
- ***COMP_Cmd()***
- ***COMP_SwitchCmd()***
- ***COMP_GetOutputLevel()***

5.2.4 Window mode control function

- ***COMP_WindowCmd()***

5.2.5 Initialization and Configuration functions

5.2.5.1 COMP_DeInit

Function Name	**void COMP_DeInit (uint32_t COMP_Selection)**
Function Description	Deinitializes COMP peripheral registers to their default reset values.
Parameters	• COMP_Selection.
Return values	• None.
Notes	• Deinitialization can't be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

5.2.5.2 COMP_Init

Function Name	**void COMP_Init (uint32_t COMP_Selection, *COMP_InitTypeDef** * **COMP_InitStruct)**
Function Description	Initializes the COMP peripheral according to the specified parameters in COMP_InitStruct.
Parameters	• **COMP_Selection :** the selected comparator. This parameter can be COMP_Selection_COMPx where x can be 1 to 7 to select the COMP peripheral. • **COMP_InitStruct :** pointer to an COMP_InitTypeDef structure that contains the configuration information for the specified COMP peripheral. COMP_InvertingInput specifies the inverting input of COMPCOMP_NonInvertingInput specifies the non inverting input of COMPCOMP_Output connect COMP output to selected timer input (Input capture / Output Compare Reference Clear / Break Input)COMP_BlankingSrce specifies the blanking source of COMPCOMP_OutputPol select output polarityCOMP_Hysteresis configures COMP hysteresis valueCOMP_Mode configures COMP power mode – ***COMP_InvertingInput :*** – ***COMP_NonInvertingInput :*** – ***COMP_Output :*** – ***COMP_BlankingSrce :*** – ***COMP_OutputPol :*** – ***COMP_Hysteresis :***

– ***COMP_Mode :***

Return values	• None.
Notes	• If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset. • By default, PA1 is selected as COMP1 non inverting input. To use PA4 as COMP1 non inverting input call COMP_SwitchCmd() after COMP_Init()

5.2.5.3 COMP_StructInit

Function Name	**void COMP_StructInit (*COMP_InitTypeDef** * COMP_InitStruct)**
Function Description	Fills each COMP_InitStruct member with its default value.
Parameters	• **COMP_InitStruct :** pointer to an COMP_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

5.2.5.4 COMP_Cmd

Function Name	**void COMP_Cmd (uint32_t COMP_Selection, *FunctionalState* NewState)**
Function Description	Enable or disable the COMP peripheral.
Parameters	• **COMP_Selection :** the selected comparator. This parameter can be COMP_Selection_COMPx where x can be 1 to 7 to select the COMP peripheral. • **NewState :** new state of the COMP peripheral. This parameter can be: ENABLE or DISABLE. When enabled, the comparator compares the non inverting input with the inverting input and the comparison result is available on comparator output. When disabled, the comparator doesn't perform comparison and the output level is low.
Return values	• None.
Notes	• If the selected comparator is locked, enable/disable can't be

performed. To unlock the configuration, perform a system reset.

5.2.5.5 COMP_SwitchCmd

Function Name	**void COMP_SwitchCmd (uint32_t COMP_Selection, *FunctionalState* NewState)**
Function Description	Close or Open the SW1 switch.
Parameters	• **NewState :** New state of the analog switch. This parameter can be ENABLE so the SW1 is closed; PA1 is connected to PA4 or DISABLE so the SW1 switch is open; PA1 is disconnected from PA4
Return values	• None.
Notes	• If the COMP1 is locked, Close/Open the SW1 switch can't be performed. To unlock the configuration, perform a system reset. • This switch is solely intended to redirect signals onto high impedance input, such as COMP1 non-inverting input (highly resistive switch)

5.2.5.6 COMP_GetOutputLevel

Function Name	**uint32_t COMP_GetOutputLevel (uint32_t COMP_Selection)**
Function Description	Return the output level (high or low) of the selected comparator.
Notes	• None.

5.2.6 Write mode control functions

5.2.6.1 COMP_WindowCmd

Function Name	**void COMP_WindowCmd (uint32_t COMP_Selection, *FunctionalState* NewState)**
Function Description	Enables or disables the window mode.
Parameters	• **COMP_Selection :** the selected comparator. This parameter can be COMP_Selection_COMPx where x can be 2, 4 or 6 to select the COMP peripheral. param NewState: new state of the window mode. This parameter can be ENABLE or DISABLE. When enbaled, COMPx and COMPx-1 non inverting inputs are connected together. When disabled, COMPx and COMPx-1 non inverting inputs are disconnected.
Return values	• None.
Notes	• If the COMPx is locked, ENABLE/DISABLE the window mode can't be performed. To unlock the configuration, perform a system reset.

5.2.7 COMP configuration locking function

5.2.7.1 COMP_LockConfig

Function Name	**void COMP_LockConfig (uint32_t COMP_Selection)**
Function Description	Lock the selected comparator (COMP1/COMP2) configuration.
Parameters	• **COMP_Selection :** the selected comparator. This parameter can be COMP_Selection_COMPx where x can be 1 to 7 to select the COMP peripheral.
Return values	• None.
Notes	• Locking the configuration means that all control bits are read-only. To unlock the comparator configuration, perform a system reset.

5.3 COMP Firmware driver defines

5.3.1 COMP

COMP

COMP_BlankingSrce

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

No blanking source

- #define: ***COMP_BlankingSrce_TIM1OC5 COMP_CSR_COMPxBLANKING_0***

TIM1 OC5 selected as blanking source for compartor

- #define: ***COMP_BlankingSrce_TIM2OC3 COMP_CSR_COMPxBLANKING_1***

TIM2 OC5 selected as blanking source for compartor

- #define: ***COMP_BlankingSrce_TIM3OC3 ((uint32_t)0x000C0000)***

TIM2 OC3 selected as blanking source for compartor

- #define: ***COMP_BlankingSrce_TIM2OC4 ((uint32_t)0x000C0000)***

TIM2 OC4 selected as blanking source for compartor

- #define: ***COMP_BlankingSrce_TIM8OC5 COMP_CSR_COMPxBLANKING_1***

TIM8 OC5 selected as blanking source for compartor

- #define: ***COMP_BlankingSrce_TIM3OC4 COMP_CSR_COMPxBLANKING_0***

TIM3 OC4 selected as blanking source for compartor

- #define: ***COMP_BlankingSrce_TIM15OC1 ((uint32_t)0x000C0000)***

TIM15 OC1 selected as blanking source for compartor

- #define: ***COMP_BlankingSrce_TIM15OC2 COMP_CSR_COMPxBLANKING_2***

TIM15 OC2 selected as blanking source for compartor

COMP_Hysteresis

- #define: ***COMP_Hysteresis_No 0x00000000***

No hysteresis

- #define: ***COMP_Hysteresis_Low COMP_CSR_COMPxHYST_0***

Hysteresis level low

- #define: ***COMP_Hysteresis_Medium COMP_CSR_COMPxHYST_1***

Hysteresis level medium

- #define: ***COMP_Hysteresis_High COMP_CSR_COMPxHYST***

Hysteresis level high

COMP_InvertingInput

- #define: ***COMP_InvertingInput_1_4VREFINT ((uint32_t)0x00000000)***

1/4 VREFINT connected to comparator inverting input

- #define: ***COMP_InvertingInput_1_2VREFINT COMP_CSR_COMPxINSEL_0***

1/2 VREFINT connected to comparator inverting input

- #define: ***COMP_InvertingInput_3_4VREFINT COMP_CSR_COMPxINSEL_1***

3/4 VREFINT connected to comparator inverting input

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

VREFINT connected to comparator inverting input

- #define: ***COMP_InvertingInput_DAC1 COMP_CSR_COMPxINSEL_2***

DAC1_OUT (PA4) connected to comparator inverting input

- #define: ***COMP_InvertingInput_DAC2 ((uint32_t)0x00000050)***

DAC2_OUT (PA5) connected to comparator inverting input

- #define: ***COMP_InvertingInput_IO1 ((uint32_t)0x00000060)***

I/O1 (PA0 for COMP1, PA2 for COMP2, PD15 for COMP3, PE8 for COMP4, PD13 for COMP5, PD10 for COMP6, PC0 for COMP7) connected to comparator inverting input

- #define: ***COMP_InvertingInput_IO2 COMP_CSR_COMPxINSEL***

I/O2 (PB12 for COMP3, PB2 for COMP4, PB10 for COMP5, PB15 for COMP6) connected to comparator inverting input

COMP_Mode

- #define: ***COMP_Mode_HighSpeed 0x00000000***

High Speed

- #define: ***COMP_Mode_MediumSpeed COMP_CSR_COMPxMODE_0***

Medium Speed

- #define: ***COMP_Mode_LowPower COMP_CSR_COMPxMODE_1***

Low power mode

- #define: ***COMP_Mode_UltraLowPower COMP_CSR_COMPxMODE***

Ultra-low power mode

COMP_NonInvertingInput

- #define: ***COMP_NonInvertingInput_IO1 ((uint32_t)0x00000000)***

I/O1 (PA1 for COMP1, PA7 for COMP2, PB14 for COMP3, PB0 for COMP4, PD12 for COMP5, PD11 for COMP6, PA0 for COMP7) connected to comparator non inverting input

- #define: ***COMP_NonInvertingInput_IO2 COMP_CSR_COMPxNONINSEL***

I/O2 (PA3 for COMP2, PD14 for COMP3, PE7 for COMP4, PB13 for COMP5, PB11 for COMP6, PC1 for COMP7) connected to comparator non inverting input

COMP_Output

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

COMP output isn't connected to other peripherals

- #define: ***COMP_Output_TIM1BKIN COMP_CSR_COMPxOUTSEL_0***

COMP output connected to TIM1 Break Input (BKIN)

- #define: ***COMP_Output_TIM1BKIN2 ((uint32_t)0x00000800)***

COMP output connected to TIM1 Break Input 2 (BKIN2)

- #define: ***COMP_Output_TIM8BKIN ((uint32_t)0x00000C00)***

COMP output connected to TIM8 Break Input (BKIN)

- #define: ***COMP_Output_TIM8BKIN2 ((uint32_t)0x00001000)***

COMP output connected to TIM8 Break Input 2 (BKIN2)

- #define: ***COMP_Output_TIM1BKIN2_TIM8BKIN2 ((uint32_t)0x00001400)***

COMP output connected to TIM1 Break Input 2 and TIM8 Break Input 2

- #define: ***COMP_Output_TIM1OCREFCLR ((uint32_t)0x00001800)***

COMP output connected to TIM1 OCREF Clear

- #define: ***COMP_Output_TIM1IC1 ((uint32_t)0x00001C00)***

COMP output connected to TIM1 Input Capture 1

- #define: ***COMP_Output_TIM2IC4 ((uint32_t)0x00002000)***

COMP output connected to TIM2 Input Capture 4

- #define: ***COMP_Output_TIM2OCREFCLR ((uint32_t)0x00002400)***

COMP output connected to TIM2 OCREF Clear

- #define: ***COMP_Output_TIM3IC1 ((uint32_t)0x00002800)***

COMP output connected to TIM3 Input Capture 1

- #define: ***COMP_Output_TIM3OCREFCLR ((uint32_t)0x00002C00)***

COMP output connected to TIM3 OCREF Clear

- #define: ***COMP_Output_TIM4IC1 ((uint32_t)0x00001C00)***

COMP output connected to TIM4 Input Capture 1

- #define: ***COMP_Output_TIM3IC2 ((uint32_t)0x00002000)***

COMP output connected to TIM3 Input Capture 2

- #define: ***COMP_Output_TIM15IC1 ((uint32_t)0x00002800)***

COMP output connected to TIM15 Input Capture 1

- #define: ***COMP_Output_TIM15BKIN ((uint32_t)0x00002C00)***

COMP output connected to TIM15 Break Input (BKIN)

- #define: ***COMP_Output_TIM3IC3 ((uint32_t)0x00001800)***

COMP output connected to TIM3 Input Capture 3

- #define: ***COMP_Output_TIM8OCREFCLR ((uint32_t)0x00001C00)***

COMP output connected to TIM8 OCREF Clear

- #define: ***COMP_Output_TIM15IC2 ((uint32_t)0x00002000)***

COMP output connected to TIM15 Input Capture 2

- #define: ***COMP_Output_TIM4IC2 ((uint32_t)0x00002400)***

COMP output connected to TIM4 Input Capture 2

- #define: ***COMP_Output_TIM15OCREFCLR ((uint32_t)0x00002800)***

COMP output connected to TIM15 OCREF Clear

- #define: ***COMP_Output_TIM2IC1 ((uint32_t)0x00001800)***

COMP output connected to TIM2 Input Capture 1

- #define: ***COMP_Output_TIM17IC1 ((uint32_t)0x00002000)***

COMP output connected to TIM17 Input Capture 1

- #define: ***COMP_Output_TIM4IC3 ((uint32_t)0x00002400)***

COMP output connected to TIM4 Input Capture 3

- #define: ***COMP_Output_TIM16BKIN ((uint32_t)0x00002800)***

COMP output connected to TIM16 Break Input (BKIN)

- #define: ***COMP_Output_TIM2IC2 ((uint32_t)0x00001800)***

COMP output connected to TIM2 Input Capture 2

- #define: ***COMP_Output_COMP6TIM2OCREFCLR ((uint32_t)0x00002000)***

COMP output connected to TIM2 OCREF Clear

- #define: ***COMP_Output_TIM16OCREFCLR ((uint32_t)0x00002400)***

COMP output connected to TIM16 OCREF Clear

- #define: ***COMP_Output_TIM16IC1 ((uint32_t)0x00002800)***

COMP output connected to TIM16 Input Capture 1

- #define: ***COMP_Output_TIM4IC4 ((uint32_t)0x00002C00)***

COMP output connected to TIM4 Input Capture 4

- #define: ***COMP_Output_TIM2IC3 ((uint32_t)0x00002000)***

COMP output connected to TIM2 Input Capture 3

- #define: ***COMP_Output_TIM1IC2 ((uint32_t)0x00002400)***

COMP output connected to TIM1 Input Capture 2

- #define: ***COMP_Output_TIM17OCREFCLR ((uint32_t)0x00002800)***

COMP output connected to TIM16 OCREF Clear

- #define: ***COMP_Output_TIM17BKIN ((uint32_t)0x00002C00)***

COMP output connected to TIM16 Break Input (BKIN)

COMP_OutputLevel

- #define: ***COMP_OutputLevel_High COMP_CSR_COMPxOUT***

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

COMP_OutputPoloarity

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

COMP output on GPIO isn't inverted

- #define: ***COMP_OutputPol_Inverted COMP_CSR_COMPxPOL***

COMP output on GPIO is inverted

COMP_Selection

- #define: ***COMP_Selection_COMP1 ((uint32_t)0x00000000)***

COMP1 Selection

- #define: ***COMP_Selection_COMP2 ((uint32_t)0x00000004)***

COMP2 Selection

- #define: ***COMP_Selection_COMP3 ((uint32_t)0x00000008)***

COMP3 Selection

- #define: ***COMP_Selection_COMP4 ((uint32_t)0x0000000C)***

COMP4 Selection

- #define: ***COMP_Selection_COMP5 ((uint32_t)0x00000010)***

COMP5 Selection

- #define: ***COMP_Selection_COMP6 ((uint32_t)0x00000014)***

COMP6 Selection

- #define: ***COMP_Selection_COMP7 ((uint32_t)0x00000018)***

COMP7 Selection

6 CRC calculation unit (CRC)

6.1 CRC Firmware driver registers structures

6.1.1 CRC_TypeDef

CRC_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t DR***
- ***__IO uint8_t IDR***
- ***uint8_t RESERVED0***
- ***uint16_t RESERVED1***
- ***__IO uint32_t CR***
- ***uint32_t RESERVED2***
- ***__IO uint32_t INIT***
- ***__IO uint32_t POL***

Field Documentation

- ***__IO uint32_t CRC_TypeDef::DR***
  - CRC Data register, Address offset: 0x00
- ***__IO uint8_t CRC_TypeDef::IDR***
  - CRC Independent data register, Address offset: 0x04
- ***uint8_t CRC_TypeDef::RESERVED0***
  - Reserved, 0x05
- ***uint16_t CRC_TypeDef::RESERVED1***
  - Reserved, 0x06
- ***__IO uint32_t CRC_TypeDef::CR***
  - CRC Control register, Address offset: 0x08
- ***uint32_t CRC_TypeDef::RESERVED2***
  - Reserved, 0x0C
- ***__IO uint32_t CRC_TypeDef::INIT***
  - Initial CRC value register, Address offset: 0x10
- ***__IO uint32_t CRC_TypeDef::POL***
  - CRC polynomial register, Address offset: 0x14

6.2 CRC Firmware driver API description

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

6.2.1 How to use this driver

1. Enable CRC AHB clock using RCC_AHBPeriphClockCmd(RCC_AHBPeriph_CRC, ENABLE) function.
2. Select the polynomial size: 7-bit, 8-bit, 16-bit or 32-bit.
3. Set the polynomial coefficients using CRC_SetPolynomial();
4. If required, select the reverse operation on input data using CRC_ReverseInputDataSelect();
5. If required, enable the reverse operation on output data using CRC_ReverseOutputDataCmd(Enable);
6. If required, set the initialization remainder value using CRC_SetInitRegister();
7. use CRC_CalcCRC() function to compute the CRC of a 32-bit data or use CRC_CalcBlockCRC() function to compute the CRC if a 32-bit data buffer.

6.2.2 CRC configuration functions

- ***CRC_DeInit()***
- ***CRC_ResetDR()***
- ***CRC_PolynomialSizeSelect()***
- ***CRC_ReverseInputDataSelect()***
- ***CRC_ReverseOutputDataCmd()***
- ***CRC_SetInitRegister()***
- ***CRC_SetPolynomial()***

6.2.3 CRC computation functions

- ***CRC_CalcCRC()***
- ***CRC_CalcCRC16bits()***
- ***CRC_CalcCRC8bits()***
- ***CRC_CalcBlockCRC()***
- ***CRC_GetCRC()***

6.2.4 CRC Independent Register (IDR) access functions

- ***CRC_SetIDRegister()***
- ***CRC_GetIDRegister()***

6.2.5 CRC Independent Register (IDR) access functions

6.2.5.1 CRC_DeInit

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

6.2.5.2 CRC_ResetDR

Function Name	**void CRC_ResetDR (void)**
Function Description	Resets the CRC calculation unit and sets INIT register content in DR register.
Parameters	• None.
Return values	• None.
Notes	• None.

6.2.5.3 CRC_PolynomialSizeSelect

Function Name	**void CRC_PolynomialSizeSelect (uint32_t CRC_PolSize)**
Function Description	Selects the polynomial size.
Parameters	• **CRC_PolSize :** Specifies the polynomial size. This parameter can be: – ***CRC_PolSize_7 :*** 7-bit polynomial for CRC calculation – ***CRC_PolSize_8 :*** 8-bit polynomial for CRC calculation – ***CRC_PolSize_16 :*** 16-bit polynomial for CRC calculation – ***CRC_PolSize_32 :*** 32-bit polynomial for CRC calculation
Return values	• None.
Notes	• None.

6.2.5.4 CRC_ReverseInputDataSelect

Function Name	**void CRC_ReverseInputDataSelect (uint32_t CRC_ReverseInputData)**
Function Description	Selects the reverse operation to be performed on input data.

Parameters	• **CRC_ReverseInputData :** Specifies the reverse operation on input data. This parameter can be: – ***CRC_ReverseInputData_No :*** No reverse operation is performed – ***CRC_ReverseInputData_8bits :*** reverse operation performed on 8 bits – ***CRC_ReverseInputData_16bits :*** reverse operation performed on 16 bits – ***CRC_ReverseInputData_32bits :*** reverse operation performed on 32 bits
Return values	• None.
Notes	• None.

6.2.5.5 CRC_ReverseOutputDataCmd

Function Name	**void CRC_ReverseOutputDataCmd (*FunctionalState* NewState)**
Function Description	Enables or disable the reverse operation on output data.
Parameters	• **NewState :** new state of the reverse operation on output data. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

6.2.5.6 CRC_SetInitRegister

Function Name	**void CRC_SetInitRegister (uint32_t CRC_InitValue)**
Function Description	Initializes the INIT register.
Parameters	• **CRC_InitValue :** Programmable initial CRC value
Return values	• None.
Notes	• After resetting CRC calculation unit, CRC_InitValue is stored in DR register

6.2.5.7 CRC_SetPolynomial

Function Name	**void CRC_SetPolynomial (uint32_t CRC_Pol)**
Function Description	Initializes the polynomail coefficients.
Parameters	• **CRC_Pol :** Polynomial to be used for CRC calculation.
Return values	• None.
Notes	• None.

6.2.6 CRC computation of one/many 32-bit data functions

6.2.6.1 CRC_CalcCRC

Function Name	**uint32_t CRC_CalcCRC (uint32_t CRC_Data)**
Function Description	Computes the 32-bit CRC of a given data word(32-bit).
Parameters	• **CRC_Data :** data word(32-bit) to compute its CRC
Return values	• **32-bit CRC**
Notes	• None.

6.2.6.2 CRC_CalcCRC16bits

Function Name	**uint32_t CRC_CalcCRC16bits (uint16_t CRC_Data)**
Function Description	Computes the 16-bit CRC of a given 16-bit data.
Parameters	• **CRC_Data :** data half-word(16-bit) to compute its CRC
Return values	• **16-bit CRC**
Notes	• None.

6.2.6.3 CRC_CalcCRC8bits

Function Name	**uint32_t CRC_CalcCRC8bits (uint8_t CRC_Data)**
Function Description	Computes the 8-bit CRC of a given 8-bit data.
Parameters	• **CRC_Data :** 8-bit data to compute its CRC
Return values	• **8-bit CRC**
Notes	• None.

6.2.6.4 CRC_CalcBlockCRC

Function Name	**uint32_t CRC_CalcBlockCRC (uint32_t pBuffer, uint32_t BufferLength)**
Function Description	Computes the 32-bit CRC of a given buffer of data word(32-bit).
Parameters	• **pBuffer :** pointer to the buffer containing the data to be computed • **BufferLength :** length of the buffer to be computed
Return values	• **32-bit CRC**
Notes	• None.

6.2.6.5 CRC_GetCRC

Function Name	**uint32_t CRC_GetCRC (void)**
Function Description	Returns the current CRC value.
Parameters	• None.
Return values	• **32-bit CRC**
Notes	• None.

6.2.7 CRC Independent Register (IDR) access functions

6.2.7.1 CRC_SetIDRegister

Function Name	**void CRC_SetIDRegister (uint8_t CRC_IDValue)**
Function Description	Stores an 8-bit data in the Independent Data(ID) register.
Parameters	• **CRC_IDValue :** 8-bit value to be stored in the ID register
Return values	• None.
Notes	• None.

6.2.7.2 CRC_GetIDRegister

Function Name	**uint8_t CRC_GetIDRegister (void)**
Function Description	Returns the 8-bit data stored in the Independent Data(ID) register.
Parameters	• None.
Return values	• **8-bit value of the ID register**
Notes	• None.

6.3 CRC Firmware driver defines

6.3.1 CRC

CRC

CRC_PolynomialSize

- #define: ***CRC_PolSize_7 CRC_CR_POLSIZE***

7-bit polynomial for CRC calculation

- #define: ***CRC_PolSize_8 CRC_CR_POLSIZE_1***

8-bit polynomial for CRC calculation

- #define: ***CRC_PolSize_16 CRC_CR_POLSIZE_0***

16-bit polynomial for CRC calculation

- #define: ***CRC_PolSize_32 ((uint32_t)0x00000000)***

32-bit polynomial for CRC calculation

CRC_ReverseInputData

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

No reverse operation of Input Data

- #define: ***CRC_ReverseInputData_8bits CRC_CR_REV_IN_0***

Reverse operation of Input Data on 8 bits

- #define: ***CRC_ReverseInputData_16bits CRC_CR_REV_IN_1***

Reverse operation of Input Data on 16 bits

- #define: ***CRC_ReverseInputData_32bits CRC_CR_REV_IN***

Reverse operation of Input Data on 32 bits

7 Digital-to-analog converter (DAC)

7.1 DAC Firmware driver registers structures

7.1.1 DAC_TypeDef

DAC_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t SWTRIGR***
- ***__IO uint32_t DHR12R1***
- ***__IO uint32_t DHR12L1***
- ***__IO uint32_t DHR8R1***
- ***__IO uint32_t DHR12R2***
- ***__IO uint32_t DHR12L2***
- ***__IO uint32_t DHR8R2***
- ***__IO uint32_t DHR12RD***
- ***__IO uint32_t DHR12LD***
- ***__IO uint32_t DHR8RD***
- ***__IO uint32_t DOR1***
- ***__IO uint32_t DOR2***
- ***__IO uint32_t SR***

Field Documentation

- ***__IO uint32_t DAC_TypeDef::CR***
  - DAC control register, Address offset: 0x00
- ***__IO uint32_t DAC_TypeDef::SWTRIGR***
  - DAC software trigger register, Address offset: 0x04
- ***__IO uint32_t DAC_TypeDef::DHR12R1***
  - DAC channel1 12-bit right-aligned data holding register, Address offset: 0x08
- ***__IO uint32_t DAC_TypeDef::DHR12L1***
  - DAC channel1 12-bit left aligned data holding register, Address offset: 0x0C
- ***__IO uint32_t DAC_TypeDef::DHR8R1***
  - DAC channel1 8-bit right aligned data holding register, Address offset: 0x10
- ***__IO uint32_t DAC_TypeDef::DHR12R2***
  - DAC channel2 12-bit right aligned data holding register, Address offset: 0x14
- ***__IO uint32_t DAC_TypeDef::DHR12L2***
  - DAC channel2 12-bit left aligned data holding register, Address offset: 0x18
- ***__IO uint32_t DAC_TypeDef::DHR8R2***
  - DAC channel2 8-bit right-aligned data holding register, Address offset: 0x1C
- ***__IO uint32_t DAC_TypeDef::DHR12RD***
  - Dual DAC 12-bit right-aligned data holding register, Address offset: 0x20
- ***__IO uint32_t DAC_TypeDef::DHR12LD***
  - DUAL DAC 12-bit left aligned data holding register, Address offset: 0x24
- ***__IO uint32_t DAC_TypeDef::DHR8RD***
  - DUAL DAC 8-bit right aligned data holding register, Address offset: 0x28

- ***__IO uint32_t DAC_TypeDef::DOR1***
  - DAC channel1 data output register, Address offset: 0x2C
- ***__IO uint32_t DAC_TypeDef::DOR2***
  - DAC channel2 data output register, Address offset: 0x30
- ***__IO uint32_t DAC_TypeDef::SR***
  - DAC status register, Address offset: 0x34

7.1.2 DAC_InitTypeDef

DAC_InitTypeDef is defined in the stm32f30x_dac.h

Data Fields

- ***uint32_t DAC_Trigger***
- ***uint32_t DAC_WaveGeneration***
- ***uint32_t DAC_LFSRUnmask_TriangleAmplitude***
- ***uint32_t DAC_OutputBuffer***

Field Documentation

- ***uint32_t DAC_InitTypeDef::DAC_Trigger***
  - Specifies the external trigger for the selected DAC channel. This parameter can be a value of ***DAC_trigger_selection***
- ***uint32_t DAC_InitTypeDef::DAC_WaveGeneration***
  - Specifies whether DAC channel noise waves or triangle waves are generated, or whether no wave is generated. This parameter can be a value of ***DAC_wave_generation***
- ***uint32_t DAC_InitTypeDef::DAC_LFSRUnmask_TriangleAmplitude***
  - Specifies the LFSR mask for noise wave generation or the maximum amplitude triangle generation for the DAC channel. This parameter can be a value of ***DAC_lfsrunmask_triangleamplitude***
- ***uint32_t DAC_InitTypeDef::DAC_OutputBuffer***
  - Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of ***DAC_output_buffer***

7.2 DAC Firmware driver API description

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

7.2.1 DAC Peripheral features

The device integrates two 12-bit Digital Analog Converters that can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 as output
2. DAC channel2 with DAC_OUT2 as output

Digital to Analog conversion can be non-triggered using DAC_Trigger_None and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register using DAC_SetChannel1Data()/DAC_SetChannel2Data.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_Pin9) using DAC_Trigger_Ext_IT9. The used pin (GPIOx_Pin9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM8/TIM3, TIM4, TIM6, TIM7, and TIM15 (DAC_Trigger_T2_TRGO, DAC_Trigger_T4_TRGO...) The timer TRGO event should be selected using TIM_SelectOutputTrigger()
   - To trigger DAC conversions by TIM3 instead of TIM8 follow this sequence:
     - Enable SYSCFG APB clock by calling RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
     - Select DAC_Trigger_T3_TRGO when calling DAC_Init()
     - Remap the DAC trigger from TIM8 to TIM3 by calling SYSCFG_TriggerRemapConfig(SYSCFG_TriggerRemap_DACTIM3, ENABLE)
3. Software using DAC_Trigger_Software

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

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

Both DAC channels can be used to generate:

- Noise wave using DAC_WaveGeneration_Noise
- Triangle wave using DAC_WaveGeneration_Triangle

Wave generation can be disabled using DAC_WaveGeneration_None

The DAC data format can be:

- 8-bit right alignment using DAC_Align_8b_R
- 12-bit left alignment using DAC_Align_12b_L
- 12-bit right alignment using DAC_Align_12b_R

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

- DAC_OUTx = VREF+ * DOR / 4095 with DOR is the Data Output Register. VREF+ is the input voltage reference (refer to the device datasheet) e.g. To set DAC_OUT1 to 0.7V, use DAC_SetChannel1Data(DAC_Align_12b_R, 868); Assuming that VREF+ = 3.3, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

A DMA request can be generated when an external trigger (but not a software trigger) occurs if DMA2 requests are enabled using DAC_DMACmd(); DMA requests are mapped as following:

- DAC channel1 is mapped on DMA2 channel3 which must be already configured.
- DAC channel2 is mapped on DMA2 channel4 which must be already configured.

7.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using DAC_Init();
- Enable the DAC channel using DAC_Cmd();

7.2.3 DAC channels configuration: trigger, output buffer, data format

- ***DAC_DeInit()***
- ***DAC_Init()***
- ***DAC_StructInit()***
- ***DAC_Cmd()***
- ***DAC_SoftwareTriggerCmd()***
- ***DAC_DualSoftwareTriggerCmd()***
- ***DAC_WaveGenerationCmd()***
- ***DAC_SetChannel1Data()***
- ***DAC_SetChannel2Data()***
- ***DAC_SetDualChannelData()***
- ***DAC_GetDataOutputValue()***

7.2.4 DMA management functions

- ***DAC_DMACmd()***

7.2.5 Interrupts and flags management functions

- ***DAC_ITConfig()***
- ***DAC_GetFlagStatus()***
- ***DAC_ClearFlag()***
- ***DAC_GetITStatus()***
- ***DAC_ClearITPendingBit()***

7.2.6 DAC channels configuration

7.2.6.1 DAC_DeInit

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

7.2.6.2 DAC_Init

Function Name	**void DAC_Init (uint32_t DAC_Channel, *DAC_InitTypeDef** * DAC_InitStruct)**
Function Description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.
Parameters	• **DAC_Channel :** the selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **DAC_InitStruct :** pointer to a DAC_InitTypeDef structure that contains the configuration information for the specified DAC channel.
Return values	• None.
Notes	• None.

7.2.6.3 DAC_StructInit

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

7.2.6.4 DAC_Cmd

Function Name	**void DAC_Cmd (uint32_t DAC_Channel, *FunctionalState* NewState)**
Function Description	Enables or disables the specified DAC channel.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected

– ***DAC_Channel_2 :*** DAC Channel2 selected

- **NewState :** new state of the DAC channel. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- When the DAC channel is enabled the trigger source can no more be modified.

7.2.6.5 DAC_SoftwareTriggerCmd

Function Name **void DAC_SoftwareTriggerCmd (uint32_t DAC_Channel, *FunctionalState* NewState)**

Function Description Enables or disables the selected DAC channel software trigger.

Parameters
- **DAC_Channel :** The selected DAC channel. This parameter can be:
  – ***DAC_Channel_1 :*** DAC Channel1 selected
  – ***DAC_Channel_2 :*** DAC Channel2 selected
- **NewState :** new state of the selected DAC channel software trigger. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

7.2.6.6 DAC_DualSoftwareTriggerCmd

Function Name **void DAC_DualSoftwareTriggerCmd (*FunctionalState* NewState)**

Function Description Enables or disables simultaneously the two DAC channels software triggers.

Parameters
- **NewState :** new state of the DAC channels software triggers. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

7.2.6.7 DAC_WaveGenerationCmd

Function Name	**void DAC_WaveGenerationCmd (uint32_t DAC_Channel, uint32_t DAC_Wave, *FunctionalState* NewState)**
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **DAC_Wave :** specifies the wave type to enable or disable. This parameter can be: – ***DAC_Wave_Noise :*** noise wave generation – ***DAC_Wave_Triangle :*** triangle wave generation • **NewState :** new state of the selected DAC channel wave generation. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

7.2.6.8 DAC_SetChannel1Data

Function Name	**void DAC_SetChannel1Data (uint32_t DAC_Align, uint16_t Data)**
Function Description	Sets the specified data holding register value for DAC channel1.
Parameters	• **DAC_Align :** Specifies the data alignment for DAC channel1. This parameter can be: – ***DAC_Align_8b_R :*** 8bit right data alignment selected – ***DAC_Align_12b_L :*** 12bit left data alignment selected – ***DAC_Align_12b_R :*** 12bit right data alignment selected • **Data :** Data to be loaded in the selected data holding register.
Return values	• None.
Notes	• None.

7.2.6.9 DAC_SetChannel2Data

Function Name	**void DAC_SetChannel2Data (uint32_t DAC_Align, uint16_t Data)**
Function Description	Sets the specified data holding register value for DAC channel2.
Parameters	• **DAC_Align :** Specifies the data alignment for DAC channel2. This parameter can be: – ***DAC_Align_8b_R :*** 8bit right data alignment selected – ***DAC_Align_12b_L :*** 12bit left data alignment selected – ***DAC_Align_12b_R :*** 12bit right data alignment selected • **Data :** Data to be loaded in the selected data holding register.
Return values	• None.
Notes	• None.

7.2.6.10 DAC_SetDualChannelData

Function Name	**void DAC_SetDualChannelData (uint32_t DAC_Align, uint16_t Data2, uint16_t Data1)**
Function Description	Sets the specified data holding register value for dual channel DAC.
Parameters	• **DAC_Align :** Specifies the data alignment for dual channel DAC. This parameter can be: – ***DAC_Align_8b_R :*** 8bit right data alignment selected – ***DAC_Align_12b_L :*** 12bit left data alignment selected – ***DAC_Align_12b_R :*** 12bit right data alignment selected • **Data2 :** Data for DAC Channel2 to be loaded in the selected data holding register. • **Data1 :** Data for DAC Channel1 to be loaded in the selected data holding register.
Return values	• None.
Notes	• In dual mode, a unique register access is required to write in both DAC channels at the same time.

7.2.6.11 DAC_GetDataOutputValue

Function Name	**uint16_t DAC_GetDataOutputValue (uint32_t DAC_Channel)**
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected
Return values	• **The selected DAC channel data output value.**
Notes	• None.

7.2.7 DAC management functions

7.2.7.1 DAC_DMACmd

Function Name	**void DAC_DMACmd (uint32_t DAC_Channel, *FunctionalState* NewState)**
Function Description	Enables or disables the specified DAC channel DMA request.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **NewState :** new state of the selected DAC channel DMA request. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• When enabled DMA1 is generated when an external trigger (EXTI Line9, TIM2, TIM4, TIM5, TIM6, TIM7 or TIM8 but not a software trigger) occurs. • The DAC channel1 is mapped on DMA1 Stream 5 channel7 which must be already configured. • The DAC channel2 is mapped on DMA1 Stream 6 channel7 which must be already configured.

7.2.8 Interrupts and flags management functions

7.2.8.1 DAC_ITConfig

Function Name	**void DAC_ITConfig (uint32_t DAC_Channel, uint32_t DAC_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified DAC interrupts.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **DAC_IT :** specifies the DAC interrupt sources to be enabled or disabled. This parameter can be the following values: – ***DAC_IT_DMAUDR :*** DMA underrun interrupt mask
Parameters	• **NewState :** new state of the specified DAC interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• The DMA underrun occurs when a second external trigger arrives before the acknowledgement for the first external trigger is received (first request).

7.2.8.2 DAC_GetFlagStatus

Function Name	**FlagStatus DAC_GetFlagStatus (uint32_t DAC_Channel, uint32_t DAC_FLAG)**
Function Description	Checks whether the specified DAC flag is set or not.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **DAC_FLAG :** specifies the flag to check. This parameter can be only of the following value: – ***DAC_FLAG_DMAUDR :*** DMA underrun flag
Return values	• **The new state of DAC_FLAG (SET or RESET).**
Notes	• The DMA underrun occurs when a second external trigger arrives before the acknowledgement for the first external trigger is received (first request).

7.2.8.3 DAC_ClearFlag

Function Name	**void DAC_ClearFlag (uint32_t DAC_Channel, uint32_t DAC_FLAG)**
Function Description	Clears the DAC channel's pending flags.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **DAC_FLAG :** specifies the flag to clear. This parameter can be of the following value: – ***DAC_FLAG_DMAUDR :*** DMA underrun flag
Return values	• None.
Notes	• None.

7.2.8.4 DAC_GetITStatus

Function Name	**ITStatus DAC_GetITStatus (uint32_t DAC_Channel, uint32_t DAC_IT)**
Function Description	Checks whether the specified DAC interrupt has occurred or not.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **DAC_IT :** specifies the DAC interrupt source to check. This parameter can be the following values: – ***DAC_IT_DMAUDR :*** DMA underrun interrupt mask
Return values	• **The new state of DAC_IT (SET or RESET).**
Notes	• The DMA underrun occurs when a second external trigger arrives before the acknowledgement for the first external trigger is received (first request).

7.2.8.5 DAC_ClearITPendingBit

Function Name	**void DAC_ClearITPendingBit (uint32_t DAC_Channel, uint32_t DAC_IT)**
Function Description	Clears the DAC channel's interrupt pending bits.
Parameters	• **DAC_Channel :** The selected DAC channel. This parameter can be: – ***DAC_Channel_1 :*** DAC Channel1 selected – ***DAC_Channel_2 :*** DAC Channel2 selected • **DAC_IT :** specifies the DAC interrupt pending bit to clear. This parameter can be the following values: – ***DAC_IT_DMAUDR :*** DMA underrun interrupt mask
Return values	• None.
Notes	• None.

7.3 DAC Firmware driver defines

7.3.1 DAC

DAC

DAC_Channel_selection

- #define: ***DAC_Channel_1 ((uint32_t)0x00000000)***

- #define: ***DAC_Channel_2 ((uint32_t)0x00000010)***

DAC_data_alignement

- #define: ***DAC_Align_12b_R ((uint32_t)0x00000000)***

- #define: ***DAC_Align_12b_L ((uint32_t)0x00000004)***

- #define: ***DAC_Align_8b_R ((uint32_t)0x00000008)***

DAC_flags_definition

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

DAC_interrupts_definition

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

DAC_lfsrunmask_triangleamplitude

- #define: ***DAC_LFSRUnmask_Bit0 ((uint32_t)0x00000000)***

Unmask DAC channel LFSR bit0 for noise wave generation

- #define: ***DAC_LFSRUnmask_Bits1_0 ((uint32_t)0x00000100)***

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

- #define: ***DAC_LFSRUnmask_Bits2_0 ((uint32_t)0x00000200)***

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

- #define: ***DAC_LFSRUnmask_Bits3_0 ((uint32_t)0x00000300)***

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

- #define: ***DAC_LFSRUnmask_Bits4_0 ((uint32_t)0x00000400)***

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

- #define: ***DAC_LFSRUnmask_Bits5_0 ((uint32_t)0x00000500)***

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

- #define: ***DAC_LFSRUnmask_Bits6_0 ((uint32_t)0x00000600)***

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

- #define: ***DAC_LFSRUnmask_Bits7_0 ((uint32_t)0x00000700)***

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

- #define: ***DAC_LFSRUnmask_Bits8_0 ((uint32_t)0x00000800)***

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

- #define: ***DAC_LFSRUnmask_Bits9_0 ((uint32_t)0x00000900)***

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

- #define: ***DAC_LFSRUnmask_Bits10_0 ((uint32_t)0x00000A00)***

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

- #define: ***DAC_LFSRUnmask_Bits11_0 ((uint32_t)0x00000B00)***

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

- #define: ***DAC_TriangleAmplitude_1 ((uint32_t)0x00000000)***

Select max triangle amplitude of 1

- #define: ***DAC_TriangleAmplitude_3 ((uint32_t)0x00000100)***

Select max triangle amplitude of 3

- #define: ***DAC_TriangleAmplitude_7 ((uint32_t)0x00000200)***

Select max triangle amplitude of 7

- #define: ***DAC_TriangleAmplitude_15 ((uint32_t)0x00000300)***

Select max triangle amplitude of 15

- #define: ***DAC_TriangleAmplitude_31 ((uint32_t)0x00000400)***

Select max triangle amplitude of 31

- #define: ***DAC_TriangleAmplitude_63 ((uint32_t)0x00000500)***

Select max triangle amplitude of 63

- #define: ***DAC_TriangleAmplitude_127 ((uint32_t)0x00000600)***

Select max triangle amplitude of 127

- #define: ***DAC_TriangleAmplitude_255 ((uint32_t)0x00000700)***

Select max triangle amplitude of 255

- #define: ***DAC_TriangleAmplitude_511 ((uint32_t)0x00000800)***

Select max triangle amplitude of 511

- #define: ***DAC_TriangleAmplitude_1023 ((uint32_t)0x00000900)***

Select max triangle amplitude of 1023

- #define: ***DAC_TriangleAmplitude_2047 ((uint32_t)0x00000A00)***

Select max triangle amplitude of 2047

- #define: ***DAC_TriangleAmplitude_4095 ((uint32_t)0x00000B00)***

Select max triangle amplitude of 4095

DAC_output_buffer

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

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

DAC_trigger_selection

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

Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger

- #define: ***DAC_Trigger_T2_TRGO ((uint32_t)0x00000024)***

TIM2 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_T3_TRGO ((uint32_t)0x0000000C)***

TIM8 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_T4_TRGO ((uint32_t)0x0000002C)***

TIM4 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_T6_TRGO ((uint32_t)0x00000004)***

TIM6 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_T7_TRGO ((uint32_t)0x00000014)***

TIM7 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_T8_TRGO ((uint32_t)0x0000000C)***

TIM8 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_T15_TRGO ((uint32_t)0x0000001C)***

TIM15 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_Ext_IT9 ((uint32_t)0x00000034)***

EXTI Line9 event selected as external conversion trigger for DAC channel

- #define: ***DAC_Trigger_Software ((uint32_t)0x0000003C)***

Conversion started by software trigger for DAC channel

DAC_wave_generation

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

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

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

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

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

8 Debug support (DBGMCU)

8.1 DBGMCU Firmware driver registers structures

8.1.1 DBGMCU_TypeDef

DBGMCU_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t IDCODE***
- ***__IO uint32_t CR***
- ***__IO uint32_t APB1FZ***
- ***__IO uint32_t APB2FZ***

Field Documentation

- ***__IO uint32_t DBGMCU_TypeDef::IDCODE***
  - MCU device ID code, Address offset: 0x00
- ***__IO uint32_t DBGMCU_TypeDef::CR***
  - Debug MCU configuration register, Address offset: 0x04
- ***__IO uint32_t DBGMCU_TypeDef::APB1FZ***
  - Debug MCU APB1 freeze register, Address offset: 0x08
- ***__IO uint32_t DBGMCU_TypeDef::APB2FZ***
  - Debug MCU APB2 freeze register, Address offset: 0x0C

8.2 DBGMCU Firmware driver API description

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

8.2.1 Device and Revision ID management functions

- ***DBGMCU_GetREVID()***
- ***DBGMCU_GetDEVID()***

8.2.2 Peripherals Configuration functions

- ***DBGMCU_Config()***
- ***DBGMCU_APB1PeriphConfig()***
- ***DBGMCU_APB2PeriphConfig()***

8.2.3 Device and Revision ID management functions

8.2.3.1 DBGMCU_GetREVID

Function Name	**uint32_t DBGMCU_GetREVID (void)**
Function Description	Returns the device revision identifier.
Parameters	• None.
Return values	• **Device revision identifier**
Notes	• None.

8.2.3.2 DBGMCU_GetDEVID

Function Name	**uint32_t DBGMCU_GetDEVID (void)**
Function Description	Returns the device identifier.
Parameters	• None.
Return values	• **Device identifier**
Notes	• None.

8.2.4 Peripherals Configuration functions

8.2.4.1 DBGMCU_Config

Function Name	**void DBGMCU_Config (uint32_t DBGMCU_Periph, *FunctionalState* NewState)**
Function Description	Configures low power mode behavior when the MCU is in Debug mode.
Parameters	• **DBGMCU_Periph :** specifies the low power mode. This parameter can be any combination of the following values: – ***DBGMCU_SLEEP :*** Keep debugger connection during SLEEP mode. – ***DBGMCU_STOP :*** Keep debugger connection during STOP mode. – ***DBGMCU_STANDBY :*** Keep debugger connection

during STANDBY mode.

- **NewState :** new state of the specified low power mode in Debug mode. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

8.2.4.2 DBGMCU_APB1PeriphConfig

Function Name: **void DBGMCU_APB1PeriphConfig (uint32_t DBGMCU_Periph, *FunctionalState* NewState)**

Function Description: Configures APB1 peripheral behavior when the MCU is in Debug mode.

Parameters
- **DBGMCU_Periph :** specifies the APB1 peripheral. This parameter can be any combination of the following values:
  - ***DBGMCU_TIM2_STOP :*** TIM2 counter stopped when Core is halted.
  - ***DBGMCU_TIM3_STOP :*** TIM3 counter stopped when Core is halted.
  - ***DBGMCU_TIM4_STOP :*** TIM4 counter stopped when Core is halted.
  - ***DBGMCU_TIM6_STOP :*** TIM6 counter stopped when Core is halted.
  - ***DBGMCU_TIM7_STOP :*** TIM7 counter stopped when Core is halted.
  - ***DBGMCU_RTC_STOP :*** RTC Calendar and Wakeup counter are stopped when Core is halted.
  - ***DBGMCU_WWDG_STOP :*** Debug WWDG stopped when Core is halted.
  - ***DBGMCU_IWDG_STOP :*** Debug IWDG stopped when Core is halted.
  - ***DBGMCU_I2C1_SMBUS_TIMEOUT :*** I2C1 SMBUS timeout mode stopped when Core is halted.
  - ***DBGMCU_I2C2_SMBUS_TIMEOUT :*** I2C2 SMBUS timeout mode stopped when Core is halted.
  - ***DBGMCU_CAN1_STOP :*** Debug CAN2 stopped when Core is halted.
- **NewState :** new state of the specified APB1 peripheral in Debug mode. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

8.2.4.3 DBGMCU_APB2PeriphConfig

Function Name	**void DBGMCU_APB2PeriphConfig (uint32_t DBGMCU_Periph, *FunctionalState* NewState)**
Function Description	Configures APB2 peripheral behavior when the MCU is in Debug mode.
Parameters	• **DBGMCU_Periph :** specifies the APB2 peripheral. This parameter can be any combination of the following values: – ***DBGMCU_TIM1_STOP :*** TIM1 counter stopped when Core is halted. – ***DBGMCU_TIM8_STOP :*** TIM8 counter stopped when Core is halted. – ***DBGMCU_TIM15_STOP :*** TIM15 counter stopped when Core is halted. – ***DBGMCU_TIM16_STOP :*** TIM16 counter stopped when Core is halted. – ***DBGMCU_TIM17_STOP :*** TIM17 counter stopped when Core is halted. • **NewState :** new state of the specified APB2 peripheral in Debug mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

8.3 DBGMCU Firmware driver defines

8.3.1 DBGMCU

DBGMCU

DBGMCU_Exported_Constants

- #define: ***DBGMCU_SLEEP ((uint32_t)0x00000001)***
- #define: ***DBGMCU_STOP ((uint32_t)0x00000002)***
- #define: ***DBGMCU_STANDBY ((uint32_t)0x00000004)***
- #define: ***DBGMCU_TIM2_STOP ((uint32_t)0x00000001)***

- #define: ***DBGMCU_TIM3_STOP ((uint32_t)0x00000002)***
- #define: ***DBGMCU_TIM4_STOP ((uint32_t)0x00000004)***
- #define: ***DBGMCU_TIM6_STOP ((uint32_t)0x00000010)***
- #define: ***DBGMCU_TIM7_STOP ((uint32_t)0x00000020)***
- #define: ***DBGMCU_RTC_STOP ((uint32_t)0x00000400)***
- #define: ***DBGMCU_WWDG_STOP ((uint32_t)0x00000800)***
- #define: ***DBGMCU_IWDG_STOP ((uint32_t)0x00001000)***
- #define: ***DBGMCU_I2C1_SMBUS_TIMEOUT ((uint32_t)0x00200000)***
- #define: ***DBGMCU_I2C2_SMBUS_TIMEOUT ((uint32_t)0x00400000)***
- #define: ***DBGMCU_CAN1_STOP ((uint32_t)0x02000000)***
- #define: ***DBGMCU_TIM1_STOP ((uint32_t)0x00000001)***
- #define: ***DBGMCU_TIM8_STOP ((uint32_t)0x00000002)***

- #define: ***DBGMCU_TIM15_STOP ((uint32_t)0x00000004)***

- #define: ***DBGMCU_TIM16_STOP ((uint32_t)0x00000008)***

- #define: ***DBGMCU_TIM17_STOP ((uint32_t)0x00000010)***

9 DMA controller (DMA)

9.1 DMA Firmware driver registers structures

9.1.1 DMA_Channel_TypeDef

DMA_Channel_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CCR***
- ***__IO uint32_t CNDTR***
- ***__IO uint32_t CPAR***
- ***__IO uint32_t CMAR***

Field Documentation

- ***__IO uint32_t DMA_Channel_TypeDef::CCR***
  - DMA channel x configuration register
- ***__IO uint32_t DMA_Channel_TypeDef::CNDTR***
  - DMA channel x number of data register
- ***__IO uint32_t DMA_Channel_TypeDef::CPAR***
  - DMA channel x peripheral address register
- ***__IO uint32_t DMA_Channel_TypeDef::CMAR***
  - DMA channel x memory address register

9.1.2 DMA_TypeDef

DMA_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t ISR***
- ***__IO uint32_t IFCR***

Field Documentation

- ***__IO uint32_t DMA_TypeDef::ISR***
  - DMA interrupt status register, Address offset: 0x00
- ***__IO uint32_t DMA_TypeDef::IFCR***
  - DMA interrupt clear flag register, Address offset: 0x04

9.1.3 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32f30x_dma.h

Data Fields

- ***uint32_t DMA_PeripheralBaseAddr***
- ***uint32_t DMA_MemoryBaseAddr***
- ***uint32_t DMA_DIR***
- ***uint16_t DMA_BufferSize***
- ***uint32_t DMA_PeripheralInc***
- ***uint32_t DMA_MemoryInc***
- ***uint32_t DMA_PeripheralDataSize***
- ***uint32_t DMA_MemoryDataSize***
- ***uint32_t DMA_Mode***
- ***uint32_t DMA_Priority***
- ***uint32_t DMA_M2M***

Field Documentation

- ***uint32_t DMA_InitTypeDef::DMA_PeripheralBaseAddr***
  - Specifies the peripheral base address for DMAy Channelx.
- ***uint32_t DMA_InitTypeDef::DMA_MemoryBaseAddr***
  - Specifies the memory base address for DMAy Channelx.
- ***uint32_t DMA_InitTypeDef::DMA_DIR***
  - Specifies if the peripheral is the source or destination. This parameter can be a value of ***DMA_data_transfer_direction***
- ***uint16_t DMA_InitTypeDef::DMA_BufferSize***
  - Specifies the buffer size, in data unit, of the specified Channel. The data unit is equal to the configuration set in DMA_PeripheralDataSize or DMA_MemoryDataSize members depending in the transfer direction.
- ***uint32_t DMA_InitTypeDef::DMA_PeripheralInc***
  - Specifies whether the Peripheral address register is incremented or not. This parameter can be a value of ***DMA_peripheral_incremented_mode***
- ***uint32_t DMA_InitTypeDef::DMA_MemoryInc***
  - Specifies whether the memory address register is incremented or not. This parameter can be a value of ***DMA_memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::DMA_PeripheralDataSize***
  - Specifies the Peripheral data width. This parameter can be a value of ***DMA_peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::DMA_MemoryDataSize***
  - Specifies the Memory data width. This parameter can be a value of ***DMA_memory_data_size***
- ***uint32_t DMA_InitTypeDef::DMA_Mode***
  - Specifies the operation mode of the DMAy Channelx. This parameter can be a value of ***DMA_circular_normal_mode***
- ***uint32_t DMA_InitTypeDef::DMA_Priority***
  - Specifies the software priority for the DMAy Channelx. This parameter can be a value of ***DMA_priority_level***
- ***uint32_t DMA_InitTypeDef::DMA_M2M***
  - Specifies if the DMAy Channelx will be used in memory-to-memory transfer. This parameter can be a value of ***DMA_memory_to_memory***

9.2 DMA Firmware driver API description

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

9.2.1 How to use this driver

1. Enable The DMA controller clock using RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE) function for DMA1 or using RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA2, ENABLE) function for DMA2.
2. Enable and configure the peripheral to be connected to the DMA channel (except for internal SRAM / FLASH memories: no initialization is necessary).
3. For a given Channel, program the Source and Destination addresses, the transfer Direction, the Buffer Size, the Peripheral and Memory Incrementation mode and Data Size, the Circular or Normal mode, the channel transfer Priority and the Memory-to-Memory transfer mode (if needed) using the DMA_Init() function.
4. Enable the NVIC and the corresponding interrupt(s) using the function DMA_ITConfig() if you need to use DMA interrupts.
5. Enable the DMA channel using the DMA_Cmd() function.
6. Activate the needed channel Request using PPP_DMACmd() function for any PPP peripheral except internal SRAM and FLASH (ie. SPI, USART ...) The function allowing this operation is provided in each PPP peripheral driver (ie. SPI_DMACmd for SPI peripheral).
7. Optionally, you can configure the number of data to be transferred when the channel is disabled (ie. after each Transfer Complete event or when a Transfer Error occurs) using the function DMA_SetCurrDataCounter(). And you can get the number of remaining data to be transferred using the function DMA_GetCurrDataCounter() at run time (when the DMA channel is enabled and running).
8. To control DMA events you can use one of the following two methods:
   a. Check on DMA channel flags using the function DMA_GetFlagStatus().
   b. Use DMA interrupts through the function DMA_ITConfig() at initialization phase and DMA_GetITStatus() function into interrupt routines in communication phase. After checking on a flag you should clear it using DMA_ClearFlag() function. And after checking on an interrupt event you should clear it using DMA_ClearITPendingBit() function.

9.2.2 Initialization and Configuration functions

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

The DMA_Init() function follows the DMA configuration procedures as described in reference manual (RM00316).

- ***DMA_DeInit()***
- ***DMA_Init()***
- ***DMA_StructInit()***
- ***DMA_Cmd()***

9.2.3 Data Counter functions

This subsection provides function allowing to configure and read the buffer size (number of data to be transferred).The DMA data counter can be written only when the DMA channel is disabled (ie. after transfer complete event).

The following function can be used to write the Channel data counter value:

- void DMA_SetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx, uint16_t DataNumber).

It is advised to use this function rather than DMA_Init() in situations where only the Data buffer needs to be reloaded.

The DMA data counter can be read to indicate the number of remaining transfers for the relative DMA channel. This counter is decremented at the end of each data transfer and when the transfer is complete:

- If Normal mode is selected: the counter is set to 0.
- If Circular mode is selected: the counter is reloaded with the initial value(configured before enabling the DMA channel).

The following function can be used to read the Channel data counter value:

- uint16_t DMA_GetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx).
- ***DMA_SetCurrDataCounter()***
- ***DMA_GetCurrDataCounter()***

9.2.4 Interrupts and flags management functions

This subsection provides functions allowing to configure the DMA Interrupt sources 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 DMA controller events: Polling mode or Interrupt mode.

Polling Mode

Each DMA channel can be managed through 4 event Flags (y : DMA Controller number, x : DMA channel number):

1. DMAy_FLAG_TCx : to indicate that a Transfer Complete event occurred.
2. DMAy_FLAG_HTx : to indicate that a Half-Transfer Complete event occurred.
3. DMAy_FLAG_TEx : to indicate that a Transfer Error occurred.
4. DMAy_FLAG_GLx : to indicate that at least one of the events described above occurred.

Clearing DMAy_FLAG_GLx results in clearing all other pending flags of the same channel (DMAy_FLAG_TCx, DMAy_FLAG_HTx and DMAy_FLAG_TEx).

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

- FlagStatus DMA_GetFlagStatus(uint32_t DMA_FLAG);

- void DMA_ClearFlag(uint32_t DMA_FLAG);

Interrupt Mode

Each DMA channel can be managed through 4 Interrupts:

- Interrupt Source
  a. DMA_IT_TC: specifies the interrupt source for the Transfer Complete event.
  b. DMA_IT_HT: specifies the interrupt source for the Half-transfer Complete event.
  c. DMA_IT_TE: specifies the interrupt source for the transfer errors event.
  d. DMA_IT_GL: to indicate that at least one of the interrupts described above occurred. Clearing DMA_IT_GL interrupt results in clearing all other interrupts of the same channel (DMA_IT_TCx, DMA_IT_HT and DMA_IT_TE).

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

- void DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, uint32_t DMA_IT, FunctionalState NewState);
- ITStatus DMA_GetITStatus(uint32_t DMA_IT);
- void DMA_ClearITPendingBit(uint32_t DMA_IT);
- ***DMA_ITConfig()***
- ***DMA_GetFlagStatus()***
- ***DMA_ClearFlag()***
- ***DMA_GetITStatus()***
- ***DMA_ClearITPendingBit()***

9.2.5 Initialization and Configuration functions

9.2.5.1 DMA_DeInit

Function Name	**void DMA_DeInit (*DMA_Channel_TypeDef ** DMAy_Channelx)**
Function Description	Deinitializes the DMAy Channelx registers to their default reset values.
Parameters	• **DMAy_Channelx :** where y can be 1 or 2 to select the DMA and x can be 1 to 7 for DMA1 and 1 to 5 for DMA2 to select the DMA Channel.
Return values	• None.
Notes	• None.

9.2.5.2 DMA_Init

Function Name	**void DMA_Init (*DMA_Channel_TypeDef ** DMAy_Channelx, *DMA_InitTypeDef ** DMA_InitStruct)**
Function Description	Initializes the DMAy Channelx according to the specified

parameters in the DMA_InitStruct.

Parameters	• **DMAy_Channelx :** where y can be 1 or 2 to select the DMA and x can be 1 to 7 for DMA1 and 1 to 5 for DMA2 to select the DMA Channel. • **DMA_InitStruct :** pointer to a DMA_InitTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• None.
Notes	• None.

9.2.5.3 DMA_StructInit

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

9.2.5.4 DMA_Cmd

Function Name	**void DMA_Cmd (*DMA_Channel_TypeDef** * DMAy_Channelx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified DMAy Channelx.
Parameters	• **DMAy_Channelx :** where y can be 1 or 2 to select the DMA and x can be 1 to 7 for DMA1 and 1 to 5 for DMA2 to select the DMA Channel. • **NewState :** new state of the DMAy Channelx. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

9.2.6 Data counter functions

9.2.6.1 DMA_SetCurrDataCounter

Function Name	**void DMA_SetCurrDataCounter (*DMA_Channel_TypeDef** * DMAy_Channelx, uint16_t DataNumber)**
Function Description	Sets the number of data units in the current DMAy Channelx transfer.
Parameters	• **DMAy_Channelx :** where y can be 1 or 2 to select the DMA and x can be 1 to 7 for DMA1 and 1 to 5 for DMA2 to select the DMA Channel. • **DataNumber :** The number of data units in the current DMAy Channelx transfer.
Return values	• **None.**
Notes	• This function can only be used when the DMAy_Channelx is disabled.

9.2.6.2 DMA_GetCurrDataCounter

Function Name	**uint16_t DMA_GetCurrDataCounter (*DMA_Channel_TypeDef** * DMAy_Channelx)**
Function Description	Returns the number of remaining data units in the current DMAy Channelx transfer.
Parameters	• **DMAy_Channelx :** where y can be 1 or 2 to select the DMA and x can be 1 to 7 for DMA1 and 1 to 5 for DMA2 to select the DMA Channel.
Return values	• **The number of remaining data units in the current DMAy Channelx transfer.**
Notes	• None.

9.2.7 Interrupts and flags management functions

9.2.7.1 DMA_ITConfig

Function Name	**void DMA_ITConfig (*DMA_Channel_TypeDef** * DMAy_Channelx, uint32_t DMA_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified DMAy Channelx interrupts.
Parameters	• **DMAy_Channelx :** where y can be 1 or 2 to select the DMA and x can be 1 to 7 for DMA1 and 1 to 5 for DMA2 to select the DMA Channel. • **DMA_IT :** specifies the DMA interrupts sources to be enabled or disabled. This parameter can be any combination of the following values: – ***DMA_IT_TC :*** Transfer complete interrupt mask – ***DMA_IT_HT :*** Half transfer interrupt mask – ***DMA_IT_TE :*** Transfer error interrupt mask • **NewState :** new state of the specified DMA interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

9.2.7.2 DMA_GetFlagStatus

Function Name	**FlagStatus DMA_GetFlagStatus (uint32_t DMAy_FLAG)**
Function Description	Checks whether the specified DMAy Channelx flag is set or not.
Parameters	• **DMAy_FLAG :** specifies the flag to check. This parameter can be one of the following values: – ***DMA1_FLAG_GL1 :*** DMA1 Channel1 global flag. – ***DMA1_FLAG_TC1 :*** DMA1 Channel1 transfer complete flag. – ***DMA1_FLAG_HT1 :*** DMA1 Channel1 half transfer flag. – ***DMA1_FLAG_TE1 :*** DMA1 Channel1 transfer error flag. – ***DMA1_FLAG_GL2 :*** DMA1 Channel2 global flag. – ***DMA1_FLAG_TC2 :*** DMA1 Channel2 transfer complete flag. – ***DMA1_FLAG_HT2 :*** DMA1 Channel2 half transfer flag. – ***DMA1_FLAG_TE2 :*** DMA1 Channel2 transfer error flag. – ***DMA1_FLAG_GL3 :*** DMA1 Channel3 global flag. – ***DMA1_FLAG_TC3 :*** DMA1 Channel3 transfer complete flag.

- ***DMA1_FLAG_HT3 :*** DMA1 Channel3 half transfer flag.
- ***DMA1_FLAG_TE3 :*** DMA1 Channel3 transfer error flag.
- ***DMA1_FLAG_GL4 :*** DMA1 Channel4 global flag.
- ***DMA1_FLAG_TC4 :*** DMA1 Channel4 transfer complete flag.
- ***DMA1_FLAG_HT4 :*** DMA1 Channel4 half transfer flag.
- ***DMA1_FLAG_TE4 :*** DMA1 Channel4 transfer error flag.
- ***DMA1_FLAG_GL5 :*** DMA1 Channel5 global flag.
- ***DMA1_FLAG_TC5 :*** DMA1 Channel5 transfer complete flag.
- ***DMA1_FLAG_HT5 :*** DMA1 Channel5 half transfer flag.
- ***DMA1_FLAG_TE5 :*** DMA1 Channel5 transfer error flag.
- ***DMA1_FLAG_GL6 :*** DMA1 Channel6 global flag.
- ***DMA1_FLAG_TC6 :*** DMA1 Channel6 transfer complete flag.
- ***DMA1_FLAG_HT6 :*** DMA1 Channel6 half transfer flag.
- ***DMA1_FLAG_TE6 :*** DMA1 Channel6 transfer error flag.
- ***DMA1_FLAG_GL7 :*** DMA1 Channel7 global flag.
- ***DMA1_FLAG_TC7 :*** DMA1 Channel7 transfer complete flag.
- ***DMA1_FLAG_HT7 :*** DMA1 Channel7 half transfer flag.
- ***DMA1_FLAG_TE7 :*** DMA1 Channel7 transfer error flag.
- ***DMA2_FLAG_GL1 :*** DMA2 Channel1 global flag.
- ***DMA2_FLAG_TC1 :*** DMA2 Channel1 transfer complete flag.
- ***DMA2_FLAG_HT1 :*** DMA2 Channel1 half transfer flag.
- ***DMA2_FLAG_TE1 :*** DMA2 Channel1 transfer error flag.
- ***DMA2_FLAG_GL2 :*** DMA2 Channel2 global flag.
- ***DMA2_FLAG_TC2 :*** DMA2 Channel2 transfer complete flag.
- ***DMA2_FLAG_HT2 :*** DMA2 Channel2 half transfer flag.
- ***DMA2_FLAG_TE2 :*** DMA2 Channel2 transfer error flag.
- ***DMA2_FLAG_GL3 :*** DMA2 Channel3 global flag.
- ***DMA2_FLAG_TC3 :*** DMA2 Channel3 transfer complete flag.
- ***DMA2_FLAG_HT3 :*** DMA2 Channel3 half transfer flag.
- ***DMA2_FLAG_TE3 :*** DMA2 Channel3 transfer error flag.
- ***DMA2_FLAG_GL4 :*** DMA2 Channel4 global flag.
- ***DMA2_FLAG_TC4 :*** DMA2 Channel4 transfer complete flag.
- ***DMA2_FLAG_HT4 :*** DMA2 Channel4 half transfer flag.
- ***DMA2_FLAG_TE4 :*** DMA2 Channel4 transfer error flag.
- ***DMA2_FLAG_GL5 :*** DMA2 Channel5 global flag.
- ***DMA2_FLAG_TC5 :*** DMA2 Channel5 transfer complete flag.
- ***DMA2_FLAG_HT5 :*** DMA2 Channel5 half transfer flag.
- ***DMA2_FLAG_TE5 :*** DMA2 Channel5 transfer error flag.

Return values

- **The new state of DMAy_FLAG (SET or RESET).**

Notes

- The Global flag (DMAy_FLAG_GLx) is set whenever any of the other flags relative to the same channel is set (Transfer Complete, Half-transfer Complete or Transfer Error flags: DMAy_FLAG_TCx, DMAy_FLAG_HTx or DMAy_FLAG_TEx).

9.2.7.3 DMA_ClearFlag

Function Name **void DMA_ClearFlag (uint32_t DMAy_FLAG)**

Function Description Clears the DMAy Channelx's pending flags.

Parameters

- **DMAy_FLAG :** specifies the flag to clear. This parameter can be any combination (for the same DMA) of the following values:
  - ***DMA1_FLAG_GL1 :*** DMA1 Channel1 global flag.
  - ***DMA1_FLAG_TC1 :*** DMA1 Channel1 transfer complete flag.
  - ***DMA1_FLAG_HT1 :*** DMA1 Channel1 half transfer flag.
  - ***DMA1_FLAG_TE1 :*** DMA1 Channel1 transfer error flag.
  - ***DMA1_FLAG_GL2 :*** DMA1 Channel2 global flag.
  - ***DMA1_FLAG_TC2 :*** DMA1 Channel2 transfer complete flag.
  - ***DMA1_FLAG_HT2 :*** DMA1 Channel2 half transfer flag.
  - ***DMA1_FLAG_TE2 :*** DMA1 Channel2 transfer error flag.
  - ***DMA1_FLAG_GL3 :*** DMA1 Channel3 global flag.
  - ***DMA1_FLAG_TC3 :*** DMA1 Channel3 transfer complete flag.
  - ***DMA1_FLAG_HT3 :*** DMA1 Channel3 half transfer flag.
  - ***DMA1_FLAG_TE3 :*** DMA1 Channel3 transfer error flag.
  - ***DMA1_FLAG_GL4 :*** DMA1 Channel4 global flag.
  - ***DMA1_FLAG_TC4 :*** DMA1 Channel4 transfer complete flag.
  - ***DMA1_FLAG_HT4 :*** DMA1 Channel4 half transfer flag.
  - ***DMA1_FLAG_TE4 :*** DMA1 Channel4 transfer error flag.
  - ***DMA1_FLAG_GL5 :*** DMA1 Channel5 global flag.
  - ***DMA1_FLAG_TC5 :*** DMA1 Channel5 transfer complete flag.
  - ***DMA1_FLAG_HT5 :*** DMA1 Channel5 half transfer flag.
  - ***DMA1_FLAG_TE5 :*** DMA1 Channel5 transfer error flag.
  - ***DMA1_FLAG_GL6 :*** DMA1 Channel6 global flag.
  - ***DMA1_FLAG_TC6 :*** DMA1 Channel6 transfer complete flag.
  - ***DMA1_FLAG_HT6 :*** DMA1 Channel6 half transfer flag.
  - ***DMA1_FLAG_TE6 :*** DMA1 Channel6 transfer error flag.
  - ***DMA1_FLAG_GL7 :*** DMA1 Channel7 global flag.
  - ***DMA1_FLAG_TC7 :*** DMA1 Channel7 transfer complete flag.
  - ***DMA1_FLAG_HT7 :*** DMA1 Channel7 half transfer flag.
  - ***DMA1_FLAG_TE7 :*** DMA1 Channel7 transfer error flag.
  - ***DMA2_FLAG_GL1 :*** DMA2 Channel1 global flag.
  - ***DMA2_FLAG_TC1 :*** DMA2 Channel1 transfer complete flag.

- ***DMA2_FLAG_HT1 :*** DMA2 Channel1 half transfer flag.
- ***DMA2_FLAG_TE1 :*** DMA2 Channel1 transfer error flag.
- ***DMA2_FLAG_GL2 :*** DMA2 Channel2 global flag.
- ***DMA2_FLAG_TC2 :*** DMA2 Channel2 transfer complete flag.
- ***DMA2_FLAG_HT2 :*** DMA2 Channel2 half transfer flag.
- ***DMA2_FLAG_TE2 :*** DMA2 Channel2 transfer error flag.
- ***DMA2_FLAG_GL3 :*** DMA2 Channel3 global flag.
- ***DMA2_FLAG_TC3 :*** DMA2 Channel3 transfer complete flag.
- ***DMA2_FLAG_HT3 :*** DMA2 Channel3 half transfer flag.
- ***DMA2_FLAG_TE3 :*** DMA2 Channel3 transfer error flag.
- ***DMA2_FLAG_GL4 :*** DMA2 Channel4 global flag.
- ***DMA2_FLAG_TC4 :*** DMA2 Channel4 transfer complete flag.
- ***DMA2_FLAG_HT4 :*** DMA2 Channel4 half transfer flag.
- ***DMA2_FLAG_TE4 :*** DMA2 Channel4 transfer error flag.
- ***DMA2_FLAG_GL5 :*** DMA2 Channel5 global flag.
- ***DMA2_FLAG_TC5 :*** DMA2 Channel5 transfer complete flag.
- ***DMA2_FLAG_HT5 :*** DMA2 Channel5 half transfer flag.
- ***DMA2_FLAG_TE5 :*** DMA2 Channel5 transfer error flag.

Return values

- None.

Notes

- Clearing the Global flag (DMAy_FLAG_GLx) results in clearing all other flags relative to the same channel (Transfer Complete, Half-transfer Complete and Transfer Error flags: DMAy_FLAG_TCx, DMAy_FLAG_HTx and DMAy_FLAG_TEx).

9.2.7.4 DMA_GetITStatus

Function Name

ITStatus DMA_GetITStatus (uint32_t DMAy_IT)

Function Description

Checks whether the specified DMAy Channelx interrupt has occurred or not.

Parameters

- **DMAy_IT :** specifies the DMAy interrupt source to check. This parameter can be one of the following values:
  - ***DMA1_IT_GL1 :*** DMA1 Channel1 global interrupt.
  - ***DMA1_IT_TC1 :*** DMA1 Channel1 transfer complete interrupt.
  - ***DMA1_IT_HT1 :*** DMA1 Channel1 half transfer interrupt.
  - ***DMA1_IT_TE1 :*** DMA1 Channel1 transfer error interrupt.
  - ***DMA1_IT_GL2 :*** DMA1 Channel2 global interrupt.
  - ***DMA1_IT_TC2 :*** DMA1 Channel2 transfer complete interrupt.

- ***DMA1_IT_HT2 :*** DMA1 Channel2 half transfer interrupt.
- ***DMA1_IT_TE2 :*** DMA1 Channel2 transfer error interrupt.
- ***DMA1_IT_GL3 :*** DMA1 Channel3 global interrupt.
- ***DMA1_IT_TC3 :*** DMA1 Channel3 transfer complete interrupt.
- ***DMA1_IT_HT3 :*** DMA1 Channel3 half transfer interrupt.
- ***DMA1_IT_TE3 :*** DMA1 Channel3 transfer error interrupt.
- ***DMA1_IT_GL4 :*** DMA1 Channel4 global interrupt.
- ***DMA1_IT_TC4 :*** DMA1 Channel4 transfer complete interrupt.
- ***DMA1_IT_HT4 :*** DMA1 Channel4 half transfer interrupt.
- ***DMA1_IT_TE4 :*** DMA1 Channel4 transfer error interrupt.
- ***DMA1_IT_GL5 :*** DMA1 Channel5 global interrupt.
- ***DMA1_IT_TC5 :*** DMA1 Channel5 transfer complete interrupt.
- ***DMA1_IT_HT5 :*** DMA1 Channel5 half transfer interrupt.
- ***DMA1_IT_TE5 :*** DMA1 Channel5 transfer error interrupt.
- ***DMA1_IT_GL6 :*** DMA1 Channel6 global interrupt.
- ***DMA1_IT_TC6 :*** DMA1 Channel6 transfer complete interrupt.
- ***DMA1_IT_HT6 :*** DMA1 Channel6 half transfer interrupt.
- ***DMA1_IT_TE6 :*** DMA1 Channel6 transfer error interrupt.
- ***DMA1_IT_GL7 :*** DMA1 Channel7 global interrupt.
- ***DMA1_IT_TC7 :*** DMA1 Channel7 transfer complete interrupt.
- ***DMA1_IT_HT7 :*** DMA1 Channel7 half transfer interrupt.
- ***DMA1_IT_TE7 :*** DMA1 Channel7 transfer error interrupt.
- ***DMA2_IT_GL1 :*** DMA2 Channel1 global interrupt.
- ***DMA2_IT_TC1 :*** DMA2 Channel1 transfer complete interrupt.
- ***DMA2_IT_HT1 :*** DMA2 Channel1 half transfer interrupt.
- ***DMA2_IT_TE1 :*** DMA2 Channel1 transfer error interrupt.
- ***DMA2_IT_GL2 :*** DMA2 Channel2 global interrupt.
- ***DMA2_IT_TC2 :*** DMA2 Channel2 transfer complete interrupt.
- ***DMA2_IT_HT2 :*** DMA2 Channel2 half transfer interrupt.
- ***DMA2_IT_TE2 :*** DMA2 Channel2 transfer error interrupt.
- ***DMA2_IT_GL3 :*** DMA2 Channel3 global interrupt.
- ***DMA2_IT_TC3 :*** DMA2 Channel3 transfer complete interrupt.
- ***DMA2_IT_HT3 :*** DMA2 Channel3 half transfer interrupt.
- ***DMA2_IT_TE3 :*** DMA2 Channel3 transfer error interrupt.
- ***DMA2_IT_GL4 :*** DMA2 Channel4 global interrupt.
- ***DMA2_IT_TC4 :*** DMA2 Channel4 transfer complete

interrupt.
- ***DMA2_IT_HT4 :*** DMA2 Channel4 half transfer interrupt.
- ***DMA2_IT_TE4 :*** DMA2 Channel4 transfer error interrupt.
- ***DMA2_IT_GL5 :*** DMA2 Channel5 global interrupt.
- ***DMA2_IT_TC5 :*** DMA2 Channel5 transfer complete interrupt.
- ***DMA2_IT_HT5 :*** DMA2 Channel5 half transfer interrupt.
- ***DMA2_IT_TE5 :*** DMA2 Channel5 transfer error interrupt.

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

Notes
- The Global interrupt (DMAy_FLAG_GLx) is set whenever any of the other interrupts relative to the same channel is set (Transfer Complete, Half-transfer Complete or Transfer Error interrupts: DMAy_IT_TCx, DMAy_IT_HTx or DMAy_IT_TEx).

9.2.7.5 DMA_ClearITPendingBit

Function Name **void DMA_ClearITPendingBit (uint32_t DMAy_IT)**

Function Description Clears the DMAy Channelx's interrupt pending bits.

Parameters
- **DMAy_IT :** specifies the DMAy interrupt pending bit to clear. This parameter can be any combination (for the same DMA) of the following values:
  - ***DMA1_IT_GL1 :*** DMA1 Channel1 global interrupt.
  - ***DMA1_IT_TC1 :*** DMA1 Channel1 transfer complete interrupt.
  - ***DMA1_IT_HT1 :*** DMA1 Channel1 half transfer interrupt.
  - ***DMA1_IT_TE1 :*** DMA1 Channel1 transfer error interrupt.
  - ***DMA1_IT_GL2 :*** DMA1 Channel2 global interrupt.
  - ***DMA1_IT_TC2 :*** DMA1 Channel2 transfer complete interrupt.
  - ***DMA1_IT_HT2 :*** DMA1 Channel2 half transfer interrupt.
  - ***DMA1_IT_TE2 :*** DMA1 Channel2 transfer error interrupt.
  - ***DMA1_IT_GL3 :*** DMA1 Channel3 global interrupt.
  - ***DMA1_IT_TC3 :*** DMA1 Channel3 transfer complete interrupt.
  - ***DMA1_IT_HT3 :*** DMA1 Channel3 half transfer interrupt.
  - ***DMA1_IT_TE3 :*** DMA1 Channel3 transfer error interrupt.
  - ***DMA1_IT_GL4 :*** DMA1 Channel4 global interrupt.
  - ***DMA1_IT_TC4 :*** DMA1 Channel4 transfer complete interrupt.
  - ***DMA1_IT_HT4 :*** DMA1 Channel4 half transfer interrupt.

- ***DMA1_IT_TE4 :*** DMA1 Channel4 transfer error interrupt.
- ***DMA1_IT_GL5 :*** DMA1 Channel5 global interrupt.
- ***DMA1_IT_TC5 :*** DMA1 Channel5 transfer complete interrupt.
- ***DMA1_IT_HT5 :*** DMA1 Channel5 half transfer interrupt.
- ***DMA1_IT_TE5 :*** DMA1 Channel5 transfer error interrupt.
- ***DMA1_IT_GL6 :*** DMA1 Channel6 global interrupt.
- ***DMA1_IT_TC6 :*** DMA1 Channel6 transfer complete interrupt.
- ***DMA1_IT_HT6 :*** DMA1 Channel6 half transfer interrupt.
- ***DMA1_IT_TE6 :*** DMA1 Channel6 transfer error interrupt.
- ***DMA1_IT_GL7 :*** DMA1 Channel7 global interrupt.
- ***DMA1_IT_TC7 :*** DMA1 Channel7 transfer complete interrupt.
- ***DMA1_IT_HT7 :*** DMA1 Channel7 half transfer interrupt.
- ***DMA1_IT_TE7 :*** DMA1 Channel7 transfer error interrupt.
- ***DMA2_IT_GL1 :*** DMA2 Channel1 global interrupt.
- ***DMA2_IT_TC1 :*** DMA2 Channel1 transfer complete interrupt.
- ***DMA2_IT_HT1 :*** DMA2 Channel1 half transfer interrupt.
- ***DMA2_IT_TE1 :*** DMA2 Channel1 transfer error interrupt.
- ***DMA2_IT_GL2 :*** DMA2 Channel2 global interrupt.
- ***DMA2_IT_TC2 :*** DMA2 Channel2 transfer complete interrupt.
- ***DMA2_IT_HT2 :*** DMA2 Channel2 half transfer interrupt.
- ***DMA2_IT_TE2 :*** DMA2 Channel2 transfer error interrupt.
- ***DMA2_IT_GL3 :*** DMA2 Channel3 global interrupt.
- ***DMA2_IT_TC3 :*** DMA2 Channel3 transfer complete interrupt.
- ***DMA2_IT_HT3 :*** DMA2 Channel3 half transfer interrupt.
- ***DMA2_IT_TE3 :*** DMA2 Channel3 transfer error interrupt.
- ***DMA2_IT_GL4 :*** DMA2 Channel4 global interrupt.
- ***DMA2_IT_TC4 :*** DMA2 Channel4 transfer complete interrupt.
- ***DMA2_IT_HT4 :*** DMA2 Channel4 half transfer interrupt.
- ***DMA2_IT_TE4 :*** DMA2 Channel4 transfer error interrupt.
- ***DMA2_IT_GL5 :*** DMA2 Channel5 global interrupt.
- ***DMA2_IT_TC5 :*** DMA2 Channel5 transfer complete interrupt.
- ***DMA2_IT_HT5 :*** DMA2 Channel5 half transfer interrupt.
- ***DMA2_IT_TE5 :*** DMA2 Channel5 transfer error interrupt.

Return values

- None.

Notes

- Clearing the Global interrupt (DMAy_IT_GLx) results in

clearing all other interrupts relative to the same channel (Transfer Complete, Half-transfer Complete and Transfer Error interrupts: DMAy_IT_TCx, DMAy_IT_HTx and DMAy_IT_TEx).

9.3 DMA Firmware driver defines

9.3.1 DMA

DMA

DMA_circular_normal_mode

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

- #define: ***DMA_Mode_Circular DMA_CCR_CIRC***

DMA_data_transfer_direction

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

- #define: ***DMA_DIR_PeripheralDST DMA_CCR_DIR***

DMA_flags_definition

- #define: ***DMA1_FLAG_GL1 ((uint32_t)0x00000001)***

- #define: ***DMA1_FLAG_TC1 ((uint32_t)0x00000002)***

- #define: ***DMA1_FLAG_HT1 ((uint32_t)0x00000004)***

- #define: ***DMA1_FLAG_TE1 ((uint32_t)0x00000008)***

- #define: ***DMA1_FLAG_GL2 ((uint32_t)0x00000010)***
- #define: ***DMA1_FLAG_TC2 ((uint32_t)0x00000020)***
- #define: ***DMA1_FLAG_HT2 ((uint32_t)0x00000040)***
- #define: ***DMA1_FLAG_TE2 ((uint32_t)0x00000080)***
- #define: ***DMA1_FLAG_GL3 ((uint32_t)0x00000100)***
- #define: ***DMA1_FLAG_TC3 ((uint32_t)0x00000200)***
- #define: ***DMA1_FLAG_HT3 ((uint32_t)0x00000400)***
- #define: ***DMA1_FLAG_TE3 ((uint32_t)0x00000800)***
- #define: ***DMA1_FLAG_GL4 ((uint32_t)0x00001000)***
- #define: ***DMA1_FLAG_TC4 ((uint32_t)0x00002000)***
- #define: ***DMA1_FLAG_HT4 ((uint32_t)0x00004000)***
- #define: ***DMA1_FLAG_TE4 ((uint32_t)0x00008000)***

- #define: ***DMA1_FLAG_GL5 ((uint32_t)0x00010000)***
- #define: ***DMA1_FLAG_TC5 ((uint32_t)0x00020000)***
- #define: ***DMA1_FLAG_HT5 ((uint32_t)0x00040000)***
- #define: ***DMA1_FLAG_TE5 ((uint32_t)0x00080000)***
- #define: ***DMA1_FLAG_GL6 ((uint32_t)0x00100000)***
- #define: ***DMA1_FLAG_TC6 ((uint32_t)0x00200000)***
- #define: ***DMA1_FLAG_HT6 ((uint32_t)0x00400000)***
- #define: ***DMA1_FLAG_TE6 ((uint32_t)0x00800000)***
- #define: ***DMA1_FLAG_GL7 ((uint32_t)0x01000000)***
- #define: ***DMA1_FLAG_TC7 ((uint32_t)0x02000000)***
- #define: ***DMA1_FLAG_HT7 ((uint32_t)0x04000000)***
- #define: ***DMA1_FLAG_TE7 ((uint32_t)0x08000000)***

- #define: ***DMA2_FLAG_GL1 ((uint32_t)0x10000001)***
- #define: ***DMA2_FLAG_TC1 ((uint32_t)0x10000002)***
- #define: ***DMA2_FLAG_HT1 ((uint32_t)0x10000004)***
- #define: ***DMA2_FLAG_TE1 ((uint32_t)0x10000008)***
- #define: ***DMA2_FLAG_GL2 ((uint32_t)0x10000010)***
- #define: ***DMA2_FLAG_TC2 ((uint32_t)0x10000020)***
- #define: ***DMA2_FLAG_HT2 ((uint32_t)0x10000040)***
- #define: ***DMA2_FLAG_TE2 ((uint32_t)0x10000080)***
- #define: ***DMA2_FLAG_GL3 ((uint32_t)0x10000100)***
- #define: ***DMA2_FLAG_TC3 ((uint32_t)0x10000200)***
- #define: ***DMA2_FLAG_HT3 ((uint32_t)0x10000400)***
- #define: ***DMA2_FLAG_TE3 ((uint32_t)0x10000800)***

- #define: ***DMA2_FLAG_GL4 ((uint32_t)0x10001000)***
- #define: ***DMA2_FLAG_TC4 ((uint32_t)0x10002000)***
- #define: ***DMA2_FLAG_HT4 ((uint32_t)0x10004000)***
- #define: ***DMA2_FLAG_TE4 ((uint32_t)0x10008000)***
- #define: ***DMA2_FLAG_GL5 ((uint32_t)0x10010000)***
- #define: ***DMA2_FLAG_TC5 ((uint32_t)0x10020000)***
- #define: ***DMA2_FLAG_HT5 ((uint32_t)0x10040000)***
- #define: ***DMA2_FLAG_TE5 ((uint32_t)0x10080000)***

DMA_interrupts_definition

- #define: ***DMA_IT_TC ((uint32_t)0x00000002)***
- #define: ***DMA_IT_HT ((uint32_t)0x00000004)***
- #define: ***DMA_IT_TE ((uint32_t)0x00000008)***
- #define: ***DMA1_IT_GL1 ((uint32_t)0x00000001)***

- #define: ***DMA1_IT_TC1 ((uint32_t)0x00000002)***
- #define: ***DMA1_IT_HT1 ((uint32_t)0x00000004)***
- #define: ***DMA1_IT_TE1 ((uint32_t)0x00000008)***
- #define: ***DMA1_IT_GL2 ((uint32_t)0x00000010)***
- #define: ***DMA1_IT_TC2 ((uint32_t)0x00000020)***
- #define: ***DMA1_IT_HT2 ((uint32_t)0x00000040)***
- #define: ***DMA1_IT_TE2 ((uint32_t)0x00000080)***
- #define: ***DMA1_IT_GL3 ((uint32_t)0x00000100)***
- #define: ***DMA1_IT_TC3 ((uint32_t)0x00000200)***
- #define: ***DMA1_IT_HT3 ((uint32_t)0x00000400)***
- #define: ***DMA1_IT_TE3 ((uint32_t)0x00000800)***
- #define: ***DMA1_IT_GL4 ((uint32_t)0x00001000)***

- #define: ***DMA1_IT_TC4 ((uint32_t)0x00002000)***
- #define: ***DMA1_IT_HT4 ((uint32_t)0x00004000)***
- #define: ***DMA1_IT_TE4 ((uint32_t)0x00008000)***
- #define: ***DMA1_IT_GL5 ((uint32_t)0x00010000)***
- #define: ***DMA1_IT_TC5 ((uint32_t)0x00020000)***
- #define: ***DMA1_IT_HT5 ((uint32_t)0x00040000)***
- #define: ***DMA1_IT_TE5 ((uint32_t)0x00080000)***
- #define: ***DMA1_IT_GL6 ((uint32_t)0x00100000)***
- #define: ***DMA1_IT_TC6 ((uint32_t)0x00200000)***
- #define: ***DMA1_IT_HT6 ((uint32_t)0x00400000)***
- #define: ***DMA1_IT_TE6 ((uint32_t)0x00800000)***
- #define: ***DMA1_IT_GL7 ((uint32_t)0x01000000)***

- #define: ***DMA1_IT_TC7 ((uint32_t)0x02000000)***
- #define: ***DMA1_IT_HT7 ((uint32_t)0x04000000)***
- #define: ***DMA1_IT_TE7 ((uint32_t)0x08000000)***
- #define: ***DMA2_IT_GL1 ((uint32_t)0x10000001)***
- #define: ***DMA2_IT_TC1 ((uint32_t)0x10000002)***
- #define: ***DMA2_IT_HT1 ((uint32_t)0x10000004)***
- #define: ***DMA2_IT_TE1 ((uint32_t)0x10000008)***
- #define: ***DMA2_IT_GL2 ((uint32_t)0x10000010)***
- #define: ***DMA2_IT_TC2 ((uint32_t)0x10000020)***
- #define: ***DMA2_IT_HT2 ((uint32_t)0x10000040)***
- #define: ***DMA2_IT_TE2 ((uint32_t)0x10000080)***
- #define: ***DMA2_IT_GL3 ((uint32_t)0x10000100)***

- #define: ***DMA2_IT_TC3 ((uint32_t)0x10000200)***
- #define: ***DMA2_IT_HT3 ((uint32_t)0x10000400)***
- #define: ***DMA2_IT_TE3 ((uint32_t)0x10000800)***
- #define: ***DMA2_IT_GL4 ((uint32_t)0x10001000)***
- #define: ***DMA2_IT_TC4 ((uint32_t)0x10002000)***
- #define: ***DMA2_IT_HT4 ((uint32_t)0x10004000)***
- #define: ***DMA2_IT_TE4 ((uint32_t)0x10008000)***
- #define: ***DMA2_IT_GL5 ((uint32_t)0x10010000)***
- #define: ***DMA2_IT_TC5 ((uint32_t)0x10020000)***
- #define: ***DMA2_IT_HT5 ((uint32_t)0x10040000)***
- #define: ***DMA2_IT_TE5 ((uint32_t)0x10080000)***

DMA_memory_data_size

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

- #define: ***DMA_MemoryDataSize_HalfWord DMA_CCR_MSIZE_0***

- #define: ***DMA_MemoryDataSize_Word DMA_CCR_MSIZE_1***

DMA_memory_incremented_mode

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

- #define: ***DMA_MemoryInc_Enable DMA_CCR_MINC***

DMA_memory_to_memory

- #define: ***DMA_M2M_Disable ((uint32_t)0x00000000)***

- #define: ***DMA_M2M_Enable DMA_CCR_MEM2MEM***

DMA_peripheral_data_size

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

- #define: ***DMA_PeripheralDataSize_HalfWord DMA_CCR_PSIZE_0***

- #define: ***DMA_PeripheralDataSize_Word DMA_CCR_PSIZE_1***

DMA_peripheral_incremented_mode

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

- #define: ***DMA_PeripheralInc_Enable DMA_CCR_PINC***

DMA_priority_level

- #define: ***DMA_Priority_VeryHigh DMA_CCR_PL***

- #define: ***DMA_Priority_High DMA_CCR_PL_1***

- #define: ***DMA_Priority_Medium DMA_CCR_PL_0***

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

10 External interrupt/event controller (EXTI)

10.1 EXTI Firmware driver registers structures

10.1.1 EXTI_TypeDef

EXTI_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t IMR***
- ***__IO uint32_t EMR***
- ***__IO uint32_t RTSR***
- ***__IO uint32_t FTSR***
- ***__IO uint32_t SWIER***
- ***__IO uint32_t PR***
- ***uint32_t RESERVED1***
- ***uint32_t RESERVED2***
- ***__IO uint32_t IMR2***
- ***__IO uint32_t EMR2***
- ***__IO uint32_t RTSR2***
- ***__IO uint32_t FTSR2***
- ***__IO uint32_t SWIER2***
- ***__IO uint32_t PR2***

Field Documentation

- ***__IO uint32_t EXTI_TypeDef::IMR***
  - EXTI Interrupt mask register, Address offset: 0x00
- ***__IO uint32_t EXTI_TypeDef::EMR***
  - EXTI Event mask register, Address offset: 0x04
- ***__IO uint32_t EXTI_TypeDef::RTSR***
  - EXTI Rising trigger selection register, Address offset: 0x08
- ***__IO uint32_t EXTI_TypeDef::FTSR***
  - EXTI Falling trigger selection register, Address offset: 0x0C
- ***__IO uint32_t EXTI_TypeDef::SWIER***
  - EXTI Software interrupt event register, Address offset: 0x10
- ***__IO uint32_t EXTI_TypeDef::PR***
  - EXTI Pending register, Address offset: 0x14
- ***uint32_t EXTI_TypeDef::RESERVED1***
  - Reserved, 0x18
- ***uint32_t EXTI_TypeDef::RESERVED2***
  - Reserved, 0x1C
- ***__IO uint32_t EXTI_TypeDef::IMR2***
  - EXTI Interrupt mask register, Address offset: 0x20
- ***__IO uint32_t EXTI_TypeDef::EMR2***
  - EXTI Event mask register, Address offset: 0x24
- ***__IO uint32_t EXTI_TypeDef::RTSR2***
  - EXTI Rising trigger selection register, Address offset: 0x28

- ***__IO uint32_t EXTI_TypeDef::FTSR2***
  - EXTI Falling trigger selection register, Address offset: 0x2C
- ***__IO uint32_t EXTI_TypeDef::SWIER2***
  - EXTI Software interrupt event register, Address offset: 0x30
- ***__IO uint32_t EXTI_TypeDef::PR2***
  - EXTI Pending register, Address offset: 0x34

10.1.2 EXTI_InitTypeDef

EXTI_InitTypeDef is defined in the stm32f30x_exti.h

Data Fields

- ***uint32_t EXTI_Line***
- ***EXTIMode_TypeDef EXTI_Mode***
- ***EXTITrigger_TypeDef EXTI_Trigger***
- ***FunctionalState EXTI_LineCmd***

Field Documentation

- ***uint32_t EXTI_InitTypeDef::EXTI_Line***
  - Specifies the EXTI lines to be enabled or disabled. This parameter can be any combination of EXTI_Lines
- ***EXTIMode_TypeDef EXTI_InitTypeDef::EXTI_Mode***
  - Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTIMode_TypeDef***
- ***EXTITrigger_TypeDef EXTI_InitTypeDef::EXTI_Trigger***
  - Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTITrigger_TypeDef***
- ***FunctionalState EXTI_InitTypeDef::EXTI_LineCmd***
  - Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE

10.2 EXTI Firmware driver API description

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

10.2.1 EXTI features

External interrupt/event lines are mapped as following:

1. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.
2. EXTI line 16 is connected to the PVD output
3. EXTI line 17 is connected to the RTC Alarm event
4. EXTI line 18 is connected to USB Device wakeup event
5. EXTI line 19 is connected to the RTC Tamper and TimeStamp events
6. EXTI line 20 is connected to the RTC wakeup event
7. EXTI line 21 is connected to the Comparator 1 wakeup event

8. EXTI line 22 is connected to the Comparator 2 wakeup event
9. EXTI line 23 is connected to the I2C1 wakeup event
10. EXTI line 24 is connected to the I2C2 wakeup event
11. EXTI line 25 is connected to the USART1 wakeup event
12. EXTI line 26 is connected to the USART2 wakeup event
13. EXTI line 27 is reserved
14. EXTI line 28 is connected to the USART3 wakeup event
15. EXTI line 29 is connected to the Comparator 3 event
16. EXTI line 30 is connected to the Comparator 4 event
17. EXTI line 31 is connected to the Comparator 5 event
18. EXTI line 32 is connected to the Comparator 6 event
19. EXTI line 33 is connected to the Comparator 7 event
20. EXTI line 34 is connected for thr UART4 wakeup event
21. EXTI line 35 is connected for the UART5 wakeup event

10.2.2 How to use this driver

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

1. Configure the I/O in input mode using GPIO_Init().
2. Select the input source pin for the EXTI line using SYSCFG_EXTILineConfig().
3. Select the mode(interrupt, event) and configure the trigger selection (Rising, falling or both) using EXTI_Init(). For the internal interrupt, the trigger selection is not needed (the active edge is always the rising one).
4. Configure NVIC IRQ channel mapped to the EXTI line using NVIC_Init().
5. Optionally, you can generate a software interrupt using the function EXTI_GenerateSWInterrupt().

SYSCFG APB clock must be enabled to get write access to SYSCFG_EXTICRx registers using RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

10.2.3 Initialization and Configuration functions

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

10.2.4 Interrupts and flags management functions

This section provides functions allowing to configure the EXTI Interrupts sources and check or clear the flags or pending bits status.

- ***EXTI_GetFlagStatus()***
- ***EXTI_ClearFlag()***
- ***EXTI_GetITStatus()***
- ***EXTI_ClearITPendingBit()***

10.2.5 Initialization and Configuration functions

10.2.5.1 EXTI_DeInit

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

10.2.5.2 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.

10.2.5.3 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.

10.2.5.4 EXTI_GenerateSWInterrupt

Function Name	**void EXTI_GenerateSWInterrupt (uint32_t EXTI_Line)**
Function Description	Generates a Software interrupt on selected EXTI line.
Parameters	• **EXTI_Line :** specifies the EXTI line on which the software interrupt will be generated. This parameter can be any combination of EXTI_Linex where x can be (0..20).
Return values	• None.
Notes	• None.

10.2.6 Interrupts and flags management functions

10.2.6.1 EXTI_GetFlagStatus

Function Name	**FlagStatus EXTI_GetFlagStatus (uint32_t EXTI_Line)**
Function Description	Checks whether the specified EXTI line flag is set or not.
Parameters	• **EXTI_Line :** specifies the EXTI line flag to check. This parameter can be any combination of EXTI_Linex where x can be (0..20).
Return values	• **The new state of EXTI_Line (SET or RESET).**
Notes	• None.

10.2.6.2 EXTI_ClearFlag

Function Name	**void EXTI_ClearFlag (uint32_t EXTI_Line)**
Function Description	Clears the EXTI's line pending flags.
Parameters	• **EXTI_Line :** specifies the EXTI lines flags to clear. This parameter can be any combination of EXTI_Linex where x can be (0..20).

Return values	• None.
Notes	• None.

10.2.6.3 EXTI_GetITStatus

Function Name	**ITStatus EXTI_GetITStatus (uint32_t EXTI_Line)**
Function Description	Checks whether the specified EXTI line is asserted or not.
Parameters	• **EXTI_Line :** specifies the EXTI line to check. This parameter can be any combination of EXTI_Linex where x can be (0..20).
Return values	• **The new state of EXTI_Line (SET or RESET).**
Notes	• None.

10.2.6.4 EXTI_ClearITPendingBit

Function Name	**void EXTI_ClearITPendingBit (uint32_t EXTI_Line)**
Function Description	Clears the EXTI's line pending bits.
Parameters	• **EXTI_Line :** specifies the EXTI lines to clear. This parameter can be any combination of EXTI_Linex where x can be (0..20).
Return values	• None.
Notes	• None.

10.3 EXTI Firmware driver defines

10.3.1 EXTI

EXTI

EXTI_Lines

- #define: ***EXTI_Line0 ((uint32_t)0x00)***

External interrupt line 0

- #define: ***EXTI_Line1 ((uint32_t)0x01)***

External interrupt line 1

- #define: ***EXTI_Line2 ((uint32_t)0x02)***

External interrupt line 2

- #define: ***EXTI_Line3 ((uint32_t)0x03)***

External interrupt line 3

- #define: ***EXTI_Line4 ((uint32_t)0x04)***

External interrupt line 4

- #define: ***EXTI_Line5 ((uint32_t)0x05)***

External interrupt line 5

- #define: ***EXTI_Line6 ((uint32_t)0x06)***

External interrupt line 6

- #define: ***EXTI_Line7 ((uint32_t)0x07)***

External interrupt line 7

- #define: ***EXTI_Line8 ((uint32_t)0x08)***

External interrupt line 8

- #define: ***EXTI_Line9 ((uint32_t)0x09)***

External interrupt line 9

- #define: ***EXTI_Line10 ((uint32_t)0x0A)***

External interrupt line 10

- #define: ***EXTI_Line11 ((uint32_t)0x0B)***

External interrupt line 11

- #define: ***EXTI_Line12 ((uint32_t)0x0C)***

External interrupt line 12

- #define: ***EXTI_Line13 ((uint32_t)0x0D)***

External interrupt line 13

- #define: ***EXTI_Line14 ((uint32_t)0x0E)***

External interrupt line 14

- #define: ***EXTI_Line15 ((uint32_t)0x0F)***

External interrupt line 15

- #define: ***EXTI_Line16 ((uint32_t)0x10)***

External interrupt line 16 Connected to the PVD Output

- #define: ***EXTI_Line17 ((uint32_t)0x11)***

Internal interrupt line 17 Connected to the RTC Alarm event

- #define: ***EXTI_Line18 ((uint32_t)0x12)***

Internal interrupt line 18 Connected to the USB Device Wakeup from suspend event

- #define: ***EXTI_Line19 ((uint32_t)0x13)***

Internal interrupt line 19 Connected to the RTC Tamper and Time Stamp events

- #define: ***EXTI_Line20 ((uint32_t)0x14)***

Internal interrupt line 20 Connected to the RTC wakeup event

- #define: ***EXTI_Line21 ((uint32_t)0x15)***

Internal interrupt line 21 Connected to the Comparator 1 event

- #define: ***EXTI_Line22 ((uint32_t)0x16)***

Internal interrupt line 22 Connected to the Comparator 2 event

- #define: ***EXTI_Line23 ((uint32_t)0x17)***

Internal interrupt line 23 Connected to the I2C1 wakeup event

- #define: ***EXTI_Line24 ((uint32_t)0x18)***

Internal interrupt line 24 Connected to the I2C2 wakeup event

- #define: ***EXTI_Line25 ((uint32_t)0x19)***

Internal interrupt line 25 Connected to the USART1 wakeup event

- #define: ***EXTI_Line26 ((uint32_t)0x1A)***

Internal interrupt line 26 Connected to the USART2 wakeup event

- #define: ***EXTI_Line27 ((uint32_t)0x1B)***

Internal interrupt line 27 reserved

- #define: ***EXTI_Line28 ((uint32_t)0x1C)***

Internal interrupt line 28 Connected to the USART3 wakeup event

- #define: ***EXTI_Line29 ((uint32_t)0x1D)***

Internal interrupt line 29 Connected to the Comparator 3 event

- #define: ***EXTI_Line30 ((uint32_t)0x1E)***

Internal interrupt line 30 Connected to the Comparator 4 event

- #define: ***EXTI_Line31 ((uint32_t)0x1F)***

Internal interrupt line 31 Connected to the Comparator 5 event

- #define: ***EXTI_Line32 ((uint32_t)0x20)***

Internal interrupt line 32 Connected to the Comparator 6 event

- #define: ***EXTI_Line33 ((uint32_t)0x21)***

Internal interrupt line 33 Connected to the Comparator 7 event

- #define: ***EXTI_Line34 ((uint32_t)0x22)***

Internal interrupt line 34 Connected to the USART4 wakeup event

- #define: ***EXTI_Line35 ((uint32_t)0x23)***

Internal interrupt line 35 Connected to the USART5 wakeup event

11 FLASH Memory (FLASH)

11.1 FLASH Firmware driver registers structures

11.1.1 FLASH_TypeDef

FLASH_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__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 RESERVED***
- ***__IO uint32_t OBR***
- ***__IO uint32_t WRPR***

Field Documentation

- ***__IO uint32_t FLASH_TypeDef::ACR***
  - FLASH access control register, Address offset: 0x00
- ***__IO uint32_t FLASH_TypeDef::KEYR***
  - FLASH key register, Address offset: 0x04
- ***__IO uint32_t FLASH_TypeDef::OPTKEYR***
  - FLASH option key register, Address offset: 0x08
- ***__IO uint32_t FLASH_TypeDef::SR***
  - FLASH status register, Address offset: 0x0C
- ***__IO uint32_t FLASH_TypeDef::CR***
  - FLASH control register, Address offset: 0x10
- ***__IO uint32_t FLASH_TypeDef::AR***
  - FLASH address register, Address offset: 0x14
- ***uint32_t FLASH_TypeDef::RESERVED***
  - Reserved, 0x18
- ***__IO uint32_t FLASH_TypeDef::OBR***
  - FLASH Option byte register, Address offset: 0x1C
- ***__IO uint32_t FLASH_TypeDef::WRPR***
  - FLASH Write register, Address offset: 0x20

11.1.2 OB_TypeDef

OB_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint16_t RDP***
- ***__IO uint16_t USER***
- ***uint16_t RESERVED0***
- ***uint16_t RESERVED1***
- ***__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 offset: 0x00
- ***__IO uint16_t OB_TypeDef::USER***
  - FLASH option byte user options, Address offset: 0x02
- ***uint16_t OB_TypeDef::RESERVED0***
  - Reserved, 0x04
- ***uint16_t OB_TypeDef::RESERVED1***
  - Reserved, 0x06
- ***__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: 0x0C
- ***__IO uint16_t OB_TypeDef::WRP2***
  - FLASH option byte write protection 2, Address offset: 0x10
- ***__IO uint16_t OB_TypeDef::WRP3***
  - FLASH option byte write protection 3, Address offset: 0x12

11.2 FLASH Firmware driver API description

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

11.2.1 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32F30x 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 Half Cycle Access.
   - 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 manage the Option Bytes:
   - Lock and Unlock the Flash Option bytes.

- Launch the Option Bytes loader
- Erase the Option Bytes
- Set/Reset the write protection
- Set the Read protection Level
- Program the user option Bytes
- Set/Reset the BOOT1 bit
- Enable/Disable the VDDA Analog Monitoring
- Enable/Disable the SRAM parity
- Get the user option bytes
- Get the Write protection
- Get the read protection status

4. FLASH Interrupts and flags 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.

11.2.2 FLASH Interface configuration functions

This group includes the following functions:

- void FLASH_SetLatency(uint32_t FLASH_Latency);
- void FLASH_HalfCycleAccessCmd(uint32_t FLASH_HalfCycleAccess);
- void FLASH_PrefetchBufferCmd(FunctionalState NewState);

The unlock sequence is not needed for these functions.

- ***FLASH_SetLatency()***
- ***FLASH_HalfCycleAccessCmd()***
- ***FLASH_PrefetchBufferCmd()***

11.2.3 FLASH Memory Programming functions

This group includes the following functions:

- void FLASH_Unlock(void);
- void FLASH_Lock(void);
- 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 operation of erase or program should follow these steps:

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

- ***FLASH_Unlock()***
- ***FLASH_Lock()***
- ***FLASH_ErasePage()***

- ***FLASH_EraseAllPages()***
- ***FLASH_ProgramWord()***
- ***FLASH_ProgramHalfWord()***

11.2.4 Option Bytes Programming functions

This group includes the following functions:

- void FLASH_OB_Unlock(void);
- void FLASH_OB_Lock(void);
- void FLASH_OB_Erase(void);
- FLASH_Status FLASH_OB_WRPConfig(uint32_t OB_WRP, FunctionalState NewState);
- FLASH_Status FLASH_OB_RDPConfig(uint8_t OB_RDP);
- FLASH_Status FLASH_OB_UserConfig(uint8_t OB_IWDG, uint8_t OB_STOP, uint8_t OB_STDBY);
- FLASH_Status FLASH_OB_BOOTConfig(uint8_t OB_BOOT1);
- FLASH_Status FLASH_OB_VDDAConfig(uint8_t OB_VDDA_ANALOG);
- FLASH_Status FLASH_OB_SRMParityConfig(uint8_t OB_SRAM_Parity);
- FLASH_Status FLASH_OB_WriteUser(uint8_t OB_USER);
- FLASH_Status FLASH_OB_Launch(void);
- uint32_t FLASH_OB_GetUser(void);
- uint8_t FLASH_OB_GetWRP(void);
- uint8_t FLASH_OB_GetRDP(void);

Any operation of erase or program should follow these steps:

1. Call the FLASH_OB_Unlock() function to enable the FLASH option control register access.
2. Call one or several functions to program the desired Option Bytes:
   - void FLASH_OB_WRPConfig(uint32_t OB_WRP, FunctionalState NewState); => to Enable/Disable the desired sector write protection.
   - FLASH_Status FLASH_OB_RDPConfig(uint8_t OB_RDP) => to set the desired read Protection Level.
   - FLASH_Status FLASH_OB_UserConfig(uint8_t OB_IWDG, uint8_t OB_STOP, uint8_t OB_STDBY); => to configure the user Option Bytes.
   - FLASH_Status FLASH_OB_BOOTConfig(uint8_t OB_BOOT1); => to set the boot1 mode
   - FLASH_Status FLASH_OB_VDDAConfig(uint8_t OB_VDDA_ANALOG); => to Enable/Disable the VDDA monotoring.
   - FLASH_Status FLASH_OB_SRMParityConfig(uint8_t OB_SRAM_Parity); => to Enable/Disable the SRAM Parity check.
   - FLASH_Status FLASH_OB_WriteUser(uint8_t OB_USER); => to write all user option bytes: OB_IWDG, OB_STOP, OB_STDBY, OB_BOOT1, OB_VDDA_ANALOG and OB_VDD_SD12.
3. Once all needed Option Bytes to be programmed are correctly written, call the FLASH_OB_Launch() function to launch the Option Bytes programming process. (#@) When changing the IWDG mode from HW to SW or from SW to HW, a system reset is needed to make the change effective.
4. Call the FLASH_OB_Lock() function to disable the FLASH option control register access (recommended to protect the Option Bytes against possible unwanted operations).

- ***FLASH_OB_Unlock()***
- ***FLASH_OB_Lock()***

- *FLASH_OB_Launch()*
- *FLASH_OB_Erase()*
- *FLASH_OB_EnableWRP()*
- *FLASH_OB_RDPConfig()*
- *FLASH_OB_UserConfig()*
- *FLASH_OB_BOOTConfig()*
- *FLASH_OB_VDDAConfig()*
- *FLASH_OB_SRAMParityConfig()*
- *FLASH_OB_WriteUser()*
- *FLASH_ProgramOptionByteData()*
- *FLASH_OB_GetUser()*
- *FLASH_OB_GetWRP()*
- *FLASH_OB_GetRDP()*

11.2.5 Interrupts and flags management functions

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

11.2.6 FLASH Interface configuration functions

11.2.6.1 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.

11.2.6.2 FLASH_HalfCycleAccessCmd

Function Name	**void FLASH_HalfCycleAccessCmd (*FunctionalState* NewState)**

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.

11.2.6.3 FLASH_PrefetchBufferCmd

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

11.2.7 FLASH Memory Programming functions

11.2.7.1 FLASH_Unlock

Function Name	**void FLASH_Unlock (void)**
Function Description	Unlocks the FLASH control register access.
Parameters	• None.
Return values	• None.
Notes	• None.

11.2.7.2 FLASH_Lock

Function Name	**void FLASH_Lock (void)**
Function Description	Locks the FLASH control register access.
Parameters	• None.
Return values	• None.
Notes	• None.

11.2.7.3 FLASH_ErasePage

Function Name	**FLASH_Status FLASH_ErasePage (uint32_t Page_Address)**
Function Description	Erases a specified page in program memory.
Parameters	• **Page_Address :** The page address in program memory to be erased.
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PROGRAM, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• To correctly run this function, the FLASH_Unlock() function must be called before. • Call the FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation) • A Page is erased in the Program memory only if the address to load is the start address of a page (multiple of 1024 bytes).

11.2.7.4 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
- To correctly run this function, the FLASH_Unlock() function must be called before. all the FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

11.2.7.5 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
- To correctly run this function, the FLASH_Unlock() function must be called before. Call the FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

11.2.7.6 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
- To correctly run this function, the FLASH_Unlock() function must be called before. Call the FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

11.2.8 Option Bytes Programming functions

11.2.8.1 FLASH_OB_Unlock

Function Name	**void FLASH_OB_Unlock (void)**
Function Description	Unlocks the option bytes block access.
Parameters	• None.
Return values	• None.
Notes	• None.

11.2.8.2 FLASH_OB_Lock

Function Name	**void FLASH_OB_Lock (void)**
Function Description	Locks the option bytes block access.
Parameters	• None.
Return values	• None.
Notes	• None.

11.2.8.3 FLASH_OB_Launch

Function Name	**void FLASH_OB_Launch (void)**
Function Description	Launch the option byte loading.
Parameters	• None.
Return values	• None.
Notes	• None.

11.2.8.4 FLASH_OB_Erase

Function Name	**FLASH_Status FLASH_OB_Erase (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).

11.2.8.5 FLASH_OB_EnableWRP

Function Name	**FLASH_Status FLASH_OB_EnableWRP (uint32_t OB_WRP)**
Function Description	Write protects the desired pages.
Parameters	• **OB_WRP :** specifies the address of the pages to be write protected. This parameter can be: – ***value :*** – ***OB_WRP_AllPages :***
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PROGRAM, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• To correctly run this function, the FLASH_OB_Unlock() function must be called before. • Call the FLASH_OB_Lock() to disable the flash control register access and the option bytes (recommended to protect the FLASH memory against possible unwanted operation)

11.2.8.6 FLASH_OB_RDPConfig

Function Name	**FLASH_Status FLASH_OB_RDPConfig (uint8_t OB_RDP)**

Function Description	Enables or disables the read out protection.
Parameters	• **FLASH_ReadProtection_Level :** specifies the read protection level. This parameter can be: – ***OB_RDP_Level_0 :*** No protection – ***OB_RDP_Level_1 :*** Read protection of the memory – ***OB_RDP_Level_2 :*** Chip protection
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PROGRAM, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• To correctly run this function, the FLASH_OB_Unlock() function must be called before. • Call the FLASH_OB_Lock() to disable the flash control register access and the option bytes (recommended to protect the FLASH memory against possible unwanted operation)

11.2.8.7 FLASH_OB_UserConfig

Function Name	**FLASH_Status FLASH_OB_UserConfig (uint8_t OB_IWDG, uint8_t OB_STOP, uint8_t OB_STDBY)**
Function Description	Programs the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY.
Parameters	• **OB_IWDG :** Selects the IWDG mode This parameter can be one of the following values: – ***OB_IWDG_SW :*** Software IWDG selected – ***OB_IWDG_HW :*** Hardware IWDG selected • **OB_STOP :** Reset event when entering STOP mode. This parameter can be one of the following values: – ***OB_STOP_NoRST :*** No reset generated when entering in STOP – ***OB_STOP_RST :*** Reset generated when entering in STOP • **OB_STDBY :** Reset event when entering Standby mode. This parameter can be one of the following values: – ***OB_STDBY_NoRST :*** No reset generated when entering in STANDBY – ***OB_STDBY_RST :*** Reset generated when entering in STANDBY
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PG, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

11.2.8.8 FLASH_OB_BOOTConfig

Function Name	**FLASH_Status FLASH_OB_BOOTConfig (uint8_t OB_BOOT1)**
Function Description	Sets or resets the BOOT1.
Parameters	• **OB_BOOT1 :** Set or Reset the BOOT1. This parameter can be one of the following values: – ***OB_BOOT1_RESET :*** BOOT1 Reset – ***OB_BOOT1_SET :*** BOOT1 Set
Return values	• None.
Notes	• None.

11.2.8.9 FLASH_OB_VDDAConfig

Function Name	**FLASH_Status FLASH_OB_VDDAConfig (uint8_t OB_VDDA_ANALOG)**
Function Description	Sets or resets the analogue monitoring on VDDA Power source.
Parameters	• **OB_VDDA_ANALOG :** Selects the analog monitoring on VDDA Power source. This parameter can be one of the following values: – ***OB_VDDA_ANALOG_ON :*** Analog monitoring on VDDA Power source ON – ***OB_VDDA_ANALOG_OFF :*** Analog monitoring on VDDA Power source OFF
Return values	• None.
Notes	• None.

11.2.8.10 FLASH_OB_SRAMParityConfig

Function Name	**FLASH_Status FLASH_OB_SRAMParityConfig (uint8_t OB_SRAM_Parity)**
Function Description	Sets or resets the SRAM partiy.

Parameters	• **OB_SRAM_Parity :** Set or Reset the SRAM partiy enable bit. This parameter can be one of the following values: – ***OB_SRAM_PARITY_SET :*** Set SRAM partiy. – ***OB_SRAM_PARITY_RESET :*** Reset SRAM partiy.
Return values	• None.
Notes	• None.

11.2.8.11 FLASH_OB_WriteUser

Function Name	**FLASH_Status FLASH_OB_WriteUser (uint8_t OB_USER)**
Function Description	Programs the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY/ BOOT1 and OB_VDDA_ANALOG.
Parameters	• **OB_USER :** Selects all user option bytes This parameter is a combination of the following values: – ***OB_IWDG_SW / OB_IWDG_HW :*** Software / Hardware WDG selected – ***OB_STOP_NoRST / OB_STOP_RST :*** No reset / Reset generated when entering in STOP – ***OB_STDBY_NoRST / OB_STDBY_RST :*** No reset / Reset generated when entering in STANDBY – ***OB_BOOT1_RESET / OB_BOOT1_SET :*** BOOT1 Reset / Set – ***OB_VDDA_ANALOG_ON / OB_VDDA_ANALOG_OFF :*** Analog monitoring on VDDA Power source ON / OFF
Return values	• **FLASH Status: The returned value can be: FLASH_ERROR_PROGRAM, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• To correctly run this function, the FLASH_OB_Unlock() function must be called before. • Call the FLASH_OB_Lock() to disable the flash control register access and the option bytes (recommended to protect the FLASH memory against possible unwanted operation)

11.2.8.12 FLASH_ProgramOptionByteData

Function Name	**FLASH_Status FLASH_ProgramOptionByteData (uint32_t Address, uint8_t Data)**

Function Description	Programs a half word at a specified Option Byte Data address.
Parameters	• **Address :** specifies the address to be programmed. This parameter can be 0x1FFFF804 or 0x1FFFF806. • **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	• To correctly run this function, the FLASH_OB_Unlock() function must be called before. Call the FLASH_OB_Lock() to disable the flash control register access and the option bytes (recommended to protect the FLASH memory against possible unwanted operation)

11.2.8.13 FLASH_OB_GetUser

Function Name	**uint8_t FLASH_OB_GetUser (void)**
Function Description	Returns the FLASH User Option Bytes values.
Parameters	• None.
Return values	• **The FLASH User Option Bytes .**
Notes	• None.

11.2.8.14 FLASH_OB_GetWRP

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

11.2.8.15 FLASH_OB_GetRDP

Function Name	**FlagStatus FLASH_OB_GetRDP (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.

11.2.9 Interrupts and flags management functions

11.2.9.1 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_EOP :*** FLASH end of programming Interrupt – ***FLASH_IT_ERR :*** FLASH Error Interrupt
Return values	• None.
Notes	• None.

11.2.9.2 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 write/erase operations in progress flag – ***FLASH_FLAG_PGERR :*** FLASH Programming error

flag flag
- ***FLASH_FLAG_WRPERR :*** FLASH Write protected error flag
- ***FLASH_FLAG_EOP :*** FLASH End of Programming flag

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

Notes
- None.

11.2.9.3 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 Programming error flag flag
  - ***FLASH_FLAG_WRPERR :*** FLASH Write protected error flag
  - ***FLASH_FLAG_EOP :*** FLASH End of Programming flag

Return values
- None.

Notes
- None.

11.2.9.4 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_PROGRAM, FLASH_ERROR_WRP or FLASH_COMPLETE.**

Notes
- None.

11.2.9.5 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_BUSY, FLASH_ERROR_PROGRAM, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.**
Notes	• None.

11.3 FLASH Firmware driver defines

11.3.1 FLASH

FLASH

FLASH_Flags

- #define: ***FLASH_FLAG_BSY FLASH_SR_BSY***

FLASH Busy flag

- #define: ***FLASH_FLAG_PGERR FLASH_SR_PGERR***

FLASH Programming error flag

- #define: ***FLASH_FLAG_WRPERR FLASH_SR_WRPERR***

FLASH Write protected error flag

- #define: ***FLASH_FLAG_EOP FLASH_SR_EOP***

FLASH End of Programming flag

FLASH_Interrupts

- #define: ***FLASH_IT_EOP FLASH_CR_EOPIE***

End of programming interrupt source

- #define: ***FLASH_IT_ERR FLASH_CR_ERRIE***

Error interrupt source

Flash_Latency

- #define: ***FLASH_Latency_0 ((uint8_t)0x0000)***

FLASH Zero Latency cycle

- #define: ***FLASH_Latency_1 FLASH_ACR_LATENCY_0***

FLASH One Latency cycle

- #define: ***FLASH_Latency_2 FLASH_ACR_LATENCY_1***

FLASH Two Latency cycles

FLASH_Option_Bytes_SRAM_Parity_Enable

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

SRAM parity enable Set

- #define: ***OB_SRAM_PARITY_RESET ((uint8_t)0x40)***

SRAM parity enable reset

12 General-purpose I/Os (GPIO)

12.1 GPIO Firmware driver registers structures

12.1.1 GPIO_TypeDef

GPIO_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t MODER***
- ***__IO uint16_t OTYPER***
- ***uint16_t RESERVED0***
- ***__IO uint32_t OSPEEDR***
- ***__IO uint32_t PUPDR***
- ***__IO uint16_t IDR***
- ***uint16_t RESERVED1***
- ***__IO uint16_t ODR***
- ***uint16_t RESERVED2***
- ***__IO uint32_t BSRR***
- ***__IO uint32_t LCKR***
- ***__IO uint32_t AFR***
- ***__IO uint16_t BRR***
- ***uint16_t RESERVED3***

Field Documentation

- ***__IO uint32_t GPIO_TypeDef::MODER***
  - GPIO port mode register, Address offset: 0x00
- ***__IO uint16_t GPIO_TypeDef::OTYPER***
  - GPIO port output type register, Address offset: 0x04
- ***uint16_t GPIO_TypeDef::RESERVED0***
  - Reserved, 0x06
- ***__IO uint32_t GPIO_TypeDef::OSPEEDR***
  - GPIO port output speed register, Address offset: 0x08
- ***__IO uint32_t GPIO_TypeDef::PUPDR***
  - GPIO port pull-up/pull-down register, Address offset: 0x0C
- ***__IO uint16_t GPIO_TypeDef::IDR***
  - GPIO port input data register, Address offset: 0x10
- ***uint16_t GPIO_TypeDef::RESERVED1***
  - Reserved, 0x12
- ***__IO uint16_t GPIO_TypeDef::ODR***
  - GPIO port output data register, Address offset: 0x14
- ***uint16_t GPIO_TypeDef::RESERVED2***
  - Reserved, 0x16
- ***__IO uint32_t GPIO_TypeDef::BSRR***
  - GPIO port bit set/reset registerBSRR, Address offset: 0x18
- ***__IO uint32_t GPIO_TypeDef::LCKR***
  - GPIO port configuration lock register, Address offset: 0x1C

- ***__IO uint32_t GPIO_TypeDef::AFR[2]***
  - GPIO alternate function low register, Address offset: 0x20-0x24
- ***__IO uint16_t GPIO_TypeDef::BRR***
  - GPIO bit reset register, Address offset: 0x28
- ***uint16_t GPIO_TypeDef::RESERVED3***
  - Reserved, 0x2A

12.1.2 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32f30x_gpio.h

Data Fields

- ***uint32_t GPIO_Pin***
- ***GPIOMode_TypeDef GPIO_Mode***
- ***GPIOSpeed_TypeDef GPIO_Speed***
- ***GPIOOType_TypeDef GPIO_OType***
- ***GPIOPuPd_TypeDef GPIO_PuPd***

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***
- ***GPIOSpeed_TypeDef GPIO_InitTypeDef::GPIO_Speed***
  - Specifies the speed for the selected pins. This parameter can be a value of ***GPIOSpeed_TypeDef***
- ***GPIOOType_TypeDef GPIO_InitTypeDef::GPIO_OType***
  - Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIOOType_TypeDef***
- ***GPIOPuPd_TypeDef GPIO_InitTypeDef::GPIO_PuPd***
  - Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIOPuPd_TypeDef***

12.2 GPIO Firmware driver API description

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

12.2.1 How to use this driver

1. Enable the GPIO AHB clock using RCC_AHBPeriphClockCmd()
2. 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). In output mode, the speed is configurable: Low, Medium, Fast or High.
  - Alternate Function: Push-Pull (Pull-up, Pull-down or no Pull), Open Drain (Pull-up, Pull-down or no Pull).
  - Analog: required mode when a pin is to be used as ADC channel, DAC output or comparator input.

3. Peripherals alternate function:
  - For ADC, DAC and comparators, configure the desired pin in analog mode using GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AN
  - For other peripherals (TIM, USART...):
    - Connect the pin to the desired peripherals' Alternate Function (AF) using GPIO_PinAFConfig() function.
    - Configure the desired pin in alternate function mode using GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF
    - Select the type, pull-up/pull-down and output speed via GPIO_PuPd, GPIO_OType and GPIO_Speed members.
    - Call GPIO_Init() function.
4. To get the level of a pin configured in input mode use GPIO_ReadInputDataBit()
5. To set/reset the level of a pin configured in output mode use GPIO_SetBits()/GPIO_ResetBits()
6. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
7. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general-purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
8. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general-purpose (PF0 and PF1 respectively) when the HSE oscillator is off. The HSE has the priority over the GPIO function.

12.2.2 Initialization and Configuration

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

12.2.3 GPIO Read and Write

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

12.2.4 GPIO Alternate functions configuration functions

- *GPIO_PinAFConfig()*

12.2.5 Initialization and Configuration

12.2.5.1 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, B, C, D, E or F) to select the GPIO peripheral.
Return values	• None.
Notes	• None.

12.2.5.2 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, C, D, E or F) to select the GPIO peripheral. • **GPIO_InitStruct :** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• None.
Notes	• GPIO_Pin: selects the pin to be configured: GPIO_Pin_0->GPIO_Pin_15 for GPIOA, GPIOB, GPIOC, GPIOD and GPIOE; GPIO_Pin_0->GPIO_Pin_2, GPIO_Pin_4, GPIO_Pin_6, GPIO_Pin_9 and GPIO_Pin_10 for GPIOF.

12.2.5.3 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.

12.2.5.4 GPIO_PinLockConfig

Function Name	**void GPIO_PinLockConfig (*GPIO_TypeDef** * GPIOx, uint16_t GPIO_Pin)**
Function Description	Locks GPIO Pins configuration registers.
Parameters	• **GPIOx :** where x can be (A or B or D) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
Return values	• None.
Notes	• The configuration of the locked GPIO pins can no longer be modified until the next reset.

12.2.6 GPIO Read and Write functions

12.2.6.1 GPIO_ReadInputDataBit

Function Name	**uint8_t GPIO_ReadInputDataBit (*GPIO_TypeDef** * GPIOx, uint16_t GPIO_Pin)**
Function Description	Reads the specified input port pin.
Parameters	• **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bit to read.
Return values	• **The input port pin value.**
Notes	• This parameter can be GPIO_Pin_x where x can be : (0..15) for GPIOA, GPIOB, GPIOC, GPIOD or GPIOE; (0..2, 4, 6, 9..10) for GPIOF.

12.2.6.2 GPIO_ReadInputData

Function Name	**uint16_t GPIO_ReadInputData (*GPIO_TypeDef** * **GPIOx)**
Function Description	Reads the specified input port pin.
Parameters	• **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral.
Return values	• **The input port pin value.**
Notes	• None.

12.2.6.3 GPIO_ReadOutputDataBit

Function Name	**uint8_t GPIO_ReadOutputDataBit (*GPIO_TypeDef** * **GPIOx, uint16_t GPIO_Pin)**
Function Description	Reads the specified output data port bit.
Parameters	• **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral. • **GPIO_Pin :** Specifies the port bit to read.
Return values	• **The output port pin value.**
Notes	• This parameter can be GPIO_Pin_x where x can be : (0..15) for GPIOA, GPIOB, GPIOC, GPIOD or GPIOE; (0..2, 4, 6, 9..10) for GPIOF.

12.2.6.4 GPIO_ReadOutputData

Function Name	**uint16_t GPIO_ReadOutputData (*GPIO_TypeDef** * **GPIOx)**
Function Description	Reads the specified GPIO output data port.
Parameters	• **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral.

Return values	• **GPIO output data port value.**
Notes	• None.

12.2.6.5 GPIO_SetBits

Function Name	**void GPIO_SetBits (*GPIO_TypeDef* * GPIOx, uint16_t GPIO_Pin)**
Function Description	Sets the selected data port bits.
Parameters	• **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bits to be written.
Return values	• None.
Notes	• This parameter can be GPIO_Pin_x where x can be : (0..15) for GPIOA, GPIOB, GPIOC, GPIOD or GPIOE; (0..2, 4, 6, 9..10) for GPIOF.

12.2.6.6 GPIO_ResetBits

Function Name	**void GPIO_ResetBits (*GPIO_TypeDef* * GPIOx, uint16_t GPIO_Pin)**
Function Description	Clears the selected data port bits.
Parameters	• **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bits to be written.
Return values	• None.
Notes	• This parameter can be GPIO_Pin_x where x can be : (0..15) for GPIOA, GPIOB, GPIOC, GPIOD or GPIOE; (0..2, 4, 6, 9..10) for GPIOF.

12.2.6.7 GPIO_WriteBit

Function Name	**void GPIO_WriteBit (*GPIO_TypeDef** * GPIOx, uint16_t GPIO_Pin, *BitAction* BitVal)**
Function Description	Sets or clears the selected data port bit.
Parameters	• **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral. • **GPIO_Pin :** specifies the port bit to be written.
Parameters	• **BitVal :** specifies the value to be written to the selected bit. This parameter can be one of the BitAction enumeration values: – ***Bit_RESET :*** to clear the port pin – ***Bit_SET :*** to set the port pin
Return values	• None.
Notes	• This parameter can be GPIO_Pin_x where x can be : (0..15) for GPIOA, GPIOB, GPIOC, GPIOD or GPIOE; (0..2, 4, 6, 9..10) for GPIOF.

12.2.6.8 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, C, D, E or F) to select the GPIO peripheral. • **PortVal :** specifies the value to be written to the port output data register.
Return values	• None.
Notes	• None.

12.2.7 GPIO Alternate functions configuration functions

12.2.7.1 GPIO_PinAFConfig

Function Name	**void GPIO_PinAFConfig (*GPIO_TypeDef** * GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF)**

Function Description — Writes data to the specified GPIO data port.

Parameters

- **GPIOx :** where x can be (A, B, C, D, E or F) to select the GPIO peripheral.
- **GPIO_PinSource :** specifies the pin for the Alternate function. This parameter can be GPIO_PinSourcex where x can be (0..15).
- **GPIO_AF :** selects the pin to be used as Alternate function. This parameter can be one of the following value:
  - ***GPIO_AF_0 :*** JTCK-SWCLK, JTDI, JTDO/TRACESW0, JTMS-SWDAT, MCO, NJTRST, TRACED, TRACECK.
  - ***GPIO_AF_1 :*** OUT, TIM2, TIM15, TIM16, TIM17.
  - ***GPIO_AF_2 :*** COMP1_OUT, TIM1, TIM2, TIM3, TIM4, TIM8, TIM15.
  - ***GPIO_AF_3 :*** COMP7_OUT, TIM8, TIM15, Touch.
  - ***GPIO_AF_4 :*** I2C1, I2C2, TIM1, TIM8, TIM16, TIM17.
  - ***GPIO_AF_5 :*** IR_OUT, I2S2, I2S3, SPI1, SPI2, TIM8, USART4, USART5
  - ***GPIO_AF_6 :*** IR_OUT, I2S2, I2S3, SPI2, SPI3, TIM1, TIM8
  - ***GPIO_AF_7 :*** AOP2_OUT, CAN, COMP3_OUT, COMP5_OUT, COMP6_OUT, USART1, USART2, USART3.
  - ***GPIO_AF_8 :*** COMP1_OUT, COMP2_OUT, COMP3_OUT, COMP4_OUT, COMP5_OUT, COMP6_OUT.
  - ***GPIO_AF_9 :*** AOP4_OUT, CAN, TIM1, TIM8, TIM15.
  - ***GPIO_AF_10 :*** AOP1_OUT, AOP3_OUT, TIM2, TIM3, TIM4, TIM8, TIM17.
  - ***GPIO_AF_11 :*** TIM1, TIM8.
  - ***GPIO_AF_12 :*** TIM1.
  - ***GPIO_AF_14 :*** USBDM, USBDP.
  - ***GPIO_AF_15 :*** OUT.

Return values

- None.

Notes

- The pin should already been configured in Alternate Function mode(AF) using GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF
- Refer to the Alternate function mapping table in the device datasheet for the detailed mapping of the system and peripherals alternate function I/O pins.

12.3 GPIO Firmware driver defines

12.3.1 GPIO

GPIO

GPIO_Alternate_function_selection_define

- #define: ***GPIO_AF_0 ((uint8_t)0x00)***

- #define: ***GPIO_AF_1 ((uint8_t)0x01)***
- #define: ***GPIO_AF_2 ((uint8_t)0x02)***
- #define: ***GPIO_AF_3 ((uint8_t)0x03)***
- #define: ***GPIO_AF_4 ((uint8_t)0x04)***
- #define: ***GPIO_AF_5 ((uint8_t)0x05)***
- #define: ***GPIO_AF_6 ((uint8_t)0x06)***
- #define: ***GPIO_AF_7 ((uint8_t)0x07)***
- #define: ***GPIO_AF_8 ((uint8_t)0x08)***
- #define: ***GPIO_AF_9 ((uint8_t)0x09)***
- #define: ***GPIO_AF_10 ((uint8_t)0x0A)***
- #define: ***GPIO_AF_11 ((uint8_t)0x0B)***
- #define: ***GPIO_AF_12 ((uint8_t)0x0E)***

- #define: ***GPIO_AF_14 ((uint8_t)0x0E)***

- #define: ***GPIO_AF_15 ((uint8_t)0x0F)***

GPIO_pins_define

- #define: ***GPIO_Pin_0 ((uint16_t)0x0001)***

Pin 0 selected

- #define: ***GPIO_Pin_1 ((uint16_t)0x0002)***

Pin 1 selected

- #define: ***GPIO_Pin_2 ((uint16_t)0x0004)***

Pin 2 selected

- #define: ***GPIO_Pin_3 ((uint16_t)0x0008)***

Pin 3 selected

- #define: ***GPIO_Pin_4 ((uint16_t)0x0010)***

Pin 4 selected

- #define: ***GPIO_Pin_5 ((uint16_t)0x0020)***

Pin 5 selected

- #define: ***GPIO_Pin_6 ((uint16_t)0x0040)***

Pin 6 selected

- #define: ***GPIO_Pin_7 ((uint16_t)0x0080)***

Pin 7 selected

- #define: ***GPIO_Pin_8 ((uint16_t)0x0100)***

Pin 8 selected

- #define: ***GPIO_Pin_9 ((uint16_t)0x0200)***

Pin 9 selected

- #define: ***GPIO_Pin_10 ((uint16_t)0x0400)***

Pin 10 selected

- #define: ***GPIO_Pin_11 ((uint16_t)0x0800)***

Pin 11 selected

- #define: ***GPIO_Pin_12 ((uint16_t)0x1000)***

Pin 12 selected

- #define: ***GPIO_Pin_13 ((uint16_t)0x2000)***

Pin 13 selected

- #define: ***GPIO_Pin_14 ((uint16_t)0x4000)***

Pin 14 selected

- #define: ***GPIO_Pin_15 ((uint16_t)0x8000)***

Pin 15 selected

- #define: ***GPIO_Pin_All ((uint16_t)0xFFFF)***

All pins selected

GPIO_Pin_sources

- #define: ***GPIO_PinSource0 ((uint8_t)0x00)***

- #define: ***GPIO_PinSource1 ((uint8_t)0x01)***

- #define: ***GPIO_PinSource2 ((uint8_t)0x02)***

- #define: ***GPIO_PinSource3 ((uint8_t)0x03)***

- #define: ***GPIO_PinSource4 ((uint8_t)0x04)***
- #define: ***GPIO_PinSource5 ((uint8_t)0x05)***
- #define: ***GPIO_PinSource6 ((uint8_t)0x06)***
- #define: ***GPIO_PinSource7 ((uint8_t)0x07)***
- #define: ***GPIO_PinSource8 ((uint8_t)0x08)***
- #define: ***GPIO_PinSource9 ((uint8_t)0x09)***
- #define: ***GPIO_PinSource10 ((uint8_t)0x0A)***
- #define: ***GPIO_PinSource11 ((uint8_t)0x0B)***
- #define: ***GPIO_PinSource12 ((uint8_t)0x0C)***
- #define: ***GPIO_PinSource13 ((uint8_t)0x0D)***
- #define: ***GPIO_PinSource14 ((uint8_t)0x0E)***
- #define: ***GPIO_PinSource15 ((uint8_t)0x0F)***

13 Inter-integrated circuit interface (I2C)

13.1 I2C Firmware driver registers structures

13.1.1 I2C_TypeDef

I2C_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t OAR1***
- ***__IO uint32_t OAR2***
- ***__IO uint32_t TIMINGR***
- ***__IO uint32_t TIMEOUTR***
- ***__IO uint32_t ISR***
- ***__IO uint32_t ICR***
- ***__IO uint32_t PECR***
- ***__IO uint32_t RXDR***
- ***__IO uint32_t TXDR***

Field Documentation

- ***__IO uint32_t I2C_TypeDef::CR1***
  - I2C Control register 1, Address offset: 0x00
- ***__IO uint32_t I2C_TypeDef::CR2***
  - I2C Control register 2, Address offset: 0x04
- ***__IO uint32_t I2C_TypeDef::OAR1***
  - I2C Own address 1 register, Address offset: 0x08
- ***__IO uint32_t I2C_TypeDef::OAR2***
  - I2C Own address 2 register, Address offset: 0x0C
- ***__IO uint32_t I2C_TypeDef::TIMINGR***
  - I2C Timing register, Address offset: 0x10
- ***__IO uint32_t I2C_TypeDef::TIMEOUTR***
  - I2C Timeout register, Address offset: 0x14
- ***__IO uint32_t I2C_TypeDef::ISR***
  - I2C Interrupt and status register, Address offset: 0x18
- ***__IO uint32_t I2C_TypeDef::ICR***
  - I2C Interrupt clear register, Address offset: 0x1C
- ***__IO uint32_t I2C_TypeDef::PECR***
  - I2C PEC register, Address offset: 0x20
- ***__IO uint32_t I2C_TypeDef::RXDR***
  - I2C Receive data register, Address offset: 0x24
- ***__IO uint32_t I2C_TypeDef::TXDR***
  - I2C Transmit data register, Address offset: 0x28

13.1.2 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32f30x_i2c.h

Data Fields

- ***uint32_t I2C_Timing***
- ***uint32_t I2C_AnalogFilter***
- ***uint32_t I2C_DigitalFilter***
- ***uint32_t I2C_Mode***
- ***uint32_t I2C_OwnAddress1***
- ***uint32_t I2C_Ack***
- ***uint32_t I2C_AcknowledgedAddress***

Field Documentation

- ***uint32_t I2C_InitTypeDef::I2C_Timing***
  - Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32_t I2C_InitTypeDef::I2C_AnalogFilter***
  - Enables or disables analog noise filter. This parameter can be a value of ***I2C_Analog_Filter***
- ***uint32_t I2C_InitTypeDef::I2C_DigitalFilter***
  - Configures the digital noise filter. This parameter can be a number between 0x00 and 0x0F
- ***uint32_t I2C_InitTypeDef::I2C_Mode***
  - Specifies the I2C mode. This parameter can be a value of ***I2C_mode***
- ***uint32_t I2C_InitTypeDef::I2C_OwnAddress1***
  - Specifies the device own address 1. This parameter can be a 7-bit or 10-bit address
- ***uint32_t I2C_InitTypeDef::I2C_Ack***
  - Enables or disables the acknowledgement. This parameter can be a value of ***I2C_acknowledgement***
- ***uint32_t I2C_InitTypeDef::I2C_AcknowledgedAddress***
  - Specifies if 7-bit or 10-bit address is acknowledged. This parameter can be a value of ***I2C_acknowledged_address***

13.2 I2C Firmware driver API description

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

13.2.1 How to use this driver

1. Enable peripheral clock using RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2Cx, ENABLE) function for I2C1 or I2C2.
2. Enable SDA, SCL and SMBA (when used) GPIO clocks using RCC_AHBPeriphClockCmd() function.
3. Peripherals alternate function:

- Connect the pin to the desired peripherals' Alternate Function (AF) using GPIO_PinAFConfig() function.
- Configure the desired pin in alternate function by: GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF
- Select the type, OpenDrain and speed via GPIO_PuPd, GPIO_OType and GPIO_Speed members
- Call GPIO_Init() function.

4. Program the Mode, Timing , Own address, Ack and Acknowledged Address using the I2C_Init() function.
5. Optionally you can enable/configure the following parameters without re-initialization (i.e there is no need to call again I2C_Init() function):
   - Enable the acknowledge feature using I2C_AcknowledgeConfig() function.
   - Enable the dual addressing mode using I2C_DualAddressCmd() function.
   - Enable the general call using the I2C_GeneralCallCmd() function.
   - Enable the clock stretching using I2C_StretchClockCmd() function.
   - Enable the PEC Calculation using I2C_CalculatePEC() function.
   - For SMBus Mode:
     - Enable the SMBusAlert pin using I2C_SMBusAlertCmd() function.
6. Enable the NVIC and the corresponding interrupt using the function I2C_ITConfig() if you need to use interrupt mode.
7. When using the DMA mode
   - Configure the DMA using DMA_Init() function.
   - Active the needed channel Request using I2C_DMACmd() function.
8. Enable the I2C using the I2C_Cmd() function.
9. Enable the DMA using the DMA_Cmd() function when using DMA mode in the transfers.

When using I2C in Fast Mode Plus, SCL and SDA pin 20mA current drive capability must be enabled by setting the driving capability control bit in SYSCFG.

13.2.2 Initialization and Configuration functions

This section provides a set of functions allowing to initialize the I2C Mode, I2C Timing, I2C filters, I2C Addressing mode, I2C OwnAddress1.

The I2C_Init() function follows the I2C configuration procedures (these procedures are available in reference manual).

When the Software Reset is performed using I2C_SoftwareResetCmd() function, the internal states machines are reset and communication control bits, as well as status bits come back to their reset value.

Before enabling Stop mode using I2C_StopModeCmd() I2C Clock source must be set to HSI and Digital filters must be disabled.

Before enabling Own Address 2 via I2C_DualAddressCmd() function, OA2 and mask should be configured using I2C_OwnAddress2Config() function.

I2C_SlaveByteControlCmd() enable Slave byte control that allow user to get control of each byte in slave mode when NBYTES is set to 0x01.

- ***I2C_DeInit()***
- ***I2C_Init()***
- ***I2C_StructInit()***

- *I2C_Cmd()*
- *I2C_SoftwareResetCmd()*
- *I2C_ITConfig()*
- *I2C_StretchClockCmd()*
- *I2C_StopModeCmd()*
- *I2C_DualAddressCmd()*
- *I2C_OwnAddress2Config()*
- *I2C_GeneralCallCmd()*
- *I2C_SlaveByteControlCmd()*
- *I2C_SlaveAddressConfig()*
- *I2C_10BitAddressingModeCmd()*

13.2.3 Communications handling functions

This section provides a set of functions that handles I2C communication.

Automatic End mode is enabled using I2C_AutoEndCmd() function. When Reload mode is enabled via I2C_ReloadCmd() AutoEnd bit has no effect.

I2C_NumberOfBytesConfig() function set the number of bytes to be transferred, this configuration should be done before generating start condition in master mode.

When switching from master write operation to read operation in 10Bit addressing mode, master can only sends the 1st 7 bits of the 10 bit address, followed by Read direction by enabling HEADR bit using I2C_10BitAddressHeader() function.

In master mode, when transferring more than 255 bytes Reload mode should be used to handle communication. In the first phase of transfer, Nbytes should be set to 255. After transferring these bytes TCR flag is set and I2C_TransferHandling() function should be called to handle remaining communication.

In master mode, when software end mode is selected when all data is transferred TC flag is set I2C_TransferHandling() function should be called to generate STOP or generate ReStart.

- *I2C_AutoEndCmd()*
- *I2C_ReloadCmd()*
- *I2C_NumberOfBytesConfig()*
- *I2C_MasterRequestConfig()*
- *I2C_GenerateSTART()*
- *I2C_GenerateSTOP()*
- *I2C_10BitAddressHeaderCmd()*
- *I2C_AcknowledgeConfig()*
- *I2C_GetAddressMatched()*
- *I2C_GetTransferDirection()*
- *I2C_TransferHandling()*

13.2.4 SMBUS management functions

This section provides a set of functions that handles SMBus communication and timeouts detection.

The SMBus Device default address (0b1100 001) is enabled by calling I2C_Init() function and setting I2C_Mode member of I2C_InitTypeDef() structure to I2C_Mode_SMBusDevice.

The SMBus Host address (0b0001 000) is enabled by calling I2C_Init() function and setting I2C_Mode member of I2C_InitTypeDef() structure to I2C_Mode_SMBusHost.

The Alert Response Address (0b0001 100) is enabled using I2C_SMBusAlertCmd() function.

To detect cumulative SCL stretch in master and slave mode, TIMEOUTB should be configured (in accordance to SMBus specification) using I2C_TimeoutBConfig() function then I2C_ExtendedClockTimeoutCmd() function should be called to enable the detection.

SCL low timeout is detected by configuring TIMEOUTB using I2C_TimeoutBConfig() function followed by the call of I2C_ClockTimeoutCmd(). When adding to this procedure the call of I2C_IdleClockTimeoutCmd() function, Bus Idle condition (both SCL and SDA high) is detected also.

- ***I2C_SMBusAlertCmd()***
- ***I2C_ClockTimeoutCmd()***
- ***I2C_ExtendedClockTimeoutCmd()***
- ***I2C_IdleClockTimeoutCmd()***
- ***I2C_TimeoutAConfig()***
- ***I2C_TimeoutBConfig()***
- ***I2C_CalculatePEC()***
- ***I2C_PECRequestCmd()***
- ***I2C_GetPEC()***

13.2.5 I2C registers management functions

This section provides a functions that allow user the management of I2C registers.

- ***I2C_ReadRegister()***

13.2.6 Data transfers management functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

The read access of the I2C_RXDR register can be done using the I2C_ReceiveData() function and returns the received value. Whereas a write access to the I2C_TXDR can be done using I2C_SendData() function and stores the written data into TXDR.

- ***I2C_SendData()***
- ***I2C_ReceiveData()***

13.2.7 DMA transfers management functions

This section provides two functions that can be used only in DMA mode.

In DMA Mode, the I2C communication can be managed by 2 DMA Channel requests:

1. I2C_DMAReq_Tx: specifies the Tx buffer DMA transfer request.
2. I2C_DMAReq_Rx: specifies the Rx buffer DMA transfer request.

In this Mode it is advised to use the following function:

- I2C_DMACmd(I2C_TypeDef* I2Cx, uint32_t I2C_DMAReq, FunctionalState NewState);
- ***I2C_DMACmd()***

13.2.8 Interrupts and flags management functions

This section provides functions allowing to configure the I2C Interrupts sources 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 I2C_Group6) .

Polling Mode

In Polling Mode, the I2C communication can be managed by 15 flags:

1. I2C_FLAG_TXE: to indicate the status of Transmit data register empty flag.
2. I2C_FLAG_TXIS: to indicate the status of Transmit interrupt status flag .
3. I2C_FLAG_RXNE: to indicate the status of Receive data register not empty flag.
4. I2C_FLAG_ADDR: to indicate the status of Address matched flag (slave mode).
5. I2C_FLAG_NACKF: to indicate the status of NACK received flag.
6. I2C_FLAG_STOPF: to indicate the status of STOP detection flag.
7. I2C_FLAG_TC: to indicate the status of Transfer complete flag(master mode).
8. I2C_FLAG_TCR: to indicate the status of Transfer complete reload flag.
9. I2C_FLAG_BERR: to indicate the status of Bus error flag.
10. I2C_FLAG_ARLO: to indicate the status of Arbitration lost flag.
11. I2C_FLAG_OVR: to indicate the status of Overrun/Underrun flag.
12. I2C_FLAG_PECERR: to indicate the status of PEC error in reception flag.
13. I2C_FLAG_TIMEOUT: to indicate the status of Timeout or Tlow detection flag.
14. I2C_FLAG_ALERT: to indicate the status of SMBus Alert flag.
15. I2C_FLAG_BUSY: to indicate the status of Bus busy flag.

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

- FlagStatus I2C_GetFlagStatus(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG);
- void I2C_ClearFlag(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG);

Do not use the BUSY flag to handle each data transmission or reception.It is better to use the TXIS and RXNE flags instead.

Interrupt Mode

In Interrupt Mode, the I2C communication can be managed by 7 interrupt sources and 15 pending bits:

Interrupt Source:

1. I2C_IT_ERRI: specifies the interrupt source for the Error interrupt.
2. I2C_IT_TCI: specifies the interrupt source for the Transfer Complete interrupt.
3. I2C_IT_STOPI: specifies the interrupt source for the Stop Detection interrupt.
4. I2C_IT_NACKI: specifies the interrupt source for the Not Acknowledge received interrupt.
5. I2C_IT_ADDRI: specifies the interrupt source for the Address Match interrupt.
6. I2C_IT_RXI: specifies the interrupt source for the RX interrupt.
7. I2C_IT_TXI: specifies the interrupt source for the TX interrupt.

Pending Bits:

1. I2C_IT_TXIS: to indicate the status of Transmit interrupt status flag.
2. I2C_IT_RXNE: to indicate the status of Receive data register not empty flag.
3. I2C_IT_ADDR: to indicate the status of Address matched flag (slave mode).

4. I2C_IT_NACKF: to indicate the status of NACK received flag.
5. I2C_IT_STOPF: to indicate the status of STOP detection flag.
6. I2C_IT_TC: to indicate the status of Transfer complete flag (master mode).
7. I2C_IT_TCR: to indicate the status of Transfer complete reload flag.
8. I2C_IT_BERR: to indicate the status of Bus error flag.
9. I2C_IT_ARLO: to indicate the status of Arbitration lost flag.
10. I2C_IT_OVR: to indicate the status of Overrun/Underrun flag.
11. I2C_IT_PECERR: to indicate the status of PEC error in reception flag.
12. I2C_IT_TIMEOUT: to indicate the status of Timeout or Tlow detection flag.
13. I2C_IT_ALERT: to indicate the status of SMBus Alert flag.

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

- void I2C_ClearITPendingBit(I2C_TypeDef* I2Cx, uint32_t I2C_IT);
- ITStatus I2C_GetITStatus(I2C_TypeDef* I2Cx, uint32_t I2C_IT);
- ***I2C_GetFlagStatus()***
- ***I2C_ClearFlag()***
- ***I2C_GetITStatus()***
- ***I2C_ClearITPendingBit()***

13.2.9 Initialization and Configuration functions

13.2.9.1 I2C_DeInit

Function Name	**void I2C_DeInit (*I2C_TypeDef* * I2Cx)**
Function Description	Deinitializes the I2Cx peripheral registers to their default reset values.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
Return values	• None.
Notes	• None.

13.2.9.2 I2C_Init

Function Name	**void I2C_Init (*I2C_TypeDef* * I2Cx, *I2C_InitTypeDef* * I2C_InitStruct)**
Function Description	Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **I2C_InitStruct :** pointer to a I2C_InitTypeDef structure that contains the configuration information for the specified I2C peripheral.
Return values	• None.

Notes	• None.

13.2.9.3 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 an I2C_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

13.2.9.4 I2C_Cmd

Function Name	**void I2C_Cmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified I2C peripheral.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.9.5 I2C_SoftwareResetCmd

Function Name	**void I2C_SoftwareResetCmd (*I2C_TypeDef** * I2Cx)**
Function Description	Enables or disables the specified I2C software reset.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.

Return values	• None.
Notes	• None.

13.2.9.6 I2C_ITConfig

Function Name	**void I2C_ITConfig (*I2C_TypeDef** * I2Cx, uint32_t I2C_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified I2C interrupts.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **I2C_IT :** specifies the I2C interrupts sources to be enabled or disabled. This parameter can be any combination of the following values: – ***I2C_IT_ERRI :*** Error interrupt mask – ***I2C_IT_TCI :*** Transfer Complete interrupt mask – ***I2C_IT_STOPI :*** Stop Detection interrupt mask – ***I2C_IT_NACKI :*** Not Acknowledge received interrupt mask – ***I2C_IT_ADDRI :*** Address Match interrupt mask – ***I2C_IT_RXI :*** RX interrupt mask – ***I2C_IT_TXI :*** TX interrupt mask • **NewState :** new state of the specified I2C interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.9.7 I2C_StretchClockCmd

Function Name	**void I2C_StretchClockCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables the I2C Clock stretching.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx Clock stretching. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.9.8 I2C_StopModeCmd

Function Name	**void I2C_StopModeCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables I2C wakeup from stop mode.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx stop mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.9.9 I2C_DualAddressCmd

Function Name	**void I2C_DualAddressCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables the I2C own address 2.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2C own address 2. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.9.10 I2C_OwnAddress2Config

Function Name	**void I2C_OwnAddress2Config (*I2C_TypeDef** * I2Cx, uint16_t Address, uint8_t Mask)**
Function Description	Configures the I2C slave own address 2 and mask.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.

- **Address :** specifies the slave address to be programmed.
- **Mask :** specifies own address 2 mask to be programmed. This parameter can be one of the following values:
  - ***I2C_OA2_NoMask :*** no mask.
  - ***I2C_OA2_Mask01 :*** OA2[1] is masked and don't care.
  - ***I2C_OA2_Mask02 :*** OA2[2:1] are masked and don't care.
  - ***I2C_OA2_Mask03 :*** OA2[3:1] are masked and don't care.
  - ***I2C_OA2_Mask04 :*** OA2[4:1] are masked and don't care.
  - ***I2C_OA2_Mask05 :*** OA2[5:1] are masked and don't care.
  - ***I2C_OA2_Mask06 :*** OA2[6:1] are masked and don't care.
  - ***I2C_OA2_Mask07 :*** OA2[7:1] are masked and don't care.

Return values
- None.

Notes
- None.

13.2.9.11 I2C_GeneralCallCmd

Function Name **void I2C_GeneralCallCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**

Function Description Enables or disables the I2C general call mode.

Parameters
- **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
- **NewState :** new state of the I2C general call mode. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

13.2.9.12 I2C_SlaveByteControlCmd

Function Name **void I2C_SlaveByteControlCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**

Function Description Enables or disables the I2C slave byte control.

Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2C slave byte control. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.9.13 I2C_SlaveAddressConfig

Function Name	**void I2C_SlaveAddressConfig (*I2C_TypeDef** * I2Cx, uint16_t Address)**
Function Description	Configures the slave address to be transmitted after start generation.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **Address :** specifies the slave address to be programmed.
Return values	• None.
Notes	• This function should be called before generating start condition.

13.2.9.14 I2C_10BitAddressingModeCmd

Function Name	**void I2C_10BitAddressingModeCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables the I2C 10-bit addressing mode for the master.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2C 10-bit addressing mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function should be called before generating start condition.

13.2.10 Communications handling functions

13.2.10.1 I2C_AutoEndCmd

Function Name	**void I2C_AutoEndCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables the I2C automatic end mode (stop condition is automatically sent when nbytes data are transferred).
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2C automatic end mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function has effect if Reload mode is disabled.

13.2.10.2 I2C_ReloadCmd

Function Name	**void I2C_ReloadCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables the I2C nbytes reload mode.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the nbytes reload mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.10.3 I2C_NumberOfBytesConfig

Function Name	**void I2C_NumberOfBytesConfig (*I2C_TypeDef** * I2Cx, uint8_t Number_Bytes)**
Function Description	Configures the number of bytes to be transmitted/received.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **Number_Bytes :** specifies the number of bytes to be

programmed.

Return values	• None.
Notes	• None.

13.2.10.4 I2C_MasterRequestConfig

Function Name	**void I2C_MasterRequestConfig (*I2C_TypeDef** * I2Cx, uint16_t I2C_Direction)**
Function Description	Configures the type of transfer request for the master.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **I2C_Direction :** specifies the transfer request direction to be programmed. This parameter can be one of the following values: – ***I2C_Direction_Transmitter :*** Master request a write transfer – ***I2C_Direction_Receiver :*** Master request a read transfer
Return values	• None.
Notes	• None.

13.2.10.5 I2C_GenerateSTART

Function Name	**void I2C_GenerateSTART (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Generates I2Cx communication START condition.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2C START condition generation. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.10.6 I2C_GenerateSTOP

Function Name	**void I2C_GenerateSTOP (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Generates I2Cx communication STOP condition.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2C STOP condition generation. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.10.7 I2C_10BitAddressHeaderCmd

Function Name	**void I2C_10BitAddressHeaderCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables the I2C 10-bit header only mode with read direction.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2C 10-bit header only mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This mode can be used only when switching from master transmitter mode to master receiver mode.

13.2.10.8 I2C_AcknowledgeConfig

Function Name	**void I2C_AcknowledgeConfig (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Generates I2C communication Acknowledge.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the Acknowledge. This parameter can be: ENABLE or DISABLE.

Return values	• None.
Notes	• None.

13.2.10.9 I2C_GetAddressMatched

Function Name	**uint8_t I2C_GetAddressMatched (*I2C_TypeDef** * I2Cx)**
Function Description	Returns the I2C slave matched address .
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
Return values	• **The value of the slave matched address .**
Notes	• None.

13.2.10.10 I2C_GetTransferDirection

Function Name	**uint16_t I2C_GetTransferDirection (*I2C_TypeDef** * I2Cx)**
Function Description	Returns the I2C slave received request.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
Return values	• **The value of the received request.**
Notes	• None.

13.2.10.11 I2C_TransferHandling

Function Name	**void I2C_TransferHandling (*I2C_TypeDef** * I2Cx, uint16_t Address, uint8_t Number_Bytes, uint32_t ReloadEndMode, uint32_t StartStopMode)**
Function Description	Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **Address :** specifies the slave address to be programmed.

- **Number_Bytes :** specifies the number of bytes to be programmed. This parameter must be a value between 0 and 255.
- **ReloadEndMode :** new state of the I2C START condition generation. This parameter can be one of the following values:
  - ***I2C_Reload_Mode :*** Enable Reload mode .
  - ***I2C_AutoEnd_Mode :*** Enable Automatic end mode.
  - ***I2C_SoftEnd_Mode :*** Enable Software end mode.
- **StartStopMode :** new state of the I2C START condition generation. This parameter can be one of the following values:
  - ***I2C_No_StartStop :*** Don't Generate stop and start condition.
  - ***I2C_Generate_Stop :*** Generate stop condition (Number_Bytes should be set to 0).
  - ***I2C_Generate_Start_Read :*** Generate Restart for read request.
  - ***I2C_Generate_Start_Write :*** Generate Restart for write request.

Return values
- None.

Notes
- None.

13.2.11 SMBUS management functions

13.2.11.1 I2C_SMBusAlertCmd

Function Name **void I2C_SMBusAlertCmd (*I2C_TypeDef* * I2Cx, *FunctionalState* NewState)**

Function Description Enables or disables I2C SMBus alert.

Parameters
- **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
- **NewState :** new state of the I2Cx SMBus alert. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

13.2.11.2 I2C_ClockTimeoutCmd

Function Name	**void I2C_ClockTimeoutCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables I2C Clock Timeout (SCL Timeout detection).
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx clock Timeout. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.11.3 I2C_ExtendedClockTimeoutCmd

Function Name	**void I2C_ExtendedClockTimeoutCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables I2C Extended Clock Timeout (SCL cumulative Timeout detection).
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx Extended clock Timeout. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.11.4 I2C_IdleClockTimeoutCmd

Function Name	**void I2C_IdleClockTimeoutCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables I2C Idle Clock Timeout (Bus idle SCL and SDA high detection).
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx Idle clock Timeout. This parameter can be: ENABLE or DISABLE.

Return values	• None.
Notes	• None.

13.2.11.5 I2C_TimeoutAConfig

Function Name	**void I2C_TimeoutAConfig (*I2C_TypeDef** * I2Cx, uint16_t Timeout)**
Function Description	Configures the I2C Bus Timeout A (SCL Timeout when TIDLE = 0 or Bus idle SCL and SDA high when TIDLE = 1).
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **Timeout :** specifies the TimeoutA to be programmed.
Return values	• None.
Notes	• None.

13.2.11.6 I2C_TimeoutBConfig

Function Name	**void I2C_TimeoutBConfig (*I2C_TypeDef** * I2Cx, uint16_t Timeout)**
Function Description	Configures the I2C Bus Timeout B (SCL cumulative Timeout).
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **Timeout :** specifies the TimeoutB to be programmed.
Return values	• None.
Notes	• None.

13.2.11.7 I2C_CalculatePEC

Function Name	**void I2C_CalculatePEC (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables I2C PEC calculation.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx PEC calculation. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.11.8 I2C_PECRequestCmd

Function Name	**void I2C_PECRequestCmd (*I2C_TypeDef** * I2Cx, *FunctionalState* NewState)**
Function Description	Enables or disables I2C PEC transmission/reception request.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **NewState :** new state of the I2Cx PEC request. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.11.9 I2C_GetPEC

Function Name	**uint8_t I2C_GetPEC (*I2C_TypeDef** * I2Cx)**
Function Description	Returns the I2C PEC.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
Return values	• **The value of the PEC .**
Notes	• None.

13.2.12 I2C registers management functions

13.2.12.1 I2C_ReadRegister

Function Name	**uint32_t I2C_ReadRegister (*I2C_TypeDef** * **I2Cx, uint8_t I2C_Register)**
Function Description	Reads the specified I2C register and returns its value.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **I2C_Register :** specifies the register to read. This parameter can be one of the following values: – ***I2C_Register_CR1 :*** CR1 register. – ***I2C_Register_CR2 :*** CR2 register. – ***I2C_Register_OAR1 :*** OAR1 register. – ***I2C_Register_OAR2 :*** OAR2 register. – ***I2C_Register_TIMINGR :*** TIMING register. – ***I2C_Register_TIMEOUTR :*** TIMEOUTR register. – ***I2C_Register_ISR :*** ISR register. – ***I2C_Register_ICR :*** ICR register. – ***I2C_Register_PECR :*** PECR register. – ***I2C_Register_RXDR :*** RXDR register. – ***I2C_Register_TXDR :*** TXDR register.
Return values	• **The value of the read register.**
Notes	• None.

13.2.13 Data transfers management functions

13.2.13.1 I2C_SendData

Function Name	**void I2C_SendData (*I2C_TypeDef** * **I2Cx, uint8_t Data)**
Function Description	Sends a data byte through the I2Cx peripheral.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **Data :** Byte to be transmitted..
Return values	• None.
Notes	• None.

13.2.13.2 I2C_ReceiveData

Function Name	**uint8_t I2C_ReceiveData (*I2C_TypeDef** * I2Cx)**
Function Description	Returns the most recent received data by the I2Cx peripheral.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
Return values	• **The value of the received data.**
Notes	• None.

13.2.14 DMA transfers management functions

13.2.14.1 I2C_DMACmd

Function Name	**void I2C_DMACmd (*I2C_TypeDef** * I2Cx, uint32_t I2C_DMAReq, *FunctionalState* NewState)**
Function Description	Enables or disables the I2C DMA interface.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **I2C_DMAReq :** specifies the I2C DMA transfer request to be enabled or disabled. This parameter can be any combination of the following values: – ***I2C_DMAReq_Tx :*** Tx DMA transfer request – ***I2C_DMAReq_Rx :*** Rx DMA transfer request • **NewState :** new state of the selected I2C DMA transfer request. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

13.2.15 Interrupts and flags management functions

13.2.15.1 I2C_GetFlagStatus

Function Name	**FlagStatus I2C_GetFlagStatus (*I2C_TypeDef** * I2Cx, uint32_t I2C_FLAG)**
Function Description	Checks whether the specified I2C flag is set or not.

Parameters
- **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
- **I2C_FLAG :** specifies the flag to check. This parameter can be one of the following values:
  - ***I2C_FLAG_TXE :*** Transmit data register empty
  - ***I2C_FLAG_TXIS :*** Transmit interrupt status
  - ***I2C_FLAG_RXNE :*** Receive data register not empty
  - ***I2C_FLAG_ADDR :*** Address matched (slave mode)
  - ***I2C_FLAG_NACKF :*** NACK received flag
  - ***I2C_FLAG_STOPF :*** STOP detection flag
  - ***I2C_FLAG_TC :*** Transfer complete (master mode)
  - ***I2C_FLAG_TCR :*** Transfer complete reload
  - ***I2C_FLAG_BERR :*** Bus error
  - ***I2C_FLAG_ARLO :*** Arbitration lost
  - ***I2C_FLAG_OVR :*** Overrun/Underrun
  - ***I2C_FLAG_PECERR :*** PEC error in reception
  - ***I2C_FLAG_TIMEOUT :*** Timeout or Tlow detection flag
  - ***I2C_FLAG_ALERT :*** SMBus Alert
  - ***I2C_FLAG_BUSY :*** Bus busy

Return values
- **The new state of I2C_FLAG (SET or RESET).**

Notes
- None.

13.2.15.2 I2C_ClearFlag

Function Name **void I2C_ClearFlag (*I2C_TypeDef* * I2Cx, uint32_t I2C_FLAG)**

Function Description Clears the I2Cx's pending flags.

Parameters
- **I2Cx :** where x can be 1 or 2 to select the I2C peripheral.
- **I2C_FLAG :** specifies the flag to clear. This parameter can be any combination of the following values:
  - ***I2C_FLAG_ADDR :*** Address matched (slave mode)
  - ***I2C_FLAG_NACKF :*** NACK received flag
  - ***I2C_FLAG_STOPF :*** STOP detection flag
  - ***I2C_FLAG_BERR :*** Bus error
  - ***I2C_FLAG_ARLO :*** Arbitration lost
  - ***I2C_FLAG_OVR :*** Overrun/Underrun
  - ***I2C_FLAG_PECERR :*** PEC error in reception
  - ***I2C_FLAG_TIMEOUT :*** Timeout or Tlow detection flag
  - ***I2C_FLAG_ALERT :*** SMBus Alert

Return values
- **The new state of I2C_FLAG (SET or RESET).**

Notes
- None.

13.2.15.3 I2C_GetITStatus

Function Name	**ITStatus I2C_GetITStatus (*I2C_TypeDef** * I2Cx, uint32_t I2C_IT)**
Function Description	Checks whether the specified I2C interrupt has occurred or not.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **I2C_IT :** specifies the interrupt source to check. This parameter can be one of the following values: – ***I2C_IT_TXIS :*** Transmit interrupt status – ***I2C_IT_RXNE :*** Receive data register not empty – ***I2C_IT_ADDR :*** Address matched (slave mode) – ***I2C_IT_NACKF :*** NACK received flag – ***I2C_IT_STOPF :*** STOP detection flag – ***I2C_IT_TC :*** Transfer complete (master mode) – ***I2C_IT_TCR :*** Transfer complete reload – ***I2C_IT_BERR :*** Bus error – ***I2C_IT_ARLO :*** Arbitration lost – ***I2C_IT_OVR :*** Overrun/Underrun – ***I2C_IT_PECERR :*** PEC error in reception – ***I2C_IT_TIMEOUT :*** Timeout or Tlow detection flag – ***I2C_IT_ALERT :*** SMBus Alert
Return values	• **The new state of I2C_IT (SET or RESET).**
Notes	• None.

13.2.15.4 I2C_ClearITPendingBit

Function Name	**void I2C_ClearITPendingBit (*I2C_TypeDef** * I2Cx, uint32_t I2C_IT)**
Function Description	Clears the I2Cx's interrupt pending bits.
Parameters	• **I2Cx :** where x can be 1 or 2 to select the I2C peripheral. • **I2C_IT :** specifies the interrupt pending bit to clear. This parameter can be any combination of the following values: – ***I2C_IT_ADDR :*** Address matched (slave mode) – ***I2C_IT_NACKF :*** NACK received flag – ***I2C_IT_STOPF :*** STOP detection flag – ***I2C_IT_BERR :*** Bus error – ***I2C_IT_ARLO :*** Arbitration lost – ***I2C_IT_OVR :*** Overrun/Underrun – ***I2C_IT_PECERR :*** PEC error in reception – ***I2C_IT_TIMEOUT :*** Timeout or Tlow detection flag

– ***I2C_IT_ALERT :*** SMBus Alert

Return values
- **The new state of I2C_IT (SET or RESET).**

Notes
- None.

13.3 I2C Firmware driver defines

13.3.1 I2C

I2C

I2C_acknowledged_address

- #define: ***I2C_AcknowledgedAddress_7bit ((uint32_t)0x00000000)***

- #define: ***I2C_AcknowledgedAddress_10bit I2C_OAR1_OA1MODE***

I2C_acknowledgement

- #define: ***I2C_Ack_Enable ((uint32_t)0x00000000)***

- #define: ***I2C_Ack_Disable I2C_CR2_NACK***

I2C_Analog_Filter

- #define: ***I2C_AnalogFilter_Enable ((uint32_t)0x00000000)***

- #define: ***I2C_AnalogFilter_Disable I2C_CR1_ANFOFF***

I2C_DMA_transfer_requests

- #define: ***I2C_DMAReq_Tx I2C_CR1_TXDMAEN***

- #define: ***I2C_DMAReq_Rx I2C_CR1_RXDMAEN***

I2C_flags_definition

- #define: ***I2C_FLAG_TXE I2C_ISR_TXE***
- #define: ***I2C_FLAG_TXIS I2C_ISR_TXIS***
- #define: ***I2C_FLAG_RXNE I2C_ISR_RXNE***
- #define: ***I2C_FLAG_ADDR I2C_ISR_ADDR***
- #define: ***I2C_FLAG_NACKF I2C_ISR_NACKF***
- #define: ***I2C_FLAG_STOPF I2C_ISR_STOPF***
- #define: ***I2C_FLAG_TC I2C_ISR_TC***
- #define: ***I2C_FLAG_TCR I2C_ISR_TCR***
- #define: ***I2C_FLAG_BERR I2C_ISR_BERR***
- #define: ***I2C_FLAG_ARLO I2C_ISR_ARLO***
- #define: ***I2C_FLAG_OVR I2C_ISR_OVR***

- #define: ***I2C_FLAG_PECERR I2C_ISR_PECERR***

- #define: ***I2C_FLAG_TIMEOUT I2C_ISR_TIMEOUT***

- #define: ***I2C_FLAG_ALERT I2C_ISR_ALERT***

- #define: ***I2C_FLAG_BUSY I2C_ISR_BUSY***

I2C_interrupts_definition

- #define: ***I2C_IT_ERRI I2C_CR1_ERRIE***

- #define: ***I2C_IT_TCI I2C_CR1_TCIE***

- #define: ***I2C_IT_STOPI I2C_CR1_STOPIE***

- #define: ***I2C_IT_NACKI I2C_CR1_NACKIE***

- #define: ***I2C_IT_ADDRI I2C_CR1_ADDRIE***

- #define: ***I2C_IT_RXI I2C_CR1_RXIE***

- #define: ***I2C_IT_TXI I2C_CR1_TXIE***

- #define: ***I2C_IT_TXIS I2C_ISR_TXIS***

- #define: ***I2C_IT_RXNE I2C_ISR_RXNE***
- #define: ***I2C_IT_ADDR I2C_ISR_ADDR***
- #define: ***I2C_IT_NACKF I2C_ISR_NACKF***
- #define: ***I2C_IT_STOPF I2C_ISR_STOPF***
- #define: ***I2C_IT_TC I2C_ISR_TC***
- #define: ***I2C_IT_TCR I2C_ISR_TCR***
- #define: ***I2C_IT_BERR I2C_ISR_BERR***
- #define: ***I2C_IT_ARLO I2C_ISR_ARLO***
- #define: ***I2C_IT_OVR I2C_ISR_OVR***
- #define: ***I2C_IT_PECERR I2C_ISR_PECERR***
- #define: ***I2C_IT_TIMEOUT I2C_ISR_TIMEOUT***
- #define: ***I2C_IT_ALERT I2C_ISR_ALERT***

I2C_mode

- #define: ***I2C_Mode_I2C ((uint32_t)0x00000000)***

- #define: ***I2C_Mode_SMBusDevice I2C_CR1_SMBDEN***

- #define: ***I2C_Mode_SMBusHost I2C_CR1_SMBHEN***

I2C_own_address2_mask

- #define: ***I2C_OA2_NoMask ((uint8_t)0x00)***

- #define: ***I2C_OA2_Mask01 ((uint8_t)0x01)***

- #define: ***I2C_OA2_Mask02 ((uint8_t)0x02)***

- #define: ***I2C_OA2_Mask03 ((uint8_t)0x03)***

- #define: ***I2C_OA2_Mask04 ((uint8_t)0x04)***

- #define: ***I2C_OA2_Mask05 ((uint8_t)0x05)***

- #define: ***I2C_OA2_Mask06 ((uint8_t)0x06)***

- #define: ***I2C_OA2_Mask07 ((uint8_t)0x07)***

I2C_registers

- #define: ***I2C_Register_CR1 ((uint8_t)0x00)***
- #define: ***I2C_Register_CR2 ((uint8_t)0x04)***
- #define: ***I2C_Register_OAR1 ((uint8_t)0x08)***
- #define: ***I2C_Register_OAR2 ((uint8_t)0x0C)***
- #define: ***I2C_Register_TIMINGR ((uint8_t)0x10)***
- #define: ***I2C_Register_TIMEOUTR ((uint8_t)0x14)***
- #define: ***I2C_Register_ISR ((uint8_t)0x18)***
- #define: ***I2C_Register_ICR ((uint8_t)0x1C)***
- #define: ***I2C_Register_PECR ((uint8_t)0x20)***
- #define: ***I2C_Register_RXDR ((uint8_t)0x24)***
- #define: ***I2C_Register_TXDR ((uint8_t)0x28)***

I2C_ReloadEndMode_definition

- #define: ***I2C_Reload_Mode I2C_CR2_RELOAD***

- #define: ***I2C_AutoEnd_Mode I2C_CR2_AUTOEND***

- #define: ***I2C_SoftEnd_Mode ((uint32_t)0x00000000)***

I2C_StartStopMode_definition

- #define: ***I2C_No_StartStop ((uint32_t)0x00000000)***

- #define: ***I2C_Generate_Stop I2C_CR2_STOP***

- #define: ***I2C_Generate_Start_Read (uint32_t)(I2C_CR2_START | I2C_CR2_RD_WRN)***

- #define: ***I2C_Generate_Start_Write I2C_CR2_START***

I2C_transfer_direction

- #define: ***I2C_Direction_Transmitter ((uint16_t)0x0000)***

- #define: ***I2C_Direction_Receiver ((uint16_t)0x0400)***

14 Independent watchdog (IWDG)

14.1 IWDG Firmware driver registers structures

14.1.1 IWDG_TypeDef

IWDG_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t KR***
- ***__IO uint32_t PR***
- ***__IO uint32_t RLR***
- ***__IO uint32_t SR***
- ***__IO uint32_t WINR***

Field Documentation

- ***__IO uint32_t IWDG_TypeDef::KR***
  - IWDG Key register, Address offset: 0x00
- ***__IO uint32_t IWDG_TypeDef::PR***
  - IWDG Prescaler register, Address offset: 0x04
- ***__IO uint32_t IWDG_TypeDef::RLR***
  - IWDG Reload register, Address offset: 0x08
- ***__IO uint32_t IWDG_TypeDef::SR***
  - IWDG Status register, Address offset: 0x0C
- ***__IO uint32_t IWDG_TypeDef::WINR***
  - IWDG Window register, Address offset: 0x10

14.2 IWDG Firmware driver API description

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

14.2.1 IWDG features

The IWDG can be started by either software or hardware (configurable through option byte).

The IWDG is clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails. Once the IWDG is started, the LSI is forced ON and cannot be disabled (LSI cannot be disabled too), and the counter starts counting down from the reset value of 0xFFF. When it reaches the end of count value (0x000) a system reset is generated. The IWDG counter should be reloaded at regular intervals to prevent an MCU reset.

The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY).

IWDGRST flag in RCC_CSR register can be used to inform when a IWDG reset occurs.

Min-max timeout value @41KHz (LSI): ~0.1ms / ~25.5s The IWDG timeout may vary due to LSI frequency dispersion. STM32F30x devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy. For more information, please refer to the STM32F30x Reference manual.

14.2.2 How to use this driver

This driver allows to use IWDG peripheral with either window option enabled or disabled. To do so follow one of the two procedures below.

1. Window option is enabled:
   - Start the IWDG using IWDG_Enable() function, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
   - Enable write access to IWDG_PR and IWDG_RLR registers using IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable) function.
   - Configure the IWDG prescaler using IWDG_SetPrescaler() function.
   - Configure the IWDG counter value using IWDG_SetReload() function. This value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
   - Wait for the IWDG registers to be updated using IWDG_GetFlagStatus() function.
   - Configure the IWDG refresh window using IWDG_SetWindowValue() function.
2. Window option is disabled:
   - Enable write access to IWDG_PR and IWDG_RLR registers using IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable) function.
   - Configure the IWDG prescaler using IWDG_SetPrescaler() function.
   - Configure the IWDG counter value using IWDG_SetReload() function. This value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
   - Wait for the IWDG registers to be updated using IWDG_GetFlagStatus() function.
   - reload the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using IWDG_ReloadCounter() function.
   - Start the IWDG using IWDG_Enable() function, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).

14.2.3 Prescaler and Counter configuration functions

- ***IWDG_WriteAccessCmd()***
- ***IWDG_SetPrescaler()***
- ***IWDG_SetReload()***
- ***IWDG_ReloadCounter()***
- ***IWDG_SetWindowValue()***

14.2.4 IWDG activation function

- ***IWDG_Enable()***

14.2.5 Flag management function

- ***IWDG_GetFlagStatus()***

14.2.6 Prescaler and counter configuration functions

14.2.6.1 IWDG_WriteAccessCmd

Function Name	**void IWDG_WriteAccessCmd (uint16_t IWDG_WriteAccess)**
Function Description	Enables or disables write access to IWDG_PR and IWDG_RLR registers.
Parameters	• **IWDG_WriteAccess :** new state of write access to IWDG_PR and IWDG_RLR registers. This parameter can be one of the following values: – ***IWDG_WriteAccess_Enable :*** Enable write access to IWDG_PR and IWDG_RLR registers – ***IWDG_WriteAccess_Disable :*** Disable write access to IWDG_PR and IWDG_RLR registers
Return values	• None.
Notes	• None.

14.2.6.2 IWDG_SetPrescaler

Function Name	**void IWDG_SetPrescaler (uint8_t IWDG_Prescaler)**
Function Description	Sets IWDG Prescaler value.
Parameters	• **IWDG_Prescaler :** specifies the IWDG Prescaler value. This parameter can be one of the following values: – ***IWDG_Prescaler_4 :*** IWDG prescaler set to 4 – ***IWDG_Prescaler_8 :*** IWDG prescaler set to 8 – ***IWDG_Prescaler_16 :*** IWDG prescaler set to 16 – ***IWDG_Prescaler_32 :*** IWDG prescaler set to 32 – ***IWDG_Prescaler_64 :*** IWDG prescaler set to 64 – ***IWDG_Prescaler_128 :*** IWDG prescaler set to 128 – ***IWDG_Prescaler_256 :*** IWDG prescaler set to 256
Return values	• None.
Notes	• None.

14.2.6.3 IWDG_SetReload

Function Name	**void IWDG_SetReload (uint16_t Reload)**
Function Description	Sets IWDG Reload value.
Parameters	• **Reload :** specifies the IWDG Reload value. This parameter must be a number between 0 and 0x0FFF.
Return values	• None.
Notes	• None.

14.2.6.4 IWDG_ReloadCounter

Function Name	**void IWDG_ReloadCounter (void)**
Function Description	Reloads IWDG counter with value defined in the reload register (write access to IWDG_PR and IWDG_RLR registers disabled).
Parameters	• None.
Return values	• None.
Notes	• None.

14.2.6.5 IWDG_SetWindowValue

Function Name	**void IWDG_SetWindowValue (uint16_t WindowValue)**
Function Description	Sets the IWDG window value.
Parameters	• **WindowValue :** specifies the window value to be compared to the downcounter.
Return values	• None.
Notes	• None.

14.2.7 IWDG activation function

14.2.7.1 IWDG_Enable

Function Name	**void IWDG_Enable (void)**
Function Description	Enables IWDG (write access to IWDG_PR and IWDG_RLR registers disabled).
Parameters	• None.
Return values	• None.
Notes	• None.

14.2.8 Flag management function

14.2.8.1 IWDG_GetFlagStatus

Function Name	**FlagStatus IWDG_GetFlagStatus (uint16_t IWDG_FLAG)**
Function Description	Checks whether the specified IWDG flag is set or not.
Parameters	• **IWDG_FLAG :** specifies the flag to check. This parameter can be one of the following values: – ***IWDG_FLAG_PVU :*** Prescaler Value Update on going – ***IWDG_FLAG_RVU :*** Reload Value Update on going – ***IWDG_FLAG_WVU :*** Counter Window Value Update on going
Return values	• **The new state of IWDG_FLAG (SET or RESET).**
Notes	• None.

14.3 IWDG Firmware driver defines

14.3.1 IWDG

IWDG

IWDG_Flag

- #define: ***IWDG_FLAG_PVU ((uint16_t)0x0001)***

- #define: ***IWDG_FLAG_RVU ((uint16_t)0x0002)***

- #define: ***IWDG_FLAG_WVU ((uint16_t)0x0002)***

IWDG_prescaler

- #define: ***IWDG_Prescaler_4 ((uint8_t)0x00)***

- #define: ***IWDG_Prescaler_8 ((uint8_t)0x01)***

- #define: ***IWDG_Prescaler_16 ((uint8_t)0x02)***

- #define: ***IWDG_Prescaler_32 ((uint8_t)0x03)***

- #define: ***IWDG_Prescaler_64 ((uint8_t)0x04)***

- #define: ***IWDG_Prescaler_128 ((uint8_t)0x05)***

- #define: ***IWDG_Prescaler_256 ((uint8_t)0x06)***

IWDG_WriteAccess

- #define: ***IWDG_WriteAccess_Enable ((uint16_t)0x5555)***

- #define: ***IWDG_WriteAccess_Disable ((uint16_t)0x0000)***

15 Miscellaneous add-on to CMSIS functions(misc)

15.1 MISC Firmware driver registers structures

15.1.1 NVIC_InitTypeDef

NVIC_InitTypeDef is defined in the stm32f30x_misc.h

Data Fields

- ***uint8_t NVIC_IRQChannel***
- ***uint8_t NVIC_IRQChannelPreemptionPriority***
- ***uint8_t NVIC_IRQChannelSubPriority***
- ***FunctionalState NVIC_IRQChannelCmd***

Field Documentation

- ***uint8_t NVIC_InitTypeDef::NVIC_IRQChannel***
  - Specifies the IRQ channel to be enabled or disabled. This parameter can be a value of ***IRQn_Type***
- ***uint8_t NVIC_InitTypeDef::NVIC_IRQChannelPreemptionPriority***
  - Specifies the pre-emption priority for the IRQ channel specified in NVIC_IRQChannel. This parameter can be a value between 0 and 15. A lower priority value indicates a higher priority
- ***uint8_t NVIC_InitTypeDef::NVIC_IRQChannelSubPriority***
  - Specifies the subpriority level for the IRQ channel specified in NVIC_IRQChannel. This parameter can be a value between 0 and 15. A lower priority value indicates a higher priority
- ***FunctionalState NVIC_InitTypeDef::NVIC_IRQChannelCmd***
  - Specifies whether the IRQ channel defined in NVIC_IRQChannel will be enabled or disabled. This parameter can be set either to ENABLE or DISABLE

15.2 MISC Firmware driver API description

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

15.2.1 How to configure Interrupts using driver

This section provide functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M4 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using NVIC_PriorityGroupConfig() function according to the following table. The table below gives the allowed values of the pre-emption priority and subpriority according to the Priority Grouping configuration performed by NVIC_PriorityGroupConfig function.
2. Enable and Configure the priority of the selected IRQ Channels.

When the NVIC_PriorityGroup_0 is selected, it will no any nested interrupt, the IRQ priority will be managed only by subpriority. The sub-priority is only used to sort pending exception priorities, and does not affect active exceptions.

Lower priority values gives higher priority.

Priority Order:

1. Lowest Preemption priority.
2. Lowest Subpriority.
3. Lowest hardware priority (IRQn position).

15.2.2 MISC functions

15.2.2.1 NVIC_PriorityGroupConfig

Function Name	**void NVIC_PriorityGroupConfig (uint32_t NVIC_PriorityGroup)**
Function Description	Configures the priority grouping: pre-emption priority and subpriority.
Parameters	• **NVIC_PriorityGroup :** specifies the priority grouping bits length. This parameter can be one of the following values: – ***NVIC_PriorityGroup_0 :*** 0 bits for pre-emption priority. 4 bits for subpriority. – ***NVIC_PriorityGroup_1 :*** 1 bits for pre-emption priority. 3 bits for subpriority. – ***NVIC_PriorityGroup_2 :*** 2 bits for pre-emption priority. 2 bits for subpriority. – ***NVIC_PriorityGroup_3 :*** 3 bits for pre-emption priority. 1 bits for subpriority. – ***NVIC_PriorityGroup_4 :*** 4 bits for pre-emption priority. 0 bits for subpriority.
Return values	• None.
Notes	• When NVIC_PriorityGroup_0 is selected, it will no be any nested interrupt. This interrupts priority is managed only with subpriority.

15.2.2.2 NVIC_Init

Function Name	**void NVIC_Init (*NVIC_InitTypeDef** * NVIC_InitStruct)**
Function Description	Initializes the NVIC peripheral according to the specified parameters in the NVIC_InitStruct.
Parameters	• **NVIC_InitStruct :** pointer to a NVIC_InitTypeDef structure that contains the configuration information for the specified NVIC peripheral.
Return values	• None.
Notes	• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

15.2.2.3 NVIC_SetVectorTable

Function Name	**void NVIC_SetVectorTable (uint32_t NVIC_VectTab, uint32_t Offset)**
Function Description	Sets the vector table location and Offset.
Parameters	• **NVIC_VectTab :** specifies if the vector table is in RAM or FLASH memory. This parameter can be one of the following values: – ***NVIC_VectTab_RAM :*** – ***NVIC_VectTab_FLASH :*** • **Offset :** Vector Table base offset field. This value must be a multiple of 0x200.
Return values	• None.
Notes	• None.

15.2.2.4 NVIC_SystemLPConfig

Function Name	**void NVIC_SystemLPConfig (uint8_t LowPowerMode, *FunctionalState* NewState)**
Function Description	Selects the condition for the system to enter low power mode.

Parameters	• **LowPowerMode :** Specifies the new mode for the system to enter low power mode. This parameter can be one of the following values: – ***NVIC_LP_SEVONPEND :*** – ***NVIC_LP_SLEEPDEEP :*** – ***NVIC_LP_SLEEPONEXIT :*** • **NewState :** new state of LP condition. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

15.2.2.5 SysTick_CLKSourceConfig

Function Name	**void SysTick_CLKSourceConfig (uint32_t SysTick_CLKSource)**
Function Description	Configures the SysTick clock source.
Parameters	• **SysTick_CLKSource :** specifies the SysTick clock source. This parameter can be one of the following values: – ***SysTick_CLKSource_HCLK_Div8 :*** AHB clock divided by 8 selected as SysTick clock source. – ***SysTick_CLKSource_HCLK :*** AHB clock selected as SysTick clock source.
Return values	• None.
Notes	• None.

15.3 MISC Firmware driver defines

15.3.1 MISC

MISC

MISC_Exported_Constants

- #define: ***SysTick_CLKSource_HCLK_Div8 ((uint32_t)0xFFFFFFFB)***

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

MISC_Preemption_Priority_Group

- #define: ***NVIC_PriorityGroup_0 ((uint32_t)0x700)***

0 bits for pre-emption priority 4 bits for subpriority

- #define: ***NVIC_PriorityGroup_1 ((uint32_t)0x600)***

1 bits for pre-emption priority 3 bits for subpriority

- #define: ***NVIC_PriorityGroup_2 ((uint32_t)0x500)***

2 bits for pre-emption priority 2 bits for subpriority

- #define: ***NVIC_PriorityGroup_3 ((uint32_t)0x400)***

3 bits for pre-emption priority 1 bits for subpriority

- #define: ***NVIC_PriorityGroup_4 ((uint32_t)0x300)***

4 bits for pre-emption priority 0 bits for subpriority

MISC_System_Low_Power

- #define: ***NVIC_LP_SEVONPEND ((uint8_t)0x10)***

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

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

MISC_Vector_Table_Base

- #define: ***NVIC_VectTab_RAM ((uint32_t)0x20000000)***

- #define: ***NVIC_VectTab_FLASH ((uint32_t)0x08000000)***

16 Operational amplifier (OPAMP)

16.1 OPAMP Firmware driver registers structures

16.1.1 OPAMP_TypeDef

OPAMP_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CSR***

Field Documentation

- ***__IO uint32_t OPAMP_TypeDef::CSR***
  - OPAMP control and status register, Address offset: 0x00

16.1.2 OPAMP_InitTypeDef

OPAMP_InitTypeDef is defined in the stm32f30x_opamp.h

Data Fields

- ***uint32_t OPAMP_InvertingInput***
- ***uint32_t OPAMP_NonInvertingInput***

Field Documentation

- ***uint32_t OPAMP_InitTypeDef::OPAMP_InvertingInput***
  - Selects the inverting input of the operational amplifier. This parameter can be a value of ***OPAMP_InvertingInput***
- ***uint32_t OPAMP_InitTypeDef::OPAMP_NonInvertingInput***
  - Selects the non inverting input of the operational amplifier. This parameter can be a value of ***OPAMP_NonInvertingInput***

16.2 OPAMP Firmware driver API description

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

16.2.1 OPAMP Peripheral Features

The device integrates 4 operational amplifiers OPAMP1, OPAMP2, OPAMP3 and OPAMP4:

- The OPAMPs non inverting input can be selected among the list shown by table below.
- The OPAMPs inverting input can be selected among the list shown by table below.
- The OPAMPs outputs can be internally connected to the inverting input (follower mode)
- The OPAMPs outputs can be internally connected to resistor feedback output (Programmable Gain Amplifier mode)
- The OPAMPs outputs can be internally connected to ADC
- The OPAMPs can be calibrated to compensate the offset compensation
- Timer-controlled Mux for automatic switch of inverting and non-inverting input

Table 14: OPAMPs inverting/non-inverting inputs

		OPAMP1	OPAMP2	OPAMP3	OPAMP4
Inverting input	PGA	OK	OK	OK	OK
	VOUT	OK	OK	OK	OK
	IO1	PC5	PC5	PB10	PB10
	IO2	PA3	PA5	PB2	PD8
Non-inverting inputs	IO1	PA7	PD14	PB13	PD11
	IO2	PA5	PB14	PA5	PB11
	IO3	PA3	PB0	PA1	PA4
	IO4	PA1	PA7	PB0	PB13

16.2.2 How to use this driver

This driver provides functions to configure and program the OPAMP of all STM32F30x devices. To use the OPAMP, perform the following steps:

1. Enable the SYSCFG APB clock to get write access to OPAMP register using RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
2. Configure the OPAMP input in analog mode using GPIO_Init()
3. Configure the OPAMP using OPAMP_Init() function:
   - Select the inverting input
   - Select the non-inverting inverting input
4. Enable the OPAMP using OPAMP_Cmd() function

16.2.3 Initialization and Configuration functions

- ***OPAMP_DeInit()***
- ***OPAMP_Init()***
- ***OPAMP_StructInit()***
- ***OPAMP_PGAConfig()***
- ***OPAMP_VrefConfig()***
- ***OPAMP_VrefConnectNonInvertingInput()***
- ***OPAMP_VrefConnectADCCmd()***
- ***OPAMP_TimerControlledMuxConfig()***
- ***OPAMP_TimerControlledMuxCmd()***

- *OPAMP_Cmd()*
- *OPAMP_GetOutputLevel()*
- *OPAMP_OffsetTrimModeSelect()*
- *OPAMP_OffsetTrimConfig()*
- *OPAMP_StartCalibration()*

16.2.4 Initialization and Configuration

16.2.4.1 OPAMP_DeInit

Function Name	**void OPAMP_DeInit (uint32_t OPAMP_Selection)**
Function Description	Deinitializes OPAMP peripheral registers to their default reset values.
Parameters	• OPAMP_Selection.
Return values	• None.
Notes	• Deinitialization can't be performed if the OPAMP configuration is locked. To unlock the configuration, perform a system reset.

16.2.4.2 OPAMP_Init

Function Name	**void OPAMP_Init (uint32_t OPAMP_Selection, *OPAMP_InitTypeDef** * **OPAMP_InitStruct)**
Function Description	Initializes the OPAMP peripheral according to the specified parameters in OPAMP_InitStruct.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **OPAMP_InitStruct :** pointer to an OPAMP_InitTypeDef structure that contains the configuration information for the specified OPAMP peripheral. OPAMP_InvertingInput specifies the inverting input of OPAMPOPAMP_NonInvertingInput specifies the non inverting input of OPAMP – *OPAMP_InvertingInput :* – *OPAMP_NonInvertingInput :*
Return values	• None.
Notes	• If the selected OPAMP is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

16.2.4.3 OPAMP_StructInit

Function Name	**void OPAMP_StructInit (*OPAMP_InitTypeDef** * OPAMP_InitStruct)**
Function Description	Fills each OPAMP_InitStruct member with its default value.
Parameters	• **OPAMP_InitStruct :** pointer to an OPAMP_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

16.2.4.4 OPAMP_PGAConfig

Function Name	**void OPAMP_PGAConfig (uint32_t OPAMP_Selection, uint32_t OPAMP_PGAGain, uint32_t OPAMP_PGAConnect)**
Function Description	Configure the feedback resistor gain.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **NewState :** new state of the OPAMP peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If the selected OPAMP is locked, gain configuration can't be performed. To unlock the configuration, perform a system reset.

16.2.4.5 OPAMP_VrefConfig

Function Name	**void OPAMP_VrefConfig (uint32_t OPAMP_Selection, uint32_t OPAMP_Vref)**

Function Description	Configure the OPAMP's internal reference.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **OPAMP_Vref :** This parameter can be: OPAMP_Vref_3VDDA: OPMAP Vref = 3.3% VDDA OPAMP_Vref_10VDDA: OPMAP Vref = 10% VDDA OPAMP_Vref_50VDDA: OPMAP Vref = 50% VDDA OPAMP_Vref_90VDDA: OPMAP Vref = 90% VDDA
Return values	• None.
Notes	• This feature is used when calibration enabled or OPAMP's reference connected to the non inverting input. • If the selected OPAMP is locked, Vref configuration can't be performed. To unlock the configuration, perform a system reset.

16.2.4.6 OPAMP_VrefConnectNonInvertingInput

Function Name	**void OPAMP_VrefConnectNonInvertingInput (uint32_t OPAMP_Selection, *FunctionalState* NewState)**
Function Description	Connnect the internal reference to the OPAMP's non inverting input.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **NewState :** new state of the OPAMP peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If the selected OPAMP is locked, Vref configuration can't be performed. To unlock the configuration, perform a system reset.

16.2.4.7 OPAMP_VrefConnectADCCmd

Function Name	**void OPAMP_VrefConnectADCCmd (uint32_t OPAMP_Selection, *FunctionalState* NewState)**

Function Description	Enables or disables connecting the OPAMP's internal reference to ADC.
Parameters	• **NewState :** new state of the Vrefint output. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If the selected OPAMP is locked, Vref connection can't be performed. To unlock the configuration, perform a system reset.

16.2.4.8 OPAMP_TimerControlledMuxConfig

Function Name	**void OPAMP_TimerControlledMuxConfig (uint32_t OPAMP_Selection, *OPAMP_InitTypeDef** * **OPAMP_InitStruct)**
Function Description	Configure the OPAMP peripheral (secondary inputs) for timer-controlled mux mode according to the specified parameters in OPAMP_InitStruct.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **OPAMP_InitStruct :** pointer to an OPAMP_InitTypeDef structure that contains the configuration information for the specified OPAMP peripheral. OPAMP_InvertingInput specifies the inverting input of OPAMPOPAMP_NonInvertingInput specifies the non inverting input of OPAMP – ***OPAMP_InvertingInput :*** – ***OPAMP_NonInvertingInput :***
Return values	• None.
Notes	• If the selected OPAMP is locked, timer-controlled mux configuration can't be performed. To unlock the configuration, perform a system reset. • PGA and Vout can't be selected as seconadry inverting input.

16.2.4.9 OPAMP_TimerControlledMuxCmd

Function Name	**void OPAMP_TimerControlledMuxCmd (uint32_t OPAMP_Selection, *FunctionalState* NewState)**

Function Description	Enable or disable the timer-controlled mux mode.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **NewState :** new state of the OPAMP peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If the selected OPAMP is locked, enable/disable can't be performed. To unlock the configuration, perform a system reset.

16.2.4.10 OPAMP_Cmd

Function Name	**void OPAMP_Cmd (uint32_t OPAMP_Selection, *FunctionalState* NewState)**
Function Description	Enable or disable the OPAMP peripheral.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **NewState :** new state of the OPAMP peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If the selected OPAMP is locked, enable/disable can't be performed. To unlock the configuration, perform a system reset.

16.2.4.11 OPAMP_GetOutputLevel

Function Name	**uint32_t OPAMP_GetOutputLevel (uint32_t OPAMP_Selection)**
Function Description	Return the output level (high or low) during calibration of the selected OPAMP.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. – ***OPAMP :***

– ***OPAMP :***

Return values
- **Returns the selected OPAMP output level: low or high.**

Notes
- OPAMP ouput level is provided only during calibration phase.

16.2.4.12 OPAMP_OffsetTrimModeSelect

Function Name
void OPAMP_OffsetTrimModeSelect (uint32_t OPAMP_Selection, uint32_t OPAMP_Trimming)

Function Description
Select the trimming mode.

Parameters
- **OffsetTrimming :** the selected offset trimming mode. This parameter can be one of the following values:
  – ***OPAMP_Trimming_Factory :*** factory trimming values are used for offset calibration
  – ***OPAMP_Trimming_User :*** user trimming values are used for offset calibration

Return values
- None.

Notes
- When OffsetTrimming_User is selected, use OPAMP_OffsetTrimConfig() function or OPAMP_OffsetTrimLowPowerConfig() function to adjust trimming value.

16.2.4.13 OPAMP_OffsetTrimConfig

Function Name
void OPAMP_OffsetTrimConfig (uint32_t OPAMP_Selection, uint32_t OPAMP_Input, uint32_t OPAMP_TrimValue)

Function Description
Configure the trimming value of the OPAMP.

Parameters
- **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral.
- **OPAMP_Input :** the selected OPAMP input. This parameter can be one of the following values:
  – ***OPAMP_Input_Inverting :*** Inverting input is selected to configure the trimming value
  – ***OPAMP_Input_NonInverting :*** Non inverting input is selected to configure the trimming value
- **OPAMP_TrimValue :** the trimming value. This parameter

can be any value lower or equal to 0x0000001F.

Return values	• None.
Notes	• None.

16.2.4.14 OPAMP_StartCalibration

Function Name	**void OPAMP_StartCalibration (uint32_t OPAMP_Selection, *FunctionalState* NewState)**
Function Description	Start or stop the calibration of selected OPAMP peripheral.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral. • **NewState :** new state of the OPAMP peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If the selected OPAMP is locked, start/stop can't be performed. To unlock the configuration, perform a system reset.

16.2.5 OPAMP configuration locking function

16.2.5.1 OPAMP_LockConfig

Function Name	**void OPAMP_LockConfig (uint32_t OPAMP_Selection)**
Function Description	Lock the selected OPAMP configuration.
Parameters	• **OPAMP_Selection :** the selected OPAMP. This parameter can be OPAMP_Selection_OPAMPx where x can be 1 to 4 to select the OPAMP peripheral.
Return values	• None.
Notes	• Locking the configuration means that all control bits are read-only. To unlock the OPAMP configuration, perform a system reset.

16.3 OPAMP Firmware driver defines

16.3.1 OPAMP

OPAMP

OPAMP_Exported_Constants

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

Factory trimming

- #define: ***OPAMP_Trimming_User OPAMP_CSR_USERTRIM***

User trimming

OPAMP_Input

- #define: ***OPAMP_Input_Inverting ((uint32_t)0x00000018)***

Inverting input

- #define: ***OPAMP_Input_NonInverting ((uint32_t)0x00000013)***

Non inverting input

OPAMP_InvertingInput

- #define: ***OPAMP_InvertingInput_IO1 ((uint32_t)0x00000000)***

IO1 (PC5 for OPAMP1 and OPAMP2, PB10 for OPAMP3 and OPAMP4) connected to OPAMPx inverting input

- #define: ***OPAMP_InvertingInput_IO2 OPAMP_CSR_VMSEL_0***

IO2 (PA3 for OPAMP1, PA5 for OPAMP2, PB2 for OPAMP3, PD8 for OPAMP4) connected to OPAMPx inverting input

- #define: ***OPAMP_InvertingInput_PGA OPAMP_CSR_VMSEL_1***

Resistor feedback output connected to OPAMPx inverting input (PGA mode)

- #define: ***OPAMP_InvertingInput_Vout OPAMP_CSR_VMSEL***

Vout connected to OPAMPx inverting input (follower mode)

OPAMP_NonInvertingInput

- #define: ***OPAMP_NonInvertingInput_IO1 ((uint32_t)0x00000000)***

IO1 (PA7 for OPAMP1, PD14 for OPAMP2, PB13 for OPAMP3, PD11 for OPAMP4) connected to OPAMPx non inverting input

- #define: ***OPAMP_NonInvertingInput_IO2 OPAMP_CSR_VPSEL_0***

IO2 (PA5 for OPAMP1, PB14 for OPAMP2, PA5 for OPAMP3, PB11 for OPAMP4) connected to OPAMPx non inverting input

- #define: ***OPAMP_NonInvertingInput_IO3 OPAMP_CSR_VPSEL_1***

IO3 (PA3 for OPAMP1, PB0 for OPAMP2, PA1 for OPAMP3, PA4 for OPAMP4) connected to OPAMPx non inverting input

- #define: ***OPAMP_NonInvertingInput_IO4 OPAMP_CSR_VPSEL***

IO4 (PA1 for OPAMP1, PA7 for OPAMP2, PB0 for OPAMP3, PB13 for OPAMP4) connected to OPAMPx non inverting input

OPAMP_OutputLevel

- #define: ***OPAMP_OutputLevel_High OPAMP_CSR_OUTCAL***

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

OPAMP_PGAConnect_Config

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

- #define: ***OPAMP_PGAConnect_IO1 OPAMP_CSR_PGGAIN_3***

- #define: ***OPAMP_PGAConnect_IO2 ((uint32_t)0x00030000)***

OPAMP_PGAGain_Config

- #define: ***OPAMP_OPAMP_PGAGain_2 ((uint32_t)0x00000000)***

- #define: ***OPAMP_OPAMP_PGAGain_4 OPAMP_CSR_PGGAIN_0***

- #define: ***OPAMP_OPAMP_PGAGain_8 OPAMP_CSR_PGGAIN_1***

- #define: ***OPAMP_OPAMP_PGAGain_16 ((uint32_t)0x0000C000)***

OPAMP_Selection

- #define: ***OPAMP_Selection_OPAMP1 ((uint32_t)0x00000000)***

OPAMP1 Selection

- #define: ***OPAMP_Selection_OPAMP2 ((uint32_t)0x00000004)***

OPAMP2 Selection

- #define: ***OPAMP_Selection_OPAMP3 ((uint32_t)0x00000008)***

OPAMP3 Selection

- #define: ***OPAMP_Selection_OPAMP4 ((uint32_t)0x0000000C)***

OPAMP4 Selection

OPAMP_Vref

- #define: ***OPAMP_Vref_3VDDA ((uint32_t)0x00000000)***

OPMAP Vref = 3.3% VDDA

- #define: ***OPAMP_Vref_10VDDA OPAMP_CSR_CALSEL_0***

OPMAP Vref = 10% VDDA

- #define: ***OPAMP_Vref_50VDDA OPAMP_CSR_CALSEL_1***

OPMAP Vref = 50% VDDA

- #define: ***OPAMP_Vref_90VDDA OPAMP_CSR_CALSEL***

OPMAP Vref = 90% VDDA

17 Power control (PWR)

17.1 PWR Firmware driver registers structures

17.1.1 PWR_TypeDef

PWR_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t CSR***

Field Documentation

- ***__IO uint32_t PWR_TypeDef::CR***
  - PWR power control register, Address offset: 0x00
- ***__IO uint32_t PWR_TypeDef::CSR***
  - PWR power control/status register, Address offset: 0x04

17.2 PWR Firmware driver API description

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

17.2.1 Backup Domain Access function

After reset, the Backup Domain Registers (RCC BDCR Register, RTC registers and RTC backup registers) are protected against possible stray write accesses.

To enable access to Backup domain use the PWR_BackupAccessCmd(ENABLE) function.

- ***PWR_DeInit()***
- ***PWR_BackupAccessCmd()***

17.2.2 PVD configuration functions

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the PWR_CR).
- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled through the EXTI registers.
- The PVD is stopped in Standby mode.
- ***PWR_PVDLevelConfig()***
- ***PWR_PVDCmd()***

17.2.3 WakeUp pins configuration functions

- WakeUp pins are used to wakeup the system from Standby mode. These pins are forced in input pull down configuration and are active on rising edges.
- There are three WakeUp pins: WakeUp Pin 1 on PA.00, WakeUp Pin 2 on PC.13 and WakeUp Pin 3 on PE.06.
- ***PWR_WakeUpPinCmd()***

17.2.4 Low Power modes configuration functions

The devices feature three low-power modes:

- Sleep mode: Cortex-M4 core stopped, peripherals kept running.
- Stop mode: all clocks are stopped, regulator running, regulator in low power mode
- Standby mode: VCORE domain powered off

Sleep mode

- Entry:
  - The Sleep mode is entered by executing the WFE() or WFI() instructions.
- Exit:
  - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

Stop mode

In Stop mode, all clocks in the VCORE domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode.

- Entry:
  - The Stop mode is entered using the PWR_EnterSTOPMode(PWR_Regulator_LowPower,) function with regulator in LowPower or with Regulator ON.
- Exit:
  - Any EXTI Line (Internal or External) configured in Interrupt/Event mode or any internal IPs (I2C or UASRT) wakeup event.

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M4 deepsleep mode, with the voltage regulator disabled. The VCORE domain is consequently powered off. The PLL, the HSI, and the HSE oscillator are also switched off. SRAM and register contents are lost except for the Backup domain (RTC registers, RTC backup registers and Standby circuitry).

The voltage regulator is OFF.

- Entry:
  - The Standby mode is entered using the PWR_EnterSTANDBYMode() function.
- Exit:

– WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, a tamper event, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop mode
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to:
    - Configure the EXTI Line 17 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
    - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function
    - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to:
    - Configure the EXTI Line 19 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
    - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.
    - Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
- RTC auto-wakeup (AWU) from the Standby mode
  - To wake up from the Standby mode with an RTC alarm event, it is necessary to:
    - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function.
    - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
  - To wake up from the Standby mode with an RTC Tamper or time stamp event, it is necessary to:
    - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.
    - Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
- Comparator auto-wakeup (AWU) from the Stop mode
  - To wake up from the Stop mode with a comparator wakeup event, it is necessary to:
    - Configure the correspondant comparator EXTI Line to be sensitive to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the EXTI_Init() function.
    - Configure the comparator to generate the event.
- ***PWR_EnterSleepMode()***
- ***PWR_EnterSTOPMode()***
- ***PWR_EnterSTANDBYMode()***

17.2.5 Flags management functions

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

17.2.6 Backup domain access function

17.2.6.1 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.

17.2.6.2 PWR_BackupAccessCmd

Function Name	**void PWR_BackupAccessCmd (*FunctionalState* NewState)**
Function Description	Enables or disables access to the RTC and backup registers.
Parameters	• **NewState :** new state of the access to the RTC and backup registers. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

17.2.7 PVD configuration function

17.2.7.1 PWR_PVDLevelConfig

Function Name	**void PWR_PVDLevelConfig (uint32_t PWR_PVDLevel)**
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	• **PWR_PVDLevel :** specifies the PVD detection level This parameter can be one of the following values: – ***PWR_PVDLevel_0 :*** PVD detection level set to 2.18V – ***PWR_PVDLevel_1 :*** PVD detection level set to 2.28V – ***PWR_PVDLevel_2 :*** PVD detection level set to 2.38V

- ***PWR_PVDLevel_3 :*** PVD detection level set to 2.48V
- ***PWR_PVDLevel_4 :*** PVD detection level set to 2.58V
- ***PWR_PVDLevel_5 :*** PVD detection level set to 2.68V
- ***PWR_PVDLevel_6 :*** PVD detection level set to 2.78V
- ***PWR_PVDLevel_7 :*** PVD detection level set to 2.88V

Return values
- None.

Notes
- None.

17.2.7.2 PWR_PVDCmd

Function Name **void PWR_PVDCmd (*FunctionalState* NewState)**

Function Description Enables or disables the Power Voltage Detector(PVD).

Parameters
- **NewState :** new state of the PVD. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

17.2.8 WakeUp pins configuration functions

17.2.8.1 PWR_WakeUpPinCmd

Function Name **void PWR_WakeUpPinCmd (uint32_t PWR_WakeUpPin, *FunctionalState* NewState)**

Function Description Enables or disables the WakeUp Pin functionality.

Parameters
- **PWR_WakeUpPin :** specifies the WakeUpPin. This parameter can be: PWR_WakeUpPin_1, PWR_WakeUpPin_2 or PWR_WakeUpPin_3.
- **NewState :** new state of the WakeUp Pin functionality. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

17.2.9 Low power mode configuration functions

17.2.9.1 PWR_EnterSleepMode

Function Name	**void PWR_EnterSleepMode (uint8_t PWR_SLEEPEntry)**
Function Description	Enters Sleep mode.
Parameters	• **PWR_SLEEPEntry :** specifies if SLEEP mode in entered with WFI or WFE instruction. This parameter can be one of the following values: – ***PWR_SLEEPEntry_WFI :*** enter SLEEP mode with WFI instruction – ***PWR_SLEEPEntry_WFE :*** enter SLEEP mode with WFE instruction
Return values	• None.
Notes	• In Sleep mode, all I/O pins keep the same state as in Run mode.

17.2.9.2 PWR_EnterSTOPMode

Function Name	**void PWR_EnterSTOPMode (uint32_t PWR_Regulator, uint8_t PWR_STOPEntry)**
Function Description	Enters STOP mode.
Parameters	• **PWR_Regulator :** specifies the regulator state in STOP mode. This parameter can be one of the following values: – ***PWR_Regulator_ON :*** STOP mode with regulator ON – ***PWR_Regulator_LowPower :*** STOP mode with regulator in low power mode • **PWR_STOPEntry :** specifies if STOP mode in entered with WFI or WFE instruction. This parameter can be one of the following values: – ***PWR_STOPEntry_WFI :*** enter STOP mode with WFI instruction – ***PWR_STOPEntry_WFE :*** enter STOP mode with WFE instruction
Return values	• None.
Notes	• In Stop mode, all I/O pins keep the same state as in Run mode. • When exiting Stop mode by issuing an interrupt or a wakeup event, the MSI RC oscillator is selected as system clock. • When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop

mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

17.2.9.3 PWR_EnterSTANDBYMode

Function Name	**void PWR_EnterSTANDBYMode (void)**
Function Description	Enters STANDBY mode.
Parameters	• None.
Return values	• None.
Notes	• In Standby mode, all I/O pins are high impedance except for: • Reset pad (still available) • RTC_AF1 pin (PC13) if configured for Wakeup pin 2 (WKUP2), tamper, time-stamp, RTC Alarm out, or RTC clock calibration out. • WKUP pin 1 (PA0) and WKUP pin 3 (PE6), if enabled.

17.2.10 Flag management functions

17.2.10.1 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 flag to check. This parameter can be one of the following values: PWR_FLAG_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm A or Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. PWR_FLAG_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode. PWR_FLAG_PVDO: PVD Output. This flag is valid only if PVD is enabled by the PWR_PVDCmd() function. PWR_FLAG_VREFINTRDY: Internal Voltage Reference Ready flag. This flag indicates the state of the internal voltage reference, VREFINT. – ***PWR_FLAG_WU :*** Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm A or Alarm B), RTC Tamper

event, RTC TimeStamp event or RTC Wakeup.
  - ***PWR_FLAG_SB :*** StandBy flag. This flag indicates that the system was resumed from StandBy mode.
  - ***PWR_FLAG_PVDO :*** PVD Output. This flag is valid only if PVD is enabled by the PWR_PVDCmd() function.
  - ***PWR_FLAG_VREFINTRDY :*** Internal Voltage Reference Ready flag. This flag indicates the state of the internal voltage reference, VREFINT.

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

Notes
- None.

17.2.10.2 PWR_ClearFlag

Function Name **void PWR_ClearFlag (uint32_t PWR_FLAG)**

Function Description Clears the PWR's pending flags.

Parameters
- **PWR_FLAG :** specifies the flag to clear. This parameter can be one of the following values:
  - ***PWR_FLAG_WU :*** Wake Up flag
  - ***PWR_FLAG_SB :*** StandBy flag

Return values
- None.

Notes
- None.

17.3 PWR Firmware driver defines

17.3.1 PWR

PWR

PWR_Flag

- #define: ***PWR_FLAG_WU PWR_CSR_WUF***

- #define: ***PWR_FLAG_SB PWR_CSR_SBF***

- #define: ***PWR_FLAG_PVDO PWR_CSR_PVDO***

- #define: ***PWR_FLAG_VREFINTRDY PWR_CSR_VREFINTRDYF***

PWR_PVD_detection_level

- #define: ***PWR_PVDLevel_0 PWR_CR_PLS_LEV0***
- #define: ***PWR_PVDLevel_1 PWR_CR_PLS_LEV1***
- #define: ***PWR_PVDLevel_2 PWR_CR_PLS_LEV2***
- #define: ***PWR_PVDLevel_3 PWR_CR_PLS_LEV3***
- #define: ***PWR_PVDLevel_4 PWR_CR_PLS_LEV4***
- #define: ***PWR_PVDLevel_5 PWR_CR_PLS_LEV5***
- #define: ***PWR_PVDLevel_6 PWR_CR_PLS_LEV6***
- #define: ***PWR_PVDLevel_7 PWR_CR_PLS_LEV7***

PWR_Regulator_state_is_Sleep_STOP_mode

- #define: ***PWR_Regulator_ON ((uint32_t)0x00000000)***
- #define: ***PWR_Regulator_LowPower PWR_CR_LPSDSR***

PWR_SLEEP_mode_entry

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

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

PWR_STOP_mode_entry

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

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

PWR_WakeUp_Pins

- #define: ***PWR_WakeUpPin_1 PWR_CSR_EWUP1***

- #define: ***PWR_WakeUpPin_2 PWR_CSR_EWUP2***

- #define: ***PWR_WakeUpPin_3 PWR_CSR_EWUP3***

18 Reset and clock control (RCC)

18.1 RCC Firmware driver registers structures

18.1.1 RCC_TypeDef

RCC_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t CFGR***
- ***__IO uint32_t CIR***
- ***__IO uint32_t APB2RSTR***
- ***__IO uint32_t APB1RSTR***
- ***__IO uint32_t AHBENR***
- ***__IO uint32_t APB2ENR***
- ***__IO uint32_t APB1ENR***
- ***__IO uint32_t BDCR***
- ***__IO uint32_t CSR***
- ***__IO uint32_t AHBRSTR***
- ***__IO uint32_t CFGR2***
- ***__IO uint32_t CFGR3***

Field Documentation

- ***__IO uint32_t RCC_TypeDef::CR***
  - RCC clock control register, Address offset: 0x00
- ***__IO uint32_t RCC_TypeDef::CFGR***
  - RCC clock configuration register, Address offset: 0x04
- ***__IO uint32_t RCC_TypeDef::CIR***
  - RCC clock interrupt register, Address offset: 0x08
- ***__IO uint32_t RCC_TypeDef::APB2RSTR***
  - RCC APB2 peripheral reset register, Address offset: 0x0C
- ***__IO uint32_t RCC_TypeDef::APB1RSTR***
  - RCC APB1 peripheral reset register, Address offset: 0x10
- ***__IO uint32_t RCC_TypeDef::AHBENR***
  - RCC AHB peripheral clock register, Address offset: 0x14
- ***__IO uint32_t RCC_TypeDef::APB2ENR***
  - RCC APB2 peripheral clock enable register, Address offset: 0x18
- ***__IO uint32_t RCC_TypeDef::APB1ENR***
  - RCC APB1 peripheral clock enable register, Address offset: 0x1C
- ***__IO uint32_t RCC_TypeDef::BDCR***
  - RCC Backup domain control register, Address offset: 0x20
- ***__IO uint32_t RCC_TypeDef::CSR***
  - RCC clock control & status register, Address offset: 0x24
- ***__IO uint32_t RCC_TypeDef::AHBRSTR***
  - RCC AHB peripheral reset register, Address offset: 0x28
- ***__IO uint32_t RCC_TypeDef::CFGR2***

  - RCC clock configuration register 2, Address offset: 0x2C
- ***__IO uint32_t RCC_TypeDef::CFGR3***
  - RCC clock configuration register 3, Address offset: 0x30

18.1.2 RCC_ClocksTypeDef

RCC_ClocksTypeDef is defined in the stm32f30x_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***
- ***uint32_t ADC12CLK_Frequency***
- ***uint32_t ADC34CLK_Frequency***
- ***uint32_t I2C1CLK_Frequency***
- ***uint32_t I2C2CLK_Frequency***
- ***uint32_t TIM1CLK_Frequency***
- ***uint32_t TIM8CLK_Frequency***
- ***uint32_t USART1CLK_Frequency***
- ***uint32_t USART2CLK_Frequency***
- ***uint32_t USART3CLK_Frequency***
- ***uint32_t UART4CLK_Frequency***
- ***uint32_t UART5CLK_Frequency***

Field Documentation

- ***uint32_t RCC_ClocksTypeDef::SYSCLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::HCLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::PCLK1_Frequency***
- ***uint32_t RCC_ClocksTypeDef::PCLK2_Frequency***
- ***uint32_t RCC_ClocksTypeDef::ADC12CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::ADC34CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::I2C1CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::I2C2CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::TIM1CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::TIM8CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::USART1CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::USART2CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::USART3CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::UART4CLK_Frequency***
- ***uint32_t RCC_ClocksTypeDef::UART5CLK_Frequency***

18.2 RCC Firmware driver API description

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

18.2.1 RCC specific features

After reset the device is running from HSI (8 MHz) with Flash 0 WS, all peripherals are off except internal SRAM, Flash and SWD.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the SWD pins which are assigned to be used for debug purpose.

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

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance).
- Configure the System clock frequency and Flash settings.
- Configure the AHB and APB busses prescalers.
- Enable the clock for the peripheral(s) to be used.
- Configure the clock source(s) for peripherals which clocks are not derived from the System clock (ADC, TIM, I2C, USART, RTC and IWDG).

18.2.2 Internal-external clocks, PLL, CSS and MCO configuration functions

This section provides functions allowing to configure the internal/external clocks, PLL, CSS and MCO.

1. HSI (high-speed internal), 8 MHz factory-trimmed RC used directly or through the PLL as System clock source. The HSI clock can be used also to clock the USART and I2C peripherals.
2. LSI (low-speed internal), 40 KHz low consumption RC used as IWDG and/or RTC clock source.
3. HSE (high-speed external), 4 to 32 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
4. LSE (low-speed external), 32 KHz oscillator used as RTC clock source. LSE can be used also to clock the USART peripherals.
5. PLL (clocked by HSI or HSE), for System clock.
6. CSS (Clock security system), once enabled and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.
7. MCO (microcontroller clock output), used to output SYSCLK, HSI, HSE, LSI, LSE, PLL clock on PA8 pin.

- ***RCC_DeInit()***
- ***RCC_HSEConfig()***
- ***RCC_WaitForHSEStartUp()***
- ***RCC_AdjustHSICalibrationValue()***
- ***RCC_HSICmd()***
- ***RCC_LSEConfig()***
- ***RCC_LSEDriveConfig()***
- ***RCC_LSICmd()***

- *RCC_PLLConfig()*
- *RCC_PLLCmd()*
- *RCC_PREDIV1Config()*
- *RCC_ClockSecuritySystemCmd()*
- *RCC_MCOConfig()*

18.2.3 System, AHB, APB1 and APB2 busses clocks configuration functions

This section provide functions allowing to configure the System, AHB, APB1 and APB2 busses clocks.

1. Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA and GPIO). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "RCC_GetClocksFreq()" function to retrieve the frequencies of these clocks.
2. The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 72 MHz. Depending on the maximum frequency, the FLASH wait states (WS) should be adapted accordingly:

Wait states	HCLK clock frequency (MHz)
0WS(1CPU cycle)	0 < HCLK <= 24
1WS(2CPU cycles)	24 < HCLK <= 48
2WS(3CPU cycles)	48 < HCLK <= 72

3. After reset, the System clock source is the HSI (8 MHz) with 0 WS and prefetch is disabled.

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

- The FLASH program/erase clock which is always HSI 8MHz clock.
- The USB 48 MHz clock which is derived from the PLL VCO clock.
- The USART clock which can be derived as well from HSI 8MHz, LSI or LSE.
- The I2C clock which can be derived as well from HSI 8MHz clock.
- The ADC clock which is derived from PLL output.
- The RTC clock which is derived from the LSE, LSI or 1 MHz HSE_RTC (HSE divided by a programmable prescaler). The System clock (SYSCLK) frequency must be higher or equal to the RTC clock frequency.
- IWDG clock which is always the LSI clock.

It is recommended to use the following software sequences to tune the number of wait states needed to access the Flash memory with the CPU frequency (HCLK).

- Increasing the CPU frequency
  - Program the Flash Prefetch buffer, using "FLASH_PrefetchBufferCmd(ENABLE)" function
  - Check that Flash Prefetch buffer activation is taken into account by reading FLASH_ACR using the FLASH_GetPrefetchBufferStatus() function
  - Program Flash WS to 1 or 2, using "FLASH_SetLatency()" function

  - Check that the new number of WS is taken into account by reading FLASH_ACR
  - Modify the CPU clock source, using "RCC_SYSCLKConfig()" function
  - If needed, modify the CPU clock prescaler by using "RCC_HCLKConfig()" function
  - Check that the new CPU clock source is taken into account by reading the clock source status, using "RCC_GetSYSCLKSource()" function
- Decreasing the CPU frequency
  - Modify the CPU clock source, using "RCC_SYSCLKConfig()" function
  - If needed, modify the CPU clock prescaler by using "RCC_HCLKConfig()" function
  - Check that the new CPU clock source is taken into account by reading the clock source status, using "RCC_GetSYSCLKSource()" function
  - Program the new number of WS, using "FLASH_SetLatency()" function
  - Check that the new number of WS is taken into account by reading FLASH_ACR
  - Disable the Flash Prefetch buffer using "FLASH_PrefetchBufferCmd(DISABLE)" function
  - Check that Flash Prefetch buffer deactivation is taken into account by reading FLASH_ACR using the FLASH_GetPrefetchBufferStatus() function.
- ***RCC_SYSCLKConfig()***
- ***RCC_GetSYSCLKSource()***
- ***RCC_HCLKConfig()***
- ***RCC_PCLK1Config()***
- ***RCC_PCLK2Config()***
- ***RCC_GetClocksFreq()***

18.2.4 Peripheral clocks configuration functions

This section provide functions allowing to configure the Peripheral clocks.

1. The RTC clock which is derived from the LSE, LSI or HSE_Div32 (HSE divided by 32).
2. After restart from Reset or wakeup from STANDBY, all peripherals are off except internal SRAM, Flash and SWD. Before to start using a peripheral you have to enable its interface clock. You can do this using RCC_AHBPeriphClockCmd(), RCC_APB2PeriphClockCmd() and RCC_APB1PeriphClockCmd() functions.
3. To reset the peripherals configuration (to the default state after device reset) you can use RCC_AHBPeriphResetCmd(), RCC_APB2PeriphResetCmd() and RCC_APB1PeriphResetCmd() functions.

- ***RCC_ADCCLKConfig()***
- ***RCC_I2CCLKConfig()***
- ***RCC_TIMCLKConfig()***
- ***RCC_USARTCLKConfig()***
- ***RCC_USBCLKConfig()***
- ***RCC_RTCCLKConfig()***
- ***RCC_I2SCLKConfig()***
- ***RCC_RTCCLKCmd()***
- ***RCC_BackupResetCmd()***
- ***RCC_AHBPeriphClockCmd()***
- ***RCC_APB2PeriphClockCmd()***
- ***RCC_APB1PeriphClockCmd()***
- ***RCC_AHBPeriphResetCmd()***
- ***RCC_APB2PeriphResetCmd()***
- ***RCC_APB1PeriphResetCmd()***

18.2.5 Interrupts and flags management functions

- ***RCC_ITConfig()***
- ***RCC_GetFlagStatus()***
- ***RCC_ClearFlag()***
- ***RCC_GetITStatus()***
- ***RCC_ClearITPendingBit()***

18.2.6 Internal and external clocks, PLL, CSS and MCO configuration functions

18.2.6.1 RCC_DeInit

Function Name	**void RCC_DeInit (void)**
Function Description	Resets the RCC 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 • HSE and PLL OFF • AHB, APB1 and APB2 prescalers set to 1. • CSS and MCO OFF • All interrupts disabled • However, this function doesn't modify the configuration of the • Peripheral clocks • LSI, LSE and RTC clocks

18.2.6.2 RCC_HSEConfig

Function Name	**void RCC_HSEConfig (uint8_t RCC_HSE)**
Function Description	Configures the External High Speed oscillator (HSE).
Parameters	• **RCC_HSE :** specifies the new state of the HSE. This parameter can be one of the following values: – ***RCC_HSE_OFF :*** turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles. – ***RCC_HSE_ON :*** turn ON the HSE oscillator – ***RCC_HSE_Bypass :*** HSE oscillator bypassed with external clock

Return values	• None.
Notes	• After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. • HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). • The HSE is stopped by hardware when entering STOP and STANDBY modes. • This function resets the CSSON bit, so if the Clock security system(CSS) was previously enabled you have to enable it again after calling this function.

18.2.6.3 RCC_WaitForHSEStartUp

Function Name	**ErrorStatus RCC_WaitForHSEStartUp (void)**
Function Description	Waits for HSE start-up.
Parameters	• None.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: HSE oscillator is stable and ready to use*** – ***ERROR: HSE oscillator not yet ready***
Notes	• This function waits on HSERDY flag to be set and return SUCCESS if this flag is set, otherwise returns ERROR if the timeout is reached and this flag is not set. The timeout value is defined by the constant HSE_STARTUP_TIMEOUT in stm32f30x.h file. You can tailor it depending on the HSE crystal used in your application.

18.2.6.4 RCC_AdjustHSICalibrationValue

Function Name	**void RCC_AdjustHSICalibrationValue (uint8_t HSICalibrationValue)**
Function Description	Adjusts the Internal High Speed oscillator (HSI) calibration value.
Parameters	• **HSICalibrationValue :** specifies the HSI calibration trimming value. This parameter must be a number between 0 and

0x1F.

Return values
- None.

Notes
- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC. Refer to the Application Note AN3300 for more details on how to calibrate the HSI.

18.2.6.5 RCC_HSICmd

Function Name **void RCC_HSICmd (*FunctionalState* NewState)**

Function Description Enables or disables the Internal High Speed oscillator (HSI).

Parameters
- **NewState :** new state of the HSI. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used to clock the PLL and/or system clock.
- HSI can not be stopped if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then stop the HSI.
- The HSI is stopped by hardware when entering STOP and STANDBY modes.
- When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

18.2.6.6 RCC_LSEConfig

Function Name **void RCC_LSEConfig (uint32_t RCC_LSE)**

Function Description Configures the External Low Speed oscillator (LSE).

Parameters
- **RCC_LSE :** specifies the new state of the LSE. This parameter can be one of the following values:
  - ***RCC_LSE_OFF :*** turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - ***RCC_LSE_ON :*** turn ON the LSE oscillator
  - ***RCC_LSE_Bypass :*** LSE oscillator bypassed with external clock

Return values	• None.
Notes	• As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using PWR_BackupAccessCmd(ENABLE) function before to configure the LSE (to be done once after reset). • After enabling the LSE (RCC_LSE_ON or RCC_LSE_Bypass), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

18.2.6.7 RCC_LSEDriveConfig

Function Name	**void RCC_LSEDriveConfig (uint32_t RCC_LSEDrive)**
Function Description	Configures the External Low Speed oscillator (LSE) drive capability.
Parameters	• **RCC_LSEDrive :** specifies the new state of the LSE drive capability. This parameter can be one of the following values: – ***RCC_LSEDrive_Low :*** LSE oscillator low drive capability. – ***RCC_LSEDrive_MediumLow :*** LSE oscillator medium low drive capability. – ***RCC_LSEDrive_MediumHigh :*** LSE oscillator medium high drive capability. – ***RCC_LSEDrive_High :*** LSE oscillator high drive capability.
Return values	• None.
Notes	• None.

18.2.6.8 RCC_LSICmd

Function Name	**void RCC_LSICmd (*FunctionalState* NewState)**
Function Description	Enables or disables the Internal Low Speed oscillator (LSI).
Parameters	• **NewState :** new state of the LSI. This parameter can be: ENABLE or DISABLE.
Return values	• None.

Notes
- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC.
- LSI can not be disabled if the IWDG is running.
- When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

18.2.6.9 RCC_PLLConfig

Function Name: **void RCC_PLLConfig (uint32_t RCC_PLLSource, uint32_t RCC_PLLMul)**

Function Description: Configures the PLL clock source and multiplication factor.

Parameters
- **RCC_PLLSource :** specifies the PLL entry clock source. This parameter can be one of the following values:
  - ***RCC_PLLSource_HSI_Div2 :*** HSI oscillator clock divided by 2 selected as PLL clock entry
  - ***RCC_PLLSource_PREDIV1 :*** PREDIV1 clock selected as PLL clock source
- **RCC_PLLMul :** specifies the PLL multiplication factor, which drive the PLLVCO clock This parameter can be RCC_PLLMul_x where x:[2,16]

Return values
- None.

Notes
- This function must be used only when the PLL is disabled.
- The minimum input clock frequency for PLL is 2 MHz (when using HSE as PLL source).

18.2.6.10 RCC_PLLCmd

Function Name: **void RCC_PLLCmd (*FunctionalState* NewState)**

Function Description: Enables or disables the PLL.

Parameters
- **NewState :** new state of the PLL. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- After enabling the PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source.

- The PLL can not be disabled if it is used as system clock source
- The PLL is disabled by hardware when entering STOP and STANDBY modes.

18.2.6.11 RCC_PREDIV1Config

Function Name	**void RCC_PREDIV1Config (uint32_t RCC_PREDIV1_Div)**
Function Description	Configures the PREDIV1 division factor.
Parameters	• **RCC_PREDIV1_Div :** specifies the PREDIV1 clock division factor. This parameter can be RCC_PREDIV1_Divx where x:[1,16]
Return values	• None.
Notes	• This function must be used only when the PLL is disabled.

18.2.6.12 RCC_ClockSecuritySystemCmd

Function Name	**void RCC_ClockSecuritySystemCmd (*FunctionalState* NewState)**
Function Description	Enables or disables the Clock Security System.
Parameters	• **NewState :** new state of the Clock Security System. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.

18.2.6.13 RCC_MCOConfig

Function Name	**void RCC_MCOConfig (uint8_t RCC_MCOSource)**
Function Description	Selects the clock source to output on MCO pin (PA8).
Parameters	• **RCC_MCOSource :** specifies the clock source to output. This parameter can be one of the following values: – ***RCC_MCOSource_NoClock :*** No clock selected. – ***RCC_MCOSource_LSI :*** LSI oscillator clock selected. – ***RCC_MCOSource_LSE :*** LSE oscillator clock selected. – ***RCC_MCOSource_SYSCLK :*** System clock selected. – ***RCC_MCOSource_HSI :*** HSI oscillator clock selected. – ***RCC_MCOSource_HSE :*** HSE oscillator clock selected. – ***RCC_MCOSource_PLLCLK_Div2 :*** PLL clock selected.
Return values	• None.
Notes	• PA8 should be configured in alternate function mode. • The MCOF flag is set once the MCO clock source switch is effective.

18.2.7 System AHB, APB1 and APB2 busses clocks configuration functions

18.2.7.1 RCC_SYSCLKConfig

Function Name	**void RCC_SYSCLKConfig (uint32_t RCC_SYSCLKSource)**
Function Description	Configures the system clock (SYSCLK).
Parameters	• **RCC_SYSCLKSource :** specifies the clock source used as system clock source This parameter can be one of the following values: – ***RCC_SYSCLKSource_HSI :*** HSI selected as system clock source – ***RCC_SYSCLKSource_HSE :*** HSE selected as system clock source – ***RCC_SYSCLKSource_PLLCLK :*** PLL selected as system clock source
Return values	• None.
Notes	• The HSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). • A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is

selected, the switch will occur when the clock source will be ready. You can use RCC_GetSYSCLKSource() function to know which clock is currently used as system clock source.

18.2.7.2 RCC_GetSYSCLKSource

Function Name	**uint8_t RCC_GetSYSCLKSource (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 values:** – ***0x00: HSI used as system clock*** – ***0x04: HSE used as system clock*** – ***0x08: PLL used as system clock***
Notes	• None.

18.2.7.3 RCC_HCLKConfig

Function Name	**void RCC_HCLKConfig (uint32_t RCC_SYSCLK)**
Function Description	Configures the AHB clock (HCLK).
Parameters	• **RCC_SYSCLK :** defines the AHB clock divider. This clock is derived from the system clock (SYSCLK). This parameter can be one of the following values: – ***RCC_SYSCLK_Div1 :*** AHB clock = SYSCLK – ***RCC_SYSCLK_Div2 :*** AHB clock = SYSCLK/2 – ***RCC_SYSCLK_Div4 :*** AHB clock = SYSCLK/4 – ***RCC_SYSCLK_Div8 :*** AHB clock = SYSCLK/8 – ***RCC_SYSCLK_Div16 :*** AHB clock = SYSCLK/16 – ***RCC_SYSCLK_Div64 :*** AHB clock = SYSCLK/64 – ***RCC_SYSCLK_Div128 :*** AHB clock = SYSCLK/128 – ***RCC_SYSCLK_Div256 :*** AHB clock = SYSCLK/256 – ***RCC_SYSCLK_Div512 :*** AHB clock = SYSCLK/512
Return values	• None.
Notes	• Depending on the device voltage range, the software has to set correctly these bits to ensure that the system frequency does not exceed the maximum allowed frequency (for more

details refer to section above "CPU, AHB and APB busses clocks configuration functions").

18.2.7.4 RCC_PCLK1Config

Function Name	**void RCC_PCLK1Config (uint32_t RCC_HCLK)**
Function Description	Configures the Low Speed APB clock (PCLK1).
Parameters	• **RCC_HCLK :** defines the APB1 clock divider. This clock is derived from the AHB clock (HCLK). This parameter can be one of the following values: – ***RCC_HCLK_Div1 :*** APB1 clock = HCLK – ***RCC_HCLK_Div2 :*** APB1 clock = HCLK/2 – ***RCC_HCLK_Div4 :*** APB1 clock = HCLK/4 – ***RCC_HCLK_Div8 :*** APB1 clock = HCLK/8 – ***RCC_HCLK_Div16 :*** APB1 clock = HCLK/16
Return values	• None.
Notes	• None.

18.2.7.5 RCC_PCLK2Config

Function Name	**void RCC_PCLK2Config (uint32_t RCC_HCLK)**
Function Description	Configures the High Speed APB clock (PCLK2).
Parameters	• **RCC_HCLK :** defines the APB2 clock divider. This clock is derived from the AHB clock (HCLK). This parameter can be one of the following values: – ***RCC_HCLK_Div1 :*** APB2 clock = HCLK – ***RCC_HCLK_Div2 :*** APB2 clock = HCLK/2 – ***RCC_HCLK_Div4 :*** APB2 clock = HCLK/4 – ***RCC_HCLK_Div8 :*** APB2 clock = HCLK/8 – ***RCC_HCLK_Div16 :*** APB2 clock = HCLK/16
Return values	• None.
Notes	• None.

18.2.7.6 RCC_GetClocksFreq

Function Name	**void RCC_GetClocksFreq (*RCC_ClocksTypeDef** * RCC_Clocks)**
Function Description	Returns the frequencies of the System, AHB, APB2 and APB1 busses clocks.
Parameters	• **RCC_Clocks :** pointer to a RCC_ClocksTypeDef structure which will hold the clocks frequencies.
Return values	• None.
Notes	• This function returns the frequencies of : System, AHB, APB2 and APB1 busses clocks, ADC1/2/3/4 clocks, USART1/2/3/4/5 clocks, I2C1/2 clocks and TIM1/8 Clocks. • The frequency returned by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the source selected by RCC_SYSCLKConfig(). • If SYSCLK source is HSI, function returns constant HSI_VALUE(*) • If SYSCLK source is HSE, function returns constant HSE_VALUE(**) • If SYSCLK source is PLL, function returns constant HSE_VALUE(**) or HSI_VALUE(*) multiplied by the PLL factors. • (*) HSI_VALUE is a constant defined in stm32f30x.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature, refer to RCC_AdjustHSICalibrationValue(). • (**) HSE_VALUE is a constant defined in stm32f30x.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may return wrong result. • The result of this function could be not correct when using fractional value for HSE crystal. • This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters. • Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 clock changes, this function must be called to update the structure's field. Otherwise, any configuration based on this function will be incorrect.

18.2.8 Peripheral clocks configuration functions

18.2.8.1 RCC_ADCCLKConfig

Function Name	**void RCC_ADCCLKConfig (uint32_t RCC_PLLCLK)**
Function Description	Configures the ADC clock (ADCCLK).
Parameters	• **RCC_PLLCLK :** defines the ADC clock divider. This clock is derived from the PLL Clock. This parameter can be one of the following values: – ***RCC_ADC12PLLCLK_OFF :*** ADC12 clock disabled – ***RCC_ADC12PLLCLK_Div1 :*** ADC12 clock = PLLCLK/1 – ***RCC_ADC12PLLCLK_Div2 :*** ADC12 clock = PLLCLK/2 – ***RCC_ADC12PLLCLK_Div4 :*** ADC12 clock = PLLCLK/4 – ***RCC_ADC12PLLCLK_Div6 :*** ADC12 clock = PLLCLK/6 – ***RCC_ADC12PLLCLK_Div8 :*** ADC12 clock = PLLCLK/8 – ***RCC_ADC12PLLCLK_Div10 :*** ADC12 clock = PLLCLK/10 – ***RCC_ADC12PLLCLK_Div12 :*** ADC12 clock = PLLCLK/12 – ***RCC_ADC12PLLCLK_Div16 :*** ADC12 clock = PLLCLK/16 – ***RCC_ADC12PLLCLK_Div32 :*** ADC12 clock = PLLCLK/32 – ***RCC_ADC12PLLCLK_Div64 :*** ADC12 clock = PLLCLK/64 – ***RCC_ADC12PLLCLK_Div128 :*** ADC12 clock = PLLCLK/128 – ***RCC_ADC12PLLCLK_Div256 :*** ADC12 clock = PLLCLK/256 – ***RCC_ADC34PLLCLK_OFF :*** ADC34 clock disabled – ***RCC_ADC34PLLCLK_Div1 :*** ADC34 clock = PLLCLK/1 – ***RCC_ADC34PLLCLK_Div2 :*** ADC34 clock = PLLCLK/2 – ***RCC_ADC34PLLCLK_Div4 :*** ADC34 clock = PLLCLK/4 – ***RCC_ADC34PLLCLK_Div6 :*** ADC34 clock = PLLCLK/6 – ***RCC_ADC34PLLCLK_Div8 :*** ADC34 clock = PLLCLK/8 – ***RCC_ADC34PLLCLK_Div10 :*** ADC34 clock = PLLCLK/10 – ***RCC_ADC34PLLCLK_Div12 :*** ADC34 clock = PLLCLK/12 – ***RCC_ADC34PLLCLK_Div16 :*** ADC34 clock = PLLCLK/16 – ***RCC_ADC34PLLCLK_Div32 :*** ADC34 clock = PLLCLK/32 – ***RCC_ADC34PLLCLK_Div64 :*** ADC34 clock = PLLCLK/64 – ***RCC_ADC34PLLCLK_Div128 :*** ADC34 clock = PLLCLK/128 – ***RCC_ADC34PLLCLK_Div256 :*** ADC34 clock = PLLCLK/256
Return values	• None.
Notes	• None.

18.2.8.2 RCC_I2CCLKConfig

Function Name	**void RCC_I2CCLKConfig (uint32_t RCC_I2CCLK)**
Function Description	Configures the I2C clock (I2CCLK).
Parameters	• **RCC_I2CCLK :** defines the I2C clock source. This clock is derived from the HSI or System clock. This parameter can be one of the following values: – ***RCC_I2CxCLK_HSI :*** I2Cx clock = HSI – ***RCC_I2CxCLK_SYSCLK :*** I2Cx clock = System Clock (x can be 1 or 2).
Return values	• None.
Notes	• None.

18.2.8.3 RCC_TIMCLKConfig

Function Name	**void RCC_TIMCLKConfig (uint32_t RCC_TIMCLK)**
Function Description	Configures the TIM1 and TIM8 clock sources(TIMCLK).
Parameters	• **RCC_TIMCLK :** defines the TIMx clock source. This parameter can be one of the following values: – ***RCC_TIMxCLK_HCLK :*** TIMx clock = APB high speed clock (doubled frequency when prescaled) – ***RCC_TIMxCLK_PLLCLK :*** TIMx clock = PLL output (running up to 144 MHz) (x can be 1 or 8).
Return values	• None.
Notes	• The configuration of the TIMx clock source is only possible when the SYSCLK = PLL and HCLK and PCLK2 clocks are not divided in respect to SYSCLK • If one of the previous conditions is missed, the TIM clock source configuration is lost and calling again this function becomes mandatory.

18.2.8.4 RCC_USARTCLKConfig

Function Name	**void RCC_USARTCLKConfig (uint32_t RCC_USARTCLK)**
Function Description	Configures the USART clock (USARTCLK).
Parameters	• **RCC_USARTCLK :** defines the USART clock source. This clock is derived from the HSI or System clock. This parameter can be one of the following values: – ***RCC_USARTxCLK_PCLK :*** USART clock = APB Clock (PCLK) – ***RCC_USARTxCLK_SYSCLK :*** USART clock = System Clock – ***RCC_USARTxCLK_LSE :*** USART clock = LSE Clock – ***RCC_USARTxCLK_HSI :*** USART clock = HSI Clock (x can be 1, 2, 3, 4 or 5).
Return values	• None.
Notes	• None.

18.2.8.5 RCC_USBCLKConfig

Function Name	**void RCC_USBCLKConfig (uint32_t RCC_USBCLKSource)**
Function Description	Configures the USB clock (USBCLK).
Parameters	• **RCC_USBCLKSource :** specifies the USB clock source. This clock is derived from the PLL output. This parameter can be one of the following values: – ***RCC_USBCLKSource_PLLCLK_1Div5 :*** PLL clock divided by 1,5 selected as USB clock source – ***RCC_USBCLKSource_PLLCLK_Div1 :*** PLL clock selected as USB clock source
Return values	• None.
Notes	• None.

18.2.8.6 RCC_RTCCLKConfig

Function Name	**void RCC_RTCCLKConfig (uint32_t RCC_RTCCLKSource)**
Function Description	Configures the RTC clock (RTCCLK).
Parameters	• **RCC_RTCCLKSource :** specifies the RTC clock source. This parameter can be one of the following values: – ***RCC_RTCCLKSource_LSE :*** LSE selected as RTC clock – ***RCC_RTCCLKSource_LSI :*** LSI selected as RTC clock – ***RCC_RTCCLKSource_HSE_Div32 :*** HSE divided by 32 selected as RTC clock
Return values	• None.
Notes	• As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using PWR_BackupAccessCmd(ENABLE) function before to configure the RTC clock source (to be done once after reset). • Once the RTC clock is configured it can't be changed unless the RTC is reset using RCC_BackupResetCmd function, or by a Power On Reset (POR) • If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. • The maximum input clock frequency for RTC is 2MHz (when using HSE as RTC clock source).

18.2.8.7 RCC_I2SCLKConfig

Function Name	**void RCC_I2SCLKConfig (uint32_t RCC_I2SCLKSource)**
Function Description	Configures the I2S clock source (I2SCLK).
Parameters	• **RCC_I2SCLKSource :** specifies the I2S clock source. This parameter can be one of the following values: – ***RCC_I2S2CLKSource_SYSCLK :*** SYSCLK clock used as I2S clock source – ***RCC_I2S2CLKSource_Ext :*** External clock mapped on the I2S_CKIN pin used as I2S clock source
Return values	• None.
Notes	• This function must be called before enabling the SPI2 and SPI3 clocks.

18.2.8.8 RCC_RTCCLKCmd

Function Name	**void RCC_RTCCLKCmd (*FunctionalState* NewState)**
Function Description	Enables or disables the RTC clock.
Parameters	• **NewState :** new state of the RTC clock. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function must be used only after the RTC clock source was selected using the RCC_RTCCLKConfig function.

18.2.8.9 RCC_BackupResetCmd

Function Name	**void RCC_BackupResetCmd (*FunctionalState* NewState)**
Function Description	Forces or releases the Backup domain reset.
Parameters	• **NewState :** new state of the Backup domain reset. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_BDCR register.

18.2.8.10 RCC_AHBPeriphClockCmd

Function Name	**void RCC_AHBPeriphClockCmd (uint32_t RCC_AHBPeriph, *FunctionalState* NewState)**
Function Description	Enables or disables the AHB peripheral clock.
Parameters	• **RCC_AHBPeriph :** specifies the AHB peripheral to gates its clock. This parameter can be any combination of the following values: – ***RCC_AHBPeriph_GPIOA :***

  - ***RCC_AHBPeriph_GPIOB :***
  - ***RCC_AHBPeriph_GPIOC :***
  - ***RCC_AHBPeriph_GPIOD :***
  - ***RCC_AHBPeriph_GPIOE :***
  - ***RCC_AHBPeriph_GPIOF :***
  - ***RCC_AHBPeriph_TS :***
  - ***RCC_AHBPeriph_CRC :***
  - ***RCC_AHBPeriph_FLITF :***
  - ***RCC_AHBPeriph_SRAM :***
  - ***RCC_AHBPeriph_DMA2 :***
  - ***RCC_AHBPeriph_DMA1 :***
  - ***RCC_AHBPeriph_ADC34 :***
  - ***RCC_AHBPeriph_ADC12 :***
- **NewState :** new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

18.2.8.11 RCC_APB2PeriphClockCmd

Function Name **void RCC_APB2PeriphClockCmd (uint32_t RCC_APB2Periph, *FunctionalState* NewState)**

Function Description Enables or disables the High Speed APB (APB2) peripheral clock.

Parameters
- **RCC_APB2Periph :** specifies the APB2 peripheral to gates its clock. This parameter can be any combination of the following values:
  - ***RCC_APB2Periph_SYSCFG :***
  - ***RCC_APB2Periph_SPI1 :***
  - ***RCC_APB2Periph_USART1 :***
  - ***RCC_APB2Periph_TIM15 :***
  - ***RCC_APB2Periph_TIM16 :***
  - ***RCC_APB2Periph_TIM17 :***
  - ***RCC_APB2Periph_TIM1 :***
  - ***RCC_APB2Periph_TIM8 :***
- **NewState :** new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

18.2.8.12 RCC_APB1PeriphClockCmd

Function Name	**void RCC_APB1PeriphClockCmd (uint32_t RCC_APB1Periph, *FunctionalState* NewState)**
Function Description	Enables or disables the Low Speed APB (APB1) peripheral clock.
Parameters	• **RCC_APB1Periph :** specifies the APB1 peripheral to gates its clock. This parameter can be any combination of the following values: – ***RCC_APB1Periph_TIM2 :*** – ***RCC_APB1Periph_TIM3 :*** – ***RCC_APB1Periph_TIM4 :*** – ***RCC_APB1Periph_TIM6 :*** – ***RCC_APB1Periph_TIM7 :*** – ***RCC_APB1Periph_WWDG :*** – ***RCC_APB1Periph_SPI2 :*** – ***RCC_APB1Periph_SPI3 :*** – ***RCC_APB1Periph_USART2 :*** – ***RCC_APB1Periph_USART3 :*** – ***RCC_APB1Periph_UART4 :*** – ***RCC_APB1Periph_UART5 :*** – ***RCC_APB1Periph_I2C1 :*** – ***RCC_APB1Periph_I2C2 :*** – ***RCC_APB1Periph_USB :*** – ***RCC_APB1Periph_CAN1 :*** – ***RCC_APB1Periph_PWR :*** – ***RCC_APB1Periph_DAC :*** • **NewState :** new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

18.2.8.13 RCC_AHBPeriphResetCmd

Function Name	**void RCC_AHBPeriphResetCmd (uint32_t RCC_AHBPeriph, *FunctionalState* NewState)**
Function Description	Forces or releases AHB peripheral reset.

Parameters

- **RCC_AHBPeriph :** specifies the AHB peripheral to reset. This parameter can be any combination of the following values:
  - ***RCC_AHBPeriph_GPIOA :***
  - ***RCC_AHBPeriph_GPIOB :***
  - ***RCC_AHBPeriph_GPIOC :***
  - ***RCC_AHBPeriph_GPIOD :***
  - ***RCC_AHBPeriph_GPIOE :***
  - ***RCC_AHBPeriph_GPIOF :***
  - ***RCC_AHBPeriph_TS :***
  - ***RCC_AHBPeriph_ADC34 :***
  - ***RCC_AHBPeriph_ADC12 :***
- **NewState :** new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

- None.

Notes

- None.

18.2.8.14 RCC_APB2PeriphResetCmd

Function Name

void RCC_APB2PeriphResetCmd (uint32_t RCC_APB2Periph, *FunctionalState* NewState)

Function Description

Forces or releases High Speed APB (APB2) peripheral reset.

Parameters

- **RCC_APB2Periph :** specifies the APB2 peripheral to reset. This parameter can be any combination of the following values:
  - ***RCC_APB2Periph_SYSCFG :***
  - ***RCC_APB2Periph_SPI1 :***
  - ***RCC_APB2Periph_USART1 :***
  - ***RCC_APB2Periph_TIM15 :***
  - ***RCC_APB2Periph_TIM16 :***
  - ***RCC_APB2Periph_TIM17 :***
  - ***RCC_APB2Periph_TIM1 :***
  - ***RCC_APB2Periph_TIM8 :***
- **NewState :** new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

- None.

Notes

- None.

18.2.8.15 RCC_APB1PeriphResetCmd

Function Name	**void RCC_APB1PeriphResetCmd (uint32_t RCC_APB1Periph, *FunctionalState* NewState)**
Function Description	Forces or releases Low Speed APB (APB1) peripheral reset.
Parameters	• **RCC_APB1Periph :** specifies the APB1 peripheral to reset. This parameter can be any combination of the following values: – ***RCC_APB1Periph_TIM2 :*** – ***RCC_APB1Periph_TIM3 :*** – ***RCC_APB1Periph_TIM4 :*** – ***RCC_APB1Periph_TIM6 :*** – ***RCC_APB1Periph_TIM7 :*** – ***RCC_APB1Periph_WWDG :*** – ***RCC_APB1Periph_SPI2 :*** – ***RCC_APB1Periph_SPI3 :*** – ***RCC_APB1Periph_USART2 :*** – ***RCC_APB1Periph_USART3 :*** – ***RCC_APB1Periph_UART4 :*** – ***RCC_APB1Periph_UART5 :*** – ***RCC_APB1Periph_I2C1 :*** – ***RCC_APB1Periph_I2C2 :*** – ***RCC_APB1Periph_USB :*** – ***RCC_APB1Periph_CAN1 :*** – ***RCC_APB1Periph_PWR :*** – ***RCC_APB1Periph_DAC :*** • **NewState :** new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

18.2.9 Interrupts and flags management functions

18.2.9.1 RCC_ITConfig

Function Name	**void RCC_ITConfig (uint8_t RCC_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified RCC interrupts.
Parameters	• **RCC_IT :** specifies the RCC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:

- ***RCC_IT_LSIRDY :*** LSI ready interrupt
- ***RCC_IT_LSERDY :*** LSE ready interrupt
- ***RCC_IT_HSIRDY :*** HSI ready interrupt
- ***RCC_IT_HSERDY :*** HSE ready interrupt
- ***RCC_IT_PLLRDY :*** PLL ready interrupt

- **NewState :** new state of the specified RCC interrupts. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- The CSS interrupt doesn't have an enable bit; once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI will be executed indefinitely, and since NMI has higher priority than any other IRQ (and main program) the application will be stacked in the NMI ISR unless the CSS interrupt pending bit is cleared.

18.2.9.2 RCC_GetFlagStatus

Function Name **FlagStatus RCC_GetFlagStatus (uint8_t RCC_FLAG)**

Function Description Checks whether the specified RCC flag is set or not.

Parameters
- **RCC_FLAG :** specifies the flag to check. This parameter can be one of the following values:
  - ***RCC_FLAG_HSIRDY :*** HSI oscillator clock ready
  - ***RCC_FLAG_HSERDY :*** HSE oscillator clock ready
  - ***RCC_FLAG_PLLRDY :*** PLL clock ready
  - ***RCC_FLAG_MCOF :*** MCO Flag
  - ***RCC_FLAG_LSERDY :*** LSE oscillator clock ready
  - ***RCC_FLAG_LSIRDY :*** LSI oscillator clock ready
  - ***RCC_FLAG_OBLRST :*** Option Byte Loader (OBL) reset
  - ***RCC_FLAG_PINRST :*** Pin reset
  - ***RCC_FLAG_PORRST :*** POR/PDR reset
  - ***RCC_FLAG_SFTRST :*** Software reset
  - ***RCC_FLAG_IWDGRST :*** Independent Watchdog reset
  - ***RCC_FLAG_WWDGRST :*** Window Watchdog reset
  - ***RCC_FLAG_LPWRRST :*** Low Power reset

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

Notes
- None.

18.2.9.3 RCC_ClearFlag

Function Name	**void RCC_ClearFlag (void)**
Function Description	Clears the RCC reset flags.
Parameters	• None.
Return values	• None.
Notes	• None.

18.2.9.4 RCC_GetITStatus

Function Name	**ITStatus RCC_GetITStatus (uint8_t RCC_IT)**
Function Description	Checks whether the specified RCC interrupt has occurred or not.
Parameters	• **RCC_IT :** specifies the RCC interrupt source to check. This parameter can be one of the following values: – ***RCC_IT_LSIRDY :*** LSI ready interrupt – ***RCC_IT_LSERDY :*** LSE ready interrupt – ***RCC_IT_HSIRDY :*** HSI ready interrupt – ***RCC_IT_HSERDY :*** HSE ready interrupt – ***RCC_IT_PLLRDY :*** PLL ready interrupt – ***RCC_IT_CSS :*** Clock Security System interrupt
Return values	• **The new state of RCC_IT (SET or RESET).**
Notes	• None.

18.2.9.5 RCC_ClearITPendingBit

Function Name	**void RCC_ClearITPendingBit (uint8_t RCC_IT)**
Function Description	Clears the RCC's interrupt pending bits.
Parameters	• **RCC_IT :** specifies the interrupt pending bit to clear. This parameter can be any combination of the following values: – ***RCC_IT_LSIRDY :*** LSI ready interrupt – ***RCC_IT_LSERDY :*** LSE ready interrupt – ***RCC_IT_HSIRDY :*** HSI ready interrupt

- ***RCC_IT_HSERDY :*** HSE ready interrupt
- ***RCC_IT_PLLRDY :*** PLL ready interrupt
- ***RCC_IT_CSS :*** Clock Security System interrupt

Return values
- None.

Notes
- None.

18.3 RCC Firmware driver defines

18.3.1 RCC

RCC

RCC_ADC_clock_source

- #define: ***RCC_ADC12PLLCLK_OFF ((uint32_t)0x00000000)***
- #define: ***RCC_ADC12PLLCLK_Div1 ((uint32_t)0x00000100)***
- #define: ***RCC_ADC12PLLCLK_Div2 ((uint32_t)0x00000110)***
- #define: ***RCC_ADC12PLLCLK_Div4 ((uint32_t)0x00000120)***
- #define: ***RCC_ADC12PLLCLK_Div6 ((uint32_t)0x00000130)***
- #define: ***RCC_ADC12PLLCLK_Div8 ((uint32_t)0x00000140)***
- #define: ***RCC_ADC12PLLCLK_Div10 ((uint32_t)0x00000150)***
- #define: ***RCC_ADC12PLLCLK_Div12 ((uint32_t)0x00000160)***

- #define: ***RCC_ADC12PLLCLK_Div16 ((uint32_t)0x00000170)***
- #define: ***RCC_ADC12PLLCLK_Div32 ((uint32_t)0x00000180)***
- #define: ***RCC_ADC12PLLCLK_Div64 ((uint32_t)0x00000190)***
- #define: ***RCC_ADC12PLLCLK_Div128 ((uint32_t)0x000001A0)***
- #define: ***RCC_ADC12PLLCLK_Div256 ((uint32_t)0x000001B0)***
- #define: ***RCC_ADC34PLLCLK_OFF ((uint32_t)0x10000000)***
- #define: ***RCC_ADC34PLLCLK_Div1 ((uint32_t)0x10002000)***
- #define: ***RCC_ADC34PLLCLK_Div2 ((uint32_t)0x10002200)***
- #define: ***RCC_ADC34PLLCLK_Div4 ((uint32_t)0x10002400)***
- #define: ***RCC_ADC34PLLCLK_Div6 ((uint32_t)0x10002600)***
- #define: ***RCC_ADC34PLLCLK_Div8 ((uint32_t)0x10002800)***
- #define: ***RCC_ADC34PLLCLK_Div10 ((uint32_t)0x10002A00)***

- #define: ***RCC_ADC34PLLCLK_Div12 ((uint32_t)0x10002C00)***
- #define: ***RCC_ADC34PLLCLK_Div16 ((uint32_t)0x10002E00)***
- #define: ***RCC_ADC34PLLCLK_Div32 ((uint32_t)0x10003000)***
- #define: ***RCC_ADC34PLLCLK_Div64 ((uint32_t)0x10003200)***
- #define: ***RCC_ADC34PLLCLK_Div128 ((uint32_t)0x10003400)***
- #define: ***RCC_ADC34PLLCLK_Div256 ((uint32_t)0x10003600)***

RCC_AHB_Clock_Source

- #define: ***RCC_SYSCLK_Div1 RCC_CFGR_HPRE_DIV1***
- #define: ***RCC_SYSCLK_Div2 RCC_CFGR_HPRE_DIV2***
- #define: ***RCC_SYSCLK_Div4 RCC_CFGR_HPRE_DIV4***
- #define: ***RCC_SYSCLK_Div8 RCC_CFGR_HPRE_DIV8***
- #define: ***RCC_SYSCLK_Div16 RCC_CFGR_HPRE_DIV16***
- #define: ***RCC_SYSCLK_Div64 RCC_CFGR_HPRE_DIV64***

- #define: ***RCC_SYSCLK_Div128 RCC_CFGR_HPRE_DIV128***

- #define: ***RCC_SYSCLK_Div256 RCC_CFGR_HPRE_DIV256***

- #define: ***RCC_SYSCLK_Div512 RCC_CFGR_HPRE_DIV512***

RCC_AHB_Peripherals

- #define: ***RCC_AHBPeriph_ADC34 RCC_AHBENR_ADC34EN***

- #define: ***RCC_AHBPeriph_ADC12 RCC_AHBENR_ADC12EN***

- #define: ***RCC_AHBPeriph_GPIOA RCC_AHBENR_GPIOAEN***

- #define: ***RCC_AHBPeriph_GPIOB RCC_AHBENR_GPIOBEN***

- #define: ***RCC_AHBPeriph_GPIOC RCC_AHBENR_GPIOCEN***

- #define: ***RCC_AHBPeriph_GPIOD RCC_AHBENR_GPIODEN***

- #define: ***RCC_AHBPeriph_GPIOE RCC_AHBENR_GPIOEEN***

- #define: ***RCC_AHBPeriph_GPIOF RCC_AHBENR_GPIOFEN***

- #define: ***RCC_AHBPeriph_TS RCC_AHBENR_TSEN***

- #define: ***RCC_AHBPeriph_CRC RCC_AHBENR_CRCEN***

- #define: ***RCC_AHBPeriph_FLITF RCC_AHBENR_FLITFEN***

- #define: ***RCC_AHBPeriph_SRAM RCC_AHBENR_SRAMEN***

- #define: ***RCC_AHBPeriph_DMA2 RCC_AHBENR_DMA2EN***

- #define: ***RCC_AHBPeriph_DMA1 RCC_AHBENR_DMA1EN***

RCC_APB1_APB2_clock_source

- #define: ***RCC_HCLK_Div1 ((uint32_t)0x00000000)***

- #define: ***RCC_HCLK_Div2 ((uint32_t)0x00000400)***

- #define: ***RCC_HCLK_Div4 ((uint32_t)0x00000500)***

- #define: ***RCC_HCLK_Div8 ((uint32_t)0x00000600)***

- #define: ***RCC_HCLK_Div16 ((uint32_t)0x00000700)***

RCC_APB1_Peripherals

- #define: ***RCC_APB1Periph_TIM2 ((uint32_t)0x00000001)***

- #define: ***RCC_APB1Periph_TIM3 ((uint32_t)0x00000002)***
- #define: ***RCC_APB1Periph_TIM4 ((uint32_t)0x00000004)***
- #define: ***RCC_APB1Periph_TIM6 ((uint32_t)0x00000010)***
- #define: ***RCC_APB1Periph_TIM7 ((uint32_t)0x00000020)***
- #define: ***RCC_APB1Periph_WWDG ((uint32_t)0x00000800)***
- #define: ***RCC_APB1Periph_SPI2 ((uint32_t)0x00004000)***
- #define: ***RCC_APB1Periph_SPI3 ((uint32_t)0x00008000)***
- #define: ***RCC_APB1Periph_USART2 ((uint32_t)0x00020000)***
- #define: ***RCC_APB1Periph_USART3 ((uint32_t)0x00040000)***
- #define: ***RCC_APB1Periph_UART4 ((uint32_t)0x00080000)***
- #define: ***RCC_APB1Periph_UART5 ((uint32_t)0x00100000)***
- #define: ***RCC_APB1Periph_I2C1 ((uint32_t)0x00200000)***

- #define: ***RCC_APB1Periph_I2C2 ((uint32_t)0x00400000)***

- #define: ***RCC_APB1Periph_USB ((uint32_t)0x00800000)***

- #define: ***RCC_APB1Periph_CAN1 ((uint32_t)0x02000000)***

- #define: ***RCC_APB1Periph_PWR ((uint32_t)0x10000000)***

- #define: ***RCC_APB1Periph_DAC ((uint32_t)0x20000000)***

RCC_APB2_Peripherals

- #define: ***RCC_APB2Periph_SYSCFG ((uint32_t)0x00000001)***

- #define: ***RCC_APB2Periph_TIM1 ((uint32_t)0x00000800)***

- #define: ***RCC_APB2Periph_SPI1 ((uint32_t)0x00001000)***

- #define: ***RCC_APB2Periph_TIM8 ((uint32_t)0x00002000)***

- #define: ***RCC_APB2Periph_USART1 ((uint32_t)0x00004000)***

- #define: ***RCC_APB2Periph_TIM15 ((uint32_t)0x00010000)***

- #define: ***RCC_APB2Periph_TIM16 ((uint32_t)0x00020000)***

- #define: ***RCC_APB2Periph_TIM17 ((uint32_t)0x00040000)***

RCC_Flag

- #define: ***RCC_FLAG_HSIRDY ((uint8_t)0x01)***
- #define: ***RCC_FLAG_HSERDY ((uint8_t)0x11)***
- #define: ***RCC_FLAG_PLLRDY ((uint8_t)0x19)***
- #define: ***RCC_FLAG_MCOF ((uint8_t)0x9C)***
- #define: ***RCC_FLAG_LSERDY ((uint8_t)0x21)***
- #define: ***RCC_FLAG_LSIRDY ((uint8_t)0x41)***
- #define: ***RCC_FLAG_OBLRST ((uint8_t)0x59)***
- #define: ***RCC_FLAG_PINRST ((uint8_t)0x5A)***
- #define: ***RCC_FLAG_PORRST ((uint8_t)0x5B)***
- #define: ***RCC_FLAG_SFTRST ((uint8_t)0x5C)***

- #define: ***RCC_FLAG_IWDGRST ((uint8_t)0x5D)***

- #define: ***RCC_FLAG_WWDGRST ((uint8_t)0x5E)***

- #define: ***RCC_FLAG_LPWRRST ((uint8_t)0x5F)***

RCC_HSE_configuration

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

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

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

RCC_I2C_clock_source

- #define: ***RCC_I2C1CLK_HSI ((uint32_t)0x00000000)***

- #define: ***RCC_I2C1CLK_SYSCLK RCC_CFGR3_I2C1SW***

- #define: ***RCC_I2C2CLK_HSI ((uint32_t)0x10000000)***

- #define: ***RCC_I2C2CLK_SYSCLK ((uint32_t)0x10000020)***

RCC_I2S_Clock_Source

- #define: ***RCC_I2S2CLKSource_SYSCLK ((uint8_t)0x00)***

- #define: ***RCC_I2S2CLKSource_Ext ((uint8_t)0x01)***

RCC_Interrupt_Source

- #define: ***RCC_IT_LSIRDY ((uint8_t)0x01)***
- #define: ***RCC_IT_LSERDY ((uint8_t)0x02)***
- #define: ***RCC_IT_HSIRDY ((uint8_t)0x04)***
- #define: ***RCC_IT_HSERDY ((uint8_t)0x08)***
- #define: ***RCC_IT_PLLRDY ((uint8_t)0x10)***
- #define: ***RCC_IT_CSS ((uint8_t)0x80)***

RCC_LSE_configuration

- #define: ***RCC_LSE_OFF ((uint32_t)0x00000000)***
- #define: ***RCC_LSE_ON RCC_BDCR_LSEON***
- #define: ***RCC_LSE_Bypass ((uint32_t)(RCC_BDCR_LSEON | RCC_BDCR_LSEBYP))***

RCC_LSE_Drive_Configuration

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

- #define: ***RCC_LSEDrive_MediumLow RCC_BDCR_LSEDRV_0***

- #define: ***RCC_LSEDrive_MediumHigh RCC_BDCR_LSEDRV_1***

- #define: ***RCC_LSEDrive_High RCC_BDCR_LSEDRV***

RCC_MCO_Clock_Source

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

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

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

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

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

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

- #define: ***RCC_MCOSource_PLLCLK_Div2 ((uint8_t)0x07)***

RCC_PLL_Clock_Source

- #define: ***RCC_PLLSource_HSI_Div2 RCC_CFGR_PLLSRC_HSI_Div2***

- #define: ***RCC_PLLSource_PREDIV1 RCC_CFGR_PLLSRC_PREDIV1***

RCC_PLL_Multiplication_Factor

- #define: ***RCC_PLLMul_2 RCC_CFGR_PLLMULL2***
- #define: ***RCC_PLLMul_3 RCC_CFGR_PLLMULL3***
- #define: ***RCC_PLLMul_4 RCC_CFGR_PLLMULL4***
- #define: ***RCC_PLLMul_5 RCC_CFGR_PLLMULL5***
- #define: ***RCC_PLLMul_6 RCC_CFGR_PLLMULL6***
- #define: ***RCC_PLLMul_7 RCC_CFGR_PLLMULL7***
- #define: ***RCC_PLLMul_8 RCC_CFGR_PLLMULL8***
- #define: ***RCC_PLLMul_9 RCC_CFGR_PLLMULL9***
- #define: ***RCC_PLLMul_10 RCC_CFGR_PLLMULL10***
- #define: ***RCC_PLLMul_11 RCC_CFGR_PLLMULL11***
- #define: ***RCC_PLLMul_12 RCC_CFGR_PLLMULL12***

- #define: ***RCC_PLLMul_13 RCC_CFGR_PLLMULL13***
- #define: ***RCC_PLLMul_14 RCC_CFGR_PLLMULL14***
- #define: ***RCC_PLLMul_15 RCC_CFGR_PLLMULL15***
- #define: ***RCC_PLLMul_16 RCC_CFGR_PLLMULL16***

RCC_PREDIV1_division_factor

- #define: ***RCC_PREDIV1_Div1 RCC_CFGR2_PREDIV1_DIV1***
- #define: ***RCC_PREDIV1_Div2 RCC_CFGR2_PREDIV1_DIV2***
- #define: ***RCC_PREDIV1_Div3 RCC_CFGR2_PREDIV1_DIV3***
- #define: ***RCC_PREDIV1_Div4 RCC_CFGR2_PREDIV1_DIV4***
- #define: ***RCC_PREDIV1_Div5 RCC_CFGR2_PREDIV1_DIV5***
- #define: ***RCC_PREDIV1_Div6 RCC_CFGR2_PREDIV1_DIV6***
- #define: ***RCC_PREDIV1_Div7 RCC_CFGR2_PREDIV1_DIV7***
- #define: ***RCC_PREDIV1_Div8 RCC_CFGR2_PREDIV1_DIV8***

- #define: ***RCC_PREDIV1_Div9 RCC_CFGR2_PREDIV1_DIV9***
- #define: ***RCC_PREDIV1_Div10 RCC_CFGR2_PREDIV1_DIV10***
- #define: ***RCC_PREDIV1_Div11 RCC_CFGR2_PREDIV1_DIV11***
- #define: ***RCC_PREDIV1_Div12 RCC_CFGR2_PREDIV1_DIV12***
- #define: ***RCC_PREDIV1_Div13 RCC_CFGR2_PREDIV1_DIV13***
- #define: ***RCC_PREDIV1_Div14 RCC_CFGR2_PREDIV1_DIV14***
- #define: ***RCC_PREDIV1_Div15 RCC_CFGR2_PREDIV1_DIV15***
- #define: ***RCC_PREDIV1_Div16 RCC_CFGR2_PREDIV1_DIV16***

RCC_RTC_Clock_Source

- #define: ***RCC_RTCCLKSource_LSE RCC_BDCR_RTCSEL_LSE***
- #define: ***RCC_RTCCLKSource_LSI RCC_BDCR_RTCSEL_LSI***
- #define: ***RCC_RTCCLKSource_HSE_Div32 RCC_BDCR_RTCSEL_HSE***

RCC_System_Clock_Source

- #define: ***RCC_SYSCLKSource_HSI RCC_CFGR_SW_HSI***

- #define: ***RCC_SYSCLKSource_HSE RCC_CFGR_SW_HSE***

- #define: ***RCC_SYSCLKSource_PLLCLK RCC_CFGR_SW_PLL***

RCC_TIM_clock_source

- #define: ***RCC_TIM1CLK_HCLK ((uint32_t)0x00000000)***

- #define: ***RCC_TIM1CLK_PLLCLK RCC_CFGR3_TIM1SW***

- #define: ***RCC_TIM8CLK_HCLK ((uint32_t)0x10000000)***

- #define: ***RCC_TIM8CLK_PLLCLK ((uint32_t)0x10000200)***

RCC_USART_clock_source

- #define: ***RCC_USART1CLK_PCLK ((uint32_t)0x10000000)***

- #define: ***RCC_USART1CLK_SYSCLK ((uint32_t)0x10000001)***

- #define: ***RCC_USART1CLK_LSE ((uint32_t)0x10000002)***

- #define: ***RCC_USART1CLK_HSI ((uint32_t)0x10000003)***

- #define: ***RCC_USART2CLK_PCLK ((uint32_t)0x20000000)***

- #define: ***RCC_USART2CLK_SYSCLK ((uint32_t)0x20010000)***
- #define: ***RCC_USART2CLK_LSE ((uint32_t)0x20020000)***
- #define: ***RCC_USART2CLK_HSI ((uint32_t)0x20030000)***
- #define: ***RCC_USART3CLK_PCLK ((uint32_t)0x30000000)***
- #define: ***RCC_USART3CLK_SYSCLK ((uint32_t)0x30040000)***
- #define: ***RCC_USART3CLK_LSE ((uint32_t)0x30080000)***
- #define: ***RCC_USART3CLK_HSI ((uint32_t)0x300C0000)***
- #define: ***RCC_UART4CLK_PCLK ((uint32_t)0x40000000)***
- #define: ***RCC_UART4CLK_SYSCLK ((uint32_t)0x40100000)***
- #define: ***RCC_UART4CLK_LSE ((uint32_t)0x40200000)***
- #define: ***RCC_UART4CLK_HSI ((uint32_t)0x40300000)***
- #define: ***RCC_UART5CLK_PCLK ((uint32_t)0x50000000)***

- #define: ***RCC_UART5CLK_SYSCLK ((uint32_t)0x50400000)***

- #define: ***RCC_UART5CLK_LSE ((uint32_t)0x50800000)***

- #define: ***RCC_UART5CLK_HSI ((uint32_t)0x50C00000)***

RCC_USB_Device_clock_source

- #define: ***RCC_USBCLKSource_PLLCLK_1Div5 ((uint8_t)0x00)***

- #define: ***RCC_USBCLKSource_PLLCLK_Div1 ((uint8_t)0x01)***

19 Real-time clock (RTC)

19.1 RTC Firmware driver registers structures

19.1.1 RTC_TypeDef

RTC_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t TR***
- ***__IO uint32_t DR***
- ***__IO uint32_t CR***
- ***__IO uint32_t ISR***
- ***__IO uint32_t PRER***
- ***__IO uint32_t WUTR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t ALRMAR***
- ***__IO uint32_t ALRMBR***
- ***__IO uint32_t WPR***
- ***__IO uint32_t SSR***
- ***__IO uint32_t SHIFTR***
- ***__IO uint32_t TSTR***
- ***__IO uint32_t TSDR***
- ***__IO uint32_t TSSSR***
- ***__IO uint32_t CALR***
- ***__IO uint32_t TAFCR***
- ***__IO uint32_t ALRMASSR***
- ***__IO uint32_t ALRMBSSR***
- ***uint32_t RESERVED7***
- ***__IO uint32_t BKP0R***
- ***__IO uint32_t BKP1R***
- ***__IO uint32_t BKP2R***
- ***__IO uint32_t BKP3R***
- ***__IO uint32_t BKP4R***
- ***__IO uint32_t BKP5R***
- ***__IO uint32_t BKP6R***
- ***__IO uint32_t BKP7R***
- ***__IO uint32_t BKP8R***
- ***__IO uint32_t BKP9R***
- ***__IO uint32_t BKP10R***
- ***__IO uint32_t BKP11R***
- ***__IO uint32_t BKP12R***
- ***__IO uint32_t BKP13R***
- ***__IO uint32_t BKP14R***
- ***__IO uint32_t BKP15R***

Field Documentation

- ***__IO uint32_t RTC_TypeDef::TR***
  - RTC time register, Address offset: 0x00
- ***__IO uint32_t RTC_TypeDef::DR***
  - RTC date register, Address offset: 0x04
- ***__IO uint32_t RTC_TypeDef::CR***
  - RTC control register, Address offset: 0x08
- ***__IO uint32_t RTC_TypeDef::ISR***
  - RTC initialization and status register, Address offset: 0x0C
- ***__IO uint32_t RTC_TypeDef::PRER***
  - RTC prescaler register, Address offset: 0x10
- ***__IO uint32_t RTC_TypeDef::WUTR***
  - RTC wakeup timer register, Address offset: 0x14
- ***uint32_t RTC_TypeDef::RESERVED0***
  - Reserved, 0x18
- ***__IO uint32_t RTC_TypeDef::ALRMAR***
  - RTC alarm A register, Address offset: 0x1C
- ***__IO uint32_t RTC_TypeDef::ALRMBR***
  - RTC alarm B register, Address offset: 0x20
- ***__IO uint32_t RTC_TypeDef::WPR***
  - RTC write protection register, Address offset: 0x24
- ***__IO uint32_t RTC_TypeDef::SSR***
  - RTC sub second register, Address offset: 0x28
- ***__IO uint32_t RTC_TypeDef::SHIFTR***
  - RTC shift control register, Address offset: 0x2C
- ***__IO uint32_t RTC_TypeDef::TSTR***
  - RTC time stamp time register, Address offset: 0x30
- ***__IO uint32_t RTC_TypeDef::TSDR***
  - RTC time stamp date register, Address offset: 0x34
- ***__IO uint32_t RTC_TypeDef::TSSSR***
  - RTC time-stamp sub second register, Address offset: 0x38
- ***__IO uint32_t RTC_TypeDef::CALR***
  - RTC calibration register, Address offset: 0x3C
- ***__IO uint32_t RTC_TypeDef::TAFCR***
  - RTC tamper and alternate function configuration register, Address offset: 0x40
- ***__IO uint32_t RTC_TypeDef::ALRMASSR***
  - RTC alarm A sub second register, Address offset: 0x44
- ***__IO uint32_t RTC_TypeDef::ALRMBSSR***
  - RTC alarm B sub second register, Address offset: 0x48
- ***uint32_t RTC_TypeDef::RESERVED7***
  - Reserved, 0x4C
- ***__IO uint32_t RTC_TypeDef::BKP0R***
  - RTC backup register 0, Address offset: 0x50
- ***__IO uint32_t RTC_TypeDef::BKP1R***
  - RTC backup register 1, Address offset: 0x54
- ***__IO uint32_t RTC_TypeDef::BKP2R***
  - RTC backup register 2, Address offset: 0x58
- ***__IO uint32_t RTC_TypeDef::BKP3R***
  - RTC backup register 3, Address offset: 0x5C
- ***__IO uint32_t RTC_TypeDef::BKP4R***
  - RTC backup register 4, Address offset: 0x60
- ***__IO uint32_t RTC_TypeDef::BKP5R***
  - RTC backup register 5, Address offset: 0x64

- ***__IO uint32_t RTC_TypeDef::BKP6R***
  - RTC backup register 6, Address offset: 0x68
- ***__IO uint32_t RTC_TypeDef::BKP7R***
  - RTC backup register 7, Address offset: 0x6C
- ***__IO uint32_t RTC_TypeDef::BKP8R***
  - RTC backup register 8, Address offset: 0x70
- ***__IO uint32_t RTC_TypeDef::BKP9R***
  - RTC backup register 9, Address offset: 0x74
- ***__IO uint32_t RTC_TypeDef::BKP10R***
  - RTC backup register 10, Address offset: 0x78
- ***__IO uint32_t RTC_TypeDef::BKP11R***
  - RTC backup register 11, Address offset: 0x7C
- ***__IO uint32_t RTC_TypeDef::BKP12R***
  - RTC backup register 12, Address offset: 0x80
- ***__IO uint32_t RTC_TypeDef::BKP13R***
  - RTC backup register 13, Address offset: 0x84
- ***__IO uint32_t RTC_TypeDef::BKP14R***
  - RTC backup register 14, Address offset: 0x88
- ***__IO uint32_t RTC_TypeDef::BKP15R***
  - RTC backup register 15, Address offset: 0x8C

19.1.2 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32f30x_rtc.h

Data Fields

- ***uint32_t RTC_HourFormat***
- ***uint32_t RTC_AsynchPrediv***
- ***uint32_t RTC_SynchPrediv***

Field Documentation

- ***uint32_t RTC_InitTypeDef::RTC_HourFormat***
  - Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats***
- ***uint32_t RTC_InitTypeDef::RTC_AsynchPrediv***
  - Specifies the RTC Asynchronous Predivider value. This parameter must be set to a value lower than 0x7F
- ***uint32_t RTC_InitTypeDef::RTC_SynchPrediv***
  - Specifies the RTC Synchronous Predivider value. This parameter must be set to a value lower than 0x1FFF

19.1.3 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32f30x_rtc.h

Data Fields

- ***uint8_t RTC_Hours***
- ***uint8_t RTC_Minutes***
- ***uint8_t RTC_Seconds***
- ***uint8_t RTC_H12***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::RTC_Hours***
  - Specifies the RTC Time Hour. This parameter must be set to a value in the 0-12 range if the RTC_HourFormat_12 is selected or 0-23 range if the RTC_HourFormat_24 is selected.
- ***uint8_t RTC_TimeTypeDef::RTC_Minutes***
  - Specifies the RTC Time Minutes. This parameter must be set to a value in the 0-59 range.
- ***uint8_t RTC_TimeTypeDef::RTC_Seconds***
  - Specifies the RTC Time Seconds. This parameter must be set to a value in the 0-59 range.
- ***uint8_t RTC_TimeTypeDef::RTC_H12***
  - Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***

19.1.4 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32f30x_rtc.h

Data Fields

- ***uint8_t RTC_WeekDay***
- ***uint8_t RTC_Month***
- ***uint8_t RTC_Date***
- ***uint8_t RTC_Year***

Field Documentation

- ***uint8_t RTC_DateTypeDef::RTC_WeekDay***
  - Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint8_t RTC_DateTypeDef::RTC_Month***
  - Specifies the RTC Date Month (in BCD format). This parameter can be a value of ***RTC_Month_Date_Definitions***
- ***uint8_t RTC_DateTypeDef::RTC_Date***
  - Specifies the RTC Date. This parameter must be set to a value in the 1-31 range.
- ***uint8_t RTC_DateTypeDef::RTC_Year***
  - Specifies the RTC Date Year. This parameter must be set to a value in the 0-99 range.

19.1.5 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32f30x_rtc.h

Data Fields

- ***RTC_TimeTypeDef RTC_AlarmTime***
- ***uint32_t RTC_AlarmMask***
- ***uint32_t RTC_AlarmDateWeekDaySel***
- ***uint8_t RTC_AlarmDateWeekDay***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::RTC_AlarmTime***
  - Specifies the RTC Alarm Time members.
- ***uint32_t RTC_AlarmTypeDef::RTC_AlarmMask***
  - Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::RTC_AlarmDateWeekDaySel***
  - Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::RTC_AlarmDateWeekDay***
  - Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***

19.2 RTC Firmware driver API description

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

19.2.1 RTC Operating Condition

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

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

1. The RTC
2. The LSE oscillator
3. PC13 to PC15 I/Os (when available)

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

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

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

1. PC14 and PC15 can be used as LSE pins only
2. PC13 can be used as the RTC_AF pin

19.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

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

19.2.3 Backup Domain Access

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

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

1. Enable the Power Controller (PWR) APB1 interface clock using the RCC_APB1PeriphClockCmd() function.
2. Enable access to Backup domain using the PWR_BackupAccessCmd() function.
3. Select the RTC clock source using the RCC_RTCCLKConfig() function.
4. Enable RTC Clock using the RCC_RTCCLKCmd() function.

19.2.4 How to use this driver

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

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the RTC_SetTime() and RTC_SetDate() functions.
- To read the RTC Calendar, use the RTC_GetTime() and RTC_GetDate() functions.
- To read the RTC subsecond, use the RTC_GetSubSecond() function.
- Use the RTC_DayLightSavingConfig() function to add or sub one hour to the RTC Calendar.

Alarm configuration

- To configure the RTC Alarm use the RTC_SetAlarm() function.
- Enable the selected RTC Alarm using the RTC_AlarmCmd() function.
- To read the RTC Alarm, use the RTC_GetAlarm() function.
- To read the RTC alarm SubSecond, use the RTC_GetAlarmSubSecond() function.

RTC Wakeup configuration

- Configure the RTC Wakeup Clock source use the RTC_WakeUpClockConfig() function.
- Configure the RTC WakeUp Counter using the RTC_SetWakeUpCounter() function
- Enable the RTC WakeUp using the RTC_WakeUpCmd() function
- To read the RTC WakeUp Counter register, use the RTC_GetWakeUpCounter() function.

Outputs configuration

The RTC has 2 different outputs:

- AFO_ALARM: this output is used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal on RTC_AF pin, use the RTC_OutputConfig() function.
- AFO_CALIB: this output is 512Hz signal or 1Hz . To output the RTC Clock on RTC_AF pin, use the RTC_CalibOutputCmd() function.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the RTC_SmoothCalibConfig() function.

TimeStamp configuration

- Configure the RTC_AF trigger and enables the RTC TimeStamp using the RTC_TimeStampCmd() function.
- To read the RTC TimeStamp Time and Date register, use the RTC_GetTimeStamp() function.
- To read the RTC TimeStamp SubSecond register, use the RTC_GetTimeStampSubSecond() function.

Tamper configuration

- Configure the Tamper filter count using RTC_TamperFilterConfig() function.
- Configure the RTC Tamper trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value using the RTC_TamperConfig() function.
- Configure the Tamper sampling frequency using RTC_TamperSamplingFreqConfig() function.
- Configure the Tamper precharge or discharge duration using RTC_TamperPinsPrechargeDuration() function.
- Enable the Tamper Pull-UP using RTC_TamperPullUpDisableCmd() function.
- Enable the RTC Tamper using the RTC_TamperCmd() function.
- Enable the Time stamp on Tamper detection event using RTC_TSOnTamperDetecCmd() function.

Backup Data Registers configuration

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

19.2.5 RTC and low power modes

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

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby lowpower modes. The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

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

19.2.6 Selection of RTC_AF alternate functions

The RTC_AF pin (PC13) can be used for the following purposes:

- Wakeup pin 2 (WKUP2) using the PWR_WakeUpPinCmd() function.
- AFO_ALARM output
- AFO_CALIB output
- AFI_TAMPER
- AFI_TIMESTAMP

Table 15: Selection of RTC_AF alternate functions

Pin configuration and functions	RTC_ALARM output enabled	RTC_CALIB output enabled	RTC_TAMPER input enabled	RTC_TIMESTAMP input enabled	PC13MODE bit	PC13VALUE bit
Alarm out output OD	1	Don't care	Don't care	Don't care	Don't care	0
Alarm out output PP	1	Don't care	Don't care	Don't care	Don't care	1
Calibration out output PP	0	1	Don't care	Don't care	Don't care	Don't care
TAMPER input floating	0	0	1	0	Don't care	Don't care
TIMESTAMP and TAMPER input floating	0	0	1	1	Don't care	Don't care
TIMESTAMP input floating	0	0	0	1	Don't care	Don't care
Output PP forced	0	0	0	0	1	PC13 output
Wakeup pin or Standard GPIO	0	0	0	0	0	Don't care

19.2.7 Initialization and Configuration functions

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

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   - A 7-bit asynchronous prescaler and A 13-bit synchronous prescaler.
   - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To Configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

- ***RTC_DeInit()***
- ***RTC_Init()***
- ***RTC_StructInit()***
- ***RTC_WriteProtectionCmd()***
- ***RTC_EnterInitMode()***
- ***RTC_ExitInitMode()***
- ***RTC_WaitForSynchro()***
- ***RTC_RefClockCmd()***
- ***RTC_BypassShadowCmd()***

19.2.8 Backup Data Registers configuration functions

- ***RTC_WriteBackupRegister()***
- ***RTC_ReadBackupRegister()***

19.2.9 Output Type Config configuration functions

- ***RTC_OutputTypeConfig()***

19.2.10 Shift control synchronisation functions

- ***RTC_SynchroShiftConfig()***

19.2.11 Interrupts and flags management functions

All RTC interrupts are connected to the EXTI controller.

- To enable the RTC Alarm interrupt, the following sequence is required:
  - Configure and enable the EXTI Line 17 in interrupt mode and select the rising edge sensitivity using the EXTI_Init() function.
  - Configure and enable the RTC_Alarm IRQ channel in the NVIC using the NVIC_Init() function.
  - Configure the RTC to generate RTC alarms (Alarm A and/or Alarm B) using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
- To enable the RTC Wakeup interrupt, the following sequence is required:
  - Configure and enable the EXTI Line 20 in interrupt mode and select the rising edge sensitivity using the EXTI_Init() function.
  - Configure and enable the RTC_WKUP IRQ channel in the NVIC using the NVIC_Init() function.
  - Configure the RTC to generate the RTC wakeup timer event using the RTC_WakeUpClockConfig(), RTC_SetWakeUpCounter() and RTC_WakeUpCmd() functions.
- To enable the RTC Tamper interrupt, the following sequence is required:
  - Configure and enable the EXTI Line 19 in interrupt mode and select the rising edge sensitivity using the EXTI_Init() function.
  - Configure and enable the TAMP_STAMP IRQ channel in the NVIC using the NVIC_Init() function.
  - Configure the RTC to detect the RTC tamper event using the RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
- To enable the RTC TimeStamp interrupt, the following sequence is required:
  - Configure and enable the EXTI Line 19 in interrupt mode and select the rising edge sensitivity using the EXTI_Init() function.
  - Configure and enable the TAMP_STAMP IRQ channel in the NVIC using the NVIC_Init() function.
  - Configure the RTC to detect the RTC time-stamp event using the RTC_TimeStampCmd() functions.
- ***RTC_ITConfig()***
- ***RTC_GetFlagStatus()***
- ***RTC_ClearFlag()***
- ***RTC_GetITStatus()***
- ***RTC_ClearITPendingBit()***

19.2.12 Time and Date configuration functions

This section provide functions allowing to program and read the RTC Calendar (Time and Date).

- ***RTC_SetTime()***
- ***RTC_TimeStructInit()***
- ***RTC_GetTime()***
- ***RTC_GetSubSecond()***
- ***RTC_SetDate()***
- ***RTC_DateStructInit()***
- ***RTC_GetDate()***

19.2.13 Alarms (Alarm A and Alarm B) configuration functions

This section provides functions allowing to program and read the RTC Alarms.

- *RTC_SetAlarm()*
- *RTC_AlarmStructInit()*
- *RTC_GetAlarm()*
- *RTC_AlarmCmd()*
- *RTC_AlarmSubSecondConfig()*
- *RTC_GetAlarmSubSecond()*

19.2.14 WakeUp Timer configuration functions

This section provide functions allowing to program and read the RTC WakeUp.

- *RTC_WakeUpClockConfig()*
- *RTC_SetWakeUpCounter()*
- *RTC_GetWakeUpCounter()*
- *RTC_WakeUpCmd()*

19.2.15 Daylight Saving configuration functions

This section provide functions allowing to configure the RTC DayLight Saving.

- *RTC_DayLightSavingConfig()*
- *RTC_GetStoreOperation()*

19.2.16 Output pin Configuration function

This section provide functions allowing to configure the RTC Output source.

- *RTC_OutputConfig()*

19.2.17 Digital Calibration configuration functions

- *RTC_CalibOutputCmd()*
- *RTC_CalibOutputConfig()*
- *RTC_SmoothCalibConfig()*

19.2.18 TimeStamp configuration functions

- *RTC_TimeStampCmd()*
- *RTC_GetTimeStamp()*
- *RTC_GetTimeStampSubSecond()*

19.2.19 Tampers configuration functions

- *RTC_TamperTriggerConfig()*
- *RTC_TamperCmd()*
- *RTC_TamperFilterConfig()*
- *RTC_TamperSamplingFreqConfig()*

- *RTC_TamperPinsPrechargeDuration()*
- *RTC_TimeStampOnTamperDetectionCmd()*
- *RTC_TamperPullUpCmd()*

19.2.20 Initialization and Configuration functions

19.2.20.1 RTC_DeInit

Function Name	**ErrorStatus RTC_DeInit (void)**
Function Description	Deinitializes the RTC registers to their default reset values.
Parameters	• None.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: RTC registers are deinitialized*** – ***ERROR: RTC registers are not deinitialized***
Notes	• This function doesn't reset the RTC Clock source and RTC Backup Data registers.

19.2.20.2 RTC_Init

Function Name	**ErrorStatus RTC_Init (*RTC_InitTypeDef** * RTC_InitStruct)**
Function Description	Initializes the RTC registers according to the specified parameters in RTC_InitStruct.
Parameters	• **RTC_InitStruct :** pointer to a RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: RTC registers are initialized*** – ***ERROR: RTC registers are not initialized***
Notes	• The RTC Prescaler register is write protected and can be written in initialization mode only.

19.2.20.3 RTC_StructInit

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

19.2.20.4 RTC_WriteProtectionCmd

Function Name	**void RTC_WriteProtectionCmd (*FunctionalState* NewState)**
Function Description	Enables or disables the RTC registers write protection.
Parameters	• **NewState :** new state of the write protection. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• All the RTC registers are write protected except for RTC_ISR[13:8], RTC_TAFCR and RTC_BKPxR. • Writing a wrong key reactivates the write protection. • The protection mechanism is not affected by system reset.

19.2.20.5 RTC_EnterInitMode

Function Name	**ErrorStatus RTC_EnterInitMode (void)**
Function Description	Enters the RTC Initialization mode.
Parameters	• None.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: RTC is in Init mode*** – ***ERROR: RTC is not in Init mode***
Notes	• The RTC Initialization mode is write protected, use the RTC_WriteProtectionCmd(DISABLE) before calling this function.

19.2.20.6 RTC_ExitInitMode

Function Name	**void RTC_ExitInitMode (void)**
Function Description	Exits the RTC Initialization mode.
Parameters	• None.
Return values	• None.
Notes	• When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles. • The RTC Initialization mode is write protected, use the RTC_WriteProtectionCmd(DISABLE) before calling this function.

19.2.20.7 RTC_WaitForSynchro

Function Name	**ErrorStatus RTC_WaitForSynchro (void)**
Function Description	Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	• None.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: RTC registers are synchronised*** – ***ERROR: RTC registers are not synchronised***
Notes	• The RTC Resynchronization mode is write protected, use the RTC_WriteProtectionCmd(DISABLE) before calling this function. • To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

19.2.20.8 RTC_RefClockCmd

Function Name	**ErrorStatus RTC_RefClockCmd (*FunctionalState* NewState)**
Function Description	Enables or disables the RTC reference clock detection.
Parameters	• **NewState :** new state of the RTC reference clock. This parameter can be: ENABLE or DISABLE.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: RTC reference clock detection is enabled*** – ***ERROR: RTC reference clock detection is disabled***
Notes	• None.

19.2.20.9 RTC_BypassShadowCmd

Function Name	**void RTC_BypassShadowCmd (*FunctionalState* NewState)**
Function Description	Enables or Disables the Bypass Shadow feature.
Parameters	• **NewState :** new state of the Bypass Shadow feature. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

19.2.21 Backup Data Registers configuration functions

19.2.21.1 RTC_WriteBackupRegister

Function Name	**void RTC_WriteBackupRegister (uint32_t RTC_BKP_DR, uint32_t Data)**
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	• **RTC_BKP_DR :** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.

- **Data :** Data to be written in the specified RTC Backup data register.

Return values
- None.

Notes
- None.

19.2.21.2 RTC_ReadBackupRegister

Function Name **uint32_t RTC_ReadBackupRegister (uint32_t RTC_BKP_DR)**

Function Description Reads data from the specified RTC Backup data Register.

Parameters
- **RTC_BKP_DR :** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.

Return values
- None.

Notes
- None.

19.2.22 Output Type Config configuration functions

19.2.22.1 RTC_OutputTypeConfig

Function Name **void RTC_OutputTypeConfig (uint32_t RTC_OutputType)**

Function Description Configures the RTC Output Pin mode.

Parameters
- **RTC_OutputType :** specifies the RTC Output (PC13) pin mode. This parameter can be one of the following values:
  - ***RTC_OutputType_OpenDrain :*** RTC Output (PC13) is configured in Open Drain mode.
  - ***RTC_OutputType_PushPull :*** RTC Output (PC13) is configured in Push Pull mode.

Return values
- None.

Notes
- None.

19.2.23 Shift control synchronisation functions

19.2.23.1 RTC_SynchroShiftConfig

Function Name	**ErrorStatus RTC_SynchroShiftConfig (uint32_t RTC_ShiftAdd1S, uint32_t RTC_ShiftSubFS)**
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	• **RTC_ShiftAdd1S :** : Select to add or not 1 second to the time Calendar. This parameter can be one of the following values : – ***RTC_ShiftAdd1S_Set :*** Add one second to the clock calendar. – ***RTC_ShiftAdd1S_Reset :*** No effect. • **RTC_ShiftSubFS :** Select the number of Second Fractions to Substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: RTC Shift registers are configured*** – ***ERROR: RTC Shift registers are not configured***
Notes	• When REFCKON is set, firmware must not write to Shift control register

19.2.24 Interrupts and flags management functions

19.2.24.1 RTC_ITConfig

Function Name	**void RTC_ITConfig (uint32_t RTC_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified RTC interrupts.
Parameters	• **RTC_IT :** specifies the RTC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values: – ***RTC_IT_TS :*** Time Stamp interrupt mask – ***RTC_IT_WUT :*** WakeUp Timer interrupt mask – ***RTC_IT_ALRB :*** Alarm B interrupt mask – ***RTC_IT_ALRA :*** Alarm A interrupt mask – ***RTC_IT_TAMP :*** Tamper event interrupt mask • **NewState :** new state of the specified RTC interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

19.2.24.2 RTC_GetFlagStatus

Function Name	**FlagStatus RTC_GetFlagStatus (uint32_t RTC_FLAG)**
Function Description	Checks whether the specified RTC flag is set or not.
Parameters	• **RTC_FLAG :** specifies the flag to check. This parameter can be one of the following values: – ***RTC_FLAG_RECALPF :*** RECALPF event flag – ***RTC_FLAG_TAMP3F :*** Tamper 3 event flag – ***RTC_FLAG_TAMP2F :*** Tamper 2 event flag – ***RTC_FLAG_TAMP1F :*** Tamper 1 event flag – ***RTC_FLAG_TSOVF :*** Time Stamp OverFlow flag – ***RTC_FLAG_TSF :*** Time Stamp event flag – ***RTC_FLAG_WUTF :*** WakeUp Timer flag – ***RTC_FLAG_ALRBF :*** Alarm B flag – ***RTC_FLAG_ALRAF :*** Alarm A flag – ***RTC_FLAG_INITF :*** Initialization mode flag – ***RTC_FLAG_RSF :*** Registers Synchronized flag – ***RTC_FLAG_INITS :*** Registers Configured flag : Shift operation pending flag. – ***RTC_FLAG_WUTWF :*** WakeUp Timer Write flag – ***RTC_FLAG_ALRBWF :*** Alarm B Write flag – ***RTC_FLAG_ALRAWF :*** Alarm A write flag
Return values	• **The new state of RTC_FLAG (SET or RESET).**
Notes	• None.

19.2.24.3 RTC_ClearFlag

Function Name	**void RTC_ClearFlag (uint32_t RTC_FLAG)**
Function Description	Clears the RTC's pending flags.
Parameters	• **RTC_FLAG :** specifies the RTC flag to clear. This parameter can be any combination of the following values: – ***RTC_FLAG_TAMP3F :*** Tamper 3 event flag – ***RTC_FLAG_TAMP2F :*** Tamper 2 event flag – ***RTC_FLAG_TAMP1F :*** Tamper 1 event flag – ***RTC_FLAG_TSOVF :*** Time Stamp Overflow flag – ***RTC_FLAG_TSF :*** Time Stamp event flag

- ***RTC_FLAG_WUTF :*** WakeUp Timer flag
- ***RTC_FLAG_ALRBF :*** Alarm B flag
- ***RTC_FLAG_ALRAF :*** Alarm A flag
- ***RTC_FLAG_RSF :*** Registers Synchronized flag

Return values
- None.

Notes
- None.

19.2.24.4 RTC_GetITStatus

Function Name **ITStatus RTC_GetITStatus (uint32_t RTC_IT)**

Function Description Checks whether the specified RTC interrupt has occurred or not.

Parameters
- **RTC_IT :** specifies the RTC interrupt source to check. This parameter can be one of the following values:
  - ***RTC_IT_TS :*** Time Stamp interrupt
  - ***RTC_IT_WUT :*** WakeUp Timer interrupt
  - ***RTC_IT_ALRB :*** Alarm B interrupt
  - ***RTC_IT_ALRA :*** Alarm A interrupt
  - ***RTC_IT_TAMP1 :*** Tamper1 event interrupt
  - ***RTC_IT_TAMP2 :*** Tamper2 event interrupt
  - ***RTC_IT_TAMP3 :*** Tamper3 event interrupt

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

Notes
- None.

19.2.24.5 RTC_ClearITPendingBit

Function Name **void RTC_ClearITPendingBit (uint32_t RTC_IT)**

Function Description Clears the RTC's interrupt pending bits.

Parameters
- **RTC_IT :** specifies the RTC interrupt pending bit to clear. This parameter can be any combination of the following values:
  - ***RTC_IT_TS :*** Time Stamp interrupt
  - ***RTC_IT_WUT :*** WakeUp Timer interrupt
  - ***RTC_IT_ALRB :*** Alarm B interrupt
  - ***RTC_IT_ALRA :*** Alarm A interrupt
  - ***RTC_IT_TAMP1 :*** Tamper1 event interrupt
  - ***RTC_IT_TAMP2 :*** Tamper2 event interrupt

– ***RTC_IT_TAMP3 :*** Tamper3 event interrupt

Return values
- None.

Notes
- None.

19.2.25 Time and Date configuration functions

19.2.25.1 RTC_SetTime

Function Name
ErrorStatus RTC_SetTime (uint32_t RTC_Format, *RTC_TimeTypeDef RTC_TimeStruct)**

Function Description
Set the RTC current time.

Parameters
- **RTC_Format :** specifies the format of the entered parameters. This parameter can be one of the following values:
  – ***RTC_Format_BIN :*** Binary data format
  – ***RTC_Format_BCD :*** BCD data format
- **RTC_TimeStruct :** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.

Return values
- **An ErrorStatus enumeration value:**
  – ***SUCCESS: RTC Time register is configured***
  – ***ERROR: RTC Time register is not configured***

Notes
- None.

19.2.25.2 RTC_TimeStructInit

Function Name
void RTC_TimeStructInit (*RTC_TimeTypeDef RTC_TimeStruct)**

Function Description
Fills each RTC_TimeStruct member with its default value (Time = 00h:00min:00sec).

Parameters
- **RTC_TimeStruct :** pointer to a RTC_TimeTypeDef structure which will be initialized.

Return values
- None.

Notes
- None.

19.2.25.3 RTC_GetTime

Function Name	**void RTC_GetTime (uint32_t RTC_Format, *RTC_TimeTypeDef ** RTC_TimeStruct)**
Function Description	Get the RTC current Time.
Parameters	• **RTC_Format :** specifies the format of the returned parameters. This parameter can be one of the following values: – ***RTC_Format_BIN :*** Binary data format – ***RTC_Format_BCD :*** BCD data format • **RTC_TimeStruct :** pointer to a RTC_TimeTypeDef structure that will contain the returned current time configuration.
Return values	• None.
Notes	• None.

19.2.25.4 RTC_GetSubSecond

Function Name	**uint32_t RTC_GetSubSecond (void)**
Function Description	Gets the RTC current Calendar Subseconds value.
Parameters	• None.
Return values	• **RTC current Calendar Subseconds value.**
Notes	• This function freeze the Time and Date registers after reading the SSR register.

19.2.25.5 RTC_SetDate

Function Name	**ErrorStatus RTC_SetDate (uint32_t RTC_Format, *RTC_DateTypeDef ** RTC_DateStruct)**
Function Description	Set the RTC current date.

Parameters
- **RTC_Format :** specifies the format of the entered parameters. This parameter can be one of the following values:
  - ***RTC_Format_BIN :*** Binary data format
  - ***RTC_Format_BCD :*** BCD data format
- **RTC_DateStruct :** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values
- **An ErrorStatus enumeration value:**
  - ***SUCCESS: RTC Date register is configured***
  - ***ERROR: RTC Date register is not configured***

Notes
- None.

19.2.25.6 RTC_DateStructInit

Function Name **void RTC_DateStructInit (*RTC_DateTypeDef** RTC_DateStruct)**

Function Description Fills each RTC_DateStruct member with its default value (Monday, January 01 xx00).

Parameters
- **RTC_DateStruct :** pointer to a RTC_DateTypeDef structure which will be initialized.

Return values
- None.

Notes
- None.

19.2.25.7 RTC_GetDate

Function Name **void RTC_GetDate (uint32_t RTC_Format, *RTC_DateTypeDef** RTC_DateStruct)**

Function Description Get the RTC current date.

Parameters
- **RTC_Format :** specifies the format of the returned parameters. This parameter can be one of the following values:
  - ***RTC_Format_BIN :*** Binary data format
  - ***RTC_Format_BCD :*** BCD data format
- **RTC_DateStruct :** pointer to a RTC_DateTypeDef structure that will contain the returned current date configuration.

Return values	• None.
Notes	• None.

19.2.26 Alarm configuration functions

19.2.26.1 RTC_SetAlarm

Function Name	**void RTC_SetAlarm (uint32_t RTC_Format, uint32_t RTC_Alarm, *RTC_AlarmTypeDef** * **RTC_AlarmStruct)**
Function Description	Set the specified RTC Alarm.
Parameters	• **RTC_Format :** specifies the format of the returned parameters. This parameter can be one of the following values: – ***RTC_Format_BIN :*** Binary data format – ***RTC_Format_BCD :*** BCD data format • **RTC_Alarm :** specifies the alarm to be configured. This parameter can be one of the following values: – ***RTC_Alarm_A :*** to select Alarm A – ***RTC_Alarm_B :*** to select Alarm B • **RTC_AlarmStruct :** pointer to a RTC_AlarmTypeDef structure that contains the alarm configuration parameters.
Return values	• None.
Notes	• The Alarm register can only be written when the corresponding Alarm is disabled (Use the RTC_AlarmCmd(DISABLE)).

19.2.26.2 RTC_AlarmStructInit

Function Name	**void RTC_AlarmStructInit (*RTC_AlarmTypeDef** * **RTC_AlarmStruct)**
Function Description	Fills each RTC_AlarmStruct member with its default value (Time = 00h:00mn:00sec / Date = 1st day of the month/Mask = all fields are masked).
Parameters	• **RTC_AlarmStruct :** pointer to a RTC_AlarmTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

19.2.26.3 RTC_GetAlarm

Function Name	**void RTC_GetAlarm (uint32_t RTC_Format, uint32_t RTC_Alarm, *RTC_AlarmTypeDef** * **RTC_AlarmStruct)**
Function Description	Get the RTC Alarm value and masks.
Parameters	• **RTC_Format :** specifies the format of the output parameters. This parameter can be one of the following values: – ***RTC_Format_BIN :*** Binary data format – ***RTC_Format_BCD :*** BCD data format • **RTC_Alarm :** specifies the alarm to be read. This parameter can be one of the following values: – ***RTC_Alarm_A :*** to select Alarm A – ***RTC_Alarm_B :*** to select Alarm B • **RTC_AlarmStruct :** pointer to a RTC_AlarmTypeDef structure that will contains the output alarm configuration values.
Return values	• None.
Notes	• None.

19.2.26.4 RTC_AlarmCmd

Function Name	**ErrorStatus RTC_AlarmCmd (uint32_t RTC_Alarm, *FunctionalState* NewState)**
Function Description	Enables or disables the specified RTC Alarm.
Parameters	• **RTC_Alarm :** specifies the alarm to be configured. This parameter can be any combination of the following values: – ***RTC_Alarm_A :*** to select Alarm A – ***RTC_Alarm_B :*** to select Alarm B • **NewState :** new state of the specified alarm. This parameter can be: ENABLE or DISABLE.
Return values	• **An ErrorStatus enumeration value:** – ***SUCCESS: RTC Alarm is enabled/disabled*** – ***ERROR: RTC Alarm is not enabled/disabled***
Notes	• None.

19.2.26.5 RTC_AlarmSubSecondConfig

Function Name	**void RTC_AlarmSubSecondConfig (uint32_t RTC_Alarm, uint32_t RTC_AlarmSubSecondValue, uint32_t RTC_AlarmSubSecondMask)**
Function Description	Configures the RTC AlarmA/B Subseconds value and mask.
Parameters	• **RTC_Alarm :** specifies the alarm to be configured. This parameter can be one of the following values: – ***RTC_Alarm_A :*** to select Alarm A – ***RTC_Alarm_B :*** to select Alarm B • **RTC_AlarmSubSecondValue :** specifies the Subseconds value. This parameter can be a value from 0 to 0x00007FFF. • **RTC_AlarmSubSecondMask :** specifies the Subseconds Mask. This parameter can be any combination of the following values: – ***RTC_AlarmSubSecondMask_All :*** All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm. – ***RTC_AlarmSubSecondMask_SS14_1 :*** SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared – ***RTC_AlarmSubSecondMask_SS14_2 :*** SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared – ***RTC_AlarmSubSecondMask_SS14_3 :*** SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared – ***RTC_AlarmSubSecondMask_SS14_4 :*** SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared – ***RTC_AlarmSubSecondMask_SS14_5 :*** SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared – ***RTC_AlarmSubSecondMask_SS14_6 :*** SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared – ***RTC_AlarmSubSecondMask_SS14_7 :*** SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared – ***RTC_AlarmSubSecondMask_SS14_8 :*** SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared – ***RTC_AlarmSubSecondMask_SS14_9 :*** SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared – ***RTC_AlarmSubSecondMask_SS14_10 :*** SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared – ***RTC_AlarmSubSecondMask_SS14_11 :*** SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared – ***RTC_AlarmSubSecondMask_SS14_12 :*** SS[14:12]

are don't care in Alarm comparison. Only SS[11:0] are compared
- ***RTC_AlarmSubSecondMask_SS14_13 :*** SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
- ***RTC_AlarmSubSecondMask_SS14 :*** SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
- ***RTC_AlarmSubSecondMask_None :*** SS[14:0] are compared and must match to activate alarm

Return values
- None.

Notes
- This function is performed only when the Alarm is disabled.

19.2.26.6 RTC_GetAlarmSubSecond

Function Name **uint32_t RTC_GetAlarmSubSecond (uint32_t RTC_Alarm)**

Function Description Gets the RTC Alarm Subseconds value.

Parameters
- RTC_Alarm.

Return values
- **RTC Alarm Subseconds value.**

Notes
- None.

19.2.27 WakeUp timer configuration functions

19.2.27.1 RTC_WakeUpClockConfig

Function Name **void RTC_WakeUpClockConfig (uint32_t RTC_WakeUpClock)**

Function Description Configures the RTC Wakeup clock source.

Parameters
- **RTC_WakeUpClock :** Wakeup Clock source. This parameter can be one of the following values:
  - ***RTC_WakeUpClock_RTCCLK_Div16 :*** RTC Wakeup Counter Clock = RTCCLK/16
  - ***RTC_WakeUpClock_RTCCLK_Div8 :*** RTC Wakeup Counter Clock = RTCCLK/8
  - ***RTC_WakeUpClock_RTCCLK_Div4 :*** RTC Wakeup Counter Clock = RTCCLK/4
  - ***RTC_WakeUpClock_RTCCLK_Div2 :*** RTC Wakeup Counter Clock = RTCCLK/2

- ***RTC_WakeUpClock_CK_SPRE_16bits :*** RTC Wakeup Counter Clock = CK_SPRE
- ***RTC_WakeUpClock_CK_SPRE_17bits :*** RTC Wakeup Counter Clock = CK_SPRE

Return values
- None.

Notes
- The WakeUp Clock source can only be changed when the RTC WakeUp is disabled (Use the RTC_WakeUpCmd(DISABLE)).

19.2.27.2 RTC_SetWakeUpCounter

Function Name **void RTC_SetWakeUpCounter (uint32_t RTC_WakeUpCounter)**

Function Description Configures the RTC Wakeup counter.

Parameters
- **RTC_WakeUpCounter :** specifies the WakeUp counter. This parameter can be a value from 0x0000 to 0xFFFF.

Return values
- None.

Notes
- The RTC WakeUp counter can only be written when the RTC WakeUp is disabled (Use the RTC_WakeUpCmd(DISABLE)).

19.2.27.3 RTC_GetWakeUpCounter

Function Name **uint32_t RTC_GetWakeUpCounter (void)**

Function Description Returns the RTC WakeUp timer counter value.

Parameters
- None.

Return values
- **The RTC WakeUp Counter value.**

Notes
- None.

19.2.27.4 RTC_WakeUpCmd

Function Name	**ErrorStatus RTC_WakeUpCmd (*FunctionalState* NewState)**
Function Description	Enables or Disables the RTC WakeUp timer.
Parameters	• **NewState :** new state of the WakeUp timer. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

19.2.28 Daylight saving configuration functions

19.2.28.1 RTC_DayLightSavingConfig

Function Name	**void RTC_DayLightSavingConfig (uint32_t RTC_DayLightSaving, uint32_t RTC_StoreOperation)**
Function Description	Adds or substract one hour from the current time.
Parameters	• **RTC_DayLightSaveOperation :** the value of hour adjustment. This parameter can be one of the following values: – ***RTC_DayLightSaving_SUB1H :*** Substract one hour (winter time) – ***RTC_DayLightSaving_ADD1H :*** Add one hour (summer time) • **RTC_StoreOperation :** Specifies the value to be written in the BCK bit in CR register to store the operation. This parameter can be one of the following values: – ***RTC_StoreOperation_Reset :*** BCK Bit Reset – ***RTC_StoreOperation_Set :*** BCK Bit Set
Return values	• None.
Notes	• None.

19.2.28.2 RTC_GetStoreOperation

Function Name	**uint32_t RTC_GetStoreOperation (void)**
Function Description	Returns the RTC Day Light Saving stored operation.
Parameters	• None.
Return values	• **RTC Day Light Saving stored operation.** – ***RTC_StoreOperation_Reset*** – ***RTC_StoreOperation_Set***
Notes	• None.

19.2.29 Output pin configuration functions

19.2.29.1 RTC_OutputConfig

Function Name	**void RTC_OutputConfig (uint32_t RTC_Output, uint32_t RTC_OutputPolarity)**
Function Description	Configures the RTC output source (AFO_ALARM).
Parameters	• **RTC_Output :** Specifies which signal will be routed to the RTC output. This parameter can be one of the following values: – ***RTC_Output_Disable :*** No output selected – ***RTC_Output_AlarmA :*** signal of AlarmA mapped to output – ***RTC_Output_AlarmB :*** signal of AlarmB mapped to output – ***RTC_Output_WakeUp :*** signal of WakeUp mapped to output • **RTC_OutputPolarity :** Specifies the polarity of the output signal. This parameter can be one of the following: – ***RTC_OutputPolarity_High :*** The output pin is high when the ALRAF/ALRBF/WUTF is high (depending on OSEL) – ***RTC_OutputPolarity_Low :*** The output pin is low when the ALRAF/ALRBF/WUTF is high (depending on OSEL)
Return values	• None.
Notes	• None.

19.2.30 Digital calibration configuration functions

19.2.30.1 RTC_CalibOutputCmd

Function Name	**void RTC_CalibOutputCmd (*FunctionalState* NewState)**
Function Description	Enables or disables the RTC clock to be output through the relative pin.
Parameters	• **NewState :** new state of the digital calibration Output. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

19.2.30.2 RTC_CalibOutputConfig

Function Name	**void RTC_CalibOutputConfig (uint32_t RTC_CalibOutput)**
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	• **RTC_CalibOutput :** : Select the Calibration output Selection . This parameter can be one of the following values: – ***RTC_CalibOutput_512Hz :*** A signal has a regular waveform at 512Hz. – ***RTC_CalibOutput_1Hz :*** A signal has a regular waveform at 1Hz.
Return values	• None.
Notes	• None.

19.2.30.3 RTC_SmoothCalibConfig

Function Name	**ErrorStatus RTC_SmoothCalibConfig (uint32_t RTC_SmoothCalibPeriod, uint32_t RTC_SmoothCalibPlusPulses, uint32_t RTC_SmouthCalibMinusPulsesValue)**
Function Description	Configures the Smooth Calibration Settings.

Parameters

- **RTC_SmoothCalibPeriod :** : Select the Smooth Calibration Period. This parameter can be can be one of the following values:
  - ***RTC_SmoothCalibPeriod_32sec :*** The smooth calibration periode is 32s.
  - ***RTC_SmoothCalibPeriod_16sec :*** The smooth calibration periode is 16s.
  - ***RTC_SmoothCalibPeriod_8sec :*** The smooth calibartion periode is 8s.
- **RTC_SmoothCalibPlusPulses :** : Select to Set or reset the CALP bit. This parameter can be one of the following values:
  - ***RTC_SmoothCalibPlusPulses_Set :*** Add one RTCCLK puls every 2**11 pulses.
  - ***RTC_SmoothCalibPlusPulses_Reset :*** No RTCCLK pulses are added.
- **RTC_SmouthCalibMinusPulsesValue :** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- **An ErrorStatus enumeration value:**
  - ***SUCCESS: RTC Calib registers are configured***
  - ***ERROR: RTC Calib registers are not configured***

Notes

- None.

19.2.31 Timestamp configuration functions

19.2.31.1 RTC_TimeStampCmd

Function Name

void RTC_TimeStampCmd (uint32_t RTC_TimeStampEdge, *FunctionalState* NewState)

Function Description

Enables or Disables the RTC TimeStamp functionality with the specified time stamp pin stimulating edge.

Parameters

- **RTC_TimeStampEdge :** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following:
  - ***RTC_TimeStampEdge_Rising :*** the Time stamp event occurs on the rising edge of the related pin.
  - ***RTC_TimeStampEdge_Falling :*** the Time stamp event occurs on the falling edge of the related pin.
- **NewState :** new state of the TimeStamp. This parameter can be: ENABLE or DISABLE.

Return values

- None.

Notes

- None.

19.2.31.2 RTC_GetTimeStamp

Function Name	**void RTC_GetTimeStamp (uint32_t RTC_Format, *RTC_TimeTypeDef** * **RTC_StampTimeStruct, *RTC_DateTypeDef** * **RTC_StampDateStruct)**
Function Description	Gets the RTC TimeStamp value and masks.
Parameters	• **RTC_Format :** specifies the format of the output parameters. This parameter can be one of the following values: – ***RTC_Format_BIN :*** Binary data format – ***RTC_Format_BCD :*** BCD data format • **RTC_StampTimeStruct :** pointer to a RTC_TimeTypeDef structure that will contains the TimeStamp time values. • **RTC_StampDateStruct :** pointer to a RTC_DateTypeDef structure that will contains the TimeStamp date values.
Return values	• None.
Notes	• None.

19.2.31.3 RTC_GetTimeStampSubSecond

Function Name	**uint32_t RTC_GetTimeStampSubSecond (void)**
Function Description	Gets the RTC timestamp Subseconds value.
Parameters	• None.
Return values	• **RTC current timestamp Subseconds value.**
Notes	• None.

19.2.32 Tamper configuration functions

19.2.32.1 RTC_TamperTriggerConfig

Function Name	**void RTC_TamperTriggerConfig (uint32_t RTC_Tamper,**

uint32_t RTC_TamperTrigger)

Function Description — Configures the select Tamper pin edge.

Parameters
- **RTC_Tamper :** Selected tamper pin. This parameter can be any combination of the following values:
  - ***RTC_Tamper_1 :*** Select Tamper 1.
  - ***RTC_Tamper_2 :*** Select Tamper 2.
  - ***RTC_Tamper_3 :*** Select Tamper 3.
- **RTC_TamperTrigger :** Specifies the trigger on the tamper pin that stimulates tamper event. This parameter can be one of the following values:
  - ***RTC_TamperTrigger_RisingEdge :*** Rising Edge of the tamper pin causes tamper event.
  - ***RTC_TamperTrigger_FallingEdge :*** Falling Edge of the tamper pin causes tamper event.
  - ***RTC_TamperTrigger_LowLevel :*** Low Level of the tamper pin causes tamper event.
  - ***RTC_TamperTrigger_HighLevel :*** High Level of the tamper pin causes tamper event.

Return values
- None.

Notes
- None.

19.2.32.2 RTC_TamperCmd

Function Name — **void RTC_TamperCmd (uint32_t RTC_Tamper, *FunctionalState* NewState)**

Function Description — Enables or Disables the Tamper detection.

Parameters
- **RTC_Tamper :** Selected tamper pin. This parameter can be any combination of the following values:
  - ***RTC_Tamper_1 :*** Select Tamper 1.
  - ***RTC_Tamper_2 :*** Select Tamper 2.
  - ***RTC_Tamper_3 :*** Select Tamper 3.
- **NewState :** new state of the tamper pin. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

19.2.32.3 RTC_TamperFilterConfig

Function Name	**void RTC_TamperFilterConfig (uint32_t RTC_TamperFilter)**
Function Description	Configures the Tampers Filter.
Parameters	• **RTC_TamperFilter :** Specifies the tampers filter. This parameter can be one of the following values: – ***RTC_TamperFilter_Disable :*** Tamper filter is disabled. – ***RTC_TamperFilter_2Sample :*** Tamper is activated after 2 consecutive samples at the active level – ***RTC_TamperFilter_4Sample :*** Tamper is activated after 4 consecutive samples at the active level – ***RTC_TamperFilter_8Sample :*** Tamper is activated after 8 consecutive samples at the active level
Return values	• None.
Notes	• None.

19.2.32.4 RTC_TamperSamplingFreqConfig

Function Name	**void RTC_TamperSamplingFreqConfig (uint32_t RTC_TamperSamplingFreq)**
Function Description	Configures the Tampers Sampling Frequency.
Parameters	• **RTC_TamperSamplingFreq :** Specifies the tampers Sampling Frequency. This parameter can be one of the following values: – ***RTC_TamperSamplingFreq_RTCCLK_Div32768 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768 – ***RTC_TamperSamplingFreq_RTCCLK_Div16384 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384 – ***RTC_TamperSamplingFreq_RTCCLK_Div8192 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192 – ***RTC_TamperSamplingFreq_RTCCLK_Div4096 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096 – ***RTC_TamperSamplingFreq_RTCCLK_Div2048 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048 – ***RTC_TamperSamplingFreq_RTCCLK_Div1024 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

- ***RTC_TamperSamplingFreq_RTCCLK_Div512 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
- ***RTC_TamperSamplingFreq_RTCCLK_Div256 :*** Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

Return values
- None.

Notes
- None.

19.2.32.5 RTC_TamperPinsPrechargeDuration

Function Name **void RTC_TamperPinsPrechargeDuration (uint32_t RTC_TamperPrechargeDuration)**

Function Description Configures the Tampers Pins input Precharge Duration.

Parameters
- **RTC_TamperPrechargeDuration :** Specifies the Tampers Pins input Precharge Duration. This parameter can be one of the following values:
  - ***RTC_TamperPrechargeDuration_1RTCCLK :*** Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
  - ***RTC_TamperPrechargeDuration_2RTCCLK :*** Tamper pins are pre-charged before sampling during 2 RTCCLK cycle
  - ***RTC_TamperPrechargeDuration_4RTCCLK :*** Tamper pins are pre-charged before sampling during 4 RTCCLK cycle
  - ***RTC_TamperPrechargeDuration_8RTCCLK :*** Tamper pins are pre-charged before sampling during 8 RTCCLK cycle

Return values
- None.

Notes
- None.

19.2.32.6 RTC_TimeStampOnTamperDetectionCmd

Function Name **void RTC_TimeStampOnTamperDetectionCmd (*FunctionalState* NewState)**

Function Description Enables or Disables the TimeStamp on Tamper Detection Event.

Parameters	• **NewState :** new state of the timestamp on tamper event. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• The timestamp is valid even the TSE bit in tamper control register is reset.

19.2.32.7 RTC_TamperPullUpCmd

Function Name	**void RTC_TamperPullUpCmd (*FunctionalState* NewState)**
Function Description	Enables or Disables the Precharge of Tamper pin.
Parameters	• **NewState :** new state of tamper pull up. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

19.3 RTC Firmware driver defines

19.3.1 RTC

RTC

RTC_Add_1_Second_Parameter_Definitions

- #define: ***RTC_ShiftAdd1S_Reset ((uint32_t)0x00000000)***

- #define: ***RTC_ShiftAdd1S_Set ((uint32_t)0x80000000)***

RTC_AlarmDateWeekDay_Definitions

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

- #define: ***RTC_AlarmDateWeekDaySel_WeekDay ((uint32_t)0x40000000)***

RTC_AlarmMask_Definitions

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

- #define: ***RTC_AlarmMask_DateWeekDay ((uint32_t)0x80000000)***

- #define: ***RTC_AlarmMask_Hours ((uint32_t)0x00800000)***

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

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

- #define: ***RTC_AlarmMask_All ((uint32_t)0x80808080)***

RTC_Alarms_Definitions

- #define: ***RTC_Alarm_A ((uint32_t)0x00000100)***

- #define: ***RTC_Alarm_B ((uint32_t)0x00000200)***

RTC_Alarm_Sub_Seconds_Masks_Definitions

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

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

- #define: ***RTC_AlarmSubSecondMask_SS14_1 ((uint32_t)0x01000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_2 ((uint32_t)0x02000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_3 ((uint32_t)0x03000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_4 ((uint32_t)0x04000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_5 ((uint32_t)0x05000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_6 ((uint32_t)0x06000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_7 ((uint32_t)0x07000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_8 ((uint32_t)0x08000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_9 ((uint32_t)0x09000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_10 ((uint32_t)0x0A000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_11 ((uint32_t)0x0B000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_12 ((uint32_t)0x0C000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14_13 ((uint32_t)0x0D000000)***

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

- #define: ***RTC_AlarmSubSecondMask_SS14 ((uint32_t)0x0E000000)***

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

- #define: ***RTC_AlarmSubSecondMask_None ((uint32_t)0x0F000000)***

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

RTC_AM_PM_Definitions

- #define: ***RTC_H12_AM ((uint8_t)0x00)***

- #define: ***RTC_H12_PM ((uint8_t)0x40)***

RTC_Backup_Registers_Definitions

- #define: ***RTC_BKP_DR0 ((uint32_t)0x00000000)***

- #define: ***RTC_BKP_DR1 ((uint32_t)0x00000001)***

- #define: ***RTC_BKP_DR2 ((uint32_t)0x00000002)***

- #define: ***RTC_BKP_DR3 ((uint32_t)0x00000003)***

- #define: ***RTC_BKP_DR4 ((uint32_t)0x00000004)***

- #define: ***RTC_BKP_DR5 ((uint32_t)0x00000005)***

- #define: ***RTC_BKP_DR6 ((uint32_t)0x00000006)***

- #define: ***RTC_BKP_DR7 ((uint32_t)0x00000007)***

- #define: ***RTC_BKP_DR8 ((uint32_t)0x00000008)***

- #define: ***RTC_BKP_DR9 ((uint32_t)0x00000009)***

- #define: ***RTC_BKP_DR10 ((uint32_t)0x0000000A)***

- #define: ***RTC_BKP_DR11 ((uint32_t)0x0000000B)***

- #define: ***RTC_BKP_DR12 ((uint32_t)0x0000000C)***

- #define: ***RTC_BKP_DR13 ((uint32_t)0x0000000D)***

- #define: ***RTC_BKP_DR14 ((uint32_t)0x0000000E)***

- #define: ***RTC_BKP_DR15 ((uint32_t)0x0000000F)***

RTC_Calib_Output_selection_Definitions

- #define: ***RTC_CalibOutput_512Hz ((uint32_t)0x00000000)***

- #define: ***RTC_CalibOutput_1Hz ((uint32_t)0x00080000)***

RTC_DayLightSaving_Definitions

- #define: ***RTC_DayLightSaving_SUB1H ((uint32_t)0x00020000)***

- #define: ***RTC_DayLightSaving_ADD1H ((uint32_t)0x00010000)***

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

- #define: ***RTC_StoreOperation_Set ((uint32_t)0x00040000)***

RTC_Digital_Calibration_Definitions

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

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

RTC_Flags_Definitions

- #define: ***RTC_FLAG_RECALPF ((uint32_t)0x00010000)***

- #define: ***RTC_FLAG_TAMP3F ((uint32_t)0x00008000)***

- #define: ***RTC_FLAG_TAMP2F ((uint32_t)0x00004000)***

- #define: ***RTC_FLAG_TAMP1F ((uint32_t)0x00002000)***

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

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

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

- #define: ***RTC_FLAG_ALRBF ((uint32_t)0x00000200)***

- #define: ***RTC_FLAG_ALRAF ((uint32_t)0x00000100)***

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

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

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

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

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

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

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

RTC_Hour_Formats

- #define: ***RTC_HourFormat_24 ((uint32_t)0x00000000)***

- #define: ***RTC_HourFormat_12 ((uint32_t)0x00000040)***

RTC_Input_parameter_format_definitions

- #define: ***RTC_Format_BIN ((uint32_t)0x000000000)***

- #define: ***RTC_Format_BCD ((uint32_t)0x000000001)***

RTC_Interrupts_Definitions

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

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

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

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

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

- #define: ***RTC_IT_TAMP1 ((uint32_t)0x00020000)***

- #define: ***RTC_IT_TAMP2 ((uint32_t)0x00040000)***

- #define: ***RTC_IT_TAMP3 ((uint32_t)0x00080000)***

RTC_Month_Date_Definitions

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

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

- #define: ***RTC_Month_March ((uint8_t)0x03)***
- #define: ***RTC_Month_April ((uint8_t)0x04)***
- #define: ***RTC_Month_May ((uint8_t)0x05)***
- #define: ***RTC_Month_June ((uint8_t)0x06)***
- #define: ***RTC_Month_July ((uint8_t)0x07)***
- #define: ***RTC_Month_August ((uint8_t)0x08)***
- #define: ***RTC_Month_September ((uint8_t)0x09)***
- #define: ***RTC_Month_October ((uint8_t)0x10)***
- #define: ***RTC_Month_November ((uint8_t)0x11)***
- #define: ***RTC_Month_December ((uint8_t)0x12)***

RTC_Output_Polarity_Definitions

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

- #define: ***RTC_OutputPolarity_Low ((uint32_t)0x00100000)***

RTC_Output_selection_Definitions

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

- #define: ***RTC_Output_AlarmA ((uint32_t)0x00200000)***

- #define: ***RTC_Output_AlarmB ((uint32_t)0x00400000)***

- #define: ***RTC_Output_WakeUp ((uint32_t)0x00600000)***

RTC_Output_Type_ALARM_OUT

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

- #define: ***RTC_OutputType_PushPull ((uint32_t)0x00040000)***

RTC_Smooth_calib_period_Definitions

- #define: ***RTC_SmoothCalibPeriod_32sec ((uint32_t)0x00000000)***

if RTCCLK = 32768 Hz, Smooth calibation period is 32s, else 2exp20 RTCCLK seconds

- #define: ***RTC_SmoothCalibPeriod_16sec ((uint32_t)0x00002000)***

if RTCCLK = 32768 Hz, Smooth calibation period is 16s, else 2exp19 RTCCLK seconds

- #define: ***RTC_SmoothCalibPeriod_8sec ((uint32_t)0x00004000)***

if RTCCLK = 32768 Hz, Smooth calibation period is 8s, else 2exp18 RTCCLK seconds

RTC_Smooth_calib_Plus_pulses_Definitions

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

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0]. with Y = 512, 256, 128 when X = 32, 16, 8

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

The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0].

RTC_Tamper_Filter_Definitions

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

Tamper filter is disabled

- #define: ***RTC_TamperFilter_2Sample ((uint32_t)0x00000800)***

Tamper is activated after 2 consecutive samples at the active level

- #define: ***RTC_TamperFilter_4Sample ((uint32_t)0x00001000)***

Tamper is activated after 4 consecutive samples at the active level

- #define: ***RTC_TamperFilter_8Sample ((uint32_t)0x00001800)***

Tamper is activated after 8 consecutive samples at the active leve.

RTC_Tamper_Pins_Definitions

- #define: ***RTC_Tamper_1 RTC_TAFCR_TAMP1E***

Tamper detection enable for input tamper 1

- #define: ***RTC_Tamper_2 RTC_TAFCR_TAMP2E***

Tamper detection enable for input tamper 2

- #define: ***RTC_Tamper_3 RTC_TAFCR_TAMP3E***

Tamper detection enable for input tamper 3

RTC_Tamper_Pin_Precharge_Duration_Definitions

- #define: ***RTC_TamperPrechargeDuration_1RTCCLK ((uint32_t)0x00000000)***

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

- #define: ***RTC_TamperPrechargeDuration_2RTCCLK ((uint32_t)0x00002000)***

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

- #define: ***RTC_TamperPrechargeDuration_4RTCCLK ((uint32_t)0x00004000)***

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

- #define: ***RTC_TamperPrechargeDuration_8RTCCLK ((uint32_t)0x00006000)***

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTC_Tamper_Sampling_Frequencies_Definitions

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div32768 ((uint32_t)0x00000000)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div16384 ((uint32_t)0x000000100)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div8192 ((uint32_t)0x00000200)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div4096 ((uint32_t)0x00000300)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div2048 ((uint32_t)0x00000400)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div1024 ((uint32_t)0x00000500)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div512 ((uint32_t)0x00000600)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

- #define: ***RTC_TamperSamplingFreq_RTCCLK_Div256 ((uint32_t)0x00000700)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTC_Tamper_Trigger_Definitions

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

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

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

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

RTC_Time_Stamp_Edges_definitions

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

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

RTC_Wakeup_Timer_Definitions

- #define: ***RTC_WakeUpClock_RTCCLK_Div16 ((uint32_t)0x00000000)***

- #define: ***RTC_WakeUpClock_RTCCLK_Div8 ((uint32_t)0x00000001)***

- #define: ***RTC_WakeUpClock_RTCCLK_Div4 ((uint32_t)0x00000002)***

- #define: ***RTC_WakeUpClock_RTCCLK_Div2 ((uint32_t)0x00000003)***

- #define: ***RTC_WakeUpClock_CK_SPRE_16bits ((uint32_t)0x00000004)***

- #define: ***RTC_WakeUpClock_CK_SPRE_17bits ((uint32_t)0x00000006)***

RTC_WeekDay_Definitions

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

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

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

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

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

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

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

20 Serial peripheral interface (SPI)

20.1 SPI Firmware driver registers structures

20.1.1 SPI_TypeDef

SPI_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint16_t CR1***
- ***uint16_t RESERVED0***
- ***__IO uint16_t CR2***
- ***uint16_t RESERVED1***
- ***__IO uint16_t SR***
- ***uint16_t RESERVED2***
- ***__IO uint16_t DR***
- ***uint16_t RESERVED3***
- ***__IO uint16_t CRCPR***
- ***uint16_t RESERVED4***
- ***__IO uint16_t RXCRCR***
- ***uint16_t RESERVED5***
- ***__IO uint16_t TXCRCR***
- ***uint16_t RESERVED6***
- ***__IO uint16_t I2SCFGR***
- ***uint16_t RESERVED7***
- ***__IO uint16_t I2SPR***
- ***uint16_t RESERVED8***

Field Documentation

- ***__IO uint16_t SPI_TypeDef::CR1***
  - SPI Control register 1 (not used in I2S mode), Address offset: 0x00
- ***uint16_t SPI_TypeDef::RESERVED0***
  - Reserved, 0x02
- ***__IO uint16_t SPI_TypeDef::CR2***
  - SPI Control register 2, Address offset: 0x04
- ***uint16_t SPI_TypeDef::RESERVED1***
  - Reserved, 0x06
- ***__IO uint16_t SPI_TypeDef::SR***
  - SPI Status register, Address offset: 0x08
- ***uint16_t SPI_TypeDef::RESERVED2***
  - Reserved, 0x0A
- ***__IO uint16_t SPI_TypeDef::DR***
  - SPI data register, Address offset: 0x0C
- ***uint16_t SPI_TypeDef::RESERVED3***
  - Reserved, 0x0E
- ***__IO uint16_t SPI_TypeDef::CRCPR***
  - SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10

- ***uint16_t SPI_TypeDef::RESERVED4***
  - Reserved, 0x12
- ***__IO uint16_t SPI_TypeDef::RXCRCR***
  - SPI Rx CRC register (not used in I2S mode), Address offset: 0x14
- ***uint16_t SPI_TypeDef::RESERVED5***
  - Reserved, 0x16
- ***__IO uint16_t SPI_TypeDef::TXCRCR***
  - SPI Tx CRC register (not used in I2S mode), Address offset: 0x18
- ***uint16_t SPI_TypeDef::RESERVED6***
  - Reserved, 0x1A
- ***__IO uint16_t SPI_TypeDef::I2SCFGR***
  - SPI_I2S configuration register, Address offset: 0x1C
- ***uint16_t SPI_TypeDef::RESERVED7***
  - Reserved, 0x1E
- ***__IO uint16_t SPI_TypeDef::I2SPR***
  - SPI_I2S prescaler register, Address offset: 0x20
- ***uint16_t SPI_TypeDef::RESERVED8***
  - Reserved, 0x22

20.1.2 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32f30x_spi.h

Data Fields

- ***uint16_t SPI_Direction***
- ***uint16_t SPI_Mode***
- ***uint16_t SPI_DataSize***
- ***uint16_t SPI_CPOL***
- ***uint16_t SPI_CPHA***
- ***uint16_t SPI_NSS***
- ***uint16_t SPI_BaudRatePrescaler***
- ***uint16_t SPI_FirstBit***
- ***uint16_t SPI_CRCPolynomial***

Field Documentation

- ***uint16_t SPI_InitTypeDef::SPI_Direction***
  - Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_data_direction***
- ***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_DataSize***
  - Specifies the SPI data size. This parameter can be a value of ***SPI_data_size***
- ***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***
- ***uint16_t SPI_InitTypeDef::SPI_NSS***
  - Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_management***
- ***uint16_t SPI_InitTypeDef::SPI_BaudRatePrescaler***
  - Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler.***
- ***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***
- ***uint16_t SPI_InitTypeDef::SPI_CRCPolynomial***
  - Specifies the polynomial used for the CRC calculation.

20.1.3 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32f30x_spi.h

Data Fields

- ***uint16_t I2S_Mode***
- ***uint16_t I2S_Standard***
- ***uint16_t I2S_DataFormat***
- ***uint16_t I2S_MCLKOutput***
- ***uint32_t I2S_AudioFreq***
- ***uint16_t I2S_CPOL***

Field Documentation

- ***uint16_t I2S_InitTypeDef::I2S_Mode***
  - Specifies the I2S operating mode. This parameter can be a value of ***I2S_Mode***
- ***uint16_t I2S_InitTypeDef::I2S_Standard***
  - Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_Standard***
- ***uint16_t I2S_InitTypeDef::I2S_DataFormat***
  - Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_Data_Format***
- ***uint16_t I2S_InitTypeDef::I2S_MCLKOutput***
  - Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_MCLK_Output***
- ***uint32_t I2S_InitTypeDef::I2S_AudioFreq***
  - Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_Audio_Frequency***
- ***uint16_t I2S_InitTypeDef::I2S_CPOL***
  - Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_Clock_Polarity***

20.2 SPI Firmware driver API description

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

20.2.1 How to use this driver

1. Enable peripheral clock using RCC_APBPeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE) function for SPI1 or using RCC_APBPeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE) function for SPI2.
2. Enable SCK, MOSI, MISO and NSS GPIO clocks using RCC_AHBPeriphClockCmd() function.
3. Peripherals alternate function:
   - Connect the pin to the desired peripherals' Alternate Function (AF) using GPIO_PinAFConfig() function.
   - Configure the desired pin in alternate function by: GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF.
   - Select the type, pull-up/pull-down and output speed via GPIO_PuPd, GPIO_OType and GPIO_Speed members.
   - Call GPIO_Init() function.
4. Program the Polarity, Phase, First Data, Baud Rate Prescaler, Slave Management, Peripheral Mode and CRC Polynomial values using the SPI_Init() function in SPI mode. In I2S mode, program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using I2S_Init() function.
5. Configure the FIFO threshold using SPI_RxFIFOThresholdConfig() to select at which threshold the RXNE event is generated.
6. Enable the NVIC and the corresponding interrupt using the function SPI_I2S_ITConfig() if you need to use interrupt mode.
7. When using the DMA mode
   - Configure the DMA using DMA_Init() function.
   - Active the needed channel Request using SPI_I2S_DMACmd() function.
8. Enable the SPI using the SPI_Cmd() function or enable the I2S using I2S_Cmd().
9. Enable the DMA using the DMA_Cmd() function when using DMA mode.
10. Optionally you can enable/configure the following parameters without re-initialization (i.e there is no need to call again SPI_Init() function):
    - When bidirectional mode (SPI_Direction_1Line_Rx or SPI_Direction_1Line_Tx) is programmed as Data direction parameter using the SPI_Init() function it can be possible to switch between SPI_Direction_Tx or SPI_Direction_Rx using the SPI_BiDirectionalLineConfig() function.
    - When SPI_NSS_Soft is selected as Slave Select Management parameter using the SPI_Init() function it can be possible to manage the NSS internal signal using the SPI_NSSInternalSoftwareConfig() function.
    - Reconfigure the data size using the SPI_DataSizeConfig() function.
    - Enable or disable the SS output using the SPI_SSOutputCmd() function.
11. To use the CRC Hardware calculation feature refer to the Peripheral CRC hardware Calculation subsection.

It is possible to use SPI in I2S full duplex mode, in this case, each SPI peripheral is able to manage sending and receiving data simultaneously using two data lines. Each SPI peripheral has an extended block called I2Sxext (ie. I2S2ext for SPI2 and I2S3ext for SPI3). The extension block is not a full SPI IP, it is used only as I2S slave to implement full duplex mode. The extension block uses the same clock sources as its master. To configure I2S full duplex you have to:

1. Configure SPIx in I2S mode (I2S_Init() function) as described above.
2. Call the I2S_FullDuplexConfig() function using the same strucutre passed to I2S_Init() function.
3. Call I2S_Cmd() for SPIx then for its extended block.
4. Configure interrupts or DMA requests and to get/clear flag status, use I2Sxext instance for the extension block.

Functions that can be called with I2Sxext instances are: I2S_Cmd(), I2S_FullDuplexConfig(), SPI_I2S_ReceiveData16(), SPI_I2S_SendData16(), SPI_I2S_DMACmd(), SPI_I2S_ITConfig(), SPI_I2S_GetFlagStatus(), SPI_I2S_ClearFlag(), SPI_I2S_GetITStatus() and SPI_I2S_ClearITPendingBit().

Example: To use SPI3 in Full duplex mode (SPI3 is Master Tx, I2S3ext is Slave Rx):

RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI3, ENABLE); I2S_StructInit(&I2SInitStruct); I2SInitStruct.Mode = I2S_Mode_MasterTx; I2S_Init(SPI3, &I2SInitStruct); I2S_FullDuplexConfig(SPI3ext, &I2SInitStruct) I2S_Cmd(SPI3, ENABLE); I2S_Cmd(SPI3ext, ENABLE); ... while (SPI_I2S_GetFlagStatus(SPI2, SPI_FLAG_TXE) == RESET) {} SPI_I2S_SendData16(SPI3, txdata[i]); ... while (SPI_I2S_GetFlagStatus(I2S3ext, SPI_FLAG_RXNE) == RESET) {} rxdata[i] = SPI_I2S_ReceiveData16(I2S3ext); ...

In SPI mode: To use the SPI TI mode, call the function SPI_TIModeCmd() just after calling the function SPI_Init().

20.2.2 Initialization and Configuration functions

This section provides a set of functions allowing to initialize the SPI Direction, SPI Mode, SPI Data Size, SPI Polarity, SPI Phase, SPI NSS Management, SPI Baud Rate Prescaler, SPI First Bit and SPI CRC Polynomial.

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

When the Software NSS management (SPI_InitStruct->SPI_NSS = SPI_NSS_Soft) is selected, use the following function to manage the NSS bit: void SPI_NSSInternalSoftwareConfig(SPI_TypeDef* SPIx, uint16_t SPI_NSSInternalSoft);

In Master mode, when the Hardware NSS management (SPI_InitStruct->SPI_NSS = SPI_NSS_Hard) is selected, use the follwoing function to enable the NSS output feature. void SPI_SSOutputCmd(SPI_TypeDef* SPIx, FunctionalState NewState);

The NSS pulse mode can be managed by the SPI TI mode when enabling it using the following function: void SPI_TIModeCmd(SPI_TypeDef* SPIx, FunctionalState NewState); And it can be managed by software in the SPI Motorola mode using this function: void SPI_NSSPulseModeCmd(SPI_TypeDef* SPIx, FunctionalState NewState);

This section provides also functions to initialize the I2S Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity.

The I2S_Init() function follows the I2S configuration procedures for Master mode and Slave mode.

- ***SPI_I2S_DeInit()***
- ***SPI_StructInit()***
- ***SPI_Init()***

- ***I2S_StructInit()***
- ***I2S_Init()***
- ***SPI_Cmd()***
- ***SPI_TIModeCmd()***
- ***I2S_Cmd()***
- ***SPI_DataSizeConfig()***
- ***SPI_RxFIFOThresholdConfig()***
- ***SPI_BiDirectionalLineConfig()***
- ***SPI_NSSInternalSoftwareConfig()***
- ***I2S_FullDuplexConfig()***
- ***SPI_SSOutputCmd()***
- ***SPI_NSSPulseModeCmd()***

20.2.3 Data transfers functions

This section provides a set of functions allowing to manage the SPI or I2S 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 SPI_DR register can be done using the SPI_I2S_ReceiveData() function and returns the Rx buffered value. Whereas a write access to the SPI_DR can be done using SPI_I2S_SendData() function and stores the written data into Tx buffer.

- ***SPI_SendData8()***
- ***SPI_I2S_SendData16()***
- ***SPI_ReceiveData8()***
- ***SPI_I2S_ReceiveData16()***

20.2.4 Hardware CRC Calculation functions

This section provides a set of functions allowing to manage the SPI CRC hardware calculation.

SPI communication using CRC is possible through the following procedure:

1. Program the Data direction, Polarity, Phase, First Data, Baud Rate Prescaler, Slave Management, Peripheral Mode and CRC Polynomial values using the SPI_Init() function.
2. Enable the CRC calculation using the SPI_CalculateCRC() function.
3. Enable the SPI using the SPI_Cmd() function
4. Before writing the last data to the TX buffer, set the CRCNext bit using the SPI_TransmitCRC() function to indicate that after transmission of the last data, the CRC should be transmitted.
5. After transmitting the last data, the SPI transmits the CRC. The SPI_CR1_CRCNEXT bit is reset. The CRC is also received and compared against the SPI_RXCRCR value. If the value does not match, the SPI_FLAG_CRCERR flag is set and an interrupt can be generated when the SPI_I2S_IT_ERR interrupt is enabled.

- It is advised to don't read the calculate CRC values during the communication.
- When the SPI is in slave mode, be careful to enable CRC calculation only when the clock is stable, that is, when the clock is in the steady state. If not, a wrong CRC calculation may be done. In fact, the CRC is sensitive to the

SCK slave input clock as soon as CRCEN is set, and this, whatever the value of the SPE bit.

- With high bitrate frequencies, be careful when transmitting the CRC. As the number of used CPU cycles has to be as low as possible in the CRC transfer phase, it is forbidden to call software functions in the CRC transmission sequence to avoid errors in the last data and CRC reception. In fact, CRCNEXT bit has to be written before the end of the transmission/reception of the last data.
- For high bit rate frequencies, it is advised to use the DMA mode to avoid the degradation of the SPI speed performance due to CPU accesses impacting the SPI bandwidth.
- When the STM32F30x are configured as slaves and the NSS hardware mode is used, the NSS pin needs to be kept low between the data phase and the CRC phase.
- When the SPI is configured in slave mode with the CRC feature enabled, CRC calculation takes place even if a high level is applied on the NSS pin. This may happen for example in case of a multislave environment where the communication master addresses slaves alternately.
- Between a slave deselection (high level on NSS) and a new slave selection (low level on NSS), the CRC value should be cleared on both master and slave sides in order to resynchronize the master and slave for their respective CRC calculation.

To clear the CRC, follow the procedure below:

1. Disable SPI using the SPI_Cmd() function.
2. Disable the CRC calculation using the SPI_CalculateCRC() function.
3. Enable the CRC calculation using the SPI_CalculateCRC() function.
4. Enable SPI using the SPI_Cmd() function.

- ***SPI_CRCLengthConfig()***
- ***SPI_CalculateCRC()***
- ***SPI_TransmitCRC()***
- ***SPI_GetCRC()***
- ***SPI_GetCRCPolynomial()***

20.2.5 DMA transfers management functions

- ***SPI_I2S_DMACmd()***
- ***SPI_LastDMATransferCmd()***

20.2.6 Interrupts and flags management functions

This section provides a set of functions allowing to configure the SPI/I2S Interrupts sources 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.

Polling Mode

In Polling Mode, the SPI/I2S communication can be managed by 9 flags:

1. SPI_I2S_FLAG_TXE : to indicate the status of the transmit buffer register.
2. SPI_I2S_FLAG_RXNE : to indicate the status of the receive buffer register.
3. SPI_I2S_FLAG_BSY : to indicate the state of the communication layer of the SPI.
4. SPI_FLAG_CRCERR : to indicate if a CRC Calculation error occur.
5. SPI_FLAG_MODF : to indicate if a Mode Fault error occur.
6. SPI_I2S_FLAG_OVR : to indicate if an Overrun error occur.
7. SPI_I2S_FLAG_FRE: to indicate a Frame Format error occurs.
8. I2S_FLAG_UDR: to indicate an Underrun error occurs.
9. I2S_FLAG_CHSIDE: to indicate Channel Side.

Do not use the BSY flag to handle each data transmission or reception. It is better to use the TXE and RXNE flags instead.

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

- FlagStatus SPI_I2S_GetFlagStatus(SPI_TypeDef* SPIx, uint16_t SPI_I2S_FLAG);
- void SPI_I2S_ClearFlag(SPI_TypeDef* SPIx, uint16_t SPI_I2S_FLAG);

Interrupt Mode

In Interrupt Mode, the SPI/I2S communication can be managed by 3 interrupt sources and 5 pending bits:

Pending Bits:

1. SPI_I2S_IT_TXE : to indicate the status of the transmit buffer register.
2. SPI_I2S_IT_RXNE : to indicate the status of the receive buffer register.
3. SPI_I2S_IT_OVR : to indicate if an Overrun error occur.
4. I2S_IT_UDR : to indicate an Underrun Error occurs.
5. SPI_I2S_FLAG_FRE : to indicate a Frame Format error occurs.

Interrupt Source:

1. SPI_I2S_IT_TXE: specifies the interrupt source for the Tx buffer empty interrupt.
2. SPI_I2S_IT_RXNE : specifies the interrupt source for the Rx buffer not empty interrupt.
3. SPI_I2S_IT_ERR : specifies the interrupt source for the errors interrupt.

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

- void SPI_I2S_ITConfig(SPI_TypeDef* SPIx, uint8_t SPI_I2S_IT, FunctionalState NewState);
- ITStatus SPI_I2S_GetITStatus(SPI_TypeDef* SPIx, uint8_t SPI_I2S_IT);

FIFO Status

It is possible to monitor the FIFO status when a transfer is ongoing using the following function:

- uint32_t SPI_GetFIFOStatus(uint8_t SPI_FIFO_Direction);

DMA Mode

In DMA Mode, the SPI communication can be managed by 2 DMA Channel requests:

1. SPI_I2S_DMAReq_Tx: specifies the Tx buffer DMA transfer request.
2. SPI_I2S_DMAReq_Rx: specifies the Rx buffer DMA transfer request.

In this Mode it is advised to use the following function:

- void SPI_I2S_DMACmd(SPI_TypeDef* SPIx, uint16_t SPI_I2S_DMAReq, FunctionalState NewState);
- ***SPI_I2S_ITConfig()***
- ***SPI_GetTransmissionFIFOStatus()***
- ***SPI_GetReceptionFIFOStatus()***
- ***SPI_I2S_GetFlagStatus()***
- ***SPI_I2S_ClearFlag()***
- ***SPI_I2S_GetITStatus()***

20.2.7 Initialization and configuration functions

20.2.7.1 SPI_I2S_DeInit

Function Name	**void SPI_I2S_DeInit (*SPI_TypeDef* * SPIx)**
Function Description	Deinitializes the SPIx peripheral registers to their default reset values.
Parameters	• **SPIx :** To select the SPIx peripheral, where x can be: 1, 2 or 3 in SPI mode.
Return values	• None.
Notes	• None.

20.2.7.2 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.

20.2.7.3 SPI_Init

Function Name	**void SPI_Init (*SPI_TypeDef* * SPIx, *SPI_InitTypeDef* ***

SPI_InitStruct)

Function Description	Initializes the SPIx peripheral according to the specified parameters in the SPI_InitStruct.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_InitStruct :** pointer to a SPI_InitTypeDef structure that contains the configuration information for the specified SPI peripheral.
Return values	• None.
Notes	• None.

20.2.7.4 I2S_StructInit

Function Name	**void I2S_StructInit (*I2S_InitTypeDef ** I2S_InitStruct)**
Function Description	Fills each I2S_InitStruct member with its default value.
Parameters	• **I2S_InitStruct :** : pointer to a I2S_InitTypeDef structure which will be initialized.
Return values	• None.
Notes	• None.

20.2.7.5 I2S_Init

Function Name	**void I2S_Init (*SPI_TypeDef ** SPIx, *I2S_InitTypeDef ** I2S_InitStruct)**
Function Description	Initializes the SPIx peripheral according to the specified parameters in the I2S_InitStruct.
Parameters	• **SPIx:To :** select the SPIx peripheral, where x can be: 2 or 3 in I2S mode. • **I2S_InitStruct :** pointer to an I2S_InitTypeDef structure that contains the configuration information for the specified SPI peripheral configured in I2S mode.
Return values	• None.
Notes	• The function calculates the optimal prescaler needed to obtain the most accurate audio frequency (depending on the I2S clock source, the PLL values and the product

configuration). But in case the prescaler value is greater than 511, the default value (0x02) will be configured instead.

20.2.7.6 SPI_Cmd

Function Name	**void SPI_Cmd (*SPI_TypeDef* * SPIx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified SPI peripheral.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **NewState :** new state of the SPIx peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

20.2.7.7 SPI_TIModeCmd

Function Name	**void SPI_TIModeCmd (*SPI_TypeDef* * SPIx, *FunctionalState* NewState)**
Function Description	Enables or disables the TI Mode.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **NewState :** new state of the selected SPI TI communication mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function can be called only after the SPI_Init() function has been called. • When TI mode is selected, the control bits SSM, SSI, CPOL and CPHA are not taken into consideration and are configured by hardware respectively to the TI mode requirements.

20.2.7.8 I2S_Cmd

Function Name	**void I2S_Cmd (*SPI_TypeDef** * SPIx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified SPI peripheral (in I2S mode).
Parameters	• **SPIx:To :** select the SPIx peripheral, where x can be: 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode. • **NewState :** new state of the SPIx peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

20.2.7.9 SPI_DataSizeConfig

Function Name	**void SPI_DataSizeConfig (*SPI_TypeDef** * SPIx, uint16_t SPI_DataSize)**
Function Description	Configures the data size for the selected SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_DataSize :** specifies the SPI data size. For the SPIx peripheral this parameter can be one of the following values: – ***SPI_DataSize_4b :*** Set data size to 4 bits – ***SPI_DataSize_5b :*** Set data size to 5 bits – ***SPI_DataSize_6b :*** Set data size to 6 bits – ***SPI_DataSize_7b :*** Set data size to 7 bits – ***SPI_DataSize_8b :*** Set data size to 8 bits – ***SPI_DataSize_9b :*** Set data size to 9 bits – ***SPI_DataSize_10b :*** Set data size to 10 bits – ***SPI_DataSize_11b :*** Set data size to 11 bits – ***SPI_DataSize_12b :*** Set data size to 12 bits – ***SPI_DataSize_13b :*** Set data size to 13 bits – ***SPI_DataSize_14b :*** Set data size to 14 bits – ***SPI_DataSize_15b :*** Set data size to 15 bits – ***SPI_DataSize_16b :*** Set data size to 16 bits
Return values	• None.
Notes	• None.

20.2.7.10 SPI_RxFIFOThresholdConfig

Function Name	**void SPI_RxFIFOThresholdConfig (*SPI_TypeDef** * SPIx, uint16_t SPI_RxFIFOThreshold)**
Function Description	Configures the FIFO reception threshold for the selected SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_RxFIFOThreshold :** specifies the FIFO reception threshold. This parameter can be one of the following values: – ***SPI_RxFIFOThreshold_HF :*** RXNE event is generated if the FIFO level is greater or equal to 1/2. – ***SPI_RxFIFOThreshold_QF :*** RXNE event is generated if the FIFO level is greater or equal to 1/4.
Return values	• None.
Notes	• None.

20.2.7.11 SPI_BiDirectionalLineConfig

Function Name	**void SPI_BiDirectionalLineConfig (*SPI_TypeDef** * SPIx, uint16_t SPI_Direction)**
Function Description	Selects the data transfer direction in bidirectional mode for the specified SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_Direction :** specifies the data transfer direction in bidirectional mode. This parameter can be one of the following values: – ***SPI_Direction_Tx :*** Selects Tx transmission direction – ***SPI_Direction_Rx :*** Selects Rx receive direction
Return values	• None.
Notes	• None.

20.2.7.12 SPI_NSSInternalSoftwareConfig

Function Name	**void SPI_NSSInternalSoftwareConfig (*SPI_TypeDef* * SPIx, uint16_t SPI_NSSInternalSoft)**
Function Description	Configures internally by software the NSS pin for the selected SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_NSSInternalSoft :** specifies the SPI NSS internal state. This parameter can be one of the following values: – ***SPI_NSSInternalSoft_Set :*** Set NSS pin internally – ***SPI_NSSInternalSoft_Reset :*** Reset NSS pin internally
Return values	• None.
Notes	• This function can be called only after the SPI_Init() function has been called.

20.2.7.13 I2S_FullDuplexConfig

Function Name	**void I2S_FullDuplexConfig (*SPI_TypeDef* * I2Sxext, *I2S_InitTypeDef* * I2S_InitStruct)**
Function Description	Configures the full duplex mode for the I2Sx peripheral using its extension I2Sxext according to the specified parameters in the I2S_InitStruct.
Parameters	• **I2Sxext :** where x can be 2 or 3 to select the I2S peripheral extension block. • **I2S_InitStruct :** pointer to an I2S_InitTypeDef structure that contains the configuration information for the specified I2S peripheral extension.
Return values	• None.
Notes	• The structure pointed by I2S_InitStruct parameter should be the same used for the master I2S peripheral. In this case, if the master is configured as transmitter, the slave will be receiver and vice versa. Or you can force a different mode by modifying the field I2S_Mode to the value I2S_SlaveRx or I2S_SlaveTx indepedently of the master configuration. • The I2S full duplex extension can be configured in slave mode only.

20.2.7.14 SPI_SSOutputCmd

Function Name	**void SPI_SSOutputCmd (*SPI_TypeDef** * **SPIx, *FunctionalState* NewState)**
Function Description	Enables or disables the SS output for the selected SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **NewState :** new state of the SPIx SS output. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function can be called only after the SPI_Init() function has been called and the NSS hardware management mode is selected.

20.2.7.15 SPI_NSSPulseModeCmd

Function Name	**void SPI_NSSPulseModeCmd (*SPI_TypeDef** * **SPIx, *FunctionalState* NewState)**
Function Description	Enables or disables the NSS pulse management mode.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **NewState :** new state of the NSS pulse management mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function can be called only after the SPI_Init() function has been called. • When TI mode is selected, the control bits NSSP is not taken into consideration and are configured by hardware respectively to the TI mode requirements.

20.2.8 Data transfer functions

20.2.8.1 SPI_SendData8

Function Name	**void SPI_SendData8 (*SPI_TypeDef* * SPIx, uint8_t Data)**
Function Description	Transmits a Data through the SPIx peripheral.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **Data :** Data to be transmitted.
Return values	• None.
Notes	• None.

20.2.8.2 SPI_I2S_SendData16

Function Name	**void SPI_I2S_SendData16 (*SPI_TypeDef* * SPIx, uint16_t Data)**
Function Description	Transmits a Data through the SPIx/I2Sx peripheral.
Parameters	• **SPIx :** To select the SPIx/I2Sx peripheral, where x can be: 1, 2 or 3 in SPI mode or 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode. • **Data :** Data to be transmitted.
Return values	• None.
Notes	• None.

20.2.8.3 SPI_ReceiveData8

Function Name	**uint8_t SPI_ReceiveData8 (*SPI_TypeDef* * SPIx)**
Function Description	Returns the most recent received data by the SPIx peripheral.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral.
Return values	• **The value of the received data.**
Notes	• None.

20.2.8.4 SPI_I2S_ReceiveData16

Function Name	**uint16_t SPI_I2S_ReceiveData16 (*SPI_TypeDef** * SPIx)**
Function Description	Returns the most recent received data by the SPIx peripheral.
Parameters	• **SPIx :** To select the SPIx/I2Sx peripheral, where x can be: 1, 2 or 3 in SPI mode or 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode.
Return values	• **The value of the received data.**
Notes	• None.

20.2.9 Hardware CRC Calculation functions

20.2.9.1 SPI_CRCLengthConfig

Function Name	**void SPI_CRCLengthConfig (*SPI_TypeDef** * SPIx, uint16_t SPI_CRCLength)**
Function Description	Configures the CRC calculation length for the selected SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_CRCLength :** specifies the SPI CRC calculation length. This parameter can be one of the following values: – ***SPI_CRCLength_8b :*** Set CRC Calculation to 8 bits – ***SPI_CRCLength_16b :*** Set CRC Calculation to 16 bits
Return values	• None.
Notes	• None.

20.2.9.2 SPI_CalculateCRC

Function Name	**void SPI_CalculateCRC (*SPI_TypeDef* * SPIx, *FunctionalState* NewState)**
Function Description	Enables or disables the CRC value calculation of the transferred bytes.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **NewState :** new state of the SPIx CRC value calculation. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

20.2.9.3 SPI_TransmitCRC

Function Name	**void SPI_TransmitCRC (*SPI_TypeDef* * SPIx)**
Function Description	Transmits the SPIx CRC value.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral.
Return values	• None.
Notes	• None.

20.2.9.4 SPI_GetCRC

Function Name	**uint16_t SPI_GetCRC (*SPI_TypeDef* * SPIx, uint8_t SPI_CRC)**
Function Description	Returns the transmit or the receive CRC register value for the specified SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_CRC :** specifies the CRC register to be read. This parameter can be one of the following values: – ***SPI_CRC_Tx :*** Selects Tx CRC register – ***SPI_CRC_Rx :*** Selects Rx CRC register
Return values	• **The selected CRC register value..**

Notes	• None.

20.2.9.5 SPI_GetCRCPolynomial

Function Name	**uint16_t SPI_GetCRCPolynomial (*SPI_TypeDef* * SPIx)**
Function Description	Returns the CRC Polynomial register value for the specified SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral.
Return values	• **The CRC Polynomial register value.**
Notes	• None.

20.2.10 DMA transfers management functions

20.2.10.1 SPI_I2S_DMACmd

Function Name	**void SPI_I2S_DMACmd (*SPI_TypeDef* * SPIx, uint16_t SPI_I2S_DMAReq, *FunctionalState* NewState)**
Function Description	Enables or disables the SPIx/I2Sx DMA interface.
Parameters	• **SPIx:To :** select the SPIx/I2Sx peripheral, where x can be: 1, 2 or 3 in SPI mode or 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode. • **SPI_I2S_DMAReq :** specifies the SPI DMA transfer request to be enabled or disabled. This parameter can be any combination of the following values: – ***SPI_I2S_DMAReq_Tx :*** Tx buffer DMA transfer request – ***SPI_I2S_DMAReq_Rx :*** Rx buffer DMA transfer request • **NewState :** new state of the selected SPI DMA transfer request. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

20.2.10.2 SPI_LastDMATransferCmd

Function Name	**void SPI_LastDMATransferCmd (*SPI_TypeDef** * **SPIx, uint16_t SPI_LastDMATransfer)**
Function Description	Configures the number of data to transfer type(Even/Odd) for the DMA last transfers and for the selected SPI.
Parameters	• **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral. • **SPI_LastDMATransfer :** specifies the SPI last DMA transfers state. This parameter can be one of the following values: – ***SPI_LastDMATransfer_TxEvenRxEven :*** Number of data for transmission Even and number of data for reception Even. – ***SPI_LastDMATransfer_TxOddRxEven :*** Number of data for transmission Odd and number of data for reception Even. – ***SPI_LastDMATransfer_TxEvenRxOdd :*** Number of data for transmission Even and number of data for reception Odd. – ***SPI_LastDMATransfer_TxOddRxOdd :*** RNumber of data for transmission Odd and number of data for reception Odd.
Return values	• None.
Notes	• This function have a meaning only if DMA mode is selected and if the packing mode is used (data length <= 8 and DMA transfer size halfword)

20.2.11 Interrupts and flags management functions

20.2.11.1 SPI_I2S_ITConfig

Function Name	**void SPI_I2S_ITConfig (*SPI_TypeDef** * **SPIx, uint8_t SPI_I2S_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified SPI/I2S interrupts.
Parameters	• **SPIx :** To select the SPIx/I2Sx peripheral, where x can be: 1, 2 or 3 in SPI mode or 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode. • **SPI_I2S_IT :** specifies the SPI interrupt source to be enabled or disabled. This parameter can be one of the following values: – ***SPI_I2S_IT_TXE :*** Tx buffer empty interrupt mask – ***SPI_I2S_IT_RXNE :*** Rx buffer not empty interrupt mask

  - ***SPI_I2S_IT_ERR :*** Error interrupt mask
- **NewState :** new state of the specified SPI interrupt. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

20.2.11.2 SPI_GetTransmissionFIFOStatus

Function Name **uint16_t SPI_GetTransmissionFIFOStatus (*SPI_TypeDef** SPIx)**

Function Description Returns the current SPIx Transmission FIFO filled level.

Parameters
- **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral.

Return values
- **The Transmission FIFO filling state.**
  - ***SPI_TransmissionFIFOStatus_Empty: when FIFO is empty***
  - ***SPI_TransmissionFIFOStatus_1QuarterFull: if more than 1 quarter-full.***
  - ***SPI_TransmissionFIFOStatus_HalfFull: if more than 1 half-full.***
  - ***SPI_TransmissionFIFOStatus_Full: when FIFO is full.***

Notes
- None.

20.2.11.3 SPI_GetReceptionFIFOStatus

Function Name **uint16_t SPI_GetReceptionFIFOStatus (*SPI_TypeDef** SPIx)**

Function Description Returns the current SPIx Reception FIFO filled level.

Parameters
- **SPIx :** where x can be 1, 2 or 3 to select the SPI peripheral.

Return values
- **The Reception FIFO filling state.**
  - ***SPI_ReceptionFIFOStatus_Empty: when FIFO is empty***
  - ***SPI_ReceptionFIFOStatus_1QuarterFull: if more than 1 quarter-full.***
  - ***SPI_ReceptionFIFOStatus_HalfFull: if more than 1 half-full.***
  - ***SPI_ReceptionFIFOStatus_Full: when FIFO is full.***

Notes	• None.

20.2.11.4 SPI_I2S_GetFlagStatus

Function Name	**FlagStatus SPI_I2S_GetFlagStatus (*SPI_TypeDef** SPIx, uint16_t SPI_I2S_FLAG)**
Function Description	Checks whether the specified SPI flag is set or not.
Parameters	• **SPIx :** To select the SPIx/I2Sx peripheral, where x can be: 1, 2 or 3 in SPI mode or 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode. • **SPI_I2S_FLAG :** specifies the SPI flag to check. This parameter can be one of the following values: – ***SPI_I2S_FLAG_TXE :*** Transmit buffer empty flag. – ***SPI_I2S_FLAG_RXNE :*** Receive buffer not empty flag. – ***SPI_I2S_FLAG_BSY :*** Busy flag. – ***SPI_I2S_FLAG_OVR :*** Overrun flag. – ***SPI_I2S_FLAG_MODF :*** Mode Fault flag. – ***SPI_I2S_FLAG_CRCERR :*** CRC Error flag. – ***SPI_I2S_FLAG_FRE :*** TI frame format error flag. – ***I2S_FLAG_UDR :*** Underrun Error flag. – ***I2S_FLAG_CHSIDE :*** Channel Side flag.
Return values	• **The new state of SPI_I2S_FLAG (SET or RESET).**
Notes	• None.

20.2.11.5 SPI_I2S_ClearFlag

Function Name	**void SPI_I2S_ClearFlag (*SPI_TypeDef** SPIx, uint16_t SPI_I2S_FLAG)**
Function Description	Clears the SPIx CRC Error (CRCERR) flag.
Parameters	• **SPIx :** To select the SPIx/I2Sx peripheral, where x can be: 1, 2 or 3 in SPI mode or 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode. • **SPI_I2S_FLAG :** specifies the SPI flag to clear. This function clears only CRCERR flag.
Return values	• None.
Notes	• OVR (OverRun error) flag is cleared by software sequence: a

read operation to SPI_DR register (SPI_I2S_ReceiveData()) followed by a read operation to SPI_SR register (SPI_I2S_GetFlagStatus()).

- MODF (Mode Fault) flag is cleared by software sequence: a read/write operation to SPI_SR register (SPI_I2S_GetFlagStatus()) followed by a write operation to SPI_CR1 register (SPI_Cmd() to enable the SPI).

20.2.11.6 SPI_I2S_GetITStatus

Function Name	**ITStatus SPI_I2S_GetITStatus (*SPI_TypeDef** * SPIx, uint8_t SPI_I2S_IT)**
Function Description	Checks whether the specified SPI/I2S interrupt has occurred or not.
Parameters	• **SPIx :** To select the SPIx/I2Sx peripheral, where x can be: 1, 2 or 3 in SPI mode or 2 or 3 in I2S mode or I2Sxext for I2S full duplex mode. • **SPI_I2S_IT :** specifies the SPI interrupt source to check. This parameter can be one of the following values: – ***SPI_I2S_IT_TXE :*** Transmit buffer empty interrupt. – ***SPI_I2S_IT_RXNE :*** Receive buffer not empty interrupt. – ***SPI_IT_MODF :*** Mode Fault interrupt. – ***SPI_I2S_IT_OVR :*** Overrun interrupt. – ***I2S_IT_UDR :*** Underrun interrupt. – ***SPI_I2S_IT_FRE :*** Format Error interrupt.
Return values	• **The new state of SPI_I2S_IT (SET or RESET).**
Notes	• None.

20.3 SPI Firmware driver defines

20.3.1 SPI

SPI

SPI_BaudRate_Prescaler

- #define: ***SPI_BaudRatePrescaler_2 ((uint16_t)0x0000)***

- #define: ***SPI_BaudRatePrescaler_4 ((uint16_t)0x0008)***

- #define: ***SPI_BaudRatePrescaler_8 ((uint16_t)0x0010)***

- #define: ***SPI_BaudRatePrescaler_16 ((uint16_t)0x0018)***

- #define: ***SPI_BaudRatePrescaler_32 ((uint16_t)0x0020)***

- #define: ***SPI_BaudRatePrescaler_64 ((uint16_t)0x0028)***

- #define: ***SPI_BaudRatePrescaler_128 ((uint16_t)0x0030)***

- #define: ***SPI_BaudRatePrescaler_256 ((uint16_t)0x0038)***

SPI_Clock_Phase

- #define: ***SPI_CPHA_1Edge ((uint16_t)0x0000)***

- #define: ***SPI_CPHA_2Edge ((uint16_t)0x0001)***

SPI_Clock_Polarity

- #define: ***SPI_CPOL_Low ((uint16_t)0x0000)***

- #define: ***SPI_CPOL_High ((uint16_t)0x0002)***

SPI_CRC_length

- #define: ***SPI_CRCLength_8b ((uint16_t)0x0000)***

- #define: ***SPI_CRCLength_16b ((uint16_t)0x0800)***

SPI_CRC_Transmit_Receive

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

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

SPI_data_direction

- #define: ***SPI_Direction_2Lines_FullDuplex ((uint16_t)0x0000)***

- #define: ***SPI_Direction_2Lines_RxOnly ((uint16_t)0x0400)***

- #define: ***SPI_Direction_1Line_Rx ((uint16_t)0x8000)***

- #define: ***SPI_Direction_1Line_Tx ((uint16_t)0xC000)***

SPI_data_size

- #define: ***SPI_DataSize_4b ((uint16_t)0x0300)***

- #define: ***SPI_DataSize_5b ((uint16_t)0x0400)***

- #define: ***SPI_DataSize_6b ((uint16_t)0x0500)***

- #define: ***SPI_DataSize_7b ((uint16_t)0x0600)***

- #define: ***SPI_DataSize_8b ((uint16_t)0x0700)***

- #define: ***SPI_DataSize_9b ((uint16_t)0x0800)***

- #define: ***SPI_DataSize_10b ((uint16_t)0x0900)***

- #define: ***SPI_DataSize_11b ((uint16_t)0x0A00)***

- #define: ***SPI_DataSize_12b ((uint16_t)0x0B00)***

- #define: ***SPI_DataSize_13b ((uint16_t)0x0C00)***

- #define: ***SPI_DataSize_14b ((uint16_t)0x0D00)***

- #define: ***SPI_DataSize_15b ((uint16_t)0x0E00)***

- #define: ***SPI_DataSize_16b ((uint16_t)0x0F00)***

SPI_direction_transmit_receive

- #define: ***SPI_Direction_Rx ((uint16_t)0xBFFF)***

- #define: ***SPI_Direction_Tx ((uint16_t)0x4000)***

SPI_FIFO_reception_threshold

- #define: ***SPI_RxFIFOThreshold_HF ((uint16_t)0x0000)***

- #define: ***SPI_RxFIFOThreshold_QF ((uint16_t)0x1000)***

SPI_I2S_DMA_transfer_requests

- #define: ***SPI_I2S_DMAReq_Tx ((uint16_t)0x0002)***

- #define: ***SPI_I2S_DMAReq_Rx ((uint16_t)0x0001)***

SPI_I2S_flags_definition

- #define: ***SPI_I2S_FLAG_RXNE ((uint16_t)0x0001)***

- #define: ***SPI_I2S_FLAG_TXE ((uint16_t)0x0002)***

- #define: ***I2S_FLAG_CHSIDE ((uint16_t)0x0004)***

- #define: ***I2S_FLAG_UDR ((uint16_t)0x0008)***

- #define: ***SPI_FLAG_CRCERR ((uint16_t)0x0010)***

- #define: ***SPI_FLAG_MODF ((uint16_t)0x0020)***

- #define: ***SPI_I2S_FLAG_OVR ((uint16_t)0x0040)***

- #define: ***SPI_I2S_FLAG_BSY ((uint16_t)0x0080)***

- #define: ***SPI_I2S_FLAG_FRE ((uint16_t)0x0100)***

SPI_I2S_interrupts_definition

- #define: ***SPI_I2S_IT_TXE ((uint8_t)0x71)***

- #define: ***SPI_I2S_IT_RXNE ((uint8_t)0x60)***

- #define: ***SPI_I2S_IT_ERR ((uint8_t)0x50)***

- #define: ***I2S_IT_UDR ((uint8_t)0x53)***

- #define: ***SPI_IT_MODF ((uint8_t)0x55)***

- #define: ***SPI_I2S_IT_OVR ((uint8_t)0x56)***

- #define: ***SPI_I2S_IT_FRE ((uint8_t)0x58)***

SPI_last_DMA_transfers

- #define: ***SPI_LastDMATransfer_TxEvenRxEven ((uint16_t)0x0000)***

- #define: ***SPI_LastDMATransfer_TxOddRxEven ((uint16_t)0x4000)***

- #define: ***SPI_LastDMATransfer_TxEvenRxOdd ((uint16_t)0x2000)***

- #define: ***SPI_LastDMATransfer_TxOddRxOdd ((uint16_t)0x6000)***

SPI_mode

- #define: ***SPI_Mode_Master ((uint16_t)0x0104)***

- #define: ***SPI_Mode_Slave ((uint16_t)0x0000)***

SPI_MSB_LSB_transmission

- #define: ***SPI_FirstBit_MSB ((uint16_t)0x0000)***

- #define: ***SPI_FirstBit_LSB ((uint16_t)0x0080)***

SPI_NSS_internal_software_management

- #define: ***SPI_NSSInternalSoft_Set ((uint16_t)0x0100)***

- #define: ***SPI_NSSInternalSoft_Reset ((uint16_t)0xFEFF)***

SPI_reception_fifo_status_level

- #define: ***SPI_ReceptionFIFOStatus_Empty ((uint16_t)0x0000)***

- #define: ***SPI_ReceptionFIFOStatus_1QuarterFull ((uint16_t)0x0200)***

- #define: ***SPI_ReceptionFIFOStatus_HalfFull ((uint16_t)0x0400)***

- #define: ***SPI_ReceptionFIFOStatus_Full ((uint16_t)0x0600)***

SPI_Slave_Select_management

- #define: ***SPI_NSS_Soft ((uint16_t)0x0200)***

- #define: ***SPI_NSS_Hard ((uint16_t)0x0000)***

SPI_transmission_fifo_status_level

- #define: ***SPI_TransmissionFIFOStatus_Empty ((uint16_t)0x0000)***

- #define: ***SPI_TransmissionFIFOStatus_1QuarterFull ((uint16_t)0x0800)***

- #define: ***SPI_TransmissionFIFOStatus_HalfFull ((uint16_t)0x1000)***

- #define: ***SPI_TransmissionFIFOStatus_Full ((uint16_t)0x1800)***

21 System configuration controller (SYSCFG)

21.1 SYSCFG Firmware driver registers structures

21.1.1 SYSCFG_TypeDef

SYSCFG_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CFGR1***
- ***__IO uint32_t RCR***
- ***__IO uint32_t EXTICR***
- ***__IO uint32_t CFGR2***

Field Documentation

- ***__IO uint32_t SYSCFG_TypeDef::CFGR1***
  - SYSCFG configuration register 1, Address offset: 0x00
- ***__IO uint32_t SYSCFG_TypeDef::RCR***
  - SYSCFG CCM SRAM protection register, Address offset: 0x04
- ***__IO uint32_t SYSCFG_TypeDef::EXTICR[4]***
  - SYSCFG external interrupt configuration registers, Address offset: 0x14-0x08
- ***__IO uint32_t SYSCFG_TypeDef::CFGR2***
  - SYSCFG configuration register 2, Address offset: 0x18

21.2 SYSCFG Firmware driver API description

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

21.2.1 How to use this driver

The SYSCFG registers can be accessed only when the SYSCFG interface APB clock is enabled.

To enable SYSCFG APB clock use:
RCC_APBPeriphClockCmd(RCC_APBPeriph_SYSCFG, ENABLE);

21.2.2 SYSCFG Initialization and Configuration functions

- ***SYSCFG_DeInit()***
- ***SYSCFG_MemoryRemapConfig()***
- ***SYSCFG_DMAChannelRemapConfig()***
- ***SYSCFG_TriggerRemapConfig()***
- ***SYSCFG_EncoderRemapConfig()***
- ***SYSCFG_USBInterruptLineRemapCmd()***

- *SYSCFG_I2CFastModePlusConfig()*
- *SYSCFG_ITConfig()*
- *SYSCFG_EXTILineConfig()*
- *SYSCFG_BreakConfig()*
- *SYSCFG_BypassParityCheckDisable()*
- *SYSCFG_SRAMWRPEnable()*
- *SYSCFG_GetFlagStatus()*
- *SYSCFG_ClearFlag()*

21.2.3 SYSCFG initialization and configuration functions

21.2.3.1 SYSCFG_DeInit

Function Name	**void SYSCFG_DeInit (void)**
Function Description	Deinitializes the SYSCFG registers to their default reset values.
Parameters	• None.
Return values	• None.
Notes	• MEM_MODE bits are not affected by APB reset. MEM_MODE bits took the value from the user option bytes.

21.2.3.2 SYSCFG_MemoryRemapConfig

Function Name	**void SYSCFG_MemoryRemapConfig (uint32_t SYSCFG_MemoryRemap)**
Function Description	Configures the memory mapping at address 0x00000000.
Parameters	• **SYSCFG_MemoryRemap :** selects the memory remapping. This parameter can be one of the following values: – ***SYSCFG_MemoryRemap_Flash :*** Main Flash memory mapped at 0x00000000 – ***SYSCFG_MemoryRemap_SystemMemory :*** System Flash memory mapped at 0x00000000 – ***SYSCFG_MemoryRemap_SRAM :*** Embedded SRAM mapped at 0x00000000
Return values	• None.
Notes	• None.

21.2.3.3 SYSCFG_DMAChannelRemapConfig

Function Name	**void SYSCFG_DMAChannelRemapConfig (uint32_t SYSCFG_DMARemap, *FunctionalState* NewState)**
Function Description	Configures the DMA channels remapping.
Parameters	• **SYSCFG_DMARemap :** selects the DMA channels remap. This parameter can be one of the following values: – ***SYSCFG_DMARemap_TIM17 :*** Remap TIM17 DMA requests from DMA1 channel1 to channel2 – ***SYSCFG_DMARemap_TIM16 :*** Remap TIM16 DMA requests from DMA1 channel3 to channel4 – ***SYSCFG_DMARemap_TIM6DAC1 :*** Remap TIM6/DAC1 DMA requests from DMA2 channel 3 to DMA1 channel 3 – ***SYSCFG_DMARemap_TIM7DAC2 :*** Remap TIM7/DAC2 DMA requests from DMA2 channel 4 to DMA1 channel 4 – ***SYSCFG_DMARemap_ADC2ADC4 :*** Remap ADC2 and ADC4 DMA requests from DMA2 channel1/channel3 to channel3/channel4 • **NewState :** new state of the DMA channel remapping. This parameter can be: Enable or Disable.
Return values	• None.
Notes	• When enabled, DMA channel of the selected peripheral is remapped • When disabled, Default DMA channel is mapped to the selected peripheral • By default TIM17 DMA requests is mapped to channel 1 use SYSCFG_DMAChannelRemapConfig(SYSCFG_DMARemap_TIM17, Enable) to remap TIM17 DMA requests to DMA1 channel 2 use SYSCFG_DMAChannelRemapConfig(SYSCFG_DMARemap_TIM17, Disable) to map TIM17 DMA requests to DMA1 channel 1 (default mapping)

21.2.3.4 SYSCFG_TriggerRemapConfig

Function Name	**void SYSCFG_TriggerRemapConfig (uint32_t SYSCFG_TriggerRemap, *FunctionalState* NewState)**
Function Description	Configures the remapping capabilities of DAC/TIM triggers.
Parameters	• **SYSCFG_TriggerRemap :** selects the trigger to be remapped. This parameter can be one of the following values: – ***SYSCFG_TriggerRemap_DACTIM3 :*** Remap DAC

trigger from TIM8 to TIM3
  - ***SYSCFG_TriggerRemap_TIM1TIM17 :*** Remap TIM1 ITR3 from TIM4 TRGO to TIM17 OC
- **NewState :** new state of the trigger mapping. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- ENABLE: Enable fast mode plus driving capability for selected pin
- DISABLE: Disable fast mode plus driving capability for selected pin

21.2.3.5 SYSCFG_EncoderRemapConfig

Function Name **void SYSCFG_EncoderRemapConfig (uint32_t SYSCFG_EncoderRemap)**

Function Description Configures the remapping capabilities of encoder mode.

Parameters
- **SYSCFG_EncoderRemap :** selects the remap option for encoder mode. This parameter can be one of the following values:
  - ***SYSCFG_EncoderRemap_No :*** No remap
  - ***SYSCFG_EncoderRemap_TIM2 :*** Timer 2 IC1 and IC2 connected to TIM15 IC1 and IC2
  - ***SYSCFG_EncoderRemap_TIM3 :*** Timer 3 IC1 and IC2 connected to TIM15 IC1 and IC2
  - ***SYSCFG_EncoderRemap_TIM4 :*** Timer 4 IC1 and IC2 connected to TIM15 IC1 and IC2

Return values
- None.

Notes
- None.

21.2.3.6 SYSCFG_USBInterruptLineRemapCmd

Function Name **void SYSCFG_USBInterruptLineRemapCmd (*FunctionalState* NewState)**

Function Description Remaps the USB interrupt lines.

Parameters
- **NewState :** new state of the mapping of USB interrupt lines. This parameter can be:

- **ENABLE :** Remap the USB interrupt line as following:
  - ***USB :***
  - ***USB :***
  - ***USB :***
- **DISABLE :** Use the default USB interrupt line:
  - ***USB :***
  - ***USB :***
  - ***USB :***

Return values
- None.

Notes
- None.

21.2.3.7 SYSCFG_I2CFastModePlusConfig

Function Name **void SYSCFG_I2CFastModePlusConfig (uint32_t SYSCFG_I2CFastModePlus, *FunctionalState* NewState)**

Function Description Configures the I2C fast mode plus driving capability.

Parameters
- **SYSCFG_I2CFastModePlus :** selects the pin. This parameter can be one of the following values:
  - ***SYSCFG_I2CFastModePlus_PB6 :*** Configure fast mode plus driving capability for PB6
  - ***SYSCFG_I2CFastModePlus_PB7 :*** Configure fast mode plus driving capability for PB7
  - ***SYSCFG_I2CFastModePlus_PB8 :*** Configure fast mode plus driving capability for PB8
  - ***SYSCFG_I2CFastModePlus_PB9 :*** Configure fast mode plus driving capability for PB9
  - ***SYSCFG_I2CFastModePlus_I2C1 :*** Configure fast mode plus driving capability for I2C1 pins
  - ***SYSCFG_I2CFastModePlus_I2C2 :*** Configure fast mode plus driving capability for I2C2 pins
- **NewState :** new state of the DMA channel remapping. This parameter can be:
  - ***ENABLE :*** Enable fast mode plus driving capability for selected I2C pin
  - ***DISABLE :*** Disable fast mode plus driving capability for selected I2C pin

Return values
- None.

Notes
- For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using SYSCFG_I2CFastModePlus_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaing I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using

SYSCFG_I2CFastModePlus_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be enabled only by using SYSCFG_I2CFastModePlus_I2C2 parameter.

21.2.3.8 SYSCFG_ITConfig

Function Name	**void SYSCFG_ITConfig (uint32_t SYSCFG_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the selected SYSCFG interrupts.
Parameters	• **SYSCFG_IT :** specifies the SYSCFG interrupt sources to be enabled or disabled. This parameter can be one of the following values: – ***SYSCFG_IT_IXC :*** Inexact Interrupt – ***SYSCFG_IT_IDC :*** Input denormal Interrupt – ***SYSCFG_IT_OFC :*** Overflow Interrupt – ***SYSCFG_IT_UFC :*** Underflow Interrupt – ***SYSCFG_IT_DZC :*** Divide-by-zero Interrupt – ***SYSCFG_IT_IOC :*** Invalid operation Interrupt • **NewState :** new state of the specified SDADC interrupts. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

21.2.3.9 SYSCFG_EXTILineConfig

Function Name	**void SYSCFG_EXTILineConfig (uint8_t EXTI_PortSourceGPIOx, uint8_t EXTI_PinSourcex)**
Function Description	Selects the GPIO pin used as EXTI Line.
Parameters	• **EXTI_PortSourceGPIOx :** : selects the GPIO port to be used as source for EXTI lines where x can be (A, B, C, D, E or F). • **EXTI_PinSourcex :** specifies the EXTI line to be configured. This parameter can be EXTI_PinSourcex where x can be (0..15)
Return values	• None.
Notes	• None.

21.2.3.10 SYSCFG_BreakConfig

Function Name	**void SYSCFG_BreakConfig (uint32_t SYSCFG_Break)**
Function Description	Connects the selected parameter to the break input of TIM1.
Parameters	• **SYSCFG_Break :** selects the configuration to be connected to break input of TIM1 This parameter can be any combination of the following values: – ***SYSCFG_Break_PVD :*** PVD interrupt is connected to the break input of TIM1. – ***SYSCFG_Break_SRAMParity :*** SRAM Parity error is connected to the break input of TIM1. – ***SYSCFG_Break_HardFault :*** Lockup output of CortexM0 is connected to the break input of TIM1.
Return values	• None.
Notes	• The selected configuration is locked and can be unlocked by system reset

21.2.3.11 SYSCFG_BypassParityCheckDisable

Function Name	**void SYSCFG_BypassParityCheckDisable (void)**
Function Description	Disables the parity check on RAM.
Parameters	• None.
Return values	• None.
Notes	• Disabling the parity check on RAM locks the configuration bit. To re-enable the parity check on RAM perform a system reset.

21.2.3.12 SYSCFG_SRAMWRPEnable

Function Name	**void SYSCFG_SRAMWRPEnable (uint32_t SYSCFG_SRAMWRP)**
Function Description	Enables the ICODE SRAM write protection.
Parameters	• None.
Return values	• None.
Notes	• Enabling the ICODE SRAM write protection locks the configuration bit. To disable the ICODE SRAM write protection perform a system reset.

21.2.3.13 SYSCFG_GetFlagStatus

Function Name	**FlagStatus SYSCFG_GetFlagStatus (uint32_t SYSCFG_Flag)**
Function Description	Checks whether the specified SYSCFG flag is set or not.
Parameters	• **SYSCFG_Flag :** specifies the SYSCFG flag to check. This parameter can be one of the following values: – ***SYSCFG_FLAG_PE :*** SRAM parity error flag.
Return values	• **The new state of SYSCFG_Flag (SET or RESET).**
Notes	• None.

21.2.3.14 SYSCFG_ClearFlag

Function Name	**void SYSCFG_ClearFlag (uint32_t SYSCFG_Flag)**
Function Description	Clears the selected SYSCFG flag.
Parameters	• **SYSCFG_Flag :** selects the flag to be cleared. This parameter can be any combination of the following values: – ***SYSCFG_FLAG_PE :*** SRAM parity error flag.
Return values	• None.
Notes	• None.

21.3 SYSCFG Firmware driver defines

21.3.1 SYSCFG

SYSCFG

SYSCFG_DMA_Remap_Config

- #define: ***SYSCFG_DMARemap_TIM17 SYSCFG_CFGR1_TIM17_DMA_RMP***

Remap TIM17 DMA requests from channel1 to channel2

- #define: ***SYSCFG_DMARemap_TIM16 SYSCFG_CFGR1_TIM16_DMA_RMP***

Remap TIM16 DMA requests from channel3 to channel4

- #define: ***SYSCFG_DMARemap_TIM6DAC1 SYSCFG_CFGR1_TIM6DAC1_DMA_RMP***

Remap TIM6/DAC1 DMA requests from DMA2 channel3 to DMA1 channel3

- #define: ***SYSCFG_DMARemap_TIM7DAC2 SYSCFG_CFGR1_TIM7DAC2_DMA_RMP***

Remap TIM7/DAC2 DMA requests from DMA2 channel4 to DMA1 channel4

- #define: ***SYSCFG_DMARemap_ADC2ADC4 SYSCFG_CFGR1_ADC24_DMA_RMP***

Remap ADC2 and ADC4 DMA requests from DMA2 channel1/channel3 to channel3/channel4

SYSCFG_EncoderRemap_Config

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

No redirection

- #define: ***SYSCFG_EncoderRemap_TIM2 SYSCFG_CFGR1_ENCODER_MODE_0***

Timer 2 IC1 and IC2 connected to TIM15 IC1 and IC2

- #define: ***SYSCFG_EncoderRemap_TIM3 SYSCFG_CFGR1_ENCODER_MODE_1***

Timer 3 IC1 and IC2 connected to TIM15 IC1 and IC2

- #define: ***SYSCFG_EncoderRemap_TIM4 SYSCFG_CFGR1_ENCODER_MODE***

Timer 4 IC1 and IC2 connected to TIM15 IC1 and IC2

SYSCFG_EXTI_Pin_sources

- #define: ***EXTI_PinSource0 ((uint8_t)0x00)***
- #define: ***EXTI_PinSource1 ((uint8_t)0x01)***
- #define: ***EXTI_PinSource2 ((uint8_t)0x02)***
- #define: ***EXTI_PinSource3 ((uint8_t)0x03)***
- #define: ***EXTI_PinSource4 ((uint8_t)0x04)***
- #define: ***EXTI_PinSource5 ((uint8_t)0x05)***
- #define: ***EXTI_PinSource6 ((uint8_t)0x06)***
- #define: ***EXTI_PinSource7 ((uint8_t)0x07)***
- #define: ***EXTI_PinSource8 ((uint8_t)0x08)***
- #define: ***EXTI_PinSource9 ((uint8_t)0x09)***
- #define: ***EXTI_PinSource10 ((uint8_t)0x0A)***
- #define: ***EXTI_PinSource11 ((uint8_t)0x0B)***

- #define: ***EXTI_PinSource12 ((uint8_t)0x0C)***

- #define: ***EXTI_PinSource13 ((uint8_t)0x0D)***

- #define: ***EXTI_PinSource14 ((uint8_t)0x0E)***

- #define: ***EXTI_PinSource15 ((uint8_t)0x0F)***

SYSCFG_EXTI_Port_Sources

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

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

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

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

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

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

SYSCFG_flags_definition

- #define: ***SYSCFG_FLAG_PE SYSCFG_CFGR2_SRAM_PE***

SYSCFG_FPU_Interrupt_Config

- #define: ***SYSCFG_IT_IXC SYSCFG_CFGR1_FPU_IE_5***

Inexact Interrupt enable (interrupt disabled by default)

- #define: ***SYSCFG_IT_IDC SYSCFG_CFGR1_FPU_IE_4***

Input denormal Interrupt enable

- #define: ***SYSCFG_IT_OFC SYSCFG_CFGR1_FPU_IE_3***

Overflow Interrupt enable

- #define: ***SYSCFG_IT_UFC SYSCFG_CFGR1_FPU_IE_2***

Underflow Interrupt enable

- #define: ***SYSCFG_IT_DZC SYSCFG_CFGR1_FPU_IE_1***

Divide-by-zero Interrupt enable

- #define: ***SYSCFG_IT_IOC SYSCFG_CFGR1_FPU_IE_0***

Invalid operation Interrupt enable

SYSCFG_I2C_FastModePlus_Config

- #define: ***SYSCFG_I2CFastModePlus_PB6 SYSCFG_CFGR1_I2C_PB6_FMP***

Enable Fast Mode Plus on PB6

- #define: ***SYSCFG_I2CFastModePlus_PB7 SYSCFG_CFGR1_I2C_PB7_FMP***

Enable Fast Mode Plus on PB7

- #define: ***SYSCFG_I2CFastModePlus_PB8 SYSCFG_CFGR1_I2C_PB8_FMP***

Enable Fast Mode Plus on PB8

- #define: ***SYSCFG_I2CFastModePlus_PB9 SYSCFG_CFGR1_I2C_PB9_FMP***

Enable Fast Mode Plus on PB9

- #define: ***SYSCFG_I2CFastModePlus_I2C1 SYSCFG_CFGR1_I2C1_FMP***

Enable Fast Mode Plus on I2C1 pins

- #define: ***SYSCFG_I2CFastModePlus_I2C2 SYSCFG_CFGR1_I2C2_FMP***

Enable Fast Mode Plus on I2C2 pins

SYSCFG_Lock_Config

- #define: ***SYSCFG_Break_PVD SYSCFG_CFGR2_PVD_LOCK***

Enables and locks the PVD connection with TIM1/8/15/16/17 Break Input and also the PVD_EN and PVDSEL[2:0] bits of the Power Control Interface

- #define: ***SYSCFG_Break_SRAMParity SYSCFG_CFGR2_SRAM_PARITY_LOCK***

Enables and locks the SRAM_PARITY error signal with Break Input of TIM1/8/15/16/17

- #define: ***SYSCFG_Break_Lockup SYSCFG_CFGR2_LOCKUP_LOCK***

Enables and locks the LOCKUP output of CortexM0 with Break Input of TIM1/8/15/16/17

SYSCFG_Memory_Remap_Config

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

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

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

SYSCFG_SRAMWRP_Config

- #define: ***SYSCFG_SRAMWRP_Page0 SYSCFG_RCR_PAGE0***

ICODE SRAM Write protection page 0

- #define: ***SYSCFG_SRAMWRP_Page1 SYSCFG_RCR_PAGE1***

ICODE SRAM Write protection page 1

- #define: ***SYSCFG_SRAMWRP_Page2 SYSCFG_RCR_PAGE2***

ICODE SRAM Write protection page 2

- #define: ***SYSCFG_SRAMWRP_Page3 SYSCFG_RCR_PAGE3***

ICODE SRAM Write protection page 3

- #define: ***SYSCFG_SRAMWRP_Page4 SYSCFG_RCR_PAGE4***

ICODE SRAM Write protection page 4

- #define: ***SYSCFG_SRAMWRP_Page5 SYSCFG_RCR_PAGE5***

ICODE SRAM Write protection page 5

- #define: ***SYSCFG_SRAMWRP_Page6 SYSCFG_RCR_PAGE6***

ICODE SRAM Write protection page 6

- #define: ***SYSCFG_SRAMWRP_Page7 SYSCFG_RCR_PAGE7***

ICODE SRAM Write protection page 7

SYSCFG_Trigger_Remap_Config

- #define: ***SYSCFG_TriggerRemap_DACTIM3 SYSCFG_CFGR1_DAC_TRIG_RMP***

Remap DAC trigger to TIM3

- #define: ***SYSCFG_TriggerRemap_TIM1TIM17 SYSCFG_CFGR1_TIM1_ITR3_RMP***

Remap TIM1 ITR3 to TIM17 OC

22 General-purpose timers (TIM)

22.1 TIM Firmware driver registers structures

22.1.1 TIM_TypeDef

TIM_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint16_t CR1***
- ***uint16_t RESERVED0***
- ***__IO uint32_t CR2***
- ***__IO uint32_t SMCR***
- ***__IO uint32_t DIER***
- ***__IO uint32_t SR***
- ***__IO uint32_t EGR***
- ***__IO uint32_t CCMR1***
- ***__IO uint32_t CCMR2***
- ***__IO uint32_t CCER***
- ***__IO uint32_t CNT***
- ***__IO uint16_t PSC***
- ***uint16_t RESERVED9***
- ***__IO uint32_t ARR***
- ***__IO uint16_t RCR***
- ***uint16_t RESERVED10***
- ***__IO uint32_t CCR1***
- ***__IO uint32_t CCR2***
- ***__IO uint32_t CCR3***
- ***__IO uint32_t CCR4***
- ***__IO uint32_t BDTR***
- ***__IO uint16_t DCR***
- ***uint16_t RESERVED12***
- ***__IO uint16_t DMAR***
- ***uint16_t RESERVED13***
- ***__IO uint16_t OR***
- ***__IO uint32_t CCMR3***
- ***__IO uint32_t CCR5***
- ***__IO uint32_t CCR6***

Field Documentation

- ***__IO uint16_t TIM_TypeDef::CR1***
  - TIM control register 1, Address offset: 0x00
- ***uint16_t TIM_TypeDef::RESERVED0***
  - Reserved, 0x02
- ***__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
- ***__IO uint32_t TIM_TypeDef::DIER***
  - TIM DMA/interrupt enable register, Address offset: 0x0C
- ***__IO uint32_t TIM_TypeDef::SR***
  - TIM status register, Address offset: 0x10
- ***__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 uint16_t TIM_TypeDef::PSC***
  - TIM prescaler, Address offset: 0x28
- ***uint16_t TIM_TypeDef::RESERVED9***
  - Reserved, 0x2A
- ***__IO uint32_t TIM_TypeDef::ARR***
  - TIM auto-reload register, Address offset: 0x2C
- ***__IO uint16_t TIM_TypeDef::RCR***
  - TIM repetition counter register, Address offset: 0x30
- ***uint16_t TIM_TypeDef::RESERVED10***
  - Reserved, 0x32
- ***__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
- ***__IO uint32_t TIM_TypeDef::BDTR***
  - TIM break and dead-time register, Address offset: 0x44
- ***__IO uint16_t TIM_TypeDef::DCR***
  - TIM DMA control register, Address offset: 0x48
- ***uint16_t TIM_TypeDef::RESERVED12***
  - Reserved, 0x4A
- ***__IO uint16_t TIM_TypeDef::DMAR***
  - TIM DMA address for full transfer, Address offset: 0x4C
- ***uint16_t TIM_TypeDef::RESERVED13***
  - Reserved, 0x4E
- ***__IO uint16_t TIM_TypeDef::OR***
  - TIM option register, Address offset: 0x50
- ***__IO uint32_t TIM_TypeDef::CCMR3***
  - TIM capture/compare mode register 3, Address offset: 0x54
- ***__IO uint32_t TIM_TypeDef::CCR5***
  - TIM capture/compare register5, Address offset: 0x58
- ***__IO uint32_t TIM_TypeDef::CCR6***
  - TIM capture/compare register 4, Address offset: 0x5C

22.1.2 TIM_TimeBaseInitTypeDef

TIM_TimeBaseInitTypeDef is defined in the stm32f30x_tim.h

Data Fields

- ***uint16_t TIM_Prescaler***
- ***uint16_t TIM_CounterMode***
- ***uint32_t TIM_Period***
- ***uint16_t TIM_ClockDivision***
- ***uint8_t TIM_RepetitionCounter***

Field Documentation

- ***uint16_t TIM_TimeBaseInitTypeDef::TIM_Prescaler***
  - Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between 0x0000 and 0xFFFF
- ***uint16_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.
- ***uint16_t TIM_TimeBaseInitTypeDef::TIM_ClockDivision***
  - Specifies the clock division. This parameter can be a value of ***TIM_Clock_Division_CKD***
- ***uint8_t TIM_TimeBaseInitTypeDef::TIM_RepetitionCounter***
  - Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to: the number of PWM periods in edge-aligned modethe number of half PWM period in center-aligned mode This parameter must be a number between 0x00 and 0xFF. This parameter is valid only for TIM1 and TIM8.

22.1.3 TIM_OCInitTypeDef

TIM_OCInitTypeDef is defined in the stm32f30x_tim.h

Data Fields

- ***uint32_t TIM_OCMode***
- ***uint16_t TIM_OutputState***
- ***uint16_t TIM_OutputNState***
- ***uint32_t TIM_Pulse***
- ***uint16_t TIM_OCPolarity***
- ***uint16_t TIM_OCNPolarity***
- ***uint16_t TIM_OCIdleState***
- ***uint16_t TIM_OCNIdleState***

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***
- ***uint16_t TIM_OCInitTypeDef::TIM_OutputState***
  - Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_Output_Compare_State***
- ***uint16_t TIM_OCInitTypeDef::TIM_OutputNState***
  - Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_Output_Compare_N_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
- ***uint16_t TIM_OCInitTypeDef::TIM_OCPolarity***
  - Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint16_t TIM_OCInitTypeDef::TIM_OCNPolarity***
  - Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
- ***uint16_t TIM_OCInitTypeDef::TIM_OCIdleState***
  - Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
- ***uint16_t TIM_OCInitTypeDef::TIM_OCNIdleState***
  - Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***

22.1.4 TIM_ICInitTypeDef

TIM_ICInitTypeDef is defined in the stm32f30x_tim.h

Data Fields

- ***uint16_t TIM_Channel***
- ***uint16_t TIM_ICPolarity***
- ***uint16_t TIM_ICSelection***
- ***uint16_t TIM_ICPrescaler***
- ***uint16_t TIM_ICFilter***

Field Documentation

- ***uint16_t TIM_ICInitTypeDef::TIM_Channel***
  - Specifies the TIM channel. This parameter can be a value of ***TIM_Channel***
- ***uint16_t TIM_ICInitTypeDef::TIM_ICPolarity***
  - Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint16_t TIM_ICInitTypeDef::TIM_ICSelection***
  - Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***

- ***uint16_t TIM_ICInitTypeDef::TIM_ICPrescaler***
  - Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint16_t TIM_ICInitTypeDef::TIM_ICFilter***
  - Specifies the input capture filter. This parameter can be a number between 0x0 and 0xF

22.1.5 TIM_BDTRInitTypeDef

TIM_BDTRInitTypeDef is defined in the stm32f30x_tim.h

Data Fields

- ***uint16_t TIM_OSSRState***
- ***uint16_t TIM_OSSIState***
- ***uint16_t TIM_LOCKLevel***
- ***uint16_t TIM_DeadTime***
- ***uint16_t TIM_Break***
- ***uint16_t TIM_BreakPolarity***
- ***uint16_t TIM_AutomaticOutput***

Field Documentation

- ***uint16_t TIM_BDTRInitTypeDef::TIM_OSSRState***
  - Specifies the Off-State selection used in Run mode. This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint16_t TIM_BDTRInitTypeDef::TIM_OSSIState***
  - Specifies the Off-State used in Idle state. This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint16_t TIM_BDTRInitTypeDef::TIM_LOCKLevel***
  - Specifies the LOCK level parameters. This parameter can be a value of ***TIM_Lock_level***
- ***uint16_t TIM_BDTRInitTypeDef::TIM_DeadTime***
  - Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between 0x00 and 0xFF
- ***uint16_t TIM_BDTRInitTypeDef::TIM_Break***
  - Specifies whether the TIM Break input is enabled or not. This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint16_t TIM_BDTRInitTypeDef::TIM_BreakPolarity***
  - Specifies the TIM Break Input pin polarity. This parameter can be a value of ***TIM_Break_Polarity***
- ***uint16_t TIM_BDTRInitTypeDef::TIM_AutomaticOutput***
  - Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

22.2 TIM Firmware driver API description

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

22.2.1 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. Enable TIM clock using RCC_APBxPeriphClockCmd(RCC_APBxPeriph_TIMx, ENABLE) function
2. Configure the TIM pins by configuring the corresponding GPIO pins
3. 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
   - Clock Division = TIM_CKD_DIV1
4. 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
5. Call TIM_OCxInit(TIMx, &TIM_OCInitStruct) to configure the desired channel with the corresponding configuration
6. 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 or DMA request are needed, the user should:

1. Enable the NVIC (or the DMA) to use the TIM interrupts (or DMA requests).
2. Enable the corresponding interrupt (or DMA request) using the function TIM_ITConfig(TIMx, TIM_IT_CCx) (or TIM_DMA_Cmd(TIMx, TIM_DMA_CCx))

- ***TIM_OC1Init()***
- ***TIM_OC2Init()***
- ***TIM_OC3Init()***
- ***TIM_OC4Init()***
- ***TIM_OC5Init()***
- ***TIM_OC6Init()***
- ***TIM_SelectGC5C1()***

- *TIM_SelectGC5C2()*
- *TIM_SelectGC5C3()*
- *TIM_OCStructInit()*
- *TIM_SelectOCxM()*
- *TIM_SetCompare1()*
- *TIM_SetCompare2()*
- *TIM_SetCompare3()*
- *TIM_SetCompare4()*
- *TIM_SetCompare5()*
- *TIM_SetCompare6()*
- *TIM_ForcedOC1Config()*
- *TIM_ForcedOC2Config()*
- *TIM_ForcedOC3Config()*
- *TIM_ForcedOC4Config()*
- *TIM_ForcedOC5Config()*
- *TIM_ForcedOC6Config()*
- *TIM_OC1PreloadConfig()*
- *TIM_OC2PreloadConfig()*
- *TIM_OC3PreloadConfig()*
- *TIM_OC4PreloadConfig()*
- *TIM_OC5PreloadConfig()*
- *TIM_OC6PreloadConfig()*
- *TIM_OC1FastConfig()*
- *TIM_OC2FastConfig()*
- *TIM_OC3FastConfig()*
- *TIM_OC4FastConfig()*
- *TIM_ClearOC1Ref()*
- *TIM_ClearOC2Ref()*
- *TIM_ClearOC3Ref()*
- *TIM_ClearOC4Ref()*
- *TIM_ClearOC5Ref()*
- *TIM_ClearOC6Ref()*
- *TIM_SelectOCREFClear()*
- *TIM_OC1PolarityConfig()*
- *TIM_OC1NPolarityConfig()*
- *TIM_OC2PolarityConfig()*
- *TIM_OC2NPolarityConfig()*
- *TIM_OC3PolarityConfig()*
- *TIM_OC3NPolarityConfig()*
- *TIM_OC4PolarityConfig()*
- *TIM_OC5PolarityConfig()*
- *TIM_OC6PolarityConfig()*
- *TIM_CCxCmd()*
- *TIM_CCxNCmd()*

22.2.2 How to use this driver

This driver provides functions to configure and program the TIM of all stm32f30x devices. These functions are split in 9 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. Advanced-control timers (TIM1 and TIM8) specific features
   - Configures the Break input, dead time, Lock level, the OSSI, the OSSR State and the AOE(automatic output enable)
   - Enable/Disable the TIM peripheral Main Outputs
   - Select the Commutation event
   - Set/Reset the Capture Compare Preload Control bit
5. TIM interrupts, DMA and flags management
   - Enable/Disable interrupt sources
   - Get flags status
   - Clear flags/ Pending bits
   - Enable/Disable DMA requests
   - Configure DMA burst mode
   - Select CaptureCompare DMA request
6. 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
7. 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
8. TIM specific interface management, this group includes all needed functions to use the specific TIM interface:
   - Encoder Interface Configuration

– Select Hall Sensor

9. TIM specific remapping management includes the Remapping configuration of specific timers

22.2.3 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. Enable TIM clock using RCC_APBxPeriphClockCmd(RCC_APBxPeriph_TIMx, ENABLE) function
2. Fill the TIM_TimeBaseInitStruct with the desired parameters.
3. Call TIM_TimeBaseInit(TIMx, &TIM_TimeBaseInitStruct) to configure the Time Base unit with the corresponding configuration
4. Enable the NVIC if you need to generate the update interrupt.
5. Enable the corresponding interrupt using the function TIM_ITConfig(TIMx, TIM_IT_Update)
6. 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.

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

22.2.4 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. Enable TIM clock using RCC_APBxPeriphClockCmd(RCC_APBxPeriph_TIMx, ENABLE) function
2. Configure the TIM pins by configuring the corresponding GPIO pins
3. 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
- Clock Division = TIM_CKD_DIV1

4. 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
5. 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
6. Enable the NVIC or the DMA to read the measured frequency.
7. Enable the corresponding interrupt (or DMA request) to read the Captured value, using the function TIM_ITConfig(TIMx, TIM_IT_CCx) (or TIM_DMA_Cmd(TIMx, TIM_DMA_CCx))
8. Call the TIM_Cmd(ENABLE) function to enable the TIM counter.
9. 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()***

22.2.5 Advanced-control timers (TIM1 and TIM8) specific features

TIM Driver: how to use the Break feature

After configuring the Timer channel(s) in the appropriate Output Compare mode:

1. Fill the TIM_BDTRInitStruct with the desired parameters for the Timer Break Polarity, dead time, Lock level, the OSSI/OSSR State and the AOE(automatic output enable).
2. Call TIM_BDTRConfig(TIMx, &TIM_BDTRInitStruct) to configure the Timer
3. Enable the Main Output using TIM_CtrlPWMOutputs(TIM1, ENABLE)
4. Once the break even occurs, the Timer's output signals are put in reset state or in a known state (according to the configuration made in TIM_BDTRConfig() function).

- ***TIM_BDTRConfig()***
- ***TIM_Break1Config()***

- *TIM_Break2Config()*
- *TIM_Break1Cmd()*
- *TIM_Break2Cmd()*
- *TIM_BDTRStructInit()*
- *TIM_CtrlPWMOutputs()*
- *TIM_SelectCOM()*
- *TIM_CCPreloadControl()*

22.2.6 Interrupts, DMA and flags management functions

- *TIM_ITConfig()*
- *TIM_GenerateEvent()*
- *TIM_GetFlagStatus()*
- *TIM_ClearFlag()*
- *TIM_GetITStatus()*
- *TIM_ClearITPendingBit()*
- *TIM_DMAConfig()*
- *TIM_DMACmd()*
- *TIM_SelectCCDMA()*

22.2.7 Clocks management functions

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

22.2.8 Synchronization management functions

TIM Driver: how to use it in synchronization Mode

Case of two/several Timers

1. Configure the Master Timers using the following functions:
   - void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_TRGOSource);
   - void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_MasterSlaveMode);
2. Configure the Slave Timers using the following functions:
   - void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource);
   - void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_SlaveMode);

Case of Timers and external trigger(ETR pin)

1. Configure the External trigger using this function:
   - void TIM_ETRConfig(TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity, uint16_t ExtTRGFilter);
2. Configure the Slave Timers using the following functions:
   - void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource);

– void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_SlaveMode);

- ***TIM_SelectInputTrigger()***
- ***TIM_SelectOutputTrigger()***
- ***TIM_SelectOutputTrigger2()***
- ***TIM_SelectSlaveMode()***
- ***TIM_SelectMasterSlaveMode()***
- ***TIM_ETRConfig()***

22.2.9 Specific interface management functions

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

22.2.10 Specific remapping management function

- ***TIM_RemapConfig()***

22.2.11 TimeBase management functions

22.2.11.1 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, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 to select the TIM peripheral.
Return values	• None.
Notes	• None.

22.2.11.2 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, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 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.

22.2.11.3 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.

22.2.11.4 TIM_PrescalerConfig

Function Name	**void TIM_PrescalerConfig (*TIM_TypeDef* * TIMx, uint16_t Prescaler, uint16_t TIM_PSCReloadMode)**
Function Description	Configures the TIMx Prescaler.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 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.

22.2.11.5 TIM_CounterModeConfig

Function Name	**void TIM_CounterModeConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_CounterMode)**
Function Description	Specifies the TIMx Counter Mode to be used.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.11.6 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, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 to select the TIM peripheral. • **Counter :** specifies the Counter register new value.
Return values	• None.
Notes	• None.

22.2.11.7 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, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 to select the TIM peripheral. • **Autoreload :** specifies the Autoreload register new value.
Return values	• None.
Notes	• None.

22.2.11.8 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, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 to select the TIM peripheral.
Return values	• **Counter Register value**
Notes	• None.

22.2.11.9 TIM_GetPrescaler

Function Name	**uint16_t TIM_GetPrescaler (*TIM_TypeDef** * TIMx)**
Function Description	Gets the TIMx Prescaler value.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 to select the TIM peripheral.
Return values	• **Prescaler Register value.**
Notes	• None.

22.2.11.10 TIM_UpdateDisableConfig

Function Name	**void TIM_UpdateDisableConfig (*TIM_TypeDef** * TIMx, *FunctionalState* NewState)**
Function Description	Enables or Disables the TIMx Update event.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 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.

22.2.11.11 TIM_UpdateRequestConfig

Function Name	**void TIM_UpdateRequestConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_UpdateSource)**
Function Description	Configures the TIMx Update Request Interrupt source.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 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.

22.2.11.12 TIM_UIFRemap

Function Name	**void TIM_UIFRemap (*TIM_TypeDef** * TIMx, *FunctionalState* NewState)**
Function Description	Sets or resets the update interrupt flag (UIF)status bit Remapping.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 to select the TIM peripheral. • **NewState :** new state of the UIFREMAP bit. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.11.13 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, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 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.

22.2.11.14 TIM_SelectOnePulseMode

Function Name	**void TIM_SelectOnePulseMode (*TIM_TypeDef** * TIMx, uint16_t TIM_OPMode)**
Function Description	Selects the TIMx's One Pulse Mode.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6 ,7 ,8, 15, 16 or 17 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.

22.2.11.15 TIM_SetClockDivision

Function Name
void TIM_SetClockDivision (*TIM_TypeDef* * TIMx, uint16_t TIM_CKD)

Function Description
Sets the TIMx Clock Division value.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17, to select the TIM peripheral.
- **TIM_CKD :** specifies the clock division value. This parameter can be one of the following value:
  - *TIM_CKD_DIV1 :* TDTS = Tck_tim
  - *TIM_CKD_DIV2 :* TDTS = 2*Tck_tim
  - *TIM_CKD_DIV4 :* TDTS = 4*Tck_tim

Return values
- None.

Notes
- None.

22.2.11.16 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, 2, 3, 4, 6, 7, 8, 15, 16 or 17 to select the TIMx peripheral.
- **NewState :** new state of the TIMx peripheral. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

22.2.12 Output Compare management functions

22.2.12.1 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, 2, 3, 4, 8, 15, 16 or 17, 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.

22.2.12.2 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, 2, 3, 4, 8 or 15 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.

22.2.12.3 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, 2, 3, 4 or 8 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.

22.2.12.4 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, 2, 3, 4 or 8 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.

22.2.12.5 TIM_OC5Init

Function Name	**void TIM_OC5Init (*TIM_TypeDef** * TIMx, *TIM_OCInitTypeDef** * TIM_OCInitStruct)**
Function Description	Initializes the TIMx Channel5 according to the specified

parameters in the TIM_OCInitStruct.

Parameters	• **TIMx :** where x can be 1 or 8 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.

22.2.12.6 TIM_OC6Init

Function Name	**void TIM_OC6Init (*TIM_TypeDef ** TIMx, *TIM_OCInitTypeDef ** TIM_OCInitStruct)**
Function Description	Initializes the TIMx Channel6 according to the specified parameters in the TIM_OCInitStruct.
Parameters	• **TIMx :** where x can be 1 or 8 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.

22.2.12.7 TIM_SelectGC5C1

Function Name	**void TIM_SelectGC5C1 (*TIM_TypeDef ** TIMx, *FunctionalState* NewState)**
Function Description	Selects the TIM Group Channel 5 and Channel 1, OC1REFC is the logical AND of OC1REFC and OC5REF.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIMx peripheral • **NewState :** new state of the Commutation event. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.12.8 TIM_SelectGC5C2

Function Name	**void TIM_SelectGC5C2 (*TIM_TypeDef** * TIMx, *FunctionalState* NewState)**
Function Description	Selects the TIM Group Channel 5 and Channel 2, OC2REFC is the logical AND of OC2REFC and OC5REF.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIMx peripheral • **NewState :** new state of the Commutation event. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.12.9 TIM_SelectGC5C3

Function Name	**void TIM_SelectGC5C3 (*TIM_TypeDef** * TIMx, *FunctionalState* NewState)**
Function Description	Selects the TIM Group Channel 5 and Channel 3, OC3REFC is the logical AND of OC3REFC and OC5REF.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIMx peripheral • **NewState :** new state of the Commutation event. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.12.10 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.

22.2.12.11 TIM_SelectOCxM

Function Name	**void TIM_SelectOCxM (*TIM_TypeDef* * TIMx, uint16_t TIM_Channel, uint32_t TIM_OCMode)**
Function Description	Selects the TIM Output Compare Mode.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 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 :*** – ***TIM_OCMode_Retrigerrable_OPM1 :*** – ***TIM_OCMode_Retrigerrable_OPM2 :*** – ***TIM_OCMode_Combined_PWM1 :*** – ***TIM_OCMode_Combined_PWM2 :*** – ***TIM_OCMode_Asymmetric_PWM1 :*** – ***TIM_OCMode_Asymmetric_PWM2 :***
Return values	• None.
Notes	• This function disables the selected channel before changing the Output Compare Mode. If needed, user has to enable this channel using TIM_CCxCmd() and TIM_CCxNCmd() functions.

22.2.12.12 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, 2, 3, 4, 8, 15, 16 or 17 to select the TIM peripheral. • **Compare1 :** specifies the Capture Compare1 register new value.
Return values	• None.
Notes	• None.

22.2.12.13 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, 2, 3, 4, 8 or 15 to select the TIM peripheral. • **Compare2 :** specifies the Capture Compare2 register new value.
Return values	• None.
Notes	• None.

22.2.12.14 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, 2, 3, 4, 5 or 8 to select the TIM peripheral. • **Compare3 :** specifies the Capture Compare3 register new

	value.
Return values	• None.
Notes	• None.

22.2.12.15 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, 2, 3, 4, 5 or 8 to select the TIM peripheral. • **Compare4 :** specifies the Capture Compare4 register new value.
Return values	• None.
Notes	• None.

22.2.12.16 TIM_SetCompare5

Function Name	**void TIM_SetCompare5 (*TIM_TypeDef* * TIMx, uint32_t Compare5)**
Function Description	Sets the TIMx Capture Compare5 Register value.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **Compare5 :** specifies the Capture Compare5 register new value.
Return values	• None.
Notes	• None.

22.2.12.17 TIM_SetCompare6

Function Name	**void TIM_SetCompare6 (*TIM_TypeDef** * TIMx, uint32_t Compare6)**
Function Description	Sets the TIMx Capture Compare6 Register value.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **Compare6 :** specifies the Capture Compare5 register new value.
Return values	• None.
Notes	• None.

22.2.12.18 TIM_ForcedOC1Config

Function Name	**void TIM_ForcedOC1Config (*TIM_TypeDef** * TIMx, uint16_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 1 waveform to active or inactive level.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 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.

22.2.12.19 TIM_ForcedOC2Config

Function Name	**void TIM_ForcedOC2Config (*TIM_TypeDef** * TIMx, uint16_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 2 waveform to active or inactive level.

Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 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.

22.2.12.20 TIM_ForcedOC3Config

Function Name	**void TIM_ForcedOC3Config (*TIM_TypeDef** * TIMx, uint16_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 3 waveform to active or inactive level.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.12.21 TIM_ForcedOC4Config

Function Name	**void TIM_ForcedOC4Config (*TIM_TypeDef** * TIMx, uint16_t TIM_ForcedAction)**
Function Description	Forces the TIMx output 4 waveform to active or inactive level.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.12.22 TIM_ForcedOC5Config

Function Name
void TIM_ForcedOC5Config (*TIM_TypeDef* * TIMx, uint16_t TIM_ForcedAction)

Function Description
Forces the TIMx output 5 waveform to active or inactive level.

Parameters
- **TIMx :** where x can be 1 or 8 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 OC5REF
  - ***TIM_ForcedAction_InActive :*** Force inactive level on OC5REF.

Return values
- None.

Notes
- None.

22.2.12.23 TIM_ForcedOC6Config

Function Name
void TIM_ForcedOC6Config (*TIM_TypeDef* * TIMx, uint16_t TIM_ForcedAction)

Function Description
Forces the TIMx output 6 waveform to active or inactive level.

Parameters
- **TIMx :** where x can be 1 or 8 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

OC5REF
– ***TIM_ForcedAction_InActive :*** Force inactive level on OC5REF.

Return values
- None.

Notes
- None.

22.2.12.24 TIM_OC1PreloadConfig

Function Name
void TIM_OC1PreloadConfig (*TIM_TypeDef TIMx, uint16_t TIM_OCPreload)**

Function Description
Enables or disables the TIMx peripheral Preload register on CCR1.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 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.

22.2.12.25 TIM_OC2PreloadConfig

Function Name
void TIM_OC2PreloadConfig (*TIM_TypeDef TIMx, uint16_t TIM_OCPreload)**

Function Description
Enables or disables the TIMx peripheral Preload register on CCR2.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 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.

22.2.12.26 TIM_OC3PreloadConfig

Function Name	**void TIM_OC3PreloadConfig (*TIM_TypeDef ** TIMx, uint16_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on CCR3.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.12.27 TIM_OC4PreloadConfig

Function Name	**void TIM_OC4PreloadConfig (*TIM_TypeDef ** TIMx, uint16_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on CCR4.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.12.28 TIM_OC5PreloadConfig

Function Name	**void TIM_OC5PreloadConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on CCR5.
Parameters	• **TIMx :** where x can be 1 or 8 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.

22.2.12.29 TIM_OC6PreloadConfig

Function Name	**void TIM_OC6PreloadConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCPreload)**
Function Description	Enables or disables the TIMx peripheral Preload register on CCR6.
Parameters	• **TIMx :** where x can be 1 or 8 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.

22.2.12.30 TIM_OC1FastConfig

Function Name	**void TIM_OC1FastConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 1 Fast feature.

Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 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.

22.2.12.31 TIM_OC2FastConfig

Function Name	**void TIM_OC2FastConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 2 Fast feature.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 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.

22.2.12.32 TIM_OC3FastConfig

Function Name	**void TIM_OC3FastConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 3 Fast feature.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.12.33 TIM_OC4FastConfig

Function Name	**void TIM_OC4FastConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCFast)**
Function Description	Configures the TIMx Output Compare 4 Fast feature.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.12.34 TIM_ClearOC1Ref

Function Name	**void TIM_ClearOC1Ref (*TIM_TypeDef** * TIMx, uint16_t TIM_OCClear)**
Function Description	Clears or safeguards the OCREF1 signal on an external event.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 to select the TIM peripheral. • **TIM_OCClear :** new state of the Output Compare Clear Enable Bit. This parameter can be one of the following values: – ***TIM_OCClear_Enable :*** TIM Output clear enable – ***TIM_OCClear_Disable :*** TIM Output clear disable
Return values	• None.
Notes	• None.

22.2.12.35 TIM_ClearOC2Ref

Function Name	**void TIM_ClearOC2Ref (*TIM_TypeDef* * TIMx, uint16_t TIM_OCClear)**
Function Description	Clears or safeguards the OCREF2 signal on an external event.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 to select the TIM peripheral. • **TIM_OCClear :** new state of the Output Compare Clear Enable Bit. This parameter can be one of the following values: – ***TIM_OCClear_Enable :*** TIM Output clear enable – ***TIM_OCClear_Disable :*** TIM Output clear disable
Return values	• None.
Notes	• None.

22.2.12.36 TIM_ClearOC3Ref

Function Name	**void TIM_ClearOC3Ref (*TIM_TypeDef* * TIMx, uint16_t TIM_OCClear)**
Function Description	Clears or safeguards the OCREF3 signal on an external event.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 to select the TIM peripheral. • **TIM_OCClear :** new state of the Output Compare Clear Enable Bit. This parameter can be one of the following values: – ***TIM_OCClear_Enable :*** TIM Output clear enable – ***TIM_OCClear_Disable :*** TIM Output clear disable
Return values	• None.
Notes	• None.

22.2.12.37 TIM_ClearOC4Ref

Function Name	**void TIM_ClearOC4Ref (*TIM_TypeDef** * TIMx, uint16_t TIM_OCClear)**
Function Description	Clears or safeguards the OCREF4 signal on an external event.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 to select the TIM peripheral. • **TIM_OCClear :** new state of the Output Compare Clear Enable Bit. This parameter can be one of the following values: – ***TIM_OCClear_Enable :*** TIM Output clear enable – ***TIM_OCClear_Disable :*** TIM Output clear disable
Return values	• None.
Notes	• None.

22.2.12.38 TIM_ClearOC5Ref

Function Name	**void TIM_ClearOC5Ref (*TIM_TypeDef** * TIMx, uint16_t TIM_OCClear)**
Function Description	Clears or safeguards the OCREF5 signal on an external event.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **TIM_OCClear :** new state of the Output Compare Clear Enable Bit. This parameter can be one of the following values: – ***TIM_OCClear_Enable :*** TIM Output clear enable – ***TIM_OCClear_Disable :*** TIM Output clear disable
Return values	• None.
Notes	• None.

22.2.12.39 TIM_ClearOC6Ref

Function Name	**void TIM_ClearOC6Ref (*TIM_TypeDef** * TIMx, uint16_t TIM_OCClear)**
Function Description	Clears or safeguards the OCREF6 signal on an external event.

Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **TIM_OCClear :** new state of the Output Compare Clear Enable Bit. This parameter can be one of the following values: – ***TIM_OCClear_Enable :*** TIM Output clear enable – ***TIM_OCClear_Disable :*** TIM Output clear disable
Return values	• None.
Notes	• None.

22.2.12.40 TIM_SelectOCREFClear

Function Name	**void TIM_SelectOCREFClear (*TIM_TypeDef ** TIMx, uint16_t TIM_OCReferenceClear)**
Function Description	Selects the OCReference Clear source.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 to select the TIM peripheral. • **TIM_OCReferenceClear :** specifies the OCReference Clear source. This parameter can be one of the following values: – ***TIM_OCReferenceClear_ETRF :*** The internal OCreference clear input is connected to ETRF. – ***TIM_OCReferenceClear_OCREFCLR :*** The internal OCreference clear input is connected to OCREF_CLR input.
Return values	• None.
Notes	• None.

22.2.12.41 TIM_OC1PolarityConfig

Function Name	**void TIM_OC1PolarityConfig (*TIM_TypeDef ** TIMx, uint16_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 1 polarity.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC1 Polarity This parameter 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.

22.2.12.42 TIM_OC1NPolarityConfig

Function Name	**void TIM_OC1NPolarityConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCNPolarity)**
Function Description	Configures the TIMx Channel 1N polarity.
Parameters	• **TIMx :** where x can be 1, 8, 15, 16 or 17 to select the TIM peripheral. • **TIM_OCNPolarity :** specifies the OC1N Polarity This parameter can be one of the following values: – ***TIM_OCNPolarity_High :*** Output Compare active high – ***TIM_OCNPolarity_Low :*** Output Compare active low
Return values	• None.
Notes	• None.

22.2.12.43 TIM_OC2PolarityConfig

Function Name	**void TIM_OC2PolarityConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 2 polarity.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 8 or 15 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC2 Polarity This parameter 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.

22.2.12.44 TIM_OC2NPolarityConfig

Function Name	**void TIM_OC2NPolarityConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCNPolarity)**
Function Description	Configures the TIMx Channel 2N polarity.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **TIM_OCNPolarity :** specifies the OC2N Polarity This parameter can be one of the following values: – ***TIM_OCNPolarity_High :*** Output Compare active high – ***TIM_OCNPolarity_Low :*** Output Compare active low
Return values	• None.
Notes	• None.

22.2.12.45 TIM_OC3PolarityConfig

Function Name	**void TIM_OC3PolarityConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 3 polarity.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC3 Polarity This parameter 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.

22.2.12.46 TIM_OC3NPolarityConfig

Function Name	**void TIM_OC3NPolarityConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_OCNPolarity)**
Function Description	Configures the TIMx Channel 3N polarity.

Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **TIM_OCNPolarity :** specifies the OC3N Polarity This parameter can be one of the following values: – ***TIM_OCNPolarity_High :*** Output Compare active high – ***TIM_OCNPolarity_Low :*** Output Compare active low
Return values	• None.
Notes	• None.

22.2.12.47 TIM_OC4PolarityConfig

Function Name	**void TIM_OC4PolarityConfig (*TIM_TypeDef* * TIMx, uint16_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 4 polarity.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC4 Polarity This parameter 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.

22.2.12.48 TIM_OC5PolarityConfig

Function Name	**void TIM_OC5PolarityConfig (*TIM_TypeDef* * TIMx, uint16_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 5 polarity.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC5 Polarity This parameter 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.

22.2.12.49 TIM_OC6PolarityConfig

Function Name	**void TIM_OC6PolarityConfig (*TIM_TypeDef ** TIMx, uint16_t TIM_OCPolarity)**
Function Description	Configures the TIMx channel 6 polarity.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **TIM_OCPolarity :** specifies the OC6 Polarity This parameter 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.

22.2.12.50 TIM_CCxCmd

Function Name	**void TIM_CCxCmd (*TIM_TypeDef ** TIMx, uint16_t TIM_Channel, uint16_t TIM_CCx)**
Function Description	Enables or disables the TIM Capture Compare Channel x.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 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_Channel_5 :*** TIM Channel 5 – ***TIM_Channel_6 :*** TIM Channel 6 • **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.

22.2.12.51 TIM_CCxNCmd

Function Name	**void TIM_CCxNCmd (*TIM_TypeDef ** TIMx, uint16_t TIM_Channel, uint16_t TIM_CCxN)**
Function Description	Enables or disables the TIM Capture Compare Channel xN.
Parameters	• **TIMx :** where x can be 1, 8, 15, 16 or 17 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_CCxN :** specifies the TIM Channel CCxNE bit new state. This parameter can be: TIM_CCxN_Enable or TIM_CCxN_Disable.
Return values	• None.
Notes	• None.

22.2.13 Input Capture management functions

22.2.13.1 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, 2, 3, 4, 8, 15, 16 or 17 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.

22.2.13.2 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.

22.2.13.3 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, 2, 3, 4, 8 or 15 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.

22.2.13.4 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, 2, 3, 4, 8, 15, 16 or 17 to select the TIM peripheral.
Return values	• **Capture Compare 1 Register value.**
Notes	• None.

22.2.13.5 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, 2, 3, 4, 8 or 15 to select the TIM peripheral.
Return values	• **Capture Compare 2 Register value.**
Notes	• None.

22.2.13.6 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, 2, 3, 4 or 8 to select the TIM peripheral.
Return values	• **Capture Compare 3 Register value.**
Notes	• None.

22.2.13.7 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, 2, 3, 4 or 8 to select the TIM peripheral.
Return values	• **Capture Compare 4 Register value.**
Notes	• None.

22.2.13.8 TIM_SetIC1Prescaler

Function Name	**void TIM_SetIC1Prescaler (*TIM_TypeDef** * TIMx, uint16_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 1 prescaler.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 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.

22.2.13.9 TIM_SetIC2Prescaler

Function Name	**void TIM_SetIC2Prescaler (*TIM_TypeDef** * TIMx, uint16_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 2 prescaler.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 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.

22.2.13.10 TIM_SetIC3Prescaler

Function Name	**void TIM_SetIC3Prescaler (*TIM_TypeDef** * TIMx, uint16_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 3 prescaler.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.13.11 TIM_SetIC4Prescaler

Function Name	**void TIM_SetIC4Prescaler (*TIM_TypeDef** * TIMx, uint16_t TIM_ICPSC)**
Function Description	Sets the TIMx Input Capture 4 prescaler.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.14 Advanced-control timers (TIM1, TIM8) specific features

22.2.14.1 TIM_BDTRConfig

Function Name	**void TIM_BDTRConfig (*TIM_TypeDef ** TIMx, *TIM_BDTRInitTypeDef ** TIM_BDTRInitStruct)**
Function Description	Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).
Parameters	• **TIMx :** where x can be 1, 8, 15, 16 or 17 to select the TIM • **TIM_BDTRInitStruct :** pointer to a TIM_BDTRInitTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.
Return values	• None.
Notes	• None.

22.2.14.2 TIM_Break1Config

Function Name	**void TIM_Break1Config (*TIM_TypeDef ** TIMx, uint32_t TIM_Break1Polarity, uint8_t TIM_Break1Filter)**
Function Description	Configures the Break1 feature.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM • **TIM_Break1Polarity :** specifies the Break1 polarity. This parameter can be one of the following values: – ***TIM_Break1Polarity_Low :*** Break1 input is active low – ***TIM_Break1Polarity_High :*** Break1 input is active high • **TIM_Break1Filter :** specifies the Break1 filter value. This parameter must be a value between 0x00 and 0x0F
Return values	• None.
Notes	• None.

22.2.14.3 TIM_Break2Config

Function Name	**void TIM_Break2Config (*TIM_TypeDef** * **TIMx, uint32_t TIM_Break2Polarity, uint8_t TIM_Break2Filter)**
Function Description	Configures the Break2 feature.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM • **TIM_Break2Polarity :** specifies the Break2 polarity. This parameter can be one of the following values: – ***TIM_Break2Polarity_Low :*** Break2 input is active low – ***TIM_Break2Polarity_High :*** Break2 input is active high • **TIM_Break2Filter :** specifies the Break2 filter value. This parameter must be a value between 0x00 and 0x0F
Return values	• None.
Notes	• None.

22.2.14.4 TIM_Break1Cmd

Function Name	**void TIM_Break1Cmd (*TIM_TypeDef** * **TIMx, *FunctionalState* NewState)**
Function Description	Enables or disables the TIM Break1 input.
Parameters	• **TIMx :** where x can be 1, 8, 1, 16 or 17 to select the TIMx peripheral. • **NewState :** new state of the TIM Break1 input. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.14.5 TIM_Break2Cmd

Function Name	**void TIM_Break2Cmd (*TIM_TypeDef** * **TIMx, *FunctionalState* NewState)**
Function Description	Enables or disables the TIM Break2 input.

Parameters	• **TIMx :** where x can be 1 or 8 to select the TIMx peripheral. • **NewState :** new state of the TIM Break2 input. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.14.6 TIM_BDTRStructInit

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

22.2.14.7 TIM_CtrlPWMOutputs

Function Name	**void TIM_CtrlPWMOutputs (*TIM_TypeDef** TIMx, *FunctionalState* NewState)**
Function Description	Enables or disables the TIM peripheral Main Outputs.
Parameters	• **TIMx :** where x can be 1, 8, 15, 16 or 17 to select the TIMx peripheral. • **NewState :** new state of the TIM peripheral Main Outputs. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.14.8 TIM_SelectCOM

Function Name	**void TIM_SelectCOM (*TIM_TypeDef** * TIMx, *FunctionalState* NewState)**
Function Description	Selects the TIM peripheral Commutation event.
Parameters	• **TIMx :** where x can be 1, 8, 15, 16 or 17 to select the TIMx peripheral • **NewState :** new state of the Commutation event. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.14.9 TIM_CCPreloadControl

Function Name	**void TIM_CCPreloadControl (*TIM_TypeDef** * TIMx, *FunctionalState* NewState)**
Function Description	Sets or Resets the TIM peripheral Capture Compare Preload Control bit.
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIMx peripheral • **NewState :** new state of the Capture Compare Preload Control bit This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

22.2.15 Interrupts DMA and flags management functions

22.2.15.1 TIM_ITConfig

Function Name	**void TIM_ITConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_IT, *FunctionalState* NewState)**
Function Description	Enables or disables the specified TIM interrupts.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6, 7, 8, 15, 16 or 17 to select the TIMx peripheral.

- **TIM_IT :** 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_COM :*** TIM Commutation Interrupt source
  - ***TIM_IT_Trigger :*** TIM Trigger Interrupt source
  - ***TIM_IT_Break :*** TIM Break Interrupt source

Parameters
- **NewState :** new state of the TIM interrupts. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- For TIM6 and TIM7 only the parameter TIM_IT_Update can be used
- For TIM9 and TIM12 only one of the following parameters can be used: TIM_IT_Update, TIM_IT_CC1, TIM_IT_CC2 or TIM_IT_Trigger.
- For TIM10, TIM11, TIM13 and TIM14 only one of the following parameters can be used: TIM_IT_Update or TIM_IT_CC1
- TIM_IT_COM and TIM_IT_Break can be used only with TIM1 and TIM8

22.2.15.2 TIM_GenerateEvent

Function Name **void TIM_GenerateEvent (*TIM_TypeDef* * TIMx, uint16_t TIM_EventSource)**

Function Description Configures the TIMx event to be generate by software.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 6, 7, 8, 15, 16 or 17 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_COM :*** Timer COM event source

- ***TIM_EventSource_Trigger :*** Timer Trigger Event source
- ***TIM_EventSource_Break :*** Timer Break event source

Return values
- None.

Notes
- TIM6 and TIM7 can only generate an update event.
- TIM_EventSource_COM and TIM_EventSource_Break are used only with TIM1 and TIM8.

22.2.15.3 TIM_GetFlagStatus

Function Name **FlagStatus TIM_GetFlagStatus (*TIM_TypeDef** TIMx, uint32_t TIM_FLAG)**

Function Description Checks whether the specified TIM flag is set or not.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 6, 7, 8, 15, 16 or 17 to select the TIM peripheral.
- **TIM_FLAG :** specifies the flag to check. This parameter can be one of the following values:
  - ***TIM_FLAG_Update :*** TIM update Flag
  - ***TIM_FLAG_CC1 :*** TIM Capture Compare 1 Flag
  - ***TIM_FLAG_CC2 :*** TIM Capture Compare 2 Flag
  - ***TIM_FLAG_CC3 :*** TIM Capture Compare 3 Flag
  - ***TIM_FLAG_CC4 :*** TIM Capture Compare 4 Flag
  - ***TIM_FLAG_CC5 :*** TIM Capture Compare 5 Flag
  - ***TIM_FLAG_CC6 :*** TIM Capture Compare 6 Flag
  - ***TIM_FLAG_COM :*** TIM Commutation Flag
  - ***TIM_FLAG_Trigger :*** TIM Trigger Flag
  - ***TIM_FLAG_Break :*** TIM Break Flag
  - ***TIM_FLAG_CC1OF :*** TIM Capture Compare 1 over capture Flag
  - ***TIM_FLAG_CC2OF :*** TIM Capture Compare 2 over capture Flag
  - ***TIM_FLAG_CC3OF :*** TIM Capture Compare 3 over capture Flag
  - ***TIM_FLAG_CC4OF :*** TIM Capture Compare 4 over capture Flag

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

Notes
- TIM6 and TIM7 can have only one update flag.
- TIM_FLAG_COM and TIM_FLAG_Break are used only with TIM1 and TIM8.

22.2.15.4 TIM_ClearFlag

Function Name	**void TIM_ClearFlag (*TIM_TypeDef** * TIMx, uint16_t TIM_FLAG)**
Function Description	Clears the TIMx's pending flags.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6, 7, 8, 15, 16 or 17 to select the TIM peripheral. • **TIM_FLAG :** specifies the flag bit to clear. This parameter can be any combination of the following values: – ***TIM_FLAG_Update :*** TIM update Flag – ***TIM_FLAG_CC1 :*** TIM Capture Compare 1 Flag – ***TIM_FLAG_CC2 :*** TIM Capture Compare 2 Flag – ***TIM_FLAG_CC3 :*** TIM Capture Compare 3 Flag – ***TIM_FLAG_CC4 :*** TIM Capture Compare 4 Flag – ***TIM_FLAG_CC5 :*** TIM Capture Compare 5 Flag – ***TIM_FLAG_CC6 :*** TIM Capture Compare 6 Flag – ***TIM_FLAG_COM :*** TIM Commutation Flag – ***TIM_FLAG_Trigger :*** TIM Trigger Flag – ***TIM_FLAG_Break :*** TIM Break Flag – ***TIM_FLAG_CC1OF :*** TIM Capture Compare 1 over capture Flag – ***TIM_FLAG_CC2OF :*** TIM Capture Compare 2 over capture Flag – ***TIM_FLAG_CC3OF :*** TIM Capture Compare 3 over capture Flag – ***TIM_FLAG_CC4OF :*** TIM Capture Compare 4 over capture Flag
Return values	• None.
Notes	• TIM6 and TIM7 can have only one update flag. • TIM_FLAG_COM and TIM_FLAG_Break are used only with TIM1 and TIM8.

22.2.15.5 TIM_GetITStatus

Function Name	**ITStatus TIM_GetITStatus (*TIM_TypeDef** * TIMx, uint16_t TIM_IT)**
Function Description	Checks whether the TIM interrupt has occurred or not.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6, 7, 8, 15, 16 or 17 to select the TIM peripheral. • **TIM_IT :** 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_COM :*** TIM Commutation Interrupt source
- ***TIM_IT_Trigger :*** TIM Trigger Interrupt source
- ***TIM_IT_Break :*** TIM Break Interrupt source

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

Notes
- TIM6 and TIM7 can generate only an update interrupt.
- TIM_IT_COM and TIM_IT_Break are used only with TIM1 and TIM8.

22.2.15.6 TIM_ClearITPendingBit

Function Name
void TIM_ClearITPendingBit (*TIM_TypeDef* * TIMx, uint16_t TIM_IT)

Function Description
Clears the TIMx's interrupt pending bits.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 6, 7, 8, 15, 16 or 17 to select the TIM peripheral.
- **TIM_IT :** 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_COM :*** TIM Commutation Interrupt source
  - ***TIM_IT_Trigger :*** TIM Trigger Interrupt source
  - ***TIM_IT_Break :*** TIM Break Interrupt source

Return values
- None.

Notes
- TIM6 and TIM7 can generate only an update interrupt.
- TIM_IT_COM and TIM_IT_Break are used only with TIM1 and TIM8.

22.2.15.7 TIM_DMAConfig

Function Name	**void TIM_DMAConfig (*TIM_TypeDef ** TIMx, uint16_t TIM_DMABase, uint16_t TIM_DMABurstLength)**
Function Description	Configures the TIMx's DMA interface.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4 or 8 to select the TIM peripheral. • **TIM_DMABase :** DMA Base address. This parameter can be one of the following values: – ***TIM_DMABase_CR1 :*** – ***TIM_DMABase_CR2 :*** – ***TIM_DMABase_SMCR :*** – ***TIM_DMABase_DIER :*** – ***TIM1_DMABase_SR :*** – ***TIM_DMABase_EGR :*** – ***TIM_DMABase_CCMR1 :*** – ***TIM_DMABase_CCMR2 :*** – ***TIM_DMABase_CCER :*** – ***TIM_DMABase_CNT :*** – ***TIM_DMABase_PSC :*** – ***TIM_DMABase_ARR :*** – ***TIM_DMABase_RCR :*** – ***TIM_DMABase_CCR1 :*** – ***TIM_DMABase_CCR2 :*** – ***TIM_DMABase_CCR3 :*** – ***TIM_DMABase_CCR4 :*** – ***TIM_DMABase_BDTR :*** – ***TIM_DMABase_DCR :*** • **TIM_DMABurstLength :** DMA Burst length. This parameter can be one value between: TIM_DMABurstLength_1Transfer and TIM_DMABurstLength_18Transfers.
Return values	• None.
Notes	• None.

22.2.15.8 TIM_DMACmd

Function Name	**void TIM_DMACmd (*TIM_TypeDef ** TIMx, uint16_t TIM_DMASource, *FunctionalState* NewState)**
Function Description	Enables or disables the TIMx's DMA Requests.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 6, 7, 8, 15, 16 or 17 to select the TIM peripheral.

- **TIM_DMASource :** specifies the DMA Request sources. This parameter can be any combination of the following values:
  - ***TIM_DMA_Update :*** TIM update Interrupt source
  - ***TIM_DMA_CC1 :*** TIM Capture Compare 1 DMA source
  - ***TIM_DMA_CC2 :*** TIM Capture Compare 2 DMA source
  - ***TIM_DMA_CC3 :*** TIM Capture Compare 3 DMA source
  - ***TIM_DMA_CC4 :*** TIM Capture Compare 4 DMA source
  - ***TIM_DMA_COM :*** TIM Commutation DMA source
  - ***TIM_DMA_Trigger :*** TIM Trigger DMA source
- **NewState :** new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

22.2.15.9 TIM_SelectCCDMA

Function Name **void TIM_SelectCCDMA (*TIM_TypeDef** * **TIMx, *FunctionalState* NewState)**

Function Description Selects the TIMx peripheral Capture Compare DMA source.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 8, 15, 16 or 17 to select the TIM peripheral.
- **NewState :** new state of the Capture Compare DMA source This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

22.2.16 Clock management functions

22.2.16.1 TIM_InternalClockConfig

Function Name **void TIM_InternalClockConfig (*TIM_TypeDef** * **TIMx)**

Function Description Configures the TIMx internal Clock.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 to select the TIM peripheral.

Return values
- None.

Notes
- None.

22.2.16.2 TIM_ITRxExternalClockConfig

Function Name	**void TIM_ITRxExternalClockConfig (*TIM_TypeDef ** TIMx, uint16_t TIM_InputTriggerSource)**
Function Description	Configures the TIMx Internal Trigger as External Clock.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 to select the TIM peripheral. • **TIM_InputTriggerSource :** Trigger source. This parameter can be one of the following values: – ***TIM_TS_ITR0 :*** Internal Trigger 0 – ***TIM_TS_ITR1 :*** Internal Trigger 1 – ***TIM_TS_ITR2 :*** Internal Trigger 2 – ***TIM_TS_ITR3 :*** Internal Trigger 3
Return values	• None.
Notes	• None.

22.2.16.3 TIM_TIxExternalClockConfig

Function Name	**void TIM_TIxExternalClockConfig (*TIM_TypeDef ** TIMx, uint16_t TIM_TIxExternalCLKSource, uint16_t TIM_ICPolarity, uint16_t ICFilter)**
Function Description	Configures the TIMx Trigger as External Clock.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 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.

22.2.16.4 TIM_ETRClockMode1Config

Function Name
void TIM_ETRClockMode1Config (*TIM_TypeDef* * TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity, uint16_t ExtTRGFilter)

Function Description
Configures the External clock Mode1.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.16.5 TIM_ETRClockMode2Config

Function Name
void TIM_ETRClockMode2Config (*TIM_TypeDef* * TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity, uint16_t ExtTRGFilter)

Function Description
Configures the External clock Mode2.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.17 Synchronization management functions

22.2.17.1 TIM_SelectInputTrigger

Function Name **void TIM_SelectInputTrigger (*TIM_TypeDef** * TIMx, uint16_t TIM_InputTriggerSource)**

Function Description Selects the Input Trigger source.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 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_ITR1 :*** Internal Trigger 1
  - ***TIM_TS_ITR2 :*** Internal Trigger 2
  - ***TIM_TS_ITR3 :*** Internal Trigger 3
  - ***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.

22.2.17.2 TIM_SelectOutputTrigger

Function Name	**void TIM_SelectOutputTrigger (*TIM_TypeDef* * TIMx, uint16_t TIM_TRGOSource)**
Function Description	Selects the TIMx Trigger Output Mode.
Parameters	• **TIMx :** where x can be 1, 2, 3, 4, 5, 6, 7, 8 or 15 to select the TIM peripheral. • **TIM_TRGOSource :** specifies the Trigger Output source. This parameter can be one of the following values:
Notes	• None.

22.2.17.3 TIM_SelectOutputTrigger2

Function Name	**void TIM_SelectOutputTrigger2 (*TIM_TypeDef* * TIMx, uint32_t TIM_TRGO2Source)**
Function Description	Selects the TIMx Trigger Output Mode2 (TRGO2).
Parameters	• **TIMx :** where x can be 1 or 8 to select the TIM peripheral. • **TIM_TRGO2Source :** specifies the Trigger Output source. This parameter can be one of the following values:
Notes	• None.

22.2.17.4 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, 2, 3, 4, 8 or 15 to select the TIM peripheral. • **TIM_SlaveMode :** specifies the Timer Slave Mode. This parameter can be one of the following values: – ***TIM_SlaveMode_Reset :*** Rising edge of the selected trigger signal(TRGI) reinitialize 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
  - ***TIM_SlaveMode_Combined_ResetTrigger :*** Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter.

Return values
- None.

Notes
- None.

22.2.17.5 TIM_SelectMasterSlaveMode

Function Name
void TIM_SelectMasterSlaveMode (*TIM_TypeDef* * TIMx, uint16_t TIM_MasterSlaveMode)

Function Description
Sets or Resets the TIMx Master/Slave Mode.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4, 8 or 15 to select the TIM peripheral.
- **TIM_MasterSlaveMode :** specifies the Timer Master Slave Mode. This parameter 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.

22.2.17.6 TIM_ETRConfig

Function Name
void TIM_ETRConfig (*TIM_TypeDef* * TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity, uint16_t ExtTRGFilter)

Function Description
Configures the TIMx External Trigger (ETR).

Parameters
- **TIMx :** where x can be 1, 2, 3, 4 or 8 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.

22.2.18 Specific interface management functions

22.2.18.1 TIM_EncoderInterfaceConfig

Function Name **void TIM_EncoderInterfaceConfig (*TIM_TypeDef* * TIMx, uint16_t TIM_EncoderMode, uint16_t TIM_IC1Polarity, uint16_t TIM_IC2Polarity)**

Function Description Configures the TIMx Encoder Interface.

Parameters
- **TIMx :** where x can be 1, 2, 3, 4 or 8 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 parameter 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 parameter can be one of the following values:
  - ***TIM_ICPolarity_Falling :*** IC Falling edge.
  - ***TIM_ICPolarity_Rising :*** IC Rising edge.

Return values
- None.

Notes	• None.

22.2.18.2 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, 3, 4, 8 or 15 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.

22.2.19 Specific remapping management functions

22.2.19.1 TIM_RemapConfig

Function Name	**void TIM_RemapConfig (*TIM_TypeDef** * TIMx, uint16_t TIM_Remap)**
Function Description	Configures the TIM16 Remapping input Capabilities.
Parameters	• **TIMx :** where x can be 1, 8 or 16 to select the TIM peripheral. • **TIM_Remap :** specifies the TIM input reampping source. This parameter can be one of the following values: – ***TIM16_GPIO :*** TIM16 Channel 1 is connected to GPIO. – ***TIM16_RTC_CLK :*** TIM16 Channel 1 is connected to RTC input clock. – ***TIM16_HSE_DIV32 :*** TIM16 Channel 1 is connected to HSE/32 clock. – ***TIM16_MCO :*** TIM16 Channel 1 is connected to MCO clock. – ***TIM1_ADC1_AWDG1 :*** TIM1 ETR is connected to ADC1 AWDG1. – ***TIM1_ADC1_AWDG2 :*** TIM1 ETR is connected to ADC1 AWDG2. – ***TIM1_ADC1_AWDG3 :*** TIM1 ETR is connected to ADC1 AWDG3.

- ***TIM1_ADC4_AWDG1 :*** TIM1 ETR is connected to ADC4 AWDG1.
- ***TIM1_ADC4_AWDG2 :*** TIM1 ETR is connected to ADC4 AWDG2.
- ***TIM1_ADC4_AWDG3 :*** TIM1 ETR is connected to ADC4 AWDG3.
- ***TIM8_ADC2_AWDG1 :*** TIM8 ETR is connected to ADC2 AWDG1.
- ***TIM8_ADC2_AWDG2 :*** TIM8 ETR is connected to ADC2 AWDG2.
- ***TIM8_ADC2_AWDG3 :*** TIM8 ETR is connected to ADC2 AWDG3.
- ***TIM8_ADC4_AWDG1 :*** TIM8 ETR is connected to ADC4 AWDG1.
- ***TIM8_ADC4_AWDG2 :*** TIM8 ETR is connected to ADC4 AWDG2.
- ***TIM8_ADC4_AWDG3 :*** TIM8 ETR is connected to ADC4 AWDG3.

Return values
- **: None**

Notes
- None.

22.3 TIM Firmware driver defines

22.3.1 TIM

TIM

TIM_AOE_Bit_Set_Reset

- #define: ***TIM_AutomaticOutput_Enable ((uint16_t)0x4000)***

- #define: ***TIM_AutomaticOutput_Disable ((uint16_t)0x0000)***

TIM_Break1_Input_enable_disable

- #define: ***TIM_Break1_Enable ((uint32_t)0x00001000)***

- #define: ***TIM_Break1_Disable ((uint32_t)0x00000000)***

TIM_Break1_Polarity

- #define: ***TIM_Break1Polarity_Low ((uint32_t)0x00000000)***

- #define: ***TIM_Break1Polarity_High ((uint32_t)0x00002000)***

TIM_Break2_Input_enable_disable

- #define: ***TIM_Break2_Enable ((uint32_t)0x01000000)***

- #define: ***TIM_Break2_Disable ((uint32_t)0x00000000)***

TIM_Break2_Polarity

- #define: ***TIM_Break2Polarity_Low ((uint32_t)0x00000000)***

- #define: ***TIM_Break2Polarity_High ((uint32_t)0x02000000)***

TIM_Break_Input_enable_disable

- #define: ***TIM_Break_Enable ((uint16_t)0x1000)***

- #define: ***TIM_Break_Disable ((uint16_t)0x0000)***

TIM_Break_Polarity

- #define: ***TIM_BreakPolarity_Low ((uint16_t)0x0000)***

- #define: ***TIM_BreakPolarity_High ((uint16_t)0x2000)***

TIM_Capture_Compare_N_State

- #define: ***TIM_CCxN_Enable ((uint16_t)0x0004)***

- #define: ***TIM_CCxN_Disable ((uint16_t)0x0000)***

TIM_Capture_Compare_State

- #define: ***TIM_CCx_Enable ((uint16_t)0x0001)***

- #define: ***TIM_CCx_Disable ((uint16_t)0x0000)***

TIM_Channel

- #define: ***TIM_Channel_1 ((uint16_t)0x0000)***

- #define: ***TIM_Channel_2 ((uint16_t)0x0004)***

- #define: ***TIM_Channel_3 ((uint16_t)0x0008)***

- #define: ***TIM_Channel_4 ((uint16_t)0x000C)***

- #define: ***TIM_Channel_5 ((uint16_t)0x0010)***

- #define: ***TIM_Channel_6 ((uint16_t)0x0014)***

TIM_Clock_Division_CKD

- #define: ***TIM_CKD_DIV1 ((uint16_t)0x0000)***

- #define: ***TIM_CKD_DIV2 ((uint16_t)0x0100)***

- #define: ***TIM_CKD_DIV4 ((uint16_t)0x0200)***

TIM_Counter_Mode

- #define: ***TIM_CounterMode_Up ((uint16_t)0x0000)***
- #define: ***TIM_CounterMode_Down ((uint16_t)0x0010)***
- #define: ***TIM_CounterMode_CenterAligned1 ((uint16_t)0x0020)***
- #define: ***TIM_CounterMode_CenterAligned2 ((uint16_t)0x0040)***
- #define: ***TIM_CounterMode_CenterAligned3 ((uint16_t)0x0060)***

TIM_DMA_Base_address

- #define: ***TIM_DMABase_CR1 ((uint16_t)0x0000)***
- #define: ***TIM_DMABase_CR2 ((uint16_t)0x0001)***
- #define: ***TIM_DMABase_SMCR ((uint16_t)0x0002)***
- #define: ***TIM_DMABase_DIER ((uint16_t)0x0003)***
- #define: ***TIM_DMABase_SR ((uint16_t)0x0004)***

- #define: ***TIM_DMABase_EGR ((uint16_t)0x0005)***
- #define: ***TIM_DMABase_CCMR1 ((uint16_t)0x0006)***
- #define: ***TIM_DMABase_CCMR2 ((uint16_t)0x0007)***
- #define: ***TIM_DMABase_CCER ((uint16_t)0x0008)***
- #define: ***TIM_DMABase_CNT ((uint16_t)0x0009)***
- #define: ***TIM_DMABase_PSC ((uint16_t)0x000A)***
- #define: ***TIM_DMABase_ARR ((uint16_t)0x000B)***
- #define: ***TIM_DMABase_RCR ((uint16_t)0x000C)***
- #define: ***TIM_DMABase_CCR1 ((uint16_t)0x000D)***
- #define: ***TIM_DMABase_CCR2 ((uint16_t)0x000E)***
- #define: ***TIM_DMABase_CCR3 ((uint16_t)0x000F)***
- #define: ***TIM_DMABase_CCR4 ((uint16_t)0x0010)***

- #define: ***TIM_DMABase_BDTR ((uint16_t)0x0011)***

- #define: ***TIM_DMABase_DCR ((uint16_t)0x0012)***

- #define: ***TIM_DMABase_OR ((uint16_t)0x0013)***

- #define: ***TIM_DMABase_CCMR3 ((uint16_t)0x0014)***

- #define: ***TIM_DMABase_CCR5 ((uint16_t)0x0015)***

- #define: ***TIM_DMABase_CCR6 ((uint16_t)0x0016)***

TIM_DMA_Burst_Length

- #define: ***TIM_DMABurstLength_1Transfer ((uint16_t)0x0000)***

- #define: ***TIM_DMABurstLength_2Transfers ((uint16_t)0x0100)***

- #define: ***TIM_DMABurstLength_3Transfers ((uint16_t)0x0200)***

- #define: ***TIM_DMABurstLength_4Transfers ((uint16_t)0x0300)***

- #define: ***TIM_DMABurstLength_5Transfers ((uint16_t)0x0400)***

- #define: ***TIM_DMABurstLength_6Transfers ((uint16_t)0x0500)***

- #define: ***TIM_DMABurstLength_7Transfers ((uint16_t)0x0600)***
- #define: ***TIM_DMABurstLength_8Transfers ((uint16_t)0x0700)***
- #define: ***TIM_DMABurstLength_9Transfers ((uint16_t)0x0800)***
- #define: ***TIM_DMABurstLength_10Transfers ((uint16_t)0x0900)***
- #define: ***TIM_DMABurstLength_11Transfers ((uint16_t)0x0A00)***
- #define: ***TIM_DMABurstLength_12Transfers ((uint16_t)0x0B00)***
- #define: ***TIM_DMABurstLength_13Transfers ((uint16_t)0x0C00)***
- #define: ***TIM_DMABurstLength_14Transfers ((uint16_t)0x0D00)***
- #define: ***TIM_DMABurstLength_15Transfers ((uint16_t)0x0E00)***
- #define: ***TIM_DMABurstLength_16Transfers ((uint16_t)0x0F00)***
- #define: ***TIM_DMABurstLength_17Transfers ((uint16_t)0x1000)***
- #define: ***TIM_DMABurstLength_18Transfers ((uint16_t)0x1100)***

TIM_DMA_sources

- #define: ***TIM_DMA_Update ((uint16_t)0x0100)***

- #define: ***TIM_DMA_CC1 ((uint16_t)0x0200)***

- #define: ***TIM_DMA_CC2 ((uint16_t)0x0400)***

- #define: ***TIM_DMA_CC3 ((uint16_t)0x0800)***

- #define: ***TIM_DMA_CC4 ((uint16_t)0x1000)***

- #define: ***TIM_DMA_COM ((uint16_t)0x2000)***

- #define: ***TIM_DMA_Trigger ((uint16_t)0x4000)***

TIM_Encoder_Mode

- #define: ***TIM_EncoderMode_TI1 ((uint16_t)0x0001)***

- #define: ***TIM_EncoderMode_TI2 ((uint16_t)0x0002)***

- #define: ***TIM_EncoderMode_TI12 ((uint16_t)0x0003)***

TIM_Event_Source

- #define: ***TIM_EventSource_Update ((uint16_t)0x0001)***

- #define: ***TIM_EventSource_CC1 ((uint16_t)0x0002)***

- #define: ***TIM_EventSource_CC2 ((uint16_t)0x0004)***

- #define: ***TIM_EventSource_CC3 ((uint16_t)0x0008)***

- #define: ***TIM_EventSource_CC4 ((uint16_t)0x0010)***

- #define: ***TIM_EventSource_COM ((uint16_t)0x0020)***

- #define: ***TIM_EventSource_Trigger ((uint16_t)0x0040)***

- #define: ***TIM_EventSource_Break ((uint16_t)0x0080)***

- #define: ***TIM_EventSource_Break2 ((uint16_t)0x0100)***

TIM_External_Trigger_Polarity

- #define: ***TIM_ExtTRGPolarity_Inverted ((uint16_t)0x8000)***

- #define: ***TIM_ExtTRGPolarity_NonInverted ((uint16_t)0x0000)***

TIM_External_Trigger_Prescaler

- #define: ***TIM_ExtTRGPSC_OFF ((uint16_t)0x0000)***

- #define: ***TIM_ExtTRGPSC_DIV2 ((uint16_t)0x1000)***

- #define: ***TIM_ExtTRGPSC_DIV4 ((uint16_t)0x2000)***

- #define: ***TIM_ExtTRGPSC_DIV8 ((uint16_t)0x3000)***

TIM_Flags

- #define: ***TIM_FLAG_Update ((uint32_t)0x00001)***

- #define: ***TIM_FLAG_CC1 ((uint32_t)0x00002)***

- #define: ***TIM_FLAG_CC2 ((uint32_t)0x00004)***

- #define: ***TIM_FLAG_CC3 ((uint32_t)0x00008)***

- #define: ***TIM_FLAG_CC4 ((uint32_t)0x00010)***

- #define: ***TIM_FLAG_COM ((uint32_t)0x00020)***

- #define: ***TIM_FLAG_Trigger ((uint32_t)0x00040)***

- #define: ***TIM_FLAG_Break ((uint32_t)0x00080)***

- #define: ***TIM_FLAG_Break2 ((uint32_t)0x00100)***

- #define: ***TIM_FLAG_CC1OF ((uint32_t)0x00200)***

- #define: ***TIM_FLAG_CC2OF ((uint32_t)0x00400)***

- #define: ***TIM_FLAG_CC3OF ((uint32_t)0x00800)***

- #define: ***TIM_FLAG_CC4OF ((uint32_t)0x01000)***

- #define: ***TIM_FLAG_CC5 ((uint32_t)0x10000)***

- #define: ***TIM_FLAG_CC6 ((uint32_t)0x20000)***

TIM_Forced_Action

- #define: ***TIM_ForcedAction_Active ((uint16_t)0x0050)***

- #define: ***TIM_ForcedAction_InActive ((uint16_t)0x0040)***

TIM_Input_Capture_Polarity

- #define: ***TIM_ICPolarity_Rising ((uint16_t)0x0000)***

- #define: ***TIM_ICPolarity_Falling ((uint16_t)0x0002)***

- #define: ***TIM_ICPolarity_BothEdge ((uint16_t)0x000A)***

TIM_Input_Capture_Prescaler

- #define: ***TIM_ICPSC_DIV1 ((uint16_t)0x0000)***

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

- #define: ***TIM_ICPSC_DIV2 ((uint16_t)0x0004)***

Capture performed once every 2 events.

- #define: ***TIM_ICPSC_DIV4 ((uint16_t)0x0008)***

Capture performed once every 4 events.

- #define: ***TIM_ICPSC_DIV8 ((uint16_t)0x000C)***

Capture performed once every 8 events.

TIM_Input_Capture_Selection

- #define: ***TIM_ICSelection_DirectTI ((uint16_t)0x0001)***

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

- #define: ***TIM_ICSelection_IndirectTI ((uint16_t)0x0002)***

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

- #define: ***TIM_ICSelection_TRC ((uint16_t)0x0003)***

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

TIM_Internal_Trigger_Selection

- #define: ***TIM_TS_ITR0 ((uint16_t)0x0000)***

- #define: ***TIM_TS_ITR1 ((uint16_t)0x0010)***

- #define: ***TIM_TS_ITR2 ((uint16_t)0x0020)***

- #define: ***TIM_TS_ITR3 ((uint16_t)0x0030)***

- #define: ***TIM_TS_TI1F_ED ((uint16_t)0x0040)***

- #define: ***TIM_TS_TI1FP1 ((uint16_t)0x0050)***

- #define: ***TIM_TS_TI2FP2 ((uint16_t)0x0060)***

- #define: ***TIM_TS_ETRF ((uint16_t)0x0070)***

TIM_interrupt_sources

- #define: ***TIM_IT_Update ((uint16_t)0x0001)***

- #define: ***TIM_IT_CC1 ((uint16_t)0x0002)***

- #define: ***TIM_IT_CC2 ((uint16_t)0x0004)***

- #define: ***TIM_IT_CC3 ((uint16_t)0x0008)***

- #define: ***TIM_IT_CC4 ((uint16_t)0x0010)***

- #define: ***TIM_IT_COM ((uint16_t)0x0020)***

- #define: ***TIM_IT_Trigger ((uint16_t)0x0040)***

- #define: ***TIM_IT_Break ((uint16_t)0x0080)***

TIM_Legacy

- #define: ***TIM_DMABurstLength_1Byte TIM_DMABurstLength_1Transfer***
- #define: ***TIM_DMABurstLength_2Bytes TIM_DMABurstLength_2Transfers***
- #define: ***TIM_DMABurstLength_3Bytes TIM_DMABurstLength_3Transfers***
- #define: ***TIM_DMABurstLength_4Bytes TIM_DMABurstLength_4Transfers***
- #define: ***TIM_DMABurstLength_5Bytes TIM_DMABurstLength_5Transfers***
- #define: ***TIM_DMABurstLength_6Bytes TIM_DMABurstLength_6Transfers***
- #define: ***TIM_DMABurstLength_7Bytes TIM_DMABurstLength_7Transfers***
- #define: ***TIM_DMABurstLength_8Bytes TIM_DMABurstLength_8Transfers***
- #define: ***TIM_DMABurstLength_9Bytes TIM_DMABurstLength_9Transfers***
- #define: ***TIM_DMABurstLength_10Bytes TIM_DMABurstLength_10Transfers***
- #define: ***TIM_DMABurstLength_11Bytes TIM_DMABurstLength_11Transfers***

- #define: ***TIM_DMABurstLength_12Bytes TIM_DMABurstLength_12Transfers***
- #define: ***TIM_DMABurstLength_13Bytes TIM_DMABurstLength_13Transfers***
- #define: ***TIM_DMABurstLength_14Bytes TIM_DMABurstLength_14Transfers***
- #define: ***TIM_DMABurstLength_15Bytes TIM_DMABurstLength_15Transfers***
- #define: ***TIM_DMABurstLength_16Bytes TIM_DMABurstLength_16Transfers***
- #define: ***TIM_DMABurstLength_17Bytes TIM_DMABurstLength_17Transfers***
- #define: ***TIM_DMABurstLength_18Bytes TIM_DMABurstLength_18Transfers***

TIM_Lock_level

- #define: ***TIM_LOCKLevel_OFF ((uint16_t)0x0000)***
- #define: ***TIM_LOCKLevel_1 ((uint16_t)0x0100)***
- #define: ***TIM_LOCKLevel_2 ((uint16_t)0x0200)***
- #define: ***TIM_LOCKLevel_3 ((uint16_t)0x0300)***

TIM_Master_Slave_Mode

- #define: ***TIM_MasterSlaveMode_Enable ((uint16_t)0x0080)***

- #define: ***TIM_MasterSlaveMode_Disable ((uint16_t)0x0000)***

TIM_OCReferenceClear

- #define: ***TIM_OCReferenceClear_ETRF ((uint16_t)0x0008)***

- #define: ***TIM_OCReferenceClear_OCREFCLR ((uint16_t)0x0000)***

TIM_One_Pulse_Mode

- #define: ***TIM_OPMode_Single ((uint16_t)0x0008)***

- #define: ***TIM_OPMode_Repetitive ((uint16_t)0x0000)***

TIM_OSSI_Off_State_Selection_for_Idle_mode_state

- #define: ***TIM_OSSIState_Enable ((uint16_t)0x0400)***

- #define: ***TIM_OSSIState_Disable ((uint16_t)0x0000)***

TIM_OSSR_Off_State_Selection_for_Run_mode_state

- #define: ***TIM_OSSRState_Enable ((uint16_t)0x0800)***

- #define: ***TIM_OSSRState_Disable ((uint16_t)0x0000)***

TIM_Output_Compare_and_PWM_modes

- #define: ***TIM_OCMode_Timing ((uint32_t)0x00000)***

- #define: ***TIM_OCMode_Active ((uint32_t)0x00010)***
- #define: ***TIM_OCMode_Inactive ((uint32_t)0x00020)***
- #define: ***TIM_OCMode_Toggle ((uint32_t)0x00030)***
- #define: ***TIM_OCMode_PWM1 ((uint32_t)0x00060)***
- #define: ***TIM_OCMode_PWM2 ((uint32_t)0x00070)***
- #define: ***TIM_OCMode_Retrigerrable_OPM1 ((uint32_t)0x10000)***
- #define: ***TIM_OCMode_Retrigerrable_OPM2 ((uint32_t)0x10010)***
- #define: ***TIM_OCMode_Combined_PWM1 ((uint32_t)0x10040)***
- #define: ***TIM_OCMode_Combined_PWM2 ((uint32_t)0x10050)***
- #define: ***TIM_OCMode_Asymmetric_PWM1 ((uint32_t)0x10060)***
- #define: ***TIM_OCMode_Asymmetric_PWM2 ((uint32_t)0x10070)***

TIM_Output_Compare_Clear_State

- #define: ***TIM_OCClear_Enable ((uint16_t)0x0080)***

- #define: ***TIM_OCClear_Disable ((uint16_t)0x0000)***

TIM_Output_Compare_Fast_State

- #define: ***TIM_OCFast_Enable ((uint16_t)0x0004)***

- #define: ***TIM_OCFast_Disable ((uint16_t)0x0000)***

TIM_Output_Compare_Idle_State

- #define: ***TIM_OCIdleState_Set ((uint16_t)0x0100)***

- #define: ***TIM_OCIdleState_Reset ((uint16_t)0x0000)***

TIM_Output_Compare_N_Idle_State

- #define: ***TIM_OCNIdleState_Set ((uint16_t)0x0200)***

- #define: ***TIM_OCNIdleState_Reset ((uint16_t)0x0000)***

TIM_Output_Compare_N_Polarity

- #define: ***TIM_OCNPolarity_High ((uint16_t)0x0000)***

- #define: ***TIM_OCNPolarity_Low ((uint16_t)0x0008)***

TIM_Output_Compare_N_State

- #define: ***TIM_OutputNState_Disable ((uint16_t)0x0000)***

- #define: ***TIM_OutputNState_Enable ((uint16_t)0x0004)***

TIM_Output_Compare_Polarity

- #define: ***TIM_OCPolarity_High ((uint16_t)0x0000)***

- #define: ***TIM_OCPolarity_Low ((uint16_t)0x0002)***

TIM_Output_Compare_Preload_State

- #define: ***TIM_OCPreload_Enable ((uint16_t)0x0008)***

- #define: ***TIM_OCPreload_Disable ((uint16_t)0x0000)***

TIM_Output_Compare_State

- #define: ***TIM_OutputState_Disable ((uint16_t)0x0000)***

- #define: ***TIM_OutputState_Enable ((uint16_t)0x0001)***

TIM_Prescaler_Reload_Mode

- #define: ***TIM_PSCReloadMode_Update ((uint16_t)0x0000)***

- #define: ***TIM_PSCReloadMode_Immediate ((uint16_t)0x0001)***

TIM_Remap

- #define: ***TIM16_GPIO ((uint16_t)0x0000)***

- #define: ***TIM16_RTC_CLK ((uint16_t)0x0001)***
- #define: ***TIM16_HSEDiv32 ((uint16_t)0x0002)***
- #define: ***TIM16_MCO ((uint16_t)0x0003)***
- #define: ***TIM1_ADC1_AWDG1 ((uint16_t)0x0001)***
- #define: ***TIM1_ADC1_AWDG2 ((uint16_t)0x0002)***
- #define: ***TIM1_ADC1_AWDG3 ((uint16_t)0x0003)***
- #define: ***TIM1_ADC4_AWDG1 ((uint16_t)0x0004)***
- #define: ***TIM1_ADC4_AWDG2 ((uint16_t)0x0008)***
- #define: ***TIM1_ADC4_AWDG3 ((uint16_t)0x000C)***
- #define: ***TIM8_ADC2_AWDG1 ((uint16_t)0x0001)***
- #define: ***TIM8_ADC2_AWDG2 ((uint16_t)0x0002)***
- #define: ***TIM8_ADC2_AWDG3 ((uint16_t)0x0003)***

- #define: ***TIM8_ADC3_AWDG1 ((uint16_t)0x0004)***

- #define: ***TIM8_ADC3_AWDG2 ((uint16_t)0x0008)***

- #define: ***TIM8_ADC3_AWDG3 ((uint16_t)0x000C)***

TIM_Slave_Mode

- #define: ***TIM_SlaveMode_Reset ((uint32_t)0x00004)***

- #define: ***TIM_SlaveMode_Gated ((uint32_t)0x00005)***

- #define: ***TIM_SlaveMode_Trigger ((uint32_t)0x00006)***

- #define: ***TIM_SlaveMode_External1 ((uint32_t)0x00007)***

- #define: ***TIM_SlaveMode_Combined_ResetTrigger ((uint32_t)0x10000)***

TIM_TIx_External_Clock_Source

- #define: ***TIM_TIxExternalCLK1Source_TI1 ((uint16_t)0x0050)***

- #define: ***TIM_TIxExternalCLK1Source_TI2 ((uint16_t)0x0060)***

- #define: ***TIM_TIxExternalCLK1Source_TI1ED ((uint16_t)0x0040)***

TIM_Trigger_Output_Source

- #define: ***TIM_TRGOSource_Reset ((uint16_t)0x0000)***
- #define: ***TIM_TRGOSource_Enable ((uint16_t)0x0010)***
- #define: ***TIM_TRGOSource_Update ((uint16_t)0x0020)***
- #define: ***TIM_TRGOSource_OC1 ((uint16_t)0x0030)***
- #define: ***TIM_TRGOSource_OC1Ref ((uint16_t)0x0040)***
- #define: ***TIM_TRGOSource_OC2Ref ((uint16_t)0x0050)***
- #define: ***TIM_TRGOSource_OC3Ref ((uint16_t)0x0060)***
- #define: ***TIM_TRGOSource_OC4Ref ((uint16_t)0x0070)***
- #define: ***TIM_TRGO2Source_Reset ((uint32_t)0x00000000)***
- #define: ***TIM_TRGO2Source_Enable ((uint32_t)0x00100000)***
- #define: ***TIM_TRGO2Source_Update ((uint32_t)0x00200000)***
- #define: ***TIM_TRGO2Source_OC1 ((uint32_t)0x00300000)***

- #define: ***TIM_TRGO2Source_OC1Ref ((uint32_t)0x00400000)***
- #define: ***TIM_TRGO2Source_OC2Ref ((uint32_t)0x00500000)***
- #define: ***TIM_TRGO2Source_OC3Ref ((uint32_t)0x00600000)***
- #define: ***TIM_TRGO2Source_OC4Ref ((uint32_t)0x00700000)***
- #define: ***TIM_TRGO2Source_OC5Ref ((uint32_t)0x00800000)***
- #define: ***TIM_TRGO2Source_OC6Ref ((uint32_t)0x00900000)***
- #define: ***TIM_TRGO2Source_OC4Ref_RisingFalling ((uint32_t)0x00A00000)***
- #define: ***TIM_TRGO2Source_OC6Ref_RisingFalling ((uint32_t)0x00B00000)***
- #define: ***TIM_TRGO2Source_OC4RefRising_OC6RefRising ((uint32_t)0x00C00000)***
- #define: ***TIM_TRGO2Source_OC4RefRising_OC6RefFalling ((uint32_t)0x00D00000)***
- #define: ***TIM_TRGO2Source_OC5RefRising_OC6RefRising ((uint32_t)0x00E00000)***
- #define: ***TIM_TRGO2Source_OC5RefRising_OC6RefFalling ((uint32_t)0x00F00000)***

TIM_Update_Source

- #define: ***TIM_UpdateSource_Global ((uint16_t)0x0000)***

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 ((uint16_t)0x0001)***

Source of update is counter overflow/underflow.

23 Universal synchronous asynchronous receiver transmitter (USART)

23.1 USART Firmware driver registers structures

23.1.1 USART_TypeDef

USART_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t CR3***
- ***__IO uint16_t BRR***
- ***uint16_t RESERVED1***
- ***__IO uint16_t GTPR***
- ***uint16_t RESERVED2***
- ***__IO uint32_t RTOR***
- ***__IO uint16_t RQR***
- ***uint16_t RESERVED3***
- ***__IO uint32_t ISR***
- ***__IO uint32_t ICR***
- ***__IO uint16_t RDR***
- ***uint16_t RESERVED4***
- ***__IO uint16_t TDR***
- ***uint16_t RESERVED5***

Field Documentation

- ***__IO uint32_t USART_TypeDef::CR1***
  - USART Control register 1, Address offset: 0x00
- ***__IO uint32_t USART_TypeDef::CR2***
  - USART Control register 2, Address offset: 0x04
- ***__IO uint32_t USART_TypeDef::CR3***
  - USART Control register 3, Address offset: 0x08
- ***__IO uint16_t USART_TypeDef::BRR***
  - USART Baud rate register, Address offset: 0x0C
- ***uint16_t USART_TypeDef::RESERVED1***
  - Reserved, 0x0E
- ***__IO uint16_t USART_TypeDef::GTPR***
  - USART Guard time and prescaler register, Address offset: 0x10
- ***uint16_t USART_TypeDef::RESERVED2***
  - Reserved, 0x12
- ***__IO uint32_t USART_TypeDef::RTOR***
  - USART Receiver Time Out register, Address offset: 0x14
- ***__IO uint16_t USART_TypeDef::RQR***

- USART Request register, Address offset: 0x18
- ***uint16_t USART_TypeDef::RESERVED3***
  - Reserved, 0x1A
- ***__IO uint32_t USART_TypeDef::ISR***
  - USART Interrupt and status register, Address offset: 0x1C
- ***__IO uint32_t USART_TypeDef::ICR***
  - USART Interrupt flag Clear register, Address offset: 0x20
- ***__IO uint16_t USART_TypeDef::RDR***
  - USART Receive Data register, Address offset: 0x24
- ***uint16_t USART_TypeDef::RESERVED4***
  - Reserved, 0x26
- ***__IO uint16_t USART_TypeDef::TDR***
  - USART Transmit Data register, Address offset: 0x28
- ***uint16_t USART_TypeDef::RESERVED5***
  - Reserved, 0x2A

23.1.2 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32f30x_usart.h

Data Fields

- ***uint32_t USART_BaudRate***
- ***uint32_t USART_WordLength***
- ***uint32_t USART_StopBits***
- ***uint32_t USART_Parity***
- ***uint32_t USART_Mode***
- ***uint32_t USART_HardwareFlowControl***

Field Documentation

- ***uint32_t USART_InitTypeDef::USART_BaudRate***
  - This member configures the USART communication baud rate. The baud rate is computed using the following formula: IntegerDivider = ((PCLKx) / (16 * (USART_InitStruct->USART_BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***uint32_t USART_InitTypeDef::USART_WordLength***
  - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_Word_Length***
- ***uint32_t USART_InitTypeDef::USART_StopBits***
  - Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***
- ***uint32_t USART_InitTypeDef::USART_Parity***
  - Specifies the parity mode. This parameter can be a value of ***USART_Parity***
- ***uint32_t USART_InitTypeDef::USART_Mode***
  - Specifies wether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***
- ***uint32_t USART_InitTypeDef::USART_HardwareFlowControl***

– Specifies wether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_Hardware_Flow_Control***

23.1.3 USART_ClockInitTypeDef

USART_ClockInitTypeDef is defined in the stm32f30x_usart.h

Data Fields

- ***uint32_t USART_Clock***
- ***uint32_t USART_CPOL***
- ***uint32_t USART_CPHA***
- ***uint32_t USART_LastBit***

Field Documentation

- ***uint32_t USART_ClockInitTypeDef::USART_Clock***
  – Specifies whether the USART clock is enabled or disabled. This parameter can be a value of ***USART_Clock***
- ***uint32_t USART_ClockInitTypeDef::USART_CPOL***
  – Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***
- ***uint32_t USART_ClockInitTypeDef::USART_CPHA***
  – Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***
- ***uint32_t USART_ClockInitTypeDef::USART_LastBit***
  – Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***

23.2 USART Firmware driver API description

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

23.2.1 How to use this driver

1. Enable peripheral clock using RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE) function for USART1 or using RCC_APB1PeriphClockCmd(RCC_APB1Periph_USARTx, ENABLE) function for USART2, USART3, UART4 and UART5.
2. According to the USART mode, enable the GPIO clocks using RCC_AHBPeriphClockCmd() function. (The I/O can be TX, RX, CTS, or and SCLK).
3. Peripheral's alternate function:
   – Connect the pin to the desired peripherals' Alternate Function (AF) using GPIO_PinAFConfig() function.

- Configure the desired pin in alternate function by: GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF.
- Select the type, pull-up/pull-down and output speed via GPIO_PuPd, GPIO_OType and GPIO_Speed members.
- Call GPIO_Init() function.

4. Program the Baud Rate, Word Length , Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) using the SPI_Init() function.
5. For synchronous mode, enable the clock and program the polarity, phase and last bit using the USART_ClockInit() function.
6. Enable the NVIC and the corresponding interrupt using the function USART_ITConfig() if you need to use interrupt mode.
7. When using the DMA mode:
   - Configure the DMA using DMA_Init() function.
   - Active the needed channel Request using USART_DMACmd() function.
8. Enable the USART using the USART_Cmd() function.
9. Enable the DMA using the DMA_Cmd() function, when using DMA mode.

Refer to Multi-Processor, LIN, half-duplex, Smartcard, IrDA sub-sections for more details.

23.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate.
  - Word Length.
  - Stop Bit.
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible USART frame formats are as listed in the following table:

M bit	PCE bit	UART frame
0	0	SB | 8 bit data | STB
0	1	SB | 7 bit data | PB | STB
1	0	SB | 9 bit data | STB
1	1	SB | 8 bit data | PB | STB

(++) Hardware flow control. (++) Receiver/transmitter modes.

The USART_Init() function follows the USART asynchronous configuration procedure(details for the procedure are available in reference manual.

- For the synchronous mode in addition to the asynchronous mode parameters these parameters should be also configured:
  - USART Clock Enabled.
  - USART polarity.
  - USART phase.
  - USART LastBit.

These parameters can be configured using the USART_ClockInit() function.

- ***USART_DeInit()***
- ***USART_Init()***
- ***USART_StructInit()***
- ***USART_ClockInit()***
- ***USART_ClockStructInit()***
- ***USART_Cmd()***
- ***USART_DirectionModeCmd()***
- ***USART_OverSampling8Cmd()***
- ***USART_OneBitMethodCmd()***
- ***USART_MSBFirstCmd()***
- ***USART_DataInvCmd()***
- ***USART_InvPinCmd()***
- ***USART_SWAPPinCmd()***
- ***USART_ReceiverTimeOutCmd()***
- ***USART_SetReceiverTimeOut()***
- ***USART_SetPrescaler()***

23.2.3 RS485 mode functions

This subsection provides a set of functions allowing to manage the USART RS485 flow control.

RS485 flow control (Driver enable feature) handling is possible through the following procedure:

1. Program the Baud rate, Word length = 8 bits, Stop bits, Parity, Transmitter/Receiver modes and hardware flow control values using the USART_Init() function.
2. Enable the Driver Enable using the USART_DECmd() function.
3. Configures the Driver Enable polarity using the USART_DEPolarityConfig() function.
4. Configures the Driver Enable assertion time using USART_SetDEAssertionTime() function and deassertion time using the USART_SetDEDeassertionTime() function.
5. Enable the USART using the USART_Cmd() function.

The assertion and dessertion times are expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).

- ***USART_DECmd()***
- ***USART_DEPolarityConfig()***
- ***USART_SetDEAssertionTime()***
- ***USART_SetDEDeassertionTime()***

23.2.4 DMA transfers management functions

This section provides two functions that can be used only in DMA mode.

In DMA Mode, the USART communication can be managed by 2 DMA Channel requests:

1. USART_DMAReq_Tx: specifies the Tx buffer DMA transfer request.
2. USART_DMAReq_Rx: specifies the Rx buffer DMA transfer request.

In this Mode it is advised to use the following function:

- void USART_DMACmd(USART_TypeDef* USARTx, uint16_t USART_DMAReq, FunctionalState NewState).
- ***USART_DMACmd()***
- ***USART_DMAReceptionErrorConfig()***

23.2.5 Interrupts and flags management functions

This subsection provides a set of functions allowing to configure the USART 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.

Polling Mode

In Polling Mode, the SPI communication can be managed by these flags:

1. USART_FLAG_REACK: to indicate the status of the Receive Enable acknowledge flag
2. USART_FLAG_TEACK: to indicate the status of the Transmit Enable acknowledge flag.
3. USART_FLAG_WUF: to indicate the status of the Wake up flag.
4. USART_FLAG_RWU: to indicate the status of the Receive Wake up flag.
5. USART_FLAG_SBK: to indicate the status of the Send Break flag.
6. USART_FLAG_CMF: to indicate the status of the Character match flag.
7. USART_FLAG_BUSY: to indicate the status of the Busy flag.
8. USART_FLAG_ABRF: to indicate the status of the Auto baud rate flag.
9. USART_FLAG_ABRE: to indicate the status of the Auto baud rate error flag.
10. USART_FLAG_EOBF: to indicate the status of the End of block flag.
11. USART_FLAG_RTOF: to indicate the status of the Receive time out flag.
12. USART_FLAG_nCTSS: to indicate the status of the Inverted nCTS input bit status.
13. USART_FLAG_TXE: to indicate the status of the transmit buffer register.
14. USART_FLAG_RXNE: to indicate the status of the receive buffer register.
15. USART_FLAG_TC: to indicate the status of the transmit operation.
16. USART_FLAG_IDLE: to indicate the status of the Idle Line.
17. USART_FLAG_CTS: to indicate the status of the nCTS input.
18. USART_FLAG_LBD: to indicate the status of the LIN break detection.
19. USART_FLAG_NE: to indicate if a noise error occur.
20. USART_FLAG_FE: to indicate if a frame error occur.
21. USART_FLAG_PE: to indicate if a parity error occur.
22. USART_FLAG_ORE: to indicate if an Overrun error occur.

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

- FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG).
- void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG).

Interrupt Mode

In Interrupt Mode, the USART communication can be managed by 8 interrupt sources and 10 pending bits:

- Pending Bits:
  a. USART_IT_WU: to indicate the status of the Wake up interrupt.

b. USART_IT_CM: to indicate the status of Character match interrupt.
c. USART_IT_EOB: to indicate the status of End of block interrupt.
d. USART_IT_RTO: to indicate the status of Receive time out interrupt.
e. USART_IT_CTS: to indicate the status of CTS change interrupt.
f. USART_IT_LBD: to indicate the status of LIN Break detection interrupt.
g. USART_IT_TC: to indicate the status of Transmission complete interrupt.
h. USART_IT_IDLE: to indicate the status of IDLE line detected interrupt.
i. USART_IT_ORE: to indicate the status of OverRun Error interrupt.
j. USART_IT_NE: to indicate the status of Noise Error interrupt.
k. USART_IT_FE: to indicate the status of Framing Error interrupt.
l. USART_IT_PE: to indicate the status of Parity Error interrupt.

- Interrupt Source:
  a. USART_IT_WU: specifies the interrupt source for Wake up interrupt.
  b. USART_IT_CM: specifies the interrupt source for Character match interrupt.
  c. USART_IT_EOB: specifies the interrupt source for End of block interrupt.
  d. USART_IT_RTO: specifies the interrupt source for Receive time-out interrupt.
  e. USART_IT_CTS: specifies the interrupt source for CTS change interrupt.
  f. USART_IT_LBD: specifies the interrupt source for LIN Break detection interrupt.
  g. USART_IT_TXE: specifies the interrupt source for Tansmit Data Register empty interrupt.
  h. USART_IT_TC: specifies the interrupt source for Transmission complete interrupt.
  i. USART_IT_RXNE: specifies the interrupt source for Receive Data register not empty interrupt.
  j. USART_IT_IDLE: specifies the interrupt source for Idle line detection interrupt.
  k. USART_IT_PE: specifies the interrupt source for Parity Error interrupt.
  l. USART_IT_ERR: specifies the interrupt source for Error interrupt (Frame error, noise error, overrun error) Some 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 USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState).
- ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT).
- void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT).
- ***USART_ITConfig()***
- ***USART_RequestCmd()***
- ***USART_OverrunDetectionConfig()***
- ***USART_GetFlagStatus()***
- ***USART_ClearFlag()***
- ***USART_GetITStatus()***
- ***USART_ClearITPendingBit()***

23.2.6 STOP Mode functions

This subsection provides a set of functions allowing to manage WakeUp from STOP mode.

The USART is able to WakeUp from Stop Mode if USART clock is set to HSI or LSI.

The WakeUp source is configured by calling USART_StopModeWakeUpSourceConfig() function.

After configuring the source of WakeUp and before entering in Stop Mode USART_STOPModeCmd() function should be called to allow USART WakeUp.

- ***USART_STOPModeCmd()***
- ***USART_StopModeWakeUpSourceConfig()***

23.2.7 AutoBaudRate functions

This subsection provides a set of functions allowing to manage the AutoBaudRate detections.

Before Enabling AutoBaudRate detection using USART_AutoBaudRateCmd () The character patterns used to calculate baudrate must be chosen by calling USART_AutoBaudRateConfig() function. These function take as parameter :

1. USART_AutoBaudRate_StartBit : any character starting with a bit 1.
2. USART_AutoBaudRate_FallingEdge : any character starting with a 10xx bit pattern.

At any later time, another request for AutoBaudRate detection can be performed using USART_RequestCmd() function.

The AutoBaudRate detection is monitored by the status of ABRF flag which indicate that the AutoBaudRate detection is completed. In addition to ABRF flag, the ABRE flag indicate that this procedure is completed without success. USART_GetFlagStatus () function should be used to monitor the status of these flags.

- ***USART_AutoBaudRateCmd()***
- ***USART_AutoBaudRateConfig()***

23.2.8 Data transfers functions

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

During an USART reception, data shifts in least significant bit first through the RX pin. When a transmission is taking place, a write instruction to the USART_TDR register stores the data in the shift register.

The read access of the USART_RDR register can be done using the USART_ReceiveData() function and returns the RDR value. Whereas a write access to the USART_TDR can be done using USART_SendData() function and stores the written data into TDR.

- ***USART_SendData()***
- ***USART_ReceiveData()***

23.2.9 Multi-Processor Communication functions

This subsection provides a set of functions allowing to manage the USART multiprocessor communication.

For instance one of the USARTs can be the master, its TX output is connected to the RX input of the other USART. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master. USART multiprocessor communication is possible through the following procedure:

1. Program the Baud rate, Word length = 9 bits, Stop bits, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Configures the USART address using the USART_SetAddress() function.
3. Configures the wake up methode (USART_WakeUp_IdleLine or USART_WakeUp_AddressMark) using USART_WakeUpConfig() function only for the slaves.
4. Enable the USART using the USART_Cmd() function.
5. Enter the USART slaves in mute mode using USART_ReceiverWakeUpCmd() function.

The USART Slave exit from mute mode when receive the wake up condition.

- ***USART_SetAddress()***
- ***USART_MuteModeCmd()***
- ***USART_MuteModeWakeUpConfig()***
- ***USART_AddressDetectionConfig()***

23.2.10 LIN mode functions

This subsection provides a set of functions allowing to manage the USART LIN Mode communication.

In LIN mode, 8-bit data format with 1 stop bit is required in accordance with the LIN standard.

Only this LIN Feature is supported by the USART IP:

- LIN Master Synchronous Break send capability and LIN slave break detection capability : 13-bit break generation and 10/11 bit break detection.

USART LIN Master transmitter communication is possible through the following procedure:

1. Program the Baud rate, Word length = 8bits, Stop bits = 1bit, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Enable the LIN mode using the USART_LINCmd() function.
3. Enable the USART using the USART_Cmd() function.
4. Send the break character using USART_SendBreak() function.

USART LIN Master receiver communication is possible through the following procedure:

1. Program the Baud rate, Word length = 8bits, Stop bits = 1bit, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Configures the break detection length using the USART_LINBreakDetectLengthConfig() function.
3. Enable the LIN mode using the USART_LINCmd() function.
4. Enable the USART using the USART_Cmd() function.

In LIN mode, the following bits must be kept cleared:

- CLKEN in the USART_CR2 register.
- STOP[1:0], SCEN, HDSEL and IREN in the USART_CR3 register.

- ***USART_LINBreakDetectLengthConfig()***
- ***USART_LINCmd()***

23.2.11 Half-duplex mode function

This subsection provides a set of functions allowing to manage the USART Half-duplex communication.

The USART can be configured to follow a single-wire half-duplex protocol where the TX and RX lines are internally connected.

USART Half duplex communication is possible through the following procedure:

1. Program the Baud rate, Word length, Stop bits, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Configures the USART address using the USART_SetAddress() function.
3. Enable the half duplex mode using USART_HalfDuplexCmd() function.
4. Enable the USART using the USART_Cmd() function.

The RX pin is no longer used.

In Half-duplex mode the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART_CR2 register.
- SCEN and IREN bits in the USART_CR3 register.

- ***USART_HalfDuplexCmd()***

23.2.12 Smartcard mode functions

This subsection provides a set of functions allowing to manage the USART Smartcard communication.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

Smartcard communication is possible through the following procedure:

1. Configures the Smartcard Prsecaler using the USART_SetPrescaler() function.
2. Configures the Smartcard Guard Time using the USART_SetGuardTime() function.
3. Program the USART clock using the USART_ClockInit() function as following:
   - USART Clock enabled.
   - USART CPOL Low.
   - USART CPHA on first edge.
   - USART Last Bit Clock Enabled.
4. Program the Smartcard interface using the USART_Init() function as following:
   - Word Length = 9 Bits.
   - 1.5 Stop Bit.
   - Even parity.
   - BaudRate = 12096 baud.

- Hardware flow control disabled (RTS and CTS signals).
- Tx and Rx enabled

5. Optionally you can enable the parity error interrupt using the USART_ITConfig() function.
6. Enable the Smartcard NACK using the USART_SmartCardNACKCmd() function.
7. Enable the Smartcard interface using the USART_SmartCardCmd() function.
8. Enable the USART using the USART_Cmd() function.

Please refer to the ISO 7816-3 specification for more details.

It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

In smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register.
- HDSEL and IREN bits in the USART_CR3 register.

- ***USART_SetGuardTime()***
- ***USART_SmartCardCmd()***
- ***USART_SmartCardNACKCmd()***
- ***USART_SetAutoRetryCount()***
- ***USART_SetBlockLength()***

23.2.13 IrDA mode functions

This subsection provides a set of functions allowing to manage the USART IrDA communication.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

IrDA communication is possible through the following procedure:

1. Program the Baud rate, Word length = 8 bits, Stop bits, Parity, Transmitter/Receiver modes and hardware flow control values using the USART_Init() function.
2. Configures the IrDA pulse width by configuring the prescaler using the USART_SetPrescaler() function.
3. Configures the IrDA USART_IrDAMode_LowPower or USART_IrDAMode_Normal mode using the USART_IrDAConfig() function.
4. Enable the IrDA using the USART_IrDACmd() function.
5. Enable the USART using the USART_Cmd() function.

A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.

The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).

In IrDA mode, the following bits must be kept cleared:

- LINEN, STOP and CLKEN bits in the USART_CR2 register.
- SCEN and HDSEL bits in the USART_CR3 register.

- ***USART_IrDAConfig()***
- ***USART_IrDACmd()***

23.2.14 Initialization and Configuration functions

23.2.14.1 USART_DeInit

Function Name	**void USART_DeInit (*USART_TypeDef** * USARTx)**
Function Description	Deinitializes the USARTx peripheral registers to their default reset values.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
Return values	• None.
Notes	• None.

23.2.14.2 USART_Init

Function Name	**void USART_Init (*USART_TypeDef** * USARTx, *USART_InitTypeDef** * USART_InitStruct)**
Function Description	Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct .
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_InitStruct :** pointer to a USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.
Return values	• None.

Notes	• None.

23.2.14.3 USART_StructInit

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

23.2.14.4 USART_ClockInit

Function Name	**void USART_ClockInit (*USART_TypeDef** * USARTx, *USART_ClockInitTypeDef** * USART_ClockInitStruct)**
Function Description	Initializes the USARTx peripheral Clock according to the specified parameters in the USART_ClockInitStruct.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3. • **USART_ClockInitStruct :** pointer to a USART_ClockInitTypeDef structure that contains the configuration information for the specified USART peripheral.
Return values	• None.
Notes	• None.

23.2.14.5 USART_ClockStructInit

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

23.2.14.6 USART_Cmd

Function Name	**void USART_Cmd (*USART_TypeDef** * **USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the specified USART peripheral.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new state of the USARTx peripheral. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.2.14.7 USART_DirectionModeCmd

Function Name	**void USART_DirectionModeCmd (*USART_TypeDef** * **USARTx, uint32_t USART_DirectionMode, *FunctionalState* NewState)**
Function Description	Enables or disables the USART's transmitter or receiver.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_Direction :** specifies the USART direction. This parameter can be any combination of the following values: – ***USART_Mode_Tx :*** USART Transmitter – ***USART_Mode_Rx :*** USART Receiver

- **NewState :** new state of the USART transfer direction. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

23.2.14.8 USART_OverSampling8Cmd

Function Name **void USART_OverSampling8Cmd (*USART_TypeDef** USARTx, *FunctionalState* NewState)**

Function Description Enables or disables the USART's 8x oversampling mode.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **NewState :** new state of the USART 8x oversampling mode. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- This function has to be called before calling USART_Init() function in order to have correct baudrate Divider value.

23.2.14.9 USART_OneBitMethodCmd

Function Name **void USART_OneBitMethodCmd (*USART_TypeDef** USARTx, *FunctionalState* NewState)**

Function Description Enables or disables the USART's one bit sampling method.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **NewState :** new state of the USART one bit sampling method. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- This function has to be called before calling USART_Cmd() function.

23.2.14.10 USART_MSBFirstCmd

Function Name	**void USART_MSBFirstCmd (*USART_TypeDef** * **USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the USART's most significant bit first transmitted/received following the start bit.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new state of the USART most significant bit first transmitted/received following the start bit. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• This function has to be called before calling USART_Cmd() function.

23.2.14.11 USART_DataInvCmd

Function Name	**void USART_DataInvCmd (*USART_TypeDef** * **USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the binary data inversion.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new defined levels for the USART data. This parameter can be: ENABLE or DISABLE. – ***ENABLE :*** Logical data from the data register are send/received in negative logic. (1=L, 0=H). The parity bit is also inverted. – ***DISABLE :*** Logical data from the data register are send/received in positive logic. (1=H, 0=L)
Return values	• None.
Notes	• This function has to be called before calling USART_Cmd() function.

23.2.14.12 USART_InvPinCmd

Function Name	**void USART_InvPinCmd (*USART_TypeDef** * USARTx, uint32_t USART_InvPin, *FunctionalState* NewState)**
Function Description	Enables or disables the Pin(s) active level inversion.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_InvPin :** specifies the USART pin(s) to invert. This parameter can be any combination of the following values: – ***USART_InvPin_Tx :*** USART Tx pin active level inversion. – ***USART_InvPin_Rx :*** USART Rx pin active level inversion. • **NewState :** new active level status for the USART pin(s). This parameter can be: ENABLE or DISABLE. – ***ENABLE :*** pin(s) signal values are inverted (Vdd =0, Gnd =1). – ***DISABLE :*** pin(s) signal works using the standard logic levels (Vdd =1, Gnd =0).
Return values	• None.
Notes	• This function has to be called before calling USART_Cmd() function.

23.2.14.13 USART_SWAPPinCmd

Function Name	**void USART_SWAPPinCmd (*USART_TypeDef** * USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the swap Tx/Rx pins.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new state of the USARTx TX/RX pins pinout. This parameter can be: ENABLE or DISABLE. – ***ENABLE :*** The TX and RX pins functions are swapped. – ***DISABLE :*** TX/RX pins are used as defined in standard pinout
Return values	• None.

Notes	• This function has to be called before calling USART_Cmd() function.

23.2.14.14 USART_ReceiverTimeOutCmd

Function Name	**void USART_ReceiverTimeOutCmd (*USART_TypeDef* * USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the receiver Time Out feature.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new state of the USARTx receiver Time Out. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.2.14.15 USART_SetReceiverTimeOut

Function Name	**void USART_SetReceiverTimeOut (*USART_TypeDef* * USARTx, uint32_t USART_ReceiverTimeOut)**
Function Description	Sets the receiver Time Out value.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_ReceiverTimeOut :** specifies the Receiver Time Out value.
Return values	• None.
Notes	• None.

23.2.14.16 USART_SetPrescaler

Function Name	**void USART_SetPrescaler (*USART_TypeDef** * USARTx, uint8_t USART_Prescaler)**
Function Description	Sets the system clock prescaler.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_Prescaler :** specifies the prescaler clock.
Return values	• None.
Notes	• This function has to be called before calling USART_Cmd() function.

23.2.15 RS485 mode function

23.2.15.1 USART_DECmd

Function Name	**void USART_DECmd (*USART_TypeDef** * USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the USART's DE functionality.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new state of the driver enable mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.2.15.2 USART_DEPolarityConfig

Function Name	**void USART_DEPolarityConfig (*USART_TypeDef** * USARTx, uint32_t USART_DEPolarity)**

Function Description	Configures the USART's DE polarity.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_DEPolarity :** specifies the DE polarity. This parameter can be one of the following values: – ***USART_DEPolarity_Low :*** – ***USART_DEPolarity_High :***
Return values	• None.
Notes	• None.

23.2.15.3 USART_SetDEAssertionTime

Function Name	**void USART_SetDEAssertionTime (*USART_TypeDef** USARTx, uint32_t USART_DEAssertionTime)**
Function Description	Sets the specified RS485 DE assertion time.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_AssertionTime :** specifies the time between the activation of the DE signal and the beginning of the start bit
Return values	• None.
Notes	• None.

23.2.15.4 USART_SetDEDeassertionTime

Function Name	**void USART_SetDEDeassertionTime (*USART_TypeDef** USARTx, uint32_t USART_DEDeassertionTime)**
Function Description	Sets the specified RS485 DE deassertion time.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_DeassertionTime :** specifies the time between the middle of the last stop bit in a transmitted message and the

de-activation of the DE signal

Return values
- None.

Notes
- None.

23.2.16 DMA transfers management functions

23.2.16.1 USART_DMACmd

Function Name **void USART_DMACmd (*USART_TypeDef** * USARTx, uint32_t USART_DMAReq, *FunctionalState* NewState)**

Function Description Enables or disables the USART's DMA interface.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4.
- **USART_DMAReq :** specifies the DMA request. This parameter can be any combination of the following values:
  - ***USART_DMAReq_Tx :*** USART DMA transmit request
  - ***USART_DMAReq_Rx :*** USART DMA receive request
- **NewState :** new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

23.2.16.2 USART_DMAReceptionErrorConfig

Function Name **void USART_DMAReceptionErrorConfig (*USART_TypeDef** * USARTx, uint32_t USART_DMAOnError)**

Function Description Enables or disables the USART's DMA interface when reception error occurs.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4.
- **USART_DMAOnError :** specifies the DMA status in case of reception error. This parameter can be any combination of the following values:

- ***USART_DMAOnError_Enable :*** DMA receive request enabled when the USART DMA reception error is asserted.
- ***USART_DMAOnError_Disable :*** DMA receive request disabled when the USART DMA reception error is asserted.

Return values
- None.

Notes
- None.

23.2.17 Interrupts and flags management functions

23.2.17.1 USART_ITConfig

Function Name **void USART_ITConfig (*USART_TypeDef** * USARTx, uint32_t USART_IT, *FunctionalState* NewState)**

Function Description Enables or disables the specified USART interrupts.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **USART_IT :** specifies the USART interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ***USART_IT_WU :*** Wake up interrupt.
  - ***USART_IT_CM :*** Character match interrupt.
  - ***USART_IT_EOB :*** End of block interrupt.
  - ***USART_IT_RTO :*** Receive time out interrupt.
  - ***USART_IT_CTS :*** CTS change interrupt.
  - ***USART_IT_LBD :*** LIN Break detection interrupt.
  - ***USART_IT_TXE :*** Tansmit Data Register empty interrupt.
  - ***USART_IT_TC :*** Transmission complete interrupt.
  - ***USART_IT_RXNE :*** Receive Data register not empty interrupt.
  - ***USART_IT_IDLE :*** Idle line detection interrupt.
  - ***USART_IT_PE :*** Parity Error interrupt.
  - ***USART_IT_ERR :*** Error interrupt(Frame error, noise error, overrun error)
- **NewState :** new state of the specified USARTx interrupts. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

23.2.17.2 USART_RequestCmd

Function Name	**void USART_RequestCmd (*USART_TypeDef** * USARTx, uint32_t USART_Request, *FunctionalState* NewState)**
Function Description	Enables the specified USART's Request.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_Request :** specifies the USART request. This parameter can be any combination of the following values: – ***USART_Request_TXFRQ :*** Transmit data flush ReQuest – ***USART_Request_RXFRQ :*** Receive data flush ReQuest – ***USART_Request_MMRQ :*** Mute Mode ReQuest – ***USART_Request_SBKRQ :*** Send Break ReQuest – ***USART_Request_ABRRQ :*** Auto Baud Rate ReQuest • **NewState :** new state of the DMA interface when reception error occurs. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.2.17.3 USART_OverrunDetectionConfig

Function Name	**void USART_OverrunDetectionConfig (*USART_TypeDef** * USARTx, uint32_t USART_OVRDetection)**
Function Description	Enables or disables the USART's Overrun detection.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_OVRDetection :** specifies the OVR detection status in case of OVR error. This parameter can be any combination of the following values: – ***USART_OVRDetection_Enable :*** OVR error detection enabled when the USART OVR error is asserted. – ***USART_OVRDetection_Disable :*** OVR error detection disabled when the USART OVR error is asserted.

Return values
- None.

Notes
- None.

23.2.17.4 USART_GetFlagStatus

Function Name
FlagStatus USART_GetFlagStatus (*USART_TypeDef USARTx, uint32_t USART_FLAG)**

Function Description
Checks whether the specified USART flag is set or not.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **USART_FLAG :** specifies the flag to check. This parameter can be one of the following values:
  - ***USART_FLAG_REACK :*** Receive Enable acknowledge flag.
  - ***USART_FLAG_TEACK :*** Transmit Enable acknowledge flag.
  - ***USART_FLAG_WUF :*** Wake up flag.
  - ***USART_FLAG_RWU :*** Receive Wake up flag.
  - ***USART_FLAG_SBK :*** Send Break flag.
  - ***USART_FLAG_CMF :*** Character match flag.
  - ***USART_FLAG_BUSY :*** Busy flag.
  - ***USART_FLAG_ABRF :*** Auto baud rate flag.
  - ***USART_FLAG_ABRE :*** Auto baud rate error flag.
  - ***USART_FLAG_EOBF :*** End of block flag.
  - ***USART_FLAG_RTOF :*** Receive time out flag.
  - ***USART_FLAG_nCTSS :*** Inverted nCTS input bit status.
  - ***USART_FLAG_CTS :*** CTS Change flag.
  - ***USART_FLAG_LBD :*** LIN Break detection flag.
  - ***USART_FLAG_TXE :*** Transmit data register empty flag.
  - ***USART_FLAG_TC :*** Transmission Complete flag.
  - ***USART_FLAG_RXNE :*** Receive data register not empty flag.
  - ***USART_FLAG_IDLE :*** Idle Line detection flag.
  - ***USART_FLAG_ORE :*** OverRun Error flag.
  - ***USART_FLAG_NE :*** Noise Error flag.
  - ***USART_FLAG_FE :*** Framing Error flag.
  - ***USART_FLAG_PE :*** Parity Error flag.

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

Notes
- None.

23.2.17.5 USART_ClearFlag

Function Name	**void USART_ClearFlag (*USART_TypeDef** * USARTx, uint32_t USART_FLAG)**
Function Description	Clears the USARTx's pending flags.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_FLAG :** specifies the flag to clear. This parameter can be any combination of the following values: – ***USART_FLAG_WUF :*** Wake up flag. – ***USART_FLAG_CMF :*** Character match flag. – ***USART_FLAG_EOBF :*** End of block flag. – ***USART_FLAG_RTOF :*** Receive time out flag. – ***USART_FLAG_CTS :*** CTS Change flag. – ***USART_FLAG_LBD :*** LIN Break detection flag. – ***USART_FLAG_TC :*** Transmission Complete flag. – ***USART_FLAG_IDLE :*** IDLE line detected flag. – ***USART_FLAG_ORE :*** OverRun Error flag. – ***USART_FLAG_NE :*** Noise Error flag. – ***USART_FLAG_FE :*** Framing Error flag. – ***USART_FLAG_PE :*** Parity Errorflag.
Return values	• None.
Notes	• RXNE pending bit is cleared by a read to the USART_RDR register (USART_ReceiveData()) or by writing 1 to the RXFRQ in the register USART_RQR (USART_RequestCmd()).TC flag can be also cleared by software sequence: a read operation to USART_SR register (USART_GetFlagStatus()) followed by a write operation to USART_TDR register (USART_SendData()).TXE flag is cleared by a write to the USART_TDR register (USART_SendData()) or by writing 1 to the TXFRQ in the register USART_RQR (USART_RequestCmd()).SBKF flag is cleared by 1 to the SBKRQ in the register USART_RQR (USART_RequestCmd()).

23.2.17.6 USART_GetITStatus

Function Name	**ITStatus USART_GetITStatus (*USART_TypeDef** * USARTx, uint32_t USART_IT)**
Function Description	Checks whether the specified USART interrupt has occurred or not.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_IT :** specifies the USART interrupt source to check. This parameter can be one of the following values: – ***USART_IT_WU :*** Wake up interrupt. – ***USART_IT_CM :*** Character match interrupt. – ***USART_IT_EOB :*** End of block interrupt. – ***USART_IT_RTO :*** Receive time out interrupt. – ***USART_IT_CTS :*** CTS change interrupt. – ***USART_IT_LBD :*** LIN Break detection interrupt. – ***USART_IT_TXE :*** Tansmit Data Register empty interrupt. – ***USART_IT_TC :*** Transmission complete interrupt. – ***USART_IT_RXNE :*** Receive Data register not empty interrupt. – ***USART_IT_IDLE :*** Idle line detection interrupt. – ***USART_IT_ORE :*** OverRun Error interrupt. – ***USART_IT_NE :*** Noise Error interrupt. – ***USART_IT_FE :*** Framing Error interrupt. – ***USART_IT_PE :*** Parity Error interrupt.
Return values	• **The new state of USART_IT (SET or RESET).**
Notes	• None.

23.2.17.7 USART_ClearITPendingBit

Function Name	**void USART_ClearITPendingBit (*USART_TypeDef** * USARTx, uint32_t USART_IT)**
Function Description	Clears the USARTx's interrupt pending bits.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_IT :** specifies the interrupt pending bit to clear. This parameter can be one of the following values:

- ***USART_IT_WU :*** Wake up interrupt.
- ***USART_IT_CM :*** Character match interrupt.
- ***USART_IT_EOB :*** End of block interrupt.
- ***USART_IT_RTO :*** Receive time out interrupt.
- ***USART_IT_CTS :*** CTS change interrupt.
- ***USART_IT_LBD :*** LIN Break detection interrupt.
- ***USART_IT_TC :*** Transmission complete interrupt.
- ***USART_IT_IDLE :*** IDLE line detected interrupt.
- ***USART_IT_ORE :*** OverRun Error interrupt.
- ***USART_IT_NE :*** Noise Error interrupt.
- ***USART_IT_FE :*** Framing Error interrupt.
- ***USART_IT_PE :*** Parity Error interrupt.

Return values
- None.

Notes
- RXNE pending bit is cleared by a read to the USART_RDR register (USART_ReceiveData()) or by writing 1 to the RXFRQ in the register USART_RQR (USART_RequestCmd()).TC pending bit can be also cleared by software sequence: a read operation to USART_SR register (USART_GetITStatus()) followed by a write operation to USART_TDR register (USART_SendData()).TXE pending bit is cleared by a write to the USART_TDR register (USART_SendData()) or by writing 1 to the TXFRQ in the register USART_RQR (USART_RequestCmd()).

23.2.18 STOP mode functions

23.2.18.1 USART_STOPModeCmd

Function Name
void USART_STOPModeCmd (*USART_TypeDef * USARTx, *FunctionalState* NewState)**

Function Description
Enables or disables the specified USART peripheral in STOP Mode.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **NewState :** new state of the USARTx peripheral state in stop mode. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- This function has to be called when USART clock is set to HSI or LSE.

23.2.18.2 USART_StopModeWakeUpSourceConfig

Function Name	**void USART_StopModeWakeUpSourceConfig (*USART_TypeDef** * **USARTx, uint32_t USART_WakeUpSource)**
Function Description	Selects the USART WakeUp method form stop mode.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_WakeUp :** specifies the selected USART wakeup method. This parameter can be one of the following values: – ***USART_WakeUpSource_AddressMatch :*** WUF active on address match. – ***USART_WakeUpSource_StartBit :*** WUF active on Start bit detection. – ***USART_WakeUpSource_RXNE :*** WUF active on RXNE.
Return values	• None.
Notes	• This function has to be called before calling USART_Cmd() function.

23.2.19 Data transfer functions

23.2.19.1 USART_SendData

Function Name	**void USART_SendData (*USART_TypeDef** * **USARTx, uint16_t Data)**
Function Description	Transmits single data through the USARTx peripheral.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **Data :** the data to transmit.
Return values	• None.
Notes	• None.

23.2.19.2 USART_ReceiveData

Function Name	**uint16_t USART_ReceiveData (*USART_TypeDef** * USARTx)**
Function Description	Returns the most recent received data by the USARTx peripheral.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
Return values	• **The received data.**
Notes	• None.

23.2.20 AutoBaudRate functions

23.2.20.1 USART_AutoBaudRateCmd

Function Name	**void USART_AutoBaudRateCmd (*USART_TypeDef** * USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the Auto Baud Rate.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new state of the USARTx auto baud rate. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.2.20.2 USART_AutoBaudRateConfig

Function Name	**void USART_AutoBaudRateConfig (*USART_TypeDef** * USARTx, uint32_t USART_AutoBaudRate)**
Function Description	Selects the USART auto baud rate method.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or

USART3 or UART4 or UART5.

- **USART_AutoBaudRate :** specifies the selected USART auto baud rate method. This parameter can be one of the following values:
  - ***USART_AutoBaudRate_StartBit :*** Start Bit duration measurement.
  - ***USART_AutoBaudRate_FallingEdge :*** Falling edge to falling edge measurement.
  - ***USART_AutoBaudRate_0x7FFrame :*** 0x7F frame.
  - ***USART_AutoBaudRate_0x55Frame :*** 0x55 frame.

Return values
- None.

Notes
- This function has to be called before calling USART_Cmd() function.

23.2.21 MultiProcessor Communication functions

23.2.21.1 USART_SetAddress

Function Name **void USART_SetAddress (*USART_TypeDef** * USARTx, uint8_t USART_Address)**

Function Description Sets the address of the USART node.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **USART_Address :** Indicates the address of the USART node.

Return values
- None.

Notes
- None.

23.2.21.2 USART_MuteModeCmd

Function Name **void USART_MuteModeCmd (*USART_TypeDef** * USARTx, *FunctionalState* NewState)**

Function Description Enables or disables the USART's mute mode.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can

be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **NewState :** new state of the USART mute mode. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

23.2.21.3 USART_MuteModeWakeUpConfig

Function Name **void USART_MuteModeWakeUpConfig (*USART_TypeDef** USARTx, uint32_t USART_WakeUp)**

Function Description Selects the USART WakeUp method from mute mode.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **USART_WakeUp :** specifies the USART wakeup method. This parameter can be one of the following values:
  - ***USART_WakeUp_IdleLine :*** WakeUp by an idle line detection
  - ***USART_WakeUp_AddressMark :*** WakeUp by an address mark

Return values
- None.

Notes
- None.

23.2.21.4 USART_AddressDetectionConfig

Function Name **void USART_AddressDetectionConfig (*USART_TypeDef** USARTx, uint32_t USART_AddressLength)**

Function Description Configure the the USART Address detection length.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **USART_AddressLength :** specifies the USART address length detection. This parameter can be one of the following values:

- *USART_AddressLength_4b :* 4-bit address length detection
- *USART_AddressLength_7b :* 7-bit address length detection

Return values
- None.

Notes
- None.

23.2.22 LIN mode functions

23.2.22.1 USART_LINBreakDetectLengthConfig

Function Name: **void USART_LINBreakDetectLengthConfig (*USART_TypeDef* * USARTx, uint32_t USART_LINBreakDetectLength)**

Function Description: Sets the USART LIN Break detection length.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **USART_LINBreakDetectLength :** specifies the LIN break detection length. This parameter can be one of the following values:
  - *USART_LINBreakDetectLength_10b :* 10-bit break detection
  - *USART_LINBreakDetectLength_11b :* 11-bit break detection

Return values
- None.

Notes
- None.

23.2.22.2 USART_LINCmd

Function Name: **void USART_LINCmd (*USART_TypeDef* * USARTx, *FunctionalState* NewState)**

Function Description: Enables or disables the USART's LIN mode.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.

- **NewState :** new state of the USART LIN mode. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

23.2.23 Halfduplex mode function

23.2.23.1 USART_HalfDuplexCmd

Function Name **void USART_HalfDuplexCmd (*USART_TypeDef** * USARTx, *FunctionalState* NewState)**

Function Description Enables or disables the USART's Half Duplex communication.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5.
- **NewState :** new state of the USART Communication. This parameter can be: ENABLE or DISABLE.

Return values
- None.

Notes
- None.

23.2.24 Smartcard mode functions

23.2.24.1 USART_SetGuardTime

Function Name **void USART_SetGuardTime (*USART_TypeDef** * USARTx, uint8_t USART_GuardTime)**

Function Description Sets the specified USART guard time.

Parameters
- **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3.
- **USART_GuardTime :** specifies the guard time.

Return values
- None.

Notes
- None.

23.2.24.2 USART_SmartCardCmd

Function Name	**void USART_SmartCardCmd (*USART_TypeDef* * USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the USART's Smart Card mode.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3. • **NewState :** new state of the Smart Card mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.2.24.3 USART_SmartCardNACKCmd

Function Name	**void USART_SmartCardNACKCmd (*USART_TypeDef* * USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables NACK transmission.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3. • **NewState :** new state of the NACK transmission. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.2.24.4 USART_SetAutoRetryCount

Function Name	**void USART_SetAutoRetryCount (*USART_TypeDef** * USARTx, uint8_t USART_AutoCount)**
Function Description	Sets the Smart Card number of retries in transmit and receive.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3. • **USART_AutoCount :** specifies the Smart Card auto retry count.
Return values	• None.
Notes	• None.

23.2.24.5 USART_SetBlockLength

Function Name	**void USART_SetBlockLength (*USART_TypeDef** * USARTx, uint8_t USART_BlockLength)**
Function Description	Sets the Smart Card Block length.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3. • **USART_BlockLength :** specifies the Smart Card block length.
Return values	• None.
Notes	• None.

23.2.25 IrDA mode functions

23.2.25.1 USART_IrDAConfig

Function Name	**void USART_IrDAConfig (*USART_TypeDef** * USARTx, uint32_t USART_IrDAMode)**

Function Description	Configures the USART's IrDA interface.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **USART_IrDAMode :** specifies the IrDA mode. This parameter can be one of the following values: – ***USART_IrDAMode_LowPower :*** – ***USART_IrDAMode_Normal :***
Return values	• None.
Notes	• None.

23.2.25.2 USART_IrDACmd

Function Name	**void USART_IrDACmd (*USART_TypeDef** * USARTx, *FunctionalState* NewState)**
Function Description	Enables or disables the USART's IrDA interface.
Parameters	• **USARTx :** Select the USART peripheral. This parameter can be one of the following values: USART1 or USART2 or USART3 or UART4 or UART5. • **NewState :** new state of the IrDA mode. This parameter can be: ENABLE or DISABLE.
Return values	• None.
Notes	• None.

23.3 USART Firmware driver defines

23.3.1 USART

USART

USART_Address_Detection

- #define: ***USART_AddressLength_4b ((uint32_t)0x00000000)***

- #define: ***USART_AddressLength_7b USART_CR2_ADDM7***

USART_AutoBaudRate_Mode

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

- #define: ***USART_AutoBaudRate_FallingEdge USART_CR2_ABRMODE_0***

- #define: ***USART_AutoBaudRate_0x7FFrame USART_CR2_ABRMODE_1***

- #define: ***USART_AutoBaudRate_0x55Frame (USART_CR2_ABRMODE_0 | USART_CR2_ABRMODE_1)***

USART_Clock

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

- #define: ***USART_Clock_Enable USART_CR2_CLKEN***

USART_Clock_Phase

- #define: ***USART_CPHA_1Edge ((uint32_t)0x00000000)***

- #define: ***USART_CPHA_2Edge USART_CR2_CPHA***

USART_Clock_Polarity

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

- #define: ***USART_CPOL_High USART_CR2_CPOL***

USART_DE_Polarity

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

- #define: ***USART_DEPolarity_Low USART_CR3_DEP***

USART_DMA_Recception_Error

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

- #define: ***USART_DMAOnError_Disable USART_CR3_DDRE***

USART_DMA_Requests

- #define: ***USART_DMAReq_Tx USART_CR3_DMAT***

- #define: ***USART_DMAReq_Rx USART_CR3_DMAR***

USART_Flags

- #define: ***USART_FLAG_REACK USART_ISR_REACK***

- #define: ***USART_FLAG_TEACK USART_ISR_TEACK***

- #define: ***USART_FLAG_WU USART_ISR_WUF***

- #define: ***USART_FLAG_RWU USART_ISR_RWU***

- #define: ***USART_FLAG_SBK USART_ISR_SBKF***

- #define: ***USART_FLAG_CM USART_ISR_CMF***
- #define: ***USART_FLAG_BUSY USART_ISR_BUSY***
- #define: ***USART_FLAG_ABRF USART_ISR_ABRF***
- #define: ***USART_FLAG_ABRE USART_ISR_ABRE***
- #define: ***USART_FLAG_EOB USART_ISR_EOBF***
- #define: ***USART_FLAG_RTO USART_ISR_RTOF***
- #define: ***USART_FLAG_nCTSS USART_ISR_CTS***
- #define: ***USART_FLAG_CTS USART_ISR_CTSIF***
- #define: ***USART_FLAG_LBD USART_ISR_LBD***
- #define: ***USART_FLAG_TXE USART_ISR_TXE***
- #define: ***USART_FLAG_TC USART_ISR_TC***
- #define: ***USART_FLAG_RXNE USART_ISR_RXNE***

- #define: ***USART_FLAG_IDLE USART_ISR_IDLE***

- #define: ***USART_FLAG_ORE USART_ISR_ORE***

- #define: ***USART_FLAG_NE USART_ISR_NE***

- #define: ***USART_FLAG_FE USART_ISR_FE***

- #define: ***USART_FLAG_PE USART_ISR_PE***

USART_Hardware_Flow_Control

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

- #define: ***USART_HardwareFlowControl_RTS USART_CR3_RTSE***

- #define: ***USART_HardwareFlowControl_CTS USART_CR3_CTSE***

- #define: ***USART_HardwareFlowControl_RTS_CTS (USART_CR3_RTSE | USART_CR3_CTSE)***

USART_Interrupt_definition

- #define: ***USART_IT_WU ((uint32_t)0x00140316)***

- #define: ***USART_IT_CM ((uint32_t)0x0011010E)***

- #define: ***USART_IT_EOB ((uint32_t)0x000C011B)***
- #define: ***USART_IT_RTO ((uint32_t)0x000B011A)***
- #define: ***USART_IT_PE ((uint32_t)0x00000108)***
- #define: ***USART_IT_TXE ((uint32_t)0x00070107)***
- #define: ***USART_IT_TC ((uint32_t)0x00060106)***
- #define: ***USART_IT_RXNE ((uint32_t)0x00050105)***
- #define: ***USART_IT_IDLE ((uint32_t)0x00040104)***
- #define: ***USART_IT_LBD ((uint32_t)0x00080206)***
- #define: ***USART_IT_CTS ((uint32_t)0x0009030A)***
- #define: ***USART_IT_ERR ((uint32_t)0x00000300)***
- #define: ***USART_IT_ORE ((uint32_t)0x00030300)***
- #define: ***USART_IT_NE ((uint32_t)0x00020300)***

- #define: ***USART_IT_FE ((uint32_t)0x00010300)***

USART_Inversion_Pins

- #define: ***USART_InvPin_Tx USART_CR2_TXINV***

- #define: ***USART_InvPin_Rx USART_CR2_RXINV***

USART_IrDA_Low_Power

- #define: ***USART_IrDAMode_LowPower USART_CR3_IRLP***

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

USART_Last_Bit

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

- #define: ***USART_LastBit_Enable USART_CR2_LBCL***

USART_LIN_Break_Detection_Length

- #define: ***USART_LINBreakDetectLength_10b ((uint32_t)0x00000000)***

- #define: ***USART_LINBreakDetectLength_11b USART_CR2_LBDL***

USART_Mode

- #define: ***USART_Mode_Rx USART_CR1_RE***

- #define: ***USART_Mode_Tx USART_CR1_TE***

USART_MuteMode_WakeUp_methods

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

- #define: ***USART_WakeUp_AddressMark USART_CR1_WAKE***

USART_OVR_DETECTION

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

- #define: ***USART_OVRDetection_Disable USART_CR3_OVRDIS***

USART_Parity

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

- #define: ***USART_Parity_Even USART_CR1_PCE***

- #define: ***USART_Parity_Odd (USART_CR1_PCE | USART_CR1_PS)***

USART_Request

- #define: ***USART_Request_ABRRQ USART_RQR_ABRRQ***

- #define: ***USART_Request_SBKRQ USART_RQR_SBKRQ***

- #define: ***USART_Request_MMRQ USART_RQR_MMRQ***

- #define: ***USART_Request_RXFRQ USART_RQR_RXFRQ***

- #define: ***USART_Request_TXFRQ USART_RQR_TXFRQ***

USART_StopMode_WakeUp_methods

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

- #define: ***USART_WakeUpSource_StartBit USART_CR3_WUS_1***

- #define: ***USART_WakeUpSource_RXNE (uint32_t)(USART_CR3_WUS_0 | USART_CR3_WUS_1)***

USART_Stop_Bits

- #define: ***USART_StopBits_1 ((uint32_t)0x00000000)***

- #define: ***USART_StopBits_2 USART_CR2_STOP_1***

- #define: ***USART_StopBits_1_5 (USART_CR2_STOP_0 | USART_CR2_STOP_1)***

USART_Word_Length

- #define: ***USART_WordLength_8b ((uint32_t)0x00000000)***

- #define: ***USART_WordLength_9b USART_CR1_M***

24 Window watchdog (WWDG)

24.1 WWDG Firmware driver registers structures

24.1.1 WWDG_TypeDef

WWDG_TypeDef is defined in the stm32f30x.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t CFR***
- ***__IO uint32_t SR***

Field Documentation

- ***__IO uint32_t WWDG_TypeDef::CR***
  - WWDG Control register, Address offset: 0x00
- ***__IO uint32_t WWDG_TypeDef::CFR***
  - WWDG Configuration register, Address offset: 0x04
- ***__IO uint32_t WWDG_TypeDef::SR***
  - WWDG Status register, Address offset: 0x08

24.2 WWDG Firmware driver API description

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

24.2.1 WWDG features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (downcounter) before to reach 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.

Once enabled the WWDG cannot be disabled except by a system reset.

WWDGRST flag in RCC_CSR register can be used to inform when a WWDG reset occurs.

The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.

WWDG counter clock = PCLK1 / Prescaler.

WWDG timeout = (WWDG counter clock) * (counter value).

Min-max timeout value @36MHz (PCLK1): ~114us / ~58.3ms.

24.2.2 How to use this driver

1. Enable WWDG clock using RCC_APB1PeriphClockCmd(RCC_APB1Periph_WWDG, ENABLE) function.
2. Configure the WWDG prescaler using WWDG_SetPrescaler() function.
3. Configure the WWDG refresh window using WWDG_SetWindowValue() function.
4. Set the WWDG counter value and start it using WWDG_Enable() function. When the WWDG is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
5. Optionally you can enable the Early wakeup interrupt which is generated when the counter reach 0x40. Once enabled this interrupt cannot be disabled except by a system reset.
6. Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using WWDG_SetCounter() function. This operation must occur only when the counter value is lower than the refresh window value, programmed using WWDG_SetWindowValue().

24.2.3 Prescaler, Refresh window and Counter configuration functions

- ***WWDG_DeInit()***
- ***WWDG_SetPrescaler()***
- ***WWDG_SetWindowValue()***
- ***WWDG_EnableIT()***
- ***WWDG_SetCounter()***

24.2.4 WWDG activation function

- ***WWDG_Enable()***

24.2.5 Interrupts and flags management functions

- ***WWDG_GetFlagStatus()***
- ***WWDG_ClearFlag()***

24.2.6 Prescaler, Refresh window and Counter configuration functions

24.2.6.1 WWDG_DeInit

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

24.2.6.2 WWDG_SetPrescaler

Function Name	**void WWDG_SetPrescaler (uint32_t WWDG_Prescaler)**
Function Description	Sets the WWDG Prescaler.
Parameters	• **WWDG_Prescaler :** specifies the WWDG Prescaler. This parameter can be one of the following values: – ***WWDG_Prescaler_1 :*** WWDG counter clock = (PCLK1/4096)/1 – ***WWDG_Prescaler_2 :*** WWDG counter clock = (PCLK1/4096)/2 – ***WWDG_Prescaler_4 :*** WWDG counter clock = (PCLK1/4096)/4 – ***WWDG_Prescaler_8 :*** WWDG counter clock = (PCLK1/4096)/8
Return values	• None.
Notes	• None.

24.2.6.3 WWDG_SetWindowValue

Function Name	**void WWDG_SetWindowValue (uint8_t WindowValue)**
Function Description	Sets the WWDG window value.
Parameters	• **WindowValue :** specifies the window value to be compared to the downcounter. This parameter value must be lower than 0x80.
Return values	• None.
Notes	• None.

24.2.6.4 WWDG_EnableIT

Function Name	**void WWDG_EnableIT (void)**
Function Description	Enables the WWDG Early Wakeup interrupt(EWI).
Parameters	• None.
Return values	• None.
Notes	• Once enabled this interrupt cannot be disabled except by a system reset.

24.2.6.5 WWDG_SetCounter

Function Name	**void WWDG_SetCounter (uint8_t Counter)**
Function Description	Sets the WWDG counter value.
Parameters	• **Counter :** specifies the watchdog counter value. This parameter must be a number between 0x40 and 0x7F (to prevent generating an immediate reset).
Return values	• None.
Notes	• None.

24.2.7 WWDG activation functions

24.2.7.1 WWDG_Enable

Function Name	**void WWDG_Enable (uint8_t Counter)**
Function Description	Enables WWDG and load the counter value.
Parameters	• **Counter :** specifies the watchdog counter value. This parameter must be a number between 0x40 and 0x7F (to prevent generating an immediate reset).
Return values	• None.
Notes	• None.

24.2.8 Interrupts and flags management functions

24.2.8.1 WWDG_GetFlagStatus

Function Name	**FlagStatus WWDG_GetFlagStatus (void)**
Function Description	Checks whether the Early Wakeup interrupt flag is set or not.
Parameters	• None.
Return values	• **The new state of the Early Wakeup interrupt flag (SET or RESET).**
Notes	• None.

24.2.8.2 WWDG_ClearFlag

Function Name	**void WWDG_ClearFlag (void)**
Function Description	Clears Early Wakeup interrupt flag.
Parameters	• None.
Return values	• None.
Notes	• None.

24.3 WWDG Firmware driver defines

24.3.1 WWDG

WWDG

WWDG_Prescaler

- #define: ***WWDG_Prescaler_1 ((uint32_t)0x00000000)***

- #define: ***WWDG_Prescaler_2 ((uint32_t)0x00000080)***

- #define: ***WWDG_Prescaler_4 ((uint32_t)0x00000100)***

- #define: ***WWDG_Prescaler_8 ((uint32_t)0x00000180)***

25 Revision history

Table 16: Revision history

Date	Revision	Changes
25-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