UM0427
User manual

ARM®-based 32-bit MCU STM32F101xx and STM32F103xx firmware library

Introduction

This document describes the ARM®-based 32-bit MCU STM32F101xx and STM32F103xx firmware library.

This library is a firmware package which contains a collection of routines, data structures and macros covering the features of all peripherals. It includes a description of the device drivers plus a set of examples for each peripheral. The firmware library allows any device to be used in the user application without the need for in-depth study of each peripheral specifications. As a result, using the firmware library saves significant time that would otherwise be spent in coding, while reducing the application development and integration cost.

Each device driver consists of a set of functions covering all peripheral functionalities. The development of each driver is driven by a common API (application programming interface) which standardizes the driver structure, the functions and the names of parameters.

The driver source code is developed in 'Strict ANSI-C' (relaxed ANSI-C for projects and examples files). It is fully documented and is MISRA-C 2004 compliant (the compliancy matrix is available upon request). Writing the whole library in 'Strict ANSI-C' makes it independent from the software toolchain. Only the start-up files depend on the toolchain.

The firmware library implements run-time failure detection by checking the input values for all library functions. This dynamic checking contributes to enhance the robustness of the software. Run-time detection is suitable for user application development and debugging. It adds an overhead and can be removed from the final application code to minimize code size and execution speed. For more details refer to *Section 2.5: Run-time checking on page 48*.

Since the firmware library is generic and covers all peripherals functionalities, the size and/or execution speed of the application code may not be optimized. For many applications, the library may be used as is. However, for applications having tough constraints in terms of code size and/or execution speed, the library drivers should be used as a reference on how to configure the peripheral and tailor them to specific application requirements.

The firmware library user manual is structured as follows:

- Definitions, document conventions and firmware library rules
- Overview of the firmware library (package content, library structure), installation guidelines, and example on how to use the library.
- Detailed description the firmware library: configuration structure and software functions for each peripheral.

STM32F101xx and STM32F103xx will be referred to as STM32F10xxx throughout the document.

Contents

List of tables

List of figures

1 Document and library rules

The user manual and the firmware library use the conventions described in the sections below.

1.1 Acronyms

Table 1 describes the acronyms used in this document.

Table 1. List of abbreviations

Acronym	Peripheral / Unit
ADC	Analog/digital converter
BKP	Backup registers
CAN	Controller area network
CRC	CRC calculation unit
DAC	Digital to analog converter (DAC)
DBGMCU	Debug MCU
DMA	DMA controller
EXTI	External interrupt/event controller
FSMC	Flexible static memory 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
PWR	Power control
RCC	Reset and clock controller
RTC	Real-time clock
SDIO	SDIO interface
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

1.2 Naming conventions

The firmware library uses the following naming conventions:

- **PPP** refers to any peripheral acronym, for example **ADC**. See *Section 1.1: Acronyms* for more information.
- System and source/header file names are preceded by ‘stm32f10x_’, for example *stm32f10x_conf.h.*
- Constants used in one file are defined within this file. A constant used in more than one file is defined in a header file. All constants are written in upper case.
- Registers are considered as constants. Their names are in upper case. In most cases, the same acronyms as in the STM32F10x reference manual document are used.
- Names of peripheral functions 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.
- Functions used to initialize the PPP peripheral according to parameters specified in ***PPP_InitTypeDef*** are named ***PPP_Init***, for example ***TIM_Init***.
- Functions used to reset the PPP peripheral registers to their default values are named ***PPP_DeInit***, for example ***TIM_DeInit***.
- Functions used to fill the ***PPP_InitTypeDef*** structure with the reset values of each member are named ***PPP_StructInit***, for example ***USART_StructInit***.
- Functions used to enable or disable the specified PPP peripheral are named ***PPP_Cmd***, for example ***USART_Cmd***.
- Functions used to enable or disable an interrupt source of the specified PPP peripheral are named ***PPP_ITConfig***, for example ***RCC_ITConfig***.
- Functions used to enable or disable the DMA interface of the specified PPP peripheral are named ***PPP_DMAConfig***, for example ***TIM_DMAConfig***.
- Functions used to configure a peripheral function always end with the string ‘Config’, for example ***GPIO_PinRemapConfig***.
- Functions used to check whether the specified PPP flag is set or reset are named ***PPP_GetFlagStatus***, for example ***I2C_GetFlagStatus***.
- Functions used to clear a PPP flag are named ***PPP_ClearFlag***, for example ***I2C_ClearFlag***.
- Functions used to check whether the specified PPP interrupt has occurred or not are named ***PPP_GetITStatus***, for example ***I2C_GetITStatus***.
- Functions used to clear a PPP interrupt pending bit are named ***PPP_ClearITPendingBit***, for example ***I2C_ClearITPendingBit***.

1.3 Coding rules

This section describes the coding rules used in the firmware Library.

1.3.1 Variables

24 specific variable types are defined. Their type and size are fixed. These types are defined in the file *stm32f10x_type.h*:

```
typedef signed long  s32;
typedef signed short s16;
typedef signed char  s8;

typedef signed long  const sc32;  /* Read Only */
typedef signed short const sc16;  /* Read Only */
typedef signed char  const sc8;   /* Read Only */

typedef volatile signed long  vs32;
typedef volatile signed short vs16;
typedef volatile signed char  vs8;

typedef volatile signed long  const vsc32;  /* Read Only */
typedef volatile signed short const vsc16;  /* Read Only */
typedef volatile signed char  const vsc8;   /* Read Only */

typedef unsigned long  u32;
typedef unsigned short u16;
typedef unsigned char  u8;

typedef unsigned long  const uc32;  /* Read Only */
typedef unsigned short const uc16;  /* Read Only */
typedef unsigned char  const uc8;   /* Read Only */

typedef volatile unsigned long  vu32;
typedef volatile unsigned short vu16;
typedef volatile unsigned char  vu8;

typedef volatile unsigned long  const vuc32;  /* Read Only */
typedef volatile unsigned short const vuc16;  /* Read Only */
typedef volatile unsigned char  const vuc8;   /* Read Only */
```

1.3.2 Boolean type

bool type is defined in the *stm32f10x_type.h* file as:

```
typedef enum
{
  FALSE = 0,
  TRUE  = !FALSE
} bool;
```

1.3.3 FlagStatus type

FlagStatus type is defined in the file *stm32f10x_type.h*. Two values can be assigned to this variable: *SET* or *RESET*.

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

1.3.4 FunctionalState type

FunctionalState type is defined in the *stm32f10x_type.h* file. Two values can be assigned to this variable: *ENABLE* or *DISABLE*.

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

1.3.5 ErrorStatus type

ErrorStatus type is defined in the *stm32f10x_type.h* file. Two values can be assigned to this variable: *SUCCESS* or *ERROR*.

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

1.3.6 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 control register structures

stm32f10x_map.h contains the definition of all peripheral structures. The example below illustrates the *SPI* register structure declaration:

```
/*------------------ Serial Peripheral Interface ----------------*/
typedef struct
{
  vu16 CR1;
  u16  RESERVED0;
  vu16 CR2;
  u16  RESERVED1;
  vu16 SR;
  u16  RESERVED2;
  vu16 DR;
  u16  RESERVED3;
  vu16 CRCPR;
```

```
  u16  RESERVED4;
  vu16 RXCRCR;
  u16  RESERVED5;
  vu16 TXCRCR;
  u16  RESERVED6;
  vu16 I2SCFGR;
  u16  RESERVED7;
  vu16 I2SPR;
  u16  RESERVED8;
} SPI_TypeDef;
```

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.

Peripheral declaration

All peripherals are declared in *stm32f10x_map.h*. The following example shows the declaration of the *SPI* peripheral:

```
#ifndef EXT
#Define EXT extern
#endif
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
...
/* SPI2 Base Address definition*/
#define SPI2_BASE             (APB1PERIPH_BASE + 0x3800)
...
/* SPI2 peripheral declaration*/
#ifndef DEBUG
...
#ifdef _SPI2
  #define SPI2                ((SPI_TypeDef *) SPI2_BASE)
#endif /*_SPI2 */
...
#else   /* DEBUG */
...
#ifdef _SPI2
  EXT SPI_TypeDef             *SPI2;
#endif /*_SPI2 */
...
#endif  /* DEBUG */
```

Define the label *_SPI, t*o include the *SPI* peripheral library in your application.

Define the label *_SPIn*, to access the *SPIn* peripheral registers. For example, the *_SPI2* label must be defined in *stm32f10x_conf.h* to be able to access the registers of *SPI2* peripheral. *_SPI* and *_SPIn* labels are defined in the *stm32f10x_conf.h* file as follows:

```
#define _SPI
#define _SPI1
#define _SPI2
#define _SPI3
```

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***_'.Flag definition is adapted to each peripheral case and defined in *stm32f10x_ppp.h*.

To enter DEBUG mode you have to define the label *DEBUG* in the file *stm32f10x_conf.h*. This creates a pointer to the peripheral structure in SRAM. Debugging consequently becomes easier and all register settings can be obtain by dumping a peripheral variable. In both cases *SPI2* is a pointer to the first address of the *SPI2* peripheral.

The *DEBUG* variable is defined in the *stm32f10x_conf.h* file as follows:

```
#define DEBUG    1
```

The DEBUG mode is initialized as follows in the *stm32f10x_lib.c* file:

```
#ifdef DEBUG
void debug(void)
{
  ...
#ifdef _SPI2
  SPI2 = (SPI_TypeDef *)  SPI2_BASE;
#endif /*_SPI2 */
  ...
}
#endif /* DEBUG*/
```

Note: 1 When the DEBUG *mode is selected, the* ***assert_param*** *macro is expanded and run time checking is enabled in the firmware library code.*

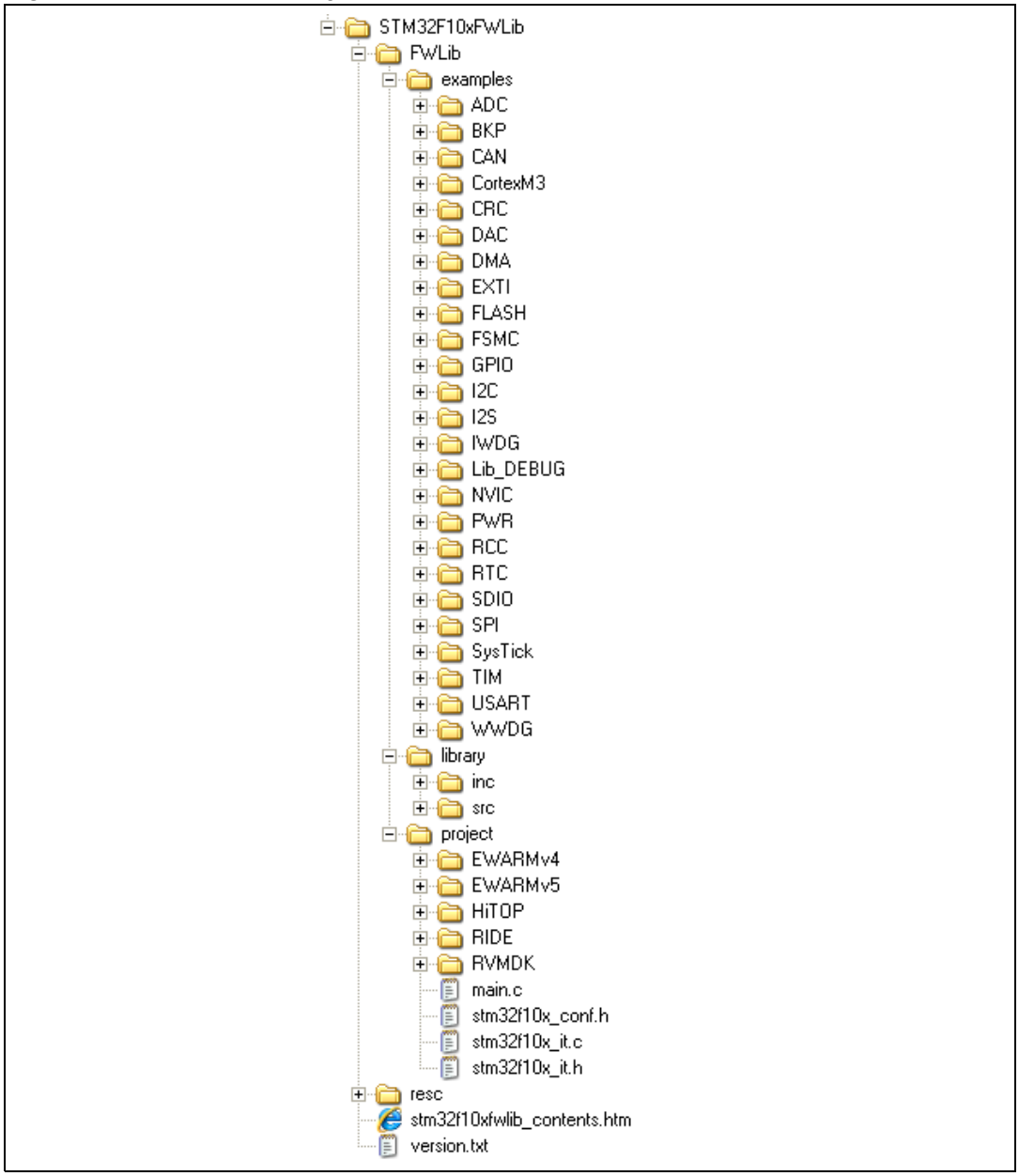
2 The DEBUG *mode increases the code size and reduces the code performance. For this reason, it is recommended to used it only when debugging the application and to remove it from the final application code.*

2 Firmware library

2.1 Package description

The firmware library is supplied in one single zip file. The extraction of the zip file generates one folder, ***STM32F10xFWLib,*** which contains the following subfolders:

Figure 1. Firmware library folder structure



2.1.1 Examples folder

This **Examples** folder contains, for each peripheral sub-folder, the minimum set of files needed to run a typical example on how to use a peripheral:

- ***readme.txt***: **-** brief text file describing the example and how to make it work,
- ***stm32f10x_conf.h -*** header file allowing to configure the peripherals that are used, and containing miscellaneous DEFINE statements,
- ***stm32f10x_it.c*** - source file containing the interrupt handlers (the function bodies may be emptied if not used),
- ***stm32f10x_it.h -*** header file including all interrupt handler prototypes,
- ***main.c*** - example of code

Note: All the examples are independent from the software toolchain.

2.1.2 Library folder

The **Library** folder contains all the subdirectories and files that make up the core of the library:

- ***inc*** sub-folder contains the firmware library header files. They do not need to be modified by the user:
  - ***stm32f10x_type.h***: common data types and enumeration used in all other files,
  - ***stm32f10x_map.h***: peripherals memory mappings and registers data structures,
  - ***stm32f10x_lib.h***: main header file including all other headers,
  - ***stm32f10x_ppp.h*** (one header file per peripheral): Function prototypes, data structures and enumeration.
  - ***cortexm3_macro.h***: header file for cortexm3_macro.s.
- ***src*** sub-folder contains the firmware library source files. They do not need to be modified by the user:
  - ***stm32f10x_ppp.c*** (one source file per peripheral): function bodies of each peripheral.
  - ***stm32f10x_lib.c***: all peripherals pointers initialization.

Note: All library files are coded in Strict ANSI-C and are independent from the software toolchain.

2.1.3 Project folder

The **Project** folder contains a standard template project program that compiles all library files plus all the user-modifiable files that are necessary to create a new project:

- ***stm32f10x_conf.h***: configuration header file with all peripherals defined by default.
- ***stm32f10x_it.c***: source file containing the interrupt handlers (the function bodies are empty in this template).
- ***stm32f10x_it.h***: header file including all interrupt handlers prototypes.
- ***main.c***: main program body.

- ***EWARM***, ***RVMDK, RIDE, HiTOP:*** it is used by the toolchain, and refers to the **readme.txt** file located in the same folder.

2.2 Description of firmware library files

Table 2 lists and describes the different files used by the firmware library.

The firmware library architecture and file inclusion relationship are shown in *Figure 2*. Each peripheral has a source code file, *stm32f10x_ppp.c*, and a header file, *stm32f10x_ppp.h*. The *stm32f10x_ppp.c* file contains all the firmware functions required to use the PPP peripheral. A single memory mapping file, *stm32f10x_map.h,* is supplied for all peripherals. It contains all the register declarations used both in Debug and release modes.

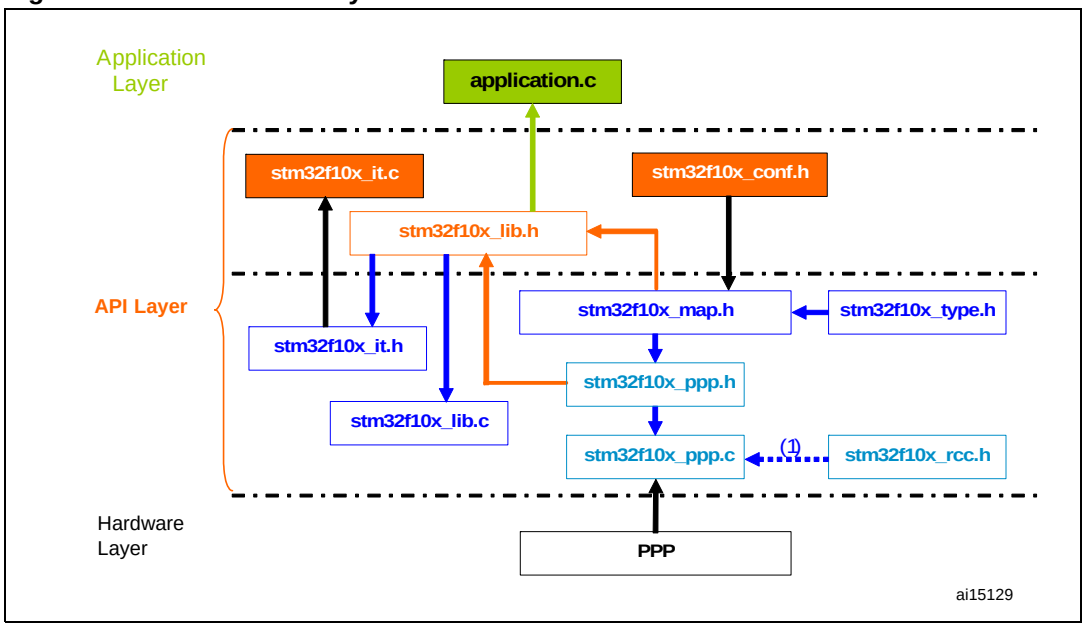
The header file *stm32f10x_lib.h* includes all the peripheral header files. This is the only file that needs to be included in the user application to interface with the library.

stm32f10x_conf.h is the only file which must be modified by the user. It is used to specify the set of parameters to interface with the library before running any application.

Table 2. Firmware library files

File name	Description
stm32f10x_conf.h	Parameter configuration file. It must be modified by the user to define the parameters used to interface with the library before running any application. The user can enable or disable peripherals by using the template, and change the external Quartz oscillator value and startup timeout value. This file can also be used to enable the Debug or Release mode before compiling the firmware library.
main.c	Main example program body.
stm32f10x_it.h	Header file including all interrupt handlers prototypes.
stm32f10x_it.c	Peripheral interrupt functions file. The user can modify it by including the code of interrupt functions used in his application. In case of multiple interrupt requests mapped to the same interrupt vector, the function polls the interrupt flags of the peripheral to identify the exact source of the interrupt. The names of these functions are already provided in the firmware library.
stm32f10x_lib.h	Header file including all peripheral header files. It is the only file that has to be included in the user application to interface with the firmware library.
stm32f10x_lib.c	Debug mode initialization file. It includes the definition of variable pointers. Each one points to the first address of a specific peripheral and to the definition of the function which is called when the Debug mode is enabled. This function initializes the defined pointers.
stm32f10x_map.h	This file implements memory mapping and physical registers address definition for both debug and release modes. It is supplied with all peripherals.
stm32f10x_type.h	Common declarations file. It includes common types and constants used by all peripheral drivers.
stm32f10x_ppp.c	Driver source code file of PPP peripheral written in C language.
stm32f10x_ppp.h	Header file of PPP peripheral. It includes the definition of PPP peripheral functions and variables used within these functions.
cortexm3_macro.h	Header file for cortexm3_macro.s.
cortexm3_macro.s	Instruction wrappers for special Cortex-M3 instructions.

Figure 2. Firmware library file architecture



1. Some RCC routines are used:
   - In PPP_DeInit function to reset the peripheral
   - To get the clock frequency of the bus to which the communication peripherals are connected.

2.3 Peripheral initialization and configuration

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

1. In the main application file, declare a ***PPP_InitTypeDef*** structure, for example:

   ```
   PPP_InitTypeDef  PPP_InitStructure;
   ```

   The `PPP_InitStructure` is a working variable located in data memory area. It allows to initialize one or more PPP instances.

2. Fill the `PPP_InitStructure` variable with the allowed values of the structure member. There are two ways of doing this:

   a) Configuring 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) Configuring only a few members of the structure: in this case the user should 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);
   PP_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 *Section 3: Peripheral firmware overview*.

Note: 1 Before configuring a peripheral, its clock must be enabled by calling one of the following functions:

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

2 ***PPP_DeInit(..)*** *function can be used to set all PPP peripheral registers to their default values:*

```
PPP_DeInit(PPP)
```

3 *To modify the peripheral settings after configuring the peripheral, the user can proceed as follows:*

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

2.4 Bit-Banding

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

All the STM32F10x peripheral registers are mapped in a bit-band region. This feature is consequently intensively used in functions which perform single bit set/reset in order to reduce and optimize code size.

Section 2.4.1 and *Section 2.4.2* give a description of how the bit-band access is used in the peripheral firmware library.

2.4.1 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.

2.4.2 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 bit-band region */
#define PERIPH_BASE       ((u32)0x40000000)

/* Peripheral address in the alias region */
#define PERIPH_BB_BASE    ((u32)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:

```
...
#define CR_PLLON_Set       ((u32)0x01000000)
#define CR_PLLON_Reset     ((u32)0xFEFFFFFF)
...
void RCC_PLLCmd(FunctionalState NewState)
{
  if (NewState != DISABLE)
  { /* Enable PLL */
    RCC->CR |= CR_PLLON_Set;
  }
  else
  { /* Disable PLL */
    RCC->CR &= CR_PLLON_Reset;
  }
}
```

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

```
void RCC_PLLCmd(FunctionalState NewState)
{
  *(vu32 *) CR_PLLON_BB = (u32)NewState;
}
```

2.5 Run-time checking

The firmware 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 to check that the input value lies within the parameter allowed values.

Example: *PWR_ClearFlag* function

stm32f10x_pwr.c:

```
void PWR_ClearFlag(u32 PWR_FLAG)
{
  /* Check the parameters */
  assert_param(IS_PWR_CLEAR_FLAG(PWR_FLAG));
   PWR->CR |=  PWR_FLAG << 2;
}
```

stm32f10x_pwr.h:

```
/* PWR Flag */
#define PWR_FLAG_WU                 ((u32)0x00000001)
#define PWR_FLAG_SB                 ((u32)0x00000002)
#define PWR_FLAG_PVDO               ((u32)0x00000004)
```

```
#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 ***stm32f10x_conf.h***:

```
/* Exported macro ------------------------------------------------------------*/
#ifdef  DEBUG
/*******************************************************************************
* Macro Name     : assert_param
* Description    : The assert_param macro is used for function's parameters check.
*                  It is used only if the library is compiled in DEBUG mode.
* Input          : - 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.
* Return         : None
*******************************************************************************/
  #define assert_param(expr) ((expr) ? (void)0 : assert_failed((u8 *)__FILE__,
__LINE__))
/* Exported functions ------------------------------------------------------- */
  void assert_failed(u8* file, u32 line);
#else
  #define assert_param(expr) ((void)0)
#endif /* DEBUG */
```

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

```
#ifdef  DEBUG
/*******************************************************************
* Function name  : assert_failed
* Description    : Reports the name of the source file and the source line number
*                  where the assert_param error has occurred.
* Input          : - file: pointer to the source file name
*                  - line: assert_param error line source number
* Output         : None
* Return         : None
*******************************************************************/
void assert_failed(u8* file, u32 line)
{
  /* User can add his own implementation to report the file name and line number,
```

```
      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */


  /* Infinite loop */
  while (1)
  {
  }
}
#endif
```

Note: ***The run-time checking, that is the assert_param macro, should only be used when the library is compiled in DEBUG mode.***

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 (because of the overhead it introduces).

However if the user wants to keep this functionality in his final application, he can re-use the **assert_param** macro defined within the library to test the parameter values before calling the library functions.

3 Peripheral firmware overview

This section describes in detail each peripheral firmware library. The related functions are fully described, an example of how to use them is provided.

The functions are described in the following format:

Table 3. Function description format

Function name	The name of the peripheral function
Function prototype	Prototype declaration
Behavior description	Brief explanation of how the function is executed
Input parameter {x}	Description of the input parameters
Output parameter {x}	Description of the output parameters
Return Value	Value returned by the function
Required preconditions	Requirements before calling the function
Called functions	Other library functions called

4 Analog/digital converter (ADC)

The analog/digital converter (ADC) consists of an input multiplexing channel selector feeding an approximation converter. The conversion resolution is of 12 bits.

The data structures used in the ADC firmware library are described in *Section 4.1*, while *Section 4.2* presents the firmware library functions.

4.1 ADC register structure

The ADC register structure, *ADC_TypeDef,* is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
vu32 SR;
vu32 CR1;
vu32 CR2;
vu32 SMPR1;
vu32 SMPR2;
vu32 JOFR1;
vu32 JOFR2;
vu32 JOFR3;
vu32 JOFR4;
vu32 HTR;
vu32 LTR;
vu32 SQR1;
vu32 SQR2;
vu32 SQR3;
vu32 JSQR;
vu32 JDR1;
vu32 JDR2;
vu32 JDR3;
vu32 JDR4;
vu32 DR;
} ADC_TypeDef;
```

Table 4 gives the lists of ADC registers:

Table 4. ADC registers

Register	Description
SR	ADC Status Register
CR1	ADC Configuration Register1
CR2	ADC Configuration Register2
SMPR1	ADC Sample Time Register1
SMPR2	ADC Sample Time Register2
JOFR1	ADC Offset Register1
JOFR2	ADC Offset Register2

Table 4. ADC registers (continued)

Register	Description
JOFR3	ADC Offset Register3
JOFR4	ADC Offset Register4
HTR	ADC High Voltage Threshold Register
LTR	ADC Low Voltage Threshold Register
SQR1	ADC Sequence Selector for Regular group Register1
SQR2	ADC Sequence Selector for Regular group Register2
SQR3	ADC Sequence Selector for Regular group Register3
JSQR	ADC Sequence Selector for Injected group Register
JDR1	ADC Data converted Injected group Register1
JDR2	ADC Data converted Injected group Register2
JDR3	ADC Data converted Injected group Register3
JDR4	ADC Data converted Injected group Register4
DR	ADC Regular group data Register

The two ADC peripherals are declared in *stm32f10x_map*:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
....
#define ADC1_BASE             (APB2PERIPH_BASE + 0x2400)
#define ADC2_BASE             (APB2PERIPH_BASE + 0x2800)
#define ADC3_BASE             (APB2PERIPH_BASE + 0x3C00)
....
#ifndef DEBUG
...
#ifdef _ADC1
  #define ADC1                ((ADC_TypeDef *) ADC1_BASE)
#endif /*_ADC1 */

#ifdef _ADC2
  #define ADC2                ((ADC_TypeDef *) ADC2_BASE)
#endif /*_ADC2 */

#ifdef _ADC3
  #define ADC3              ((ADC_TypeDef *) ADC3_BASE)
#endif /*_ADC3 */
...
#else   /* DEBUG */
...
#ifdef _ADC1
  EXT ADC_TypeDef            *ADC1;
#endif /*_ADC1 */
```

```
#ifdef _ADC2
  EXT ADC_TypeDef              *ADC2;
#endif /*_ADC2 */

#ifdef _ADC3
  EXT ADC_TypeDef              *ADC3;
#endif /*_ADC3 */
...
#endif
```

When using the Debug mode, _ADC1, _ADC2 and _ADC3 pointers are initialized in *stm32f10x_lib.c* file:

```
...
#ifdef _ADC1
  ADC1 = (ADC_TypeDef *)  ADC1_BASE;
#endif /*_ADC1 */

#ifdef _ADC2
  ADC2 = (ADC_TypeDef *)  ADC2_BASE;
#endif /*_ADC2 */

#ifdef _ADC3
  ADC3 = (ADC_TypeDef *)  ADC3_BASE;
#endif /*_ADC3 */
...
```

To access the ADC registers, _ADC, _ADC1, _ADC2 and _ADC3 must be defined in *stm32f10x_conf.h*, as follows:

```
...
#define _ADC
#define _ADC1
#define _ADC2
#define _ADC3
...
```

4.2 ADC library functions

Table 5 lists the ADC firmware library functions.

Table 5. ADC firmware library functions

Function name	Description
ADC_DeInit	Resets the ADCx peripheral registers to their default reset values.
ADC_Init	Initializes the ADCx peripheral according to the parameters specified in the ADC_InitStruct.
ADC_StructInit	Fills each ADC_InitStruct member with its default value.
ADC_Cmd	Enables or disables the specified ADC peripheral.
ADC_DMACmd	Enables or disables the specified ADC DMA request
ADC_ITConfig	Enables or disables the specified ADC interrupts.
ADC_ResetCalibration	Resets the selected ADC calibration registers
ADC_GetResetCalibrationStatus	Gets the selected ADC reset calibration registers status.
ADC_StartCalibration	Starts the selected ADC calibration process.
ADC_GetCalibrationStatus	Gets the selected ADC calibration status.
ADC_SoftwareStartConvCmd	Enables or disables the selected ADC software start conversion.
ADC_GetSoftwareStartConvStatus	Gets the selected ADC Software start conversion Status.
ADC_DiscModeChannelCountConfig	Configures the discontinuous mode for the selected ADC regular group channel.
ADC_DiscModeCmd	Enables or disables the discontinuous mode on regular group channel for the specified ADC.
ADC_RegularChannelConfig	Configures for the selected ADC regular channel the corresponding rank in the sequencer and the sample time.
ADC_ExternalTrigConvCmd	Enables or disables the ADCx conversion through external trigger
ADC_GetConversionValue	Returns the last ADCx conversion result data for regular channel
ADC_GetDualModeConversionValue	Returns the last ADCs conversion result data in dual mode
ADC_AutoInjectedConvCmd	Enables or disables the selected ADC automatic injected group conversion after regular one
ADC_InjectedDiscModeCmd	Enables or disables the discontinuous mode for injected group channel for the specified ADC
ADC_ExternalTrigInjectedConvConfig	Configures the ADCx external trigger for injected channels conversion

Table 5. ADC firmware library functions (continued)

Function name	Description
ADC_ExternalTrigInjectedConvCmd	Enables or disables the ADCx injected channels conversion through external trigger
ADC_SoftwareStartInjectedConvCmd	Enables or disables the selected ADC start of the injected channels conversion
ADC_GetSoftwareStartInjectedConvStatus	Gets the selected ADC Software start injected conversion Status.
ADC_InjectedChannelConfig	Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.
ADC_InjectedSequencerLengthConfig	Configures the sequencer length for injected channels
ADC_SetInjectedOffset	Sets the injected channels conversion value offset
ADC_GetInjectedConversionValue	Returns the ADC conversion result data for the selected injected channel
ADC_AnalogWatchdogCmd	Enables or disables the analog watchdog on single/all regular or injected channels
ADC_AnalogWatchdogThresholdsConfig	Configures the high and low thresholds of the analog watchdog
ADC_AnalogWatchdogSingleChannelConfig	Configures the analog watchdog guarded single channel
ADC_TempSensorVrefintCmd	Enables or disables the temperature sensor and Vrefint channel.
ADC_GetFlagStatus	Checks whether the specified ADC flag is set or not.
ADC_ClearFlag	Clears the ADCx pending flags.
ADC_GetITStatus	Checks whether the specified ADC interrupt has occurred or not.
ADC_ClearITPendingBit	Clears the ADCx interrupt pending bits.

4.2.1 ADC_DeInit function

Table 6 describes the ADC_DeInit function.

Table 6. ADC_DeInit function

Function name	ADC_DeInit
Function prototype	void ADC_DeInit(ADC_TypeDef* ADCx)
Behavior description	Resets the ADCx peripheral registers to their default reset values.
Input parameter	ADCx: where x can be either 1 or 2 to select ADC peripheral ADC1 or ADC2.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB2PeriphClockCmd().

Example:

```
/* Resets ADC2 */
ADC_DeInit(ADC2);
```

4.2.2 ADC_Init function

Table 7 describes the ADC_Init function.

Table 7. ADC_Init function

Function name	ADC_Init
Function prototype	void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)
Behavior description	Initializes the ADCx peripheral according to the parameters specified in the ADC_InitStruct.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_InitStruct: pointer to an ADC_InitTypeDef structure that contains the configuration information for the specified ADC peripheral. Refer to the *Section 4.2.3: ADC_StructInit function* for a full description of the ADC_InitStruct values.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ADC_InitTypeDef structure

The **ADC_InitTypeDef** structure is defined in the *stm32f10x_adc.h* file:

```
typedef struct
{
  u32 ADC_Mode;
  FunctionalState ADC_ScanConvMode;
  FunctionalState ADC_ContinuousConvMode;
  u32 ADC_ExternalTrigConv;
  u32 ADC_DataAlign;
  u8 ADC_NbrOfChannel;
} ADC_InitTypeDef
```

ADC_Mode

ADC_Mode configures the ADC to operate in independent or dual mode. See *Table 8* for the values taken by this member.

Table 8. ADC_Mode definition

ADC_Mode	Description
ADC_Mode_Independent	ADC1 and ADC2 operate in independent mode
ADC_Mode_RegInjecSimult	ADC1 and ADC2 operate in simultaneous sample/hold 1 and 2 mode
ADC_Mode_RegSimult_AlterTrig	ADC1 and ADC2 operate in simultaneous sample/hold 2 and Alternate trigger mode
ADC_Mode_InjecSimult_FastInterl	ADC1 and ADC2 operate in simultaneous sample/hold 1 and Interleaved 1 mode
ADC_Mode_InjecSimult_SlowInterl	ADC1 and ADC2 operate in simultaneous sample/hold 1 and Interleaved 2 mode
ADC_Mode_InjecSimult	ADC1 and ADC2 operate in simultaneous sample/hold 1 mode
ADC_Mode_RegSimult	ADC1 and ADC2 operate in simultaneous sample/hold 2 mode
ADC_Mode_FastInterl	ADC1 and ADC2 operate in interleaved 1 mode
ADC_Mode_SlowInterl	ADC1 and ADC2 operate in interleaved 2 mode
ADC_Mode_AlterTrig	ADC1 and ADC2 operate in alternate trigger mode

ADC_ScanConvMode

ADC_ScanConvMode specifies whether the conversion is performed in Scan (multi-channels) or Single (one channel) mode. This member can be set to ENABLE or DISABLE.

ADC_ContinuousConvMode

ADC_ContinuousConvMode specifies whether the conversion is performed in Continuous or Single mode. This member can be set to ENABLE or DISABLE.

ADC_ExternalTrigConv

ADC_ExternalTrigConv defines the external trigger used to start the analog to digital conversion of regular channels. The values taken by this member are given in *Table 9*.

Table 9. ADC_ExternalTrigConv definition

ADC_ExternalTrigConv	Description
ADC_ExternalTrigConv_T1_CC3	Timer1 Capture Compare3 selected as external trigger conversion (ADC1, ADC2 and ADC3)
ADC_ExternalTrigConv_None	Conversion started by software and not by external trigger (ADC1, ADC2 and ADC3)
ADC_ExternalTrigConv_T1_CC1	Timer1 Capture Compare1 selected as external trigger conversion (ADC1 and ADC2)
ADC_ExternalTrigConv_T1_CC2	Timer1 Capture Compare2 selected as external trigger conversion (ADC1 and ADC2)
ADC_ExternalTrigConv_T2_CC2	Timer2 Capture Compare2 selected as external trigger conversion (ADC1 and ADC2)

Table 9. ADC_ExternalTrigConv definition (continued)

ADC_ExternalTrigConv	Description
ADC_ExternalTrigConv_T3_TRGO	Timer3 TRGO selected as external trigger conversion (ADC1 and ADC2)
ADC_ExternalTrigConv_T4_CC4	Timer4 Capture Compare4 selected as external trigger conversion (ADC1 and ADC2)
ADC_ExternalTrigConv_Ext_IT11_TIM8_TRGO	External interrupt 11 event/Timer8 TRGO selected as external trigger conversion (ADC1 and ADC2)
ADC_ExternalTrigConv_T3_CC1	Timer3 Capture Compare1 selected as external trigger conversion (ADC3 only)
ADC_ExternalTrigConv_T2_CC3	Timer2 Capture Compare3 selected as external trigger conversion (ADC3 only)
ADC_ExternalTrigConv_T8_CC1	Timer8 Capture Compare1 selected as external trigger conversion (ADC3 only)
ADC_ExternalTrigConv_T8_TRGO	Timer8 TRGO selected as external trigger conversion (ADC3 only)
ADC_ExternalTrigConv_T5_CC1	Timer5 Capture Compare1 selected as external trigger conversion (ADC3 only)
ADC_ExternalTrigConv_T5_CC3	Timer5 Capture Compare3 selected as external trigger conversion (ADC3 only)

Note: The TIM8_TRGO event exists only in High-density devices. The EXTI line11 or TIM8_TRGO external trigger event for regular channels is selected through AFIO configuration bits ADC1_ETRGREG_REMAP and ADC2_ETRGREG_REMAP for ADC1 and ADC2, respectively.

ADC_DataAlign

ADC_DataAlign specifies whether the ADC data alignment is left or right. The values taken by this member are given in *Table 10*.

Table 10. ADC_DataAlign definition

ADC_DataAlign	Description
ADC_DataAlign_Right	ADC data right aligned
ADC_DataAlign_Left	ADC data left aligned

ADC_NbrOfChannel

ADC_NbreOfChannel specifies the number of ADC channels that will be converted using the sequencer for regular channel group. This number must range from 1 to 16.

Example:

```
/* Initialize the ADC1 according to the ADC_InitStructure members */
ADC_InitTypeDef  ADC_InitStructure;
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_T1_CC3;
```

```
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 16;
ADC_Init(ADC1, &ADC_InitStructure);
```

Note: To correctly configure the ADC channels conversion, the user must call the ADC_ChannelConfig() function after ADC_Init() to configure the sequencer rank and sample time for each used channel.

4.2.3 ADC_StructInit function

Table 11 describes the ADC_StructInit function.

Table 11. ADC_StructInit function

Function name	ADC_StructInit
Function prototype	void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)
Behavior description	Fills each ADC_InitStruct member with its default value.
Input parameter	ADC_InitStruct: pointer to the ADC_InitTypeDef structure to initialize.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The ADC_InitStruct members have the following default values:

Table 12. ADC_IniyStruct default values

Member	Default value
ADC_Mode	ADC_Mode_Independent
ADC_ScanConvMode	DISABLE
ADC_ContinuousConvMode	DISABLE
ADC_ExternalTrigConv	ADC_ExternalTrigConv_T1_CC1
ADC_DataAlign	ADC_DataAlign_Right
ADC_NbrOfChannel	1

Example:

```
/* Initialize a ADC_InitTypeDef structure. */
ADC_InitTypeDef  ADC_InitStructure;
ADC_StructInit(&ADC_InitStructure);
```

4.2.4 ADC_Cmd function

Table 13 describes the ADC_Cmd function.

Table 13. ADC_Cmd function

Function name	Description
Function name	ADC_Cmd
Function prototype	void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the specified ADC peripheral.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the ADCx peripheral. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
```

Note: The ADC_Cmd function must be called after all other ADC configuration functions.

4.2.5 ADC_DMACmd function

Table 14 describes the ADC_DMACmd function.

Table 14. ADC_DMACmd function

Function name	ADC_DMACmd
Function prototype	ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the specified ADC DMA request.
Input parameter1	ADC*x*: where x can be 1 or 3 to select ADC1 or ADC3 peripheral.
Input parameter2	NewState: new state of the ADC DMA transfer. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable ADC1 DMA transfer */
ADC_DMACmd(ADC1, ENABLE);
```

4.2.6 ADC_ITConfig function

Table 15 describes the ADC_ITConfig function.

Table 15. ADC_ITConfig function

Function name	ADC_ITConfig
Function prototype	void ADC_ITConfig(ADC_TypeDef* ADCx, u16 ADC_IT, FunctionalState NewState)
Behavior description	Enables or disables the specified ADC interrupts.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_IT: specifies the ADC interrupt sources to be enabled or disabled. Refer to *ADC_IT* for details on the allowed values for this parameter.
Input parameter3	NewState: new state of the specified ADC interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ADC_IT

ADC_IT is used to enable or disable ADC interrupts. One or a combination of the following values can be used:

Table 16. ADC_IT definition

ADC_IT	Description
ADC_IT_EOC	EOC interrupt mask
ADC_IT_AWD	AWDOG interrupt mask
ADC_IT_JEOC	JEOC interrupt mask

Example:

```
/* Enable ADC2 EOC and AWDOG interrupts */
ADC_ITConfig(ADC2, ADC_IT_EOC | ADC_IT_AWD, ENABLE);
```

4.2.7 ADC_ResetCalibration function

Table 17 describes the ADC_ResetCalibration function.

Table 17. ADC_ResetCalibration function

Function name	ADC_ResetCalibration
Function prototype	void ADC_ResetCalibration(ADC_TypeDef* ADCx)
Behavior description	Resets the selected ADC calibration registers.
Input parameter	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Reset the ADC1 Calibration registers */
ADC_ResetCalibration(ADC1);
```

4.2.8 ADC_GetResetCalibrationStatus function

Table 18 describes the ADC_GetResetCalibration function.

Table 18. ADC_GetResetCalibration function

Function name	ADC_GetResetCalibrationStatus
Function prototype	FlagStatus ADC_GetResetCalibrationStatus(ADC_TypeDef* ADCx)
Behavior description	Gets the selected ADC reset calibration registers status.
Input parameter	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Output parameter	None
Return parameter	The new state of ADC Reset Calibration registers (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the ADC2 reset calibration registers status */
FlagStatus Status;
Status = ADC_GetResetCalibrationStatus(ADC2);
```

4.2.9 ADC_StartCalibration function

Table 19 describes the ADC_StartCalibration function.

Table 19. ADC_StartCalibration function

Function name	ADC_StartCalibration
Function prototype	void ADC_StartCalibration(ADC_TypeDef* ADCx)
Behavior description	Starts the selected ADC calibration process.
Input parameter	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Start the ADC2 Calibration */
ADC_StartCalibration(ADC2);
```

4.2.10 ADC_GetCalibrationStatus function

Table 20 describes the ADC_GetCalibrationStatus function.

Table 20. ADC_GetCalibrationStatus function

Function name	ADC_GetCalibrationStatus
Function prototype	FlagStatus ADC_GetCalibrationStatus(ADC_TypeDef* ADCx)
Behavior description	Gets the selected ADC calibration status.
Input parameter	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Output parameter	None
Return parameter	The new state of ADC Calibration (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the ADC2 calibration status */
FlagStatus Status;
Status = ADC_GetCalibrationStatus(ADC2);
```

4.2.11 ADC_SoftwareStartConvCmd function

Table 21 describes the ADC_SoftwareStartConvCmd function.

Table 21. ADC_SoftwareStartConvCmd function

Function name	ADC_SoftwareStartConvCmd
Function prototype	void ADC_SoftwareStartConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the selected ADC software start conversion.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the selected ADC software start conversion. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Start by software the ADC1 Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

4.2.12 ADC_GetSoftwareStartConvStatus function

Table 22 describes the ADC_GetSoftwareStartConvStatus function.

Table 22. ADC_GetSoftwareStartConvStatus function

Function name	ADC_GetSoftwareStartConvStatus
Function prototype	FlagStatus ADC_GetSoftwareStartConvStatus(ADC_TypeDef* ADCx)
Behavior description	Gets the selected ADC Software start conversion Status.
Input parameter	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Output parameter	None
Return parameter	The new state of ADC software start conversion (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the ADC1 conversion start bit */
FlagStatus Status;
Status = ADC_GetSoftwareStartConvStatus(ADC1);
```

4.2.13 ADC_DiscModeChannelCountConfig function

Table 23 describes the ADC_DiscModeChannelCountConfig function.

Table 23. ADC_DiscModeChannelCountConfig function

Function name	ADC_DiscModeChannelCountConfig
Function prototype	void ADC_DiscModeChannelCountConfig(ADC_TypeDef* ADCx, u8 Number)
Behavior description	Configures the discontinuous mode for the selected ADC regular group channel.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	Number: the discontinuous mode regular channel count value. This number ranges from 1 to 8.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the discontinuous mode channel count to 2 for ADC1 */
ADC_DiscModeChannelCountConfig(ADC1, 2);
```

4.2.14 ADC_DiscModeCmd function

Table 24 describes the ADC_DiscModeCmd function.

Table 24. ADC_DiscModeCmd function

Function name	ADC_DiscModeCmd
Function prototype	void ADC_DiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the discontinuous mode on regular group channel for the specified ADC
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the ADC discontinuous mode on regular group channel. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the discontinuous mode for ADC1 regular group channel */
ADC_DiscModeCmd(ADC1, ENABLE);
```

4.2.15 ADC_RegularChannelConfig function

Table 25 describes the ADC_RegularChannelConfig function.

Table 25. ADC_RegularChannelConfig function

Function name	ADC_RegularChannelConfig
Function prototype	void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel, u8 Rank, u8 ADC_SampleTime)
Behavior description	Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_Channel: the ADC channel to be configured. Refer to *ADC_Channel* for details on the allowed values for this parameter.
Input parameter3	Rank: The rank in the regular group sequencer. This parameter ranges from 1 to 16.
Input parameter4	ADC_SampleTime: The sample time value to be set for the selected channel. Refer to section *ADC_SampleTime* for details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ADC_Channel

The ADC_Channel parameter specifies the ADC channel that will be configured by issuing a ADC_RegularChannelConfig function. *Table 26* shows the values taken by ADC_Channel:

Table 26. ADC_Channel values

ADC_Channel	Description
ADC_Channel_0	ADC Channel0 selected
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

Table 26. ADC_Channel values (continued)

ADC_Channel	Description
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_SampleTime

This parameter specifies the ADC samples time for the selected channel. *Table 27* gives the values taken by ADC_SampleTime.

Table 27. ADC_SampleTime values

ADC_SampleTime	Description
ADC_SampleTime_1Cycles5	Sample time equal to 1.5 cycles
ADC_SampleTime_7Cycles5	Sample time equal to 7.5 cycles
ADC_SampleTime_13Cycles5	Sample time equal to 13.5 cycles
ADC_SampleTime_28Cycles5	Sample time equal to 28.5 cycles
ADC_SampleTime_41Cycles5	Sample time equal to 41.5 cycles
ADC_SampleTime_55Cycles5	Sample time equal to 55.5 cycles
ADC_SampleTime_71Cycles5	Sample time equal to 71.5 cycles
ADC_SampleTime_239Cycles5	Sample time equal to 239.5 cycles

Example:

```
/* Configures ADC1 Channel2 as: first converted channel with an 7.5
cycles sample time */
ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 1,
ADC_SampleTime_7Cycles5);

/* Configures ADC1 Channel8 as: second converted channel with an 1.5
cycles sample time */
ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 2,
ADC_SampleTime_1Cycles5);
```

4.2.16 ADC_ExternalTrigConvCmd function

Table 28 describes the ADC_ExternalTrigConvCmd function.

Table 28. ADC_ExternalTrigConvCmd function

Function name	ADC_ExternalTrigConvCmd
Function prototype	void ADC_ExternalTrigConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the ADCx conversion through external Trigger.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the selected ADC external trigger starting the conversion. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/*Enable the start of conversion for ADC1 through exteral trigger */
ADC_ExternalTrigConvCmd(ADC1, ENABLE);
```

4.2.17 ADC_GetConversionValue function

Table 29 describes the ADC_GetConversionValue function.

Table 29. ADC_GetConversionValue function

Function name	ADC_GetConversionValue
Function prototype	u16 ADC_GetConversionValue(ADC_TypeDef* ADCx)
Behavior description	Returns the last ADCx conversion result data for regular channel.
Input parameter	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Output parameter	None
Return parameter	The Data conversion value.
Required preconditions	None
Called functions	None

Example:

```
/*Returns the ADC1 Master data value of the last converted channel*/
u16 DataValue;
DataValue = ADC_GetConversionValue(ADC1);
```

4.2.18 ADC_GetDualModeConversionValue function

Table 30 describes the ADC_GetDualModeConversionValue function.

Table 30. ADC_GetDualModeConversionValue function

Function name	ADC_GetDualModeConversionValue
Function prototype	u32 ADC_GetDualModeConversionValue()
Behavior description	Returns the last ADC converted data in dual mode
Output parameter	None
Return parameter	The Data conversion value.
Required preconditions	None
Called functions	None

Example:

```
/* Returns the ADC1 and ADC2 last converted values*/
u32 DataValue;
DataValue = ADC_GetDualModeConversionValue();
```

4.2.19 ADC_AutoInjectedConvCmd function

Table 31 describes the ADC_AutoInjectedConvCmd function.

Table 31. ADC_AutoInjectedConvCmd function

Function name	ADC_AutoInjectedConvCmd
Function prototype	void ADC_AutoInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the selected ADC automatic injected group conversion after regular group.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the selected ADC auto injected conversion. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the auto injected conversion for ADC2 */
ADC_AutoInjectedConvCmd(ADC2, ENABLE);
```

4.2.20 ADC_InjectedDiscModeCmd function

Table 32 describes the ADC_InjectedDiscModeCmd function.

Table 32. ADC_InjectedDiscModeCmd function

Function name	ADC_InjectedDiscModeCmd
Function prototype	void ADC_InjectedDiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the discontinuous mode for injected group channel for the specified ADC
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the selected ADC discontinuous mode on injected group channel. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the injected discontinuous mode for ADC2 */
ADC_InjectedDiscModeCmd(ADC2, ENABLE);
```

4.2.21 ADC_ExternalTrigInjectedConvConfig function

Table 33 describes the ADC_ExternalTrigInjectedConvConfig function.

Table 33. ADC_ExternalTrigInjectedConvConfig function

Function name	ADC_ExternalTrigInjectedConvConfig
Function prototype	void ADC_ExternalTrigInjectedConvConfig(ADC_TypeDef* ADCx, u32 ADC_ExternalTrigInjecConv)
Behavior description	Configures the ADCx external trigger for injected channels conversion.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_ExternalTrigInjecConv: the ADC trigger to start injected conversion. Refer to *ADC_ExternalTrigInjecConv* for details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ADC_ExternalTrigInjecConv

This parameter specifies the ADC trigger that is used to start injected conversion. *Table 34* gives the values taken by ADC_ExternalTrigInjecConv.

Table 34. ADC_ExternalTrigInjecConv values

ADC_ExternalTrigInjecConv	Description
ADC_ExternalTrigInjecConv_T1_TRGO	Timer1 TRGO event selected as external trigger for injected conversion (ADC1, ADC2 and ADC3)
ADC_ExternalTrigInjecConv_T1_CC4	Timer1 capture compare4 selected as external trigger for injected conversion (ADC1, ADC2 and ADC3)
ADC_ExternalTrigInjecConv_None	Injected conversion started by software and not by external trigger (ADC1, ADC2 and ADC3)
ADC_ExternalTrigInjecConv_T2_TRGO	Timer2 TRGO selected as external trigger for injected conversion (ADC1 and ADC2)
ADC_ExternalTrigInjecConv_T2_CC1	Timer2 capture compare1 selected as external trigger for injected conversion (ADC1 and ADC2)
ADC_ExternalTrigInjecConv_T3_CC4	Timer3 capture compare4 selected as external trigger for injected conversion (ADC1 and ADC2)
ADC_ExternalTrigInjecConv_T4_TRGO	Timer4 TRGO selected as external trigger for injected conversion (ADC1 and ADC2)
ADC_ExternalTrigInjecConv_Ext_IT15_TIM8_CC4	External interrupt 15 event/Timer8 capture compare4 selected as external trigger for injected conversion (ADC1 and ADC2)
ADC_ExternalTrigInjecConv_T4_CC3	Timer4 capture compare3 selected as external trigger for injected conversion (ADC3 only)
ADC_ExternalTrigInjecConv_T8_CC2	Timer8 capture compare2 selected as external trigger for injected conversion (ADC3 only)
ADC_ExternalTrigInjecConv_T8_CC4	Timer8 capture compare4 selected as external trigger for injected conversion (ADC3 only)
ADC_ExternalTrigInjecConv_T5_TRGO	Timer5 TRGO selected as external trigger for injected conversion (ADC3 only)
ADC_ExternalTrigInjecConv_T5_CC4	Timer5 capture compare4 selected as external trigger for injected conversion (ADC3 only)

Note: The TIM8_CC4 event exists only in High-density devices. The EXTI line15 or TIM8_CC4 external trigger event for injected channels is selected through AFIO configuration bits ADC1_ETRGINJ_REMAP and ADC2_ETRGINJ_REMAP for ADC1 and ADC2, respectively.

Example:

```
/* Set ADC1 injected external trigger conversion start to Timer1
capture compare4 */
ADC_ExternalTrigInjectedConvConfig(ADC1,
ADC_ExternalTrigConv_T1_CC4);
```

4.2.22 ADC_ExternalTrigInjectedConvCmd function

Table 35 describes the ADC_ExternalTrigInjectedConvCmd function.

Table 35. ADC_ExternalTrigInjectedConvCmd function

Function name	ADC_ExternalTrigInjectedConvCmd
Function prototype	void ADC_ExternalTrigInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the ADCx injected channels conversion through external trigger
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the selected ADC external trigger used to start injected conversion. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the start of injected conversion for ADC1 through exteral
trigger */
ADC_ExternalTrigInjectedConvCmd(ADC1, ENABLE);
```

4.2.23 ADC_SoftwareStartInjectedConvCmd function

Table 36 describes the ADC_SoftwareStartInjectedConvCmd function.

Table 36. ADC_SoftwareStartInjectedConvCmd function

Function name	ADC_SoftwareStartInjectedConvCmd
Function prototype	void ADC_SoftwareStartInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
Behavior description	Enables or disables the start of the injected channels conversion.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	NewState: new state of the selected ADC software used to start injected conversion. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Start by software the ADC2 Conversion */
ADC_SoftwareStartInjectedConvCmd(ADC2, ENABLE);
```

4.2.24 ADC_GetSoftwareStartInjectedConvStatus function

Table 37 describes the ADC_GetSoftwareStartInjectedConvStatus function.

Table 37. ADC_GetSoftwareStartInjectedConvStatus function

Function name	ADC_GetSoftwareStartInjectedConvStatus
Function prototype	FlagStatus ADC_GetSoftwareStartInjectedConvStatus(ADC_TypeDef* ADCx)
Behavior description	Gets the selected ADC Software start injected conversion Status.
Input parameter	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Output parameter	None
Return parameter	The new state of ADC software start injected conversion (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the ADC1 injected conversion start bit */
FlagStatus Status;
Status = ADC_GetSoftwareStartInjectedConvStatus(ADC1);
```

4.2.25 ADC_InjectedChannelConfig function

Table 38 describes the ADC_InjectedChannelConfig function.

Table 38. ADC_InjectedChannelConfig function

Function name	ADC_InjectedChannelConfig
Function prototype	void ADC_InjectedChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel, u8 Rank, u8 ADC_SampleTime)
Behavior description	Configures for the selected ADC injected channel the corresponding rank in the sequencer and the sample time.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_Channel: ADC channel to be configured. Refer to *ADC_Channel* for more details on the allowed values for this parameter.
Input parameter3	Rank: The rank in the injected group sequencer. This parameter ranges from 1 to 4.
Input parameter4	ADC_SampleTime: sample time value to be set for the selected channel. Refer to *ADC_SampleTime* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	ADC_InjectedSequencerLengthConfig must be called before to specify the total injected channel number. This is necessary specially when this number is less than 4 to properly configure the rank of each injected channel
Called functions	None

ADC_Channel

ADC_Channel specifies the ADC channel to be configured. Refer to *Table 26* for the values taken by this parameter.

ADC_SampleTime

ADC_SampleTime specifies the ADC Sample Time for the selected channel. Refer to *Table 27* for the values taken by this parameter.

Example:

```
/* Configures ADC1 Channel12 as: second converted channel with a
28.5 cycle sample time */
ADC_InjectedChannelConfig(ADC1, ADC_Channel_12, 2,
ADC_SampleTime_28Cycles5);

/* Configures ADC2 Channel4 as: fourth converted channel with a 71.5
cycle sample time */
ADC_InjectedChannelConfig(ADC2, ADC_Channel_4, 4,
ADC_SampleTime_71Cycles5);
```

4.2.26 ADC_InjectedSequencerLengthConfig function

Table 39 describes the ADC_InjectedSequencerLengthConfig function.

Table 39. ADC_InjectedSequencerLengthConfig function

Function name	ADC_InjectedSequencerLengthConfig
Function prototype	void ADC_InjectedSequencerLengthConfig(ADC_TypeDef* ADCx, u8 Length)
Behavior description	Configures the sequencer length for injected channels
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	Length: sequencer length. This parameter ranges from 1 to 4.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the ADC1 Sequencer length to 4 channels */
ADC_InjectedSequencerLengthConfig(ADC1, 4);
```

4.2.27 ADC_SetInjectedOffset function

Table 40 describes the ADC_SetInjectedOffset function.

Table 40. ADC_SetInjectedOffset function

Function name	ADC_SetInjectedOffset
Function prototype	void ADC_SetInjectedOffset(ADC_TypeDef* ADCx, u8 ADC_InjectedChannel, u16 Offset)
Behavior description	Set the injected channels conversion offset value
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_InjectedChannel: ADC injected channel for which the offset must be set Refer to *ADC_InjectedChannel* for more details on the allowed values for this parameter.
Input parameter3	Offset: offset value for the selected ADC injected channel This parameter is a 12-bit value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ADC_InjectedChannel

ADC_InjectedChannel specifies the ADC injected channel for which the offset must be set. *Table 41* gives the values of this parameter.

Table 41. ADC_InjectedChannel values

ADC_InjectedChannel	Description
ADC_InjectedChannel_1	Injected Channel1 selected
ADC_InjectedChannel_2	Injected Channel2 selected
ADC_InjectedChannel_3	Injected Channel3 selected
ADC_InjectedChannel_4	Injected Channel4 selected

Example:

```
/* Set the offset 0x100 for the 3rd injected Channel of ADC1 */
ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_3, 0x100);
```

4.2.28 ADC_GetInjectedConversionValue function

Table 42 describes the ADC_GetInjectedConversionValue function.

Table 42. ADC_GetInjectedConversionValue function

Function name	ADC_GetInjectedConversionValue
Function prototype	u16 ADC_GetInjectedConversionValue(ADC_TypeDef* ADCx, u8 ADC_InjectedChannel)
Behavior description	Returns the selected ADC injected channel conversion result
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_InjectedChannel: converted ADC injected channel. Refer to *ADC_InjectedChannel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	Data conversion value.
Required preconditions	None
Called functions	None

Example:

```
/* Return the ADC1 injected channel1 converted data value */
u16 InjectedDataValue;
InjectedDataValue = ADC_GetInjectedConversionValue(ADC1,
ADC_InjectedChannel_1);
```

4.2.29 ADC_AnalogWatchdogCmd function

Table 43 describes the ADC_AnalogWatchdogCmd function.

Table 43. ADC_AnalogWatchdogCmd function

Function name	ADC_AnalogWatchdogCmd
Function prototype	void ADC_AnalogWatchdogCmd(ADC_TypeDef* ADCx, u32 ADC_AnalogWatchdog)
Behavior description	Enables or disables the analog watchdog on one or all regular or injected channels
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_AnalogWatchdog: ADC analog watchdog configuration. Refer to *ADC_AnalogWatchdog* for more details on the values taken by this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ADC_AnalogWatchdog

ADC_AnalogWatchdog specifies the ADC analog watchdog configuration. *Table 44* gives the value taken by this parameter.

Table 44. ADC_AnalogWatchdog values

ADC_AnalogWatchdog	Description
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 channels
ADC_AnalogWatchdog_AllInjecEnable	Analog watchdog on all injected channels
ADC_AnalogWatchdog_AllRegAllInjecEnable	Analog watchdog on all regular and injected channels
ADC_AnalogWatchdog_None	No channel guarded by the analog watchdog

Example:

```
/* Configue the Analog watchdog on all regular and injected channels
of ADC2 */
ADC_AnalogWatchdogCmd(ADC2,
ADC_AnalogWatchdog_AllRegAllInjecEnable);
```

4.2.30 ADC_AnalogWatchdogThresholdsConfig function

Table 45 describes the ADC_AnalogWatchdogThresholdsConfig function.

Table 45. ADC_AnalogWatchdogThresholdsConfig function

Function name	ADC_AnalogWatchdogThresholdsConfig
Function prototype	void ADC_AnalogWatchdogThresholdsConfig(ADC_TypeDef* ADCx, u16 HighThreshold, u16 LowThreshold)
Behavior description	Configures the high and low thresholds of the analog watchdog
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	HighThreshold: ADC analog watchdog High threshold value. This parameter must be a 12-bit value.
Input parameter3	LowThreshold: ADC analog watchdog Low threshold value. This parameter must be a 12-bit value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Configue the Analog watchdog High and Low thresholds for ADC1 */
ADC_AnalogWatchdogThresholdsConfig(ADC1, 0x400, 0x100);
```

4.2.31 ADC_AnalogWatchdogSingleChannelConfig function

Table 46 describes the AnalogWatchdogSingleChannelConfig function.

Table 46. AnalogWatchdogSingleChannelConfig function

Function name	ADC_AnalogWatchdogSingleChannelConfig
Function prototype	void ADC_AnalogWatchdogSingleChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel)
Behavior description	Configures the analog watchdog guarded single channel
Input parameter1	ADCx: where x can be 1, 2 or 3 to select ADC1, ADC2 or ADC3 peripheral
Input parameter2	ADC_Channel: ADC channel for which the analog watchdog will be configured. Refer to *ADC_Channel* or more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Configue the Analog watchdog on Channel1 of ADC1 */
ADC_AnalogWatchdogSingleChannelConfig(ADC1, ADC_Channel_1);
```

4.2.32 ADC_TempSensorVrefintCmd function

Table 47 describes the ADC_TempSensorVrefintCmd function.

Table 47. ADC_TempSensorVrefintCmd function

Function name	ADC_TempSensorVrefintCmd
Function prototype	void ADC_TempSensorVrefintCmd(FunctionalState NewState)
Behavior description	Enables or disables the temperature sensor and Vrefint channel.
Input parameter	NewState: new state of the temperature sensor and Vrefint channel This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the temperature sensor and vref internal channel */
ADC_TempSensorVrefintCmd(ENABLE);
```

4.2.33 ADC_GetFlagStatus function

Table 48 describes the ADC_GetFlagStatus function.

Table 48. ADC_GetFlagStatus function

Function name	ADC_GetFlagStatus
Function prototype	FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, u8 ADC_FLAG)
Behavior description	Checks whether the specified ADC flag is set or not.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_FLAG: specifies the flag to check. Refer to *ADC_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	New state of ADC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

ADC_FLAG

The values of the ADC_FLAG are given in *Table 49*.

Table 49. ADC_FLAG values

ADC_FLAG	Description
ADC_FLAG_AWD	Analog watchdog flag
ADC_FLAG_EOC	End of conversion flag
ADC_FLAG_JEOC	End of injected group conversion flag
ADC_FLAG_JSTRT	Start of injected group conversion flag
ADC_FLAG_STRT	Start of regular group conversion flag

Example:

```
/* Test if the ADC1 EOC flag is set or not */
FlagStatus Status;
Status = ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC);
```

4.2.34 ADC_ClearFlag function

Table 50 describes the ADC_ClearFlag function.

Table 50. ADC_ClearFlag function

Function name	ADC_ClearFlag
Function prototype	void ADC_ClearFlag(ADC_TypeDef* ADCx, u8 ADC_FLAG)
Behavior description	Clears the ADCx's pending flags.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_FLAG: flag to clear. More than one flag can be cleared using the “|” operator. Refer to *ADC_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the ADC2 STRT pending flag */
ADC_ClearFlag(ADC2, ADC_FLAG_STRT);
```

4.2.35 ADC_GetITStatus function

Table 51 describes the ADC_GetITStatus function.

Table 51. ADC_GetITStatus function

Function name	ADC_GetITStatus
Function prototype	ITStatus ADC_GetITStatus(ADC_TypeDef* ADCx, u16 ADC_IT)
Behavior description	Checks whether the specified ADC interrupt has occurred or not.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_IT: ADC interrupt source to check. Refer to *ADC_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of ADC_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Test if the ADC1 AWD interrupt has occurred or not */
ITStatus Status;
Status = ADC_GetITStatus(ADC1, ADC_IT_AWD);
```

4.2.36 ADC_ClearITPendingBit function

Table 52 describes the ADC_ClearITPendingBit function.

Table 52. ADC_ClearITPendingBit function

Function name	ADC_ClearITPending Bit
Function prototype	void ADC_ClearITPendingBit(ADC_TypeDef* ADCx, u16 ADC_IT)
Behavior description	Clears the ADCx's interrupt pending bits.
Input parameter1	ADCx: where x can be 1, 2 or 3 to select the ADC1, ADC2 or ADC3 peripheral.
Input parameter2	ADC_IT: interrupt pending bit to clear. Refer to *ADC_IT*" for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the ADC2 JEOC interrupt pending bit */
ADC_ClearITPendingBit(ADC2, ADC_IT_JEOC);
```

5 Backup registers (BKP)

There are forty-two 16-bit backup registers used to store 84 bytes of user application data. They are implemented in the backup domain that remains powered on by V_{BAT} when V_{DD} is switched off.

The BKP registers are also used to manage Tamper detection feature and RTC calibration.

Section 5.1: BKP register structure describes the data structures used in the BKP firmware library. *Section 5.2: Firmware library functions* presents the firmware library functions.

5.1 BKP register structure

The BKP register structure, *BKP_TypeDef,* is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  u32  RESERVED0;
  vu16 DR1;
  u16  RESERVED1;
  vu16 DR2;
  u16  RESERVED2;
  vu16 DR3;
  u16  RESERVED3;
  vu16 DR4;
  u16  RESERVED4;
  vu16 DR5;
  u16  RESERVED5;
  vu16 DR6;
  u16  RESERVED6;
  vu16 DR7;
  u16  RESERVED7;
  vu16 DR8;
  u16  RESERVED8;
  vu16 DR9;
  u16  RESERVED9;
  vu16 DR10;
  u16  RESERVED10;
  vu16 RTCCR;
  u16  RESERVED11;
  vu16 CR;
  u16  RESERVED12;
  vu16 CSR;
  u16  RESERVED13[5];
  vu16 DR11;
  u16  RESERVED14;
  vu16 DR12;
  u16  RESERVED15;
  vu16 DR13;
  u16  RESERVED16;
  vu16 DR14;
  u16  RESERVED17;
```

```
vu16 DR15;
u16  RESERVED18;
vu16 DR16;
u16  RESERVED19;
vu16 DR17;
u16  RESERVED20;
vu16 DR18;
u16  RESERVED21;
vu16 DR19;
u16  RESERVED22;
vu16 DR20;
u16  RESERVED23;
vu16 DR21;
u16  RESERVED24;
vu16 DR22;
u16  RESERVED25;
vu16 DR23;
u16  RESERVED26;
vu16 DR24;
u16  RESERVED27;
vu16 DR25;
u16  RESERVED28;
vu16 DR26;
u16  RESERVED29;
vu16 DR27;
u16  RESERVED30;
vu16 DR28;
u16  RESERVED31;
vu16 DR29;
u16  RESERVED32;
vu16 DR30;
u16  RESERVED33;
vu16 DR31;
u16  RESERVED34;
vu16 DR32;
u16  RESERVED35;
vu16 DR33;
u16  RESERVED36;
vu16 DR34;
u16  RESERVED37;
vu16 DR35;
u16  RESERVED38;
vu16 DR36;
u16  RESERVED39;
vu16 DR37;
u16  RESERVED40;
vu16 DR38;
u16  RESERVED41;
vu16 DR39;
u16  RESERVED42;
vu16 DR40;
u16  RESERVED43;
```

```
  vu16 DR41;
  u16  RESERVED44;
  vu16 DR42;
  u16  RESERVED45;
} BKP_TypeDef;
```

Table 53 gives the list of the BKP registers:

Table 53. BKP registers

Register	Description
DR 1-10	Data Backup Register 1 to 42
RTCCR	RTC Clock Calibration Register
CR	Backup Control Register
CSR	Backup Control Status Register

The BKP peripheral is also declared in *stm32f10x_map.h*:

```
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
#define BKP_BASE              (APB1PERIPH_BASE + 0x6C00)
#ifndef DEBUG
...
#ifdef _BKP
  #define BKP                   ((BKP_TypeDef *) BKP_BASE)
#endif /*_BKP */
...
#else   /* DEBUG */
...
#ifdef _BKP
  EXT BKP_TypeDef             *BKP;
#endif /*_BKP */
...
#endif
```

When using the Debug mode, the BKP pointer is initialized in *stm32f10x_lib.c*:

```
#ifdef _BKP
  BKP = (BKP_TypeDef *)  BKP_BASE;
#endif /*_BKP */
```

To access the backup registers, _BKP must be defined in *stm32f10x_conf.h,* as follows:

```
#define _BKP
```

5.2 Firmware library functions

Table 54 lists the BKP library functions.

Table 54. BKP library functions

Function name	Description
BKP_DeInit	Resets the BKP peripheral registers to their default reset values.
BKP_TamperPinLevelConfig	Configures the Tamper Pin active level.
BKP_TamperPinCmd	Enables or disables the Tamper Pin activation.
BKP_ITConfig	Enables or disables the Tamper Pin Interrupt.
BKP_RTCOutputConfig	Selects the RTC output source to output on the Tamper pin.
BKP_SetRTCCalibrationValue	Sets RTC Clock Calibration value.
BKP_WriteBackupRegister	Writes user data to the specified Data Backup Register.
BKP_ReadBackupRegister	Reads data from the specified Data Backup Register.
BKP_GetFlagStatus	Checks whether the Tamper Pin Event flag is set or not.
BKP_ClearFlag	Clears Tamper Pin Event pending flag.
BKP_GetITStatus	Checks whether the Tamper Pin Interrupt has occurred or not.
BKP_ClearITPendingBit	Clears Tamper Pin Interrupt pending bit.

5.2.1 BKP_DeInit function

Table 55 describes the BKP_DeInit function.

Table 55. BKP_DeInit function

Function name	BKP_DeInit
Function prototype	void BKP_DeInit(void)
Behavior description	Resets the BKP registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_BackupResetCmd

Example:

```
/* Reset the BKP registers */
BKP_DeInit();
```

5.2.2 BKP_TamperPinLevelConfig function

Table 56 describes the BKP_TamperPinLevelConfig function.

Table 56. BKP_TamperPinLevelConfig function

Function name	BKP_TamperPinLevelConfig
Function prototype	void BKP_TamperPinLevelConfig(u16 BKP_TamperPinLevel)
Behavior description	Configures the Tamper Pin active level.
Input parameter	BKP_TamperPinLevel: Tamper Pin active level. Refer to *BKP_TamperPinLevel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

BKP_TamperPinLevel

The BKP_TamperPinLevel input parameter is used to select the Tamper Pin active level. It can take one of the following values:

Table 57. BKP_TamperPinLevel values

BKP_TamperPinLevel	Description
BKP_TamperPinLevel_High	Tamper pin active on high level
BKP_TamperPinLevel_Low	Tamper pin active on low level

Example:

```
/* Configure Tamper pin to be active on high level*/
BKP_TamperPinLevelConfig(BKP_TamperPinLevel_High);
```

5.2.3 BKP_TamperPinCmd function

Table 58 describes the BKP_TamperPinCmd function.

Table 58. BKP_TamperPinCmd function

Function name	BKP_TamperPinCmd
Function prototype	void BKP_TamperPinCmd(FunctionalState NewState)
Behavior description	Enables or disables the Tamper Pin activation.
Input parameter	NewState: new state of the Tamper Pin activation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable Tamper Pin functionality */
BKP_TamperPinCmd(ENABLE);
```

5.2.4 BKP_ITConfig function

Table 59 describes the BKP_ITConfig function.

Table 59. BKP_ITConfig function

Function name	BKP_ITConfig
Function prototype	void BKP_ITConfig(FunctionalState NewState)
Behavior description	Enables or disables the Tamper Pin Interrupt.
Input parameter	NewState: new state of the Tamper Pin Interrupt. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable Tamper Pin interrupt */
BKP_ITConfig(ENABLE);
```

5.2.5 BKP_RTCOutputConfig function

Table 60 describes the BKP_RTCOutputConfig function.

Table 60. BKP_RTCOutputConfig function

Function name	BKP_RTCOutputConfig
Function prototype	void BKP_RTCOutputConfig(u16 BKP_RTCOutputSource)
Behavior description	Selects the RTC output source to output on the Tamper pin.
Input parameter	BKP_RTCOutputSource: specifies the RTC output source. Refer to *BKP_RTCOutputSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	The Tamper Pin functionality must be disabled before using this function.
Called functions	None

BKP_RTCOutputSource

The BKP_RTCOutputSource input parameter is used to select the RTC output source. It can take one of the following values:

Table 61. BKP_RTCOutputSource values

BKP_RTCOutputSource	Description
BKP_RTCOutputSource_None	No RTC output on the Tamper pin.
BKP_RTCOutputSource_CalibClock	Output the RTC clock with frequency divided by 64 on the Tamper pin
BKP_RTCOutputSource_Alarm	Output the RTC Alarm pulse signal on the Tamper pin.
BKP_RTCOutputSource_Second	Output the RTC Second pulse signal on the Tamper pin.

Example:

```
/* Output the RTC clock source with frequency divided by 64 on the
Tamper pad(if the Tamper Pin functionality is disabled) */
BKP_RTCOutputConfig(BKP_RTCOutputSource_CalibClock);
```

5.2.6 BKP_SetRTCCalibrationValue function

Table 62 describes the BKP_SetRTCCalibrationValue function.

Table 62. BKP_SetRTCCalibrationValue function

Function name	BKP_SetRTCCalibrationValue
Function prototype	void BKP_SetRTCCalibrationValue(u8 CalibrationValue)
Behavior description	Sets RTC Clock Calibration value.
Input parameter	CalibrationValue: RTC Clock Calibration value. This parameter ranges from 0 to 0x7F.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set RTC clock calibration value to 0x7F (maximum) */
BKP_SetRTCCalibrationValue(0x7F);
```

5.2.7 BKP_WriteBackupRegister function

Table 63 describes the BKP_WriteBackupRegister function.

Table 63. BKP_WriteBackupRegister function

Function name	BKP_WriteBackupRegister
Function prototype	void BKP_WriteBackupRegister(u16 BKP_DR, u16 Data)
Behavior description	Writes user data to the specified Data Backup Register.
Input parameter1	BKP_DR: Data Backup Register. Refer to *BKP_DR* for more details on the allowed values for this parameter.
Input parameter2	Data: data to write.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

BKP_DR

BKP_DR is used to select the Data Backup Register. *Table 64* shows the values taken by this parameter.

Table 64. BKP_DR values

BKP_DR	Description
BKP_DRx	Data Backup Register x is selected, where x is a value between 1 and 42

Example:

```
/* Write 0xA587 to Data Backup Register1 */
BKP_WriteBackupRegister(BKP_DR1, 0xA587);
```

5.2.8 BKP_ReadBackupRegister function

Table 65 describes the BKP_ReadBackupRegister function.

Table 65. BKP_ReadBackupRegister function

Function name	BKP_ReadBackupRegister
Function prototype	u16 BKP_ReadBackupRegister(u16 BKP_DR)
Behavior description	Reads data from the specified Data Backup Register.
Input parameter	BKP_DR: Data Backup Register. Refer to *BKP_DR* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The content of the specified Data Backup Register.
Required preconditions	None
Called functions	None

Example:

```
/* Read Data Backup Register1 */
u16 Data;
Data = BKP_ReadBackupRegister(BKP_DR1);
```

5.2.9 BKP_GetFlagStatus function

Table 66 describes the BKP_GetFlagStatus function.

Table 66. BKP_GetFlagStatus function

Function name	BKP_GetFlagStatus
Function prototype	FlagStatus BKP_GetFlagStatus(void)
Behavior description	Checks whether the Tamper Pin Event flag is set or not.
Input parameter	None
Output parameter	None
Return parameter	The new state of the Tamper Pin Event flag (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Test if the Tamper Pin Event flag is set or not */
FlagStatus Status;
Status = BKP_GetFlagStatus();
if(Status == RESET)
{
...
}
else
{
...
}
```

5.2.10 BKP_ClearFlag function

Table 67 describes the BKP_ClearFlag function.

Table 67. BKP_ClearFlag function

Function name	BKP_ClearFlag
Function prototype	void BKP_ClearFlag(void)
Behavior description	Clears Tamper Pin Event pending flag.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear Tamper Pin Event pending flag */
BKP_ClearFlag();
```

5.2.11 BKP_GetITStatus function

Table 68 describes the BKP_GetITStatus function.

Table 68. BKP_GetITStatus function

Function name	BKP_GetITStatus
Function prototype	ITStatus BKP_GetITStatus(void)
Behavior description	Checks whether the Tamper Pin Interrupt has occurred or not.
Input parameter	None
Output parameter	None
Return parameter	The new state of the Tamper Pin Interrupt (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Test if the Tamper Pin interrupt has occurred or not */
ITStatus Status;
Status = BKP_GetITStatus();
if(Status == RESET)
{
...
}
else
{
...
}
```

5.2.12 BKP_ClearITPendingBit function

Table 69 describes the BKP_ClearITPendingBit function.

Table 69. BKP_ClearITPendingBit function

Function name	BKP_ClearITPendingBit
Function prototype	void BKP_ClearITPendingBit(void)
Behavior description	Clears Tamper Pin Interrupt pending bit.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear Tamper Pin interrupt pending bit */
BKP_ClearITPendingBit();
```

6 Controller area network (CAN)

This peripheral interfaces the CAN network. It supports the CAN protocols version 2.0A and B. It has been designed to manage efficiently a high number of incoming messages with a minimum CPU load. It also meets the priority requirements for transmit messages.

Section 6.1 describes the data structures used in the CAN firmware library. *Section 6.2* presents the firmware library functions.

6.1 CAN register structure

The CAN register structure, CAN_*TypeDef,* is defined in *stm32f10x_map.h* as follows:

```
typedef struct
{
  vu32 MCR;
  vu32 MSR;
  vu32 TSR;
  vu32 RF0R;
  vu32 RF1R;
  vu32 IER;
  vu32 ESR;
  vu32 BTR;
  u32 RESERVED0[88];
  CAN_TxMailBox_TypeDef sTxMailBox[3];
  CAN_FIFOMailBox_TypeDef sFIFOMailBox[2];
  u32 RESERVED1[12];
  vu32 FMR;
  vu32 FM0R;
  u32 RESERVED2[1];
  vu32 FS0R;
  u32 RESERVED3[1];
  vu32 FFA0R;
  u32 RESERVED4[1];
  vu32 FA0R;
  u32 RESERVED5[8];
  CAN_FilterRegister_TypeDef sFilterRegister[14];
} CAN_TypeDef;

typedef struct
{
  vu32 TIR;
  vu32 TDTR;
  vu32 TDLR;
  vu32 TDHR;
} CAN_TxMailBox_TypeDef;

typedef struct
{
  vu32 RIR;
  vu32 RDTR;
```

```
  vu32 RDLR;
  vu32 RDHR;
} CAN_FIFOMailBox_TypeDef;
typedef struct
{
vu32 FR0;
vu32 FR1;
} CAN_FilterRegister_TypeDef;
```

Table 70 shows the list of all CAN registers.

Table 70. CAN registers

Register	Description
CAN_MCR	CAN Master Control Register
CAN_MSR	CAN Master Status Register
CAN_TSR	CAN Transmit Status Register
CAN_RF0R	CAN Receive FIFO 0 Register
CAN_RF1R	CAN Receive FIFO 1 Register
CAN_IER	CAN Interrupt Enable Register
CAN_ESR	CAN Error Status Register
CAN_BTR	CAN Bit Timing Register
TIR	Tx Mailbox Identifier Register
TDTR	Mailbox Data Length Control and Time Stamp Register
TDLR	Mailbox Data Low Register
TDHR	Mailbox Data High Register
RIR	Rx FIFO Mailbox Identifier Register
RDTR	Receive FIFO Mailbox Data Length Control and Time Stamp Register
RDLR	Receive FIFO Mailbox Data Low Register
RDHR	Receive FIFO Mailbox Data High Register
CAN_FMR	CAN Filter Master Register
CAN_FM0R	CAN Filter Mode Register
CAN_FSC0R	CAN Filter Scale Register
CAN_FFA0R	CAN Filter FIFO Assignment Register
CAN_FA0R	CAN Filter Activation Register
CAN_FR0	Filter x Register 0
CAN_FR1	Filter x Register 1

The CAN peripheral is also declared in *stm32f10x_map.h*:

```
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
```

```
#define CAN_BASE              (APB1PERIPH_BASE + 0x6400)

#ifndef DEBUG
...
#ifdef _CAN
  #define CAN                 ((CAN_TypeDef *) CAN_BASE)
#endif /*_CAN */
...
#else   /* DEBUG */
...
#ifdef _CAN
  EXT CAN_TypeDef             *CAN;
#endif /*_CAN */
...
#endif
```

When using the Debug mode, the CAN pointer is initialized in *stm32f10x_lib.c*:

```
#ifdef _CAN
  CAN = (CAN_TypeDef *)  CAN_BASE;
#endif /*_CAN */
```

To access the CAN registers, _CAN must be defined in stm32f10x_conf.h:

```
#define _CAN
```

6.2 Firmware library functions

Table 71 gives the list of the CAN library functions.

Table 71. CAN firmware library functions

Function name	Description
CAN_DeInit	Resets the CAN peripheral registers to their default reset values.
CAN_Init	Initializes the CAN peripheral according to the parameters specified in the CAN_InitStruct.
CAN_FilterInit	Initializes the CAN peripheral according to the parameters specified in the CAN_FilterInitStruct.
CAN_StructInit	Fills each CAN_InitStruct member with its default value.
CAN_ITConfig	Enables or disables the specified CAN interrupts.
CAN_Transmit	Initiates the transmission of a message
CAN_TransmitStatus	Checks the transmission of a message
CAN_CancelTransmit	Cancels a transmit request
CAN_FIFORelease	Releases a FIFO
CAN_MessagePending	Returns the number of pending messages
CAN_Receive	Receives a message
CAN_Sleep	Enters the low power mode
CAN_WakeUp	Wakes the CAN up
CAN_GetFlagStatus	Checks whether the specified CAN flag is set or not.
CAN_ClearFlag	Clears the CAN pending flags.
CAN_GetITStatus	Checks whether the specified CAN interrupt has occurred or not.
CAN_ClearITPendingBit	Clears the CAN interrupt pending bits.

6.2.1 CAN_DeInit function

Table 72 describes the CAN_DeInit function.

Table 72. CAN_DeInit function

Function name	CAN_DeInit
Function prototype	void CAN_DeInit(void)
Behavior description	Resets the CAN peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB1PeriphResetCmd()

Example:

```
/* Deinitialize the CAN */
CAN_DeInit();
```

6.2.2 CAN_Init function

Table 73 describes the CAN_Init function.

Table 73. CAN_Init function

Function name	CAN_Init
Function prototype	u8 CAN_Init(CAN_InitTypeDef* CAN_InitStruct)
Behavior description	Initializes the CAN peripheral according to the parameters specified in the CAN_InitStruct.
Input parameter	CAN_InitStruct: pointer to a CAN_InitTypeDef structure that contains the configuration information for the CAN peripheral. Refer to *CAN_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	Constant indicating that the CAN initialization has been successful. CANINITFAILED = initialization failed CANINITOK = initialization successful
Required preconditions	None
Called functions	None

CAN_InitTypeDef structure

The CAN_InitTypeDef structure is defined in the *stm32f10x_can.h* file:

```
typedef struct
{
  FunctionnalState CAN_TTCM;
  FunctionnalState CAN_ABOM;
  FunctionnalState CAN_AWUM;
  FunctionnalState CAN_NART;
  FunctionnalState CAN_RFLM;
  FunctionnalState CAN_TXFP;
  u8 CAN_Mode;
  u8 CAN_SJW;
  u8 CAN_BS1;
  u8 CAN_BS2;
  u16 CAN_Prescaler;
} CAN_InitTypeDef;
```

CAN_TTCM

CAN_TTCM is used to enable or disable the time triggered communication mode. This member can be set either to ENABLE or DISABLE.

CAN_ABOM

CAN_ABOM is used to enable or disable the automatic bus-off management. This member can be set either to ENABLE or DISABLE.

CAN_AWUM

CAN_AWUM is used to enable or disable the automatic wake-up mode. This member can be set either to ENABLE or DISABLE.

CAN_NART

CAN_NART is used to enable or disable the no-automatic retransmission mode. This member can be either set to ENABLE or DISABLE.

CAN_RFLM

CAN_RFLM is used to enable or disable the Receive Fifo Locked mode. This member can be either set to ENABLE or DISABLE.

CAN_TXFP

CAN_TXFP is used to enable or disable the transmit FIFO priority. This member can be set either to ENABLE or DISABLE.

CAN_Mode

CAN_Mode configures the CAN operating mode. The values taken by this member are given in *Table 74*.

Table 74. CAN_Mode values

CAN_Mode	Description
CAN_Mode_Normal	CAN hardware operates in normal mode
CAN_Mode_Silent	CAN hardware operates in silent mode
CAN_Mode_LoopBack	CAN hardware operates in loop back mode
CAN_Mode_Silent_LoopBack	CAN hardware operates in loop back combined with silent mode

CAN_SJW

CAN_SJW configures the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. The values taken by this member are given in *Table 75*.

Table 75. CAN_SJW values

CAN_SJW	Description
CAN_SJW_1tq	Resynchronization Jump Width=1 time quantum
CAN_SJW_2tq	Resynchronization Jump Width= 2 time quantum
CAN_SJW_3tq	Resynchronization Jump Width= 3 time quantum
CAN_SJW_4tq	Resynchronization Jump Width= 4 time quantum

CAN_BS1

CAN_BS1 configures the number of time quanta in Bit Segment 1. The values taken by this member are given in *Table 76*.

Table 76. CAN_BS1 values

CAN_BS1	Description
CAN_BS1_1tq	Bit Segment 1= 1 time quantum
...	...
CAN_BS1_16tq	Bit Segment 1= 16 time quantum

CAN_BS2

CAN_BS2 configures the number of time quanta in Bit Segment 2. The values taken by this member are given in *Table 77*.

Table 77. CAN_BS2 values

CAN_BS2	Description
CAN_BS2_1tq	Bit Segment 2= 1 time quantum
...	...
CAN_BS2_8tq	Bit Segment 2= 8 time quantum

CAN_Prescaler

CAN_Prescaler configures the length of a time quantum. It ranges from 1 to 1024.

Example:

```
/* Initialize the CAN as 1Mb/s in normal mode, receive FIFO locked:
*/
CAN_InitTypeDef CAN_InitStructure;

CAN_InitStructure.CAN_TTCM = DISABLE;
CAN_InitStructure.CAN_ABOM = DISABLE;
CAN_InitStructure.CAN_AWUM = DISABLE;
CAN_InitStructure.CAN_NART = DISABLE;
CAN_InitStructure.CAN_RFLM = ENABLE;
CAN_InitStructure.CAN_TXFP = DISABLE;
CAN_InitStructure.CAN_Mode = CAN_Mode_Normal;
CAN_InitStructure.CAN_BS1 = CAN_BS1_4tq;
CAN_InitStructure.CAN_BS2 = CAN_BS2_3tq;
CAN_InitStructure.CAN_Prescaler = 0;
CAN_Init(&CAN_InitStructure);
```

6.2.3 CAN_FilterInit function

Table 78 describes the CAN_FilterInit function.

Table 78. CAN_FilterInit function

Function name	CAN_FilterInit
Function prototype	void CAN_FilterInit(CAN_FilterInitTypeDef* CAN_FilterInitStruct)
Behavior description	Initializes the CAN peripheral according to the specified parameters in the CAN_FilterInitStruct.
Input parameter	CAN_FilterInitStruct: pointer to a CAN_FilterInitTypeDef structure containing the configuration information. Refer to *CAN_FilterInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

CAN_FilterInitTypeDef structure

The CAN_FilterInitTypeDef structure is defined in the *stm32f10x_can.h* file:

```
typedef struct
{
  u8 CAN_FilterNumber;
  u8 CAN_FilterMode;
  u8 CAN_FilterScale;
  u16 CAN_FilterIdHigh;
  u16 CAN_FilterIdLow;
  u16 CAN_FilterMaskIdHigh;
  u16 CAN_FilterMaskIdLow;
  u16 CAN_FilterFIFOAssignment;
  FunctionalState CAN_FilterActivation;
} CAN_FilterInitTypeDef;
```

CAN_FilterNumber

CAN_FilterNumber selects the filter which will be initialized. It ranges from 0 to 13.

CAN_FilterMode

CAN_FilterMode selects the mode to be initialized. The values taken by this member are given in *Table 79*.

Table 79. CAN_FilterMode values

CAN_FilterMode	Description
CAN_FilterMode_IdMask	id/mask mode
CAN_FilterMode_IdList	identifier list mode

CAN_FilterScale

CAN_FilterScale configures the filter scale. The values taken by this member are given in *Table 80*.

Table 80. CAN_FilterScale values

CAN_FilterScale	Description
CAN_FilterScale_Two16bit	Two 16-bit filters
CAN_FilterScale_One32bit	One 32-bit filter

CAN_FilterIdHigh

CAN_FilterIdHigh is used to select the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterIdLow

CAN_FilterIdLow is used to select the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterMaskIdHigh

CAN_FilterMaskIdHigh is used to select the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterMaskIdLow

CAN_FilterMaskIdLow is used to select the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). It ranges from 0x0000 to 0xFFFF.

CAN_FilterFIFO

CAN_FilterFIFO is used to select the FIFO (0 or 1) which will be assigned to the filter. The values taken by this member are given in *Table 81*.

Table 81. CAN_FilterFIFO values

CAN_FilterFIFO	Description
CAN_FilterFIFO0	Filter FIFO 0 assignment for filter x
CAN_FilterFIFO1	Filter FIFO 1assignment for filter x

CAN_FilterActivation

CAN_FilterActivation enables or disables the filter. It can be set either to ENABLE or DISABLE.

Example:

```
/* Initialize the CAN filter 2 */
CAN_FilterInitTypeDef CAN_FilterInitStructure;

CAN_FilterInitStructure.CAN_FilterNumber = 2;
CAN_FilterInitStructure.CAN_FilterMode = CAN_FilterMode_IdMask;
CAN_FilterInitStructure.CAN_FilterScale = CAN_FilterScale_One32bit;
CAN_FilterInitStructure.CAN_FilterIdHigh = 0x0F0F;
CAN_FilterInitStructure.CAN_FilterIdLow = 0xF0F0;
CAN_FilterInitStructure.CAN_FilterMaskIdHigh = 0xFF00;
CAN_FilterInitStructure.CAN_FilterMaskIdLow = 0x00FF;
CAN_FilterInitStructure.CAN_FilterFIFO = CAN_FilterFIFO0;
CAN_FilterInitStructure.CAN_FilterActivation = ENABLE;
CAN_FilterInit(&CAN_InitStructure);
```

6.2.4 CAN_StructInit function

Table 82 describes the CAN_StructInit function.

Table 82. CAN_StructInit function

Function name	CAN_StructInit
Function prototype	void CAN_StructInit(CAN_InitTypeDef* CAN_InitStruct)
Behavior description	Fills each CAN_InitStruct member with its default value.
Input parameter	CAN_InitStruct: pointer to a CAN_InitTypeDef structure which will be initialized. Refer to *Table 83* for the default values of the CAN_InitStruct members.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 83. CAN_InitStruct default values

Member	Default value
CAN_TTCM	DISABLE
CAN_ABOM	DISABLE
CAN_AWUM	DISABLE
CAN_NART	DISABLE
CAN_RFLM	DISABLE
CAN_TXFP	DISABLE
CAN_Mode	CAN_Mode_Normal
CAN_SJW	CAN_SJW_1tq
CAN_BS1	CAN_BS1_4tq
CAN_BS2	CAN_BS2_3tq
CAN_Prescaler	1

Example:

```
/* Initialize a CAN_InitTypeDef structure. */
CAN_InitTypeDef  CAN_InitStructure;
CAN_StructInit(&CAN_InitStructure);
```

6.2.5 CAN_ITConfig function

Table 84 describes the CAN_ITConfig function.

Table 84. CAN_ITConfig function

Function name	CAN_ITConfig
Function prototype	void CAN_ITConfig(u32 CAN_IT, FunctionalState NewState)
Behavior description	Enables or disables the specified CAN interrupts.
Input parameter1	CAN_IT: CAN interrupt sources to be enabled or disabled. Refer to *CAN_IT* for details on the allowed values for this parameter.
Input parameter2	NewState: new state of the CAN interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

CAN_IT

The CAN_IT input parameter enables or disables CAN interrupts. One or a combination of the following values can be used:

Table 85. CAN_IT values

CAN_IT	Description
CAN_IT_TME	Transmit Mailbox Empty Mask
CAN_IT_FMP0	FIFO 0 Message Pending Mask
CAN_IT_FF0	FIFO 0 Full Mask
CAN_IT_FOV0	FIFO 0 Overrun Mask
CAN_IT_FMP1	FIFO 1 Message Pending Mask
CAN_IT_FF1	FIFO 1 Full Mask
CAN_IT_FOV1	FIFO 1 Overrun Mask
CAN_IT_EWG	Error Warning Mask
CAN_IT_EPV	Error Passive Mask
CAN_IT_BOF	Bus-Off Mask
CAN_IT_LEC	Last Error Code Mask
CAN_IT_ERR	Error Mask
CAN_IT_WKU	Wake-Up Mask
CAN_IT_SLK	Sleep Flag Mask

Example:

```
/* Enable CAN FIFO 0 overrun interrupt */
CAN_ITConfig(CAN_IT_FOV0, ENABLE);
```

6.2.6 CAN_Transmit function

Table 86 describes the CAN_Transmit function.

Table 86. CAN_Transmit function

Function name	CAN_Transmit
Function prototype	u8 CAN_Transmit(CanTxMsg* TxMessage)
Behavior description	Initiates the transmission of a message.
Input parameter	TxMessage: pointer to a structure which contains CAN Id, CAN DLC and CAN data.
Output parameter	None
Return parameter	Number of the mailbox that is used for transmission or CAN_NO_MB if there is no empty mailbox.
Required preconditions	None
Called functions	None

CanTxMsg

The CanTxMsg structure is defined in the *stm32f10x_can.h* file:

```
typedef struct
{
  u32 StdId;
  u32 ExtId;
  u8 IDE;
  u8 RTR;
  u8 DLC;
  u8 Data[8];
} CanTxMsg;
```

StdId

StdId is used to configure the standard identifier. This member ranges from 0 to 0x7FF.

ExtId

ExtId is used to configure the extended identifier. This member ranges from 0 to 0x1FFF FFFF.

IDE

IDE is used to configure the type of identifier for the message that will be transmitted. See *Table 87* for the values taken by this member.

Table 87. IDE values

IDE	Description
CAN_ID_STD	standard ID used
CAN_ID_EXT	extended ID used

RTR

RTR is used to select the type of frame for the message that will be transmitted. It can be set either to data frame or remote frame.

Table 88. RTR values

RTR	Description
CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame

DLC

DLC is used to configure the length of the frame that will be transmitted. It ranges from 0 to 0x8.

Data[8]

Data[8] contain the data to be transmitted. It ranges from 0 to 0xFF.

Example:

```
/* Send a message with the CAN */
CanTxMsg TxMessage;

TxMessage.StdId = 0x1F;
TxMessage.ExtId = 0x00;
TxMessage.IDE = CAN_ID_STD;
TxMessage.RTR = CAN_RTR_DATA;
TxMessage.DLC = 2;
TxMessage.Data[0] = 0xAA;
TxMessage.Data[1] = 0x55;
CAN_Transmit(&TxMessage);
```

6.2.7 CAN_TransmitStatus function

Table 89 describes the CAN_TransmitStatus function.

Table 89. CAN_TransmitStatus function

Function name	CAN_Transmit
Function prototype	u8 CAN_TransmitStatus(u8 TransmitMailbox)
Behavior description	Checks message transmission status
Input parameter	TransmitMailbox: the number of the mailbox that is used for the transmission.
Output parameter	None
Return parameter	CANTXOK if the CAN driver is transmitting the message CANTXPENDING if the message is pending CANTXFAILED otherwise
Required preconditions	Transmission ongoing
Called functions	None

Example:

```
/* Check the status of a transmission with the CAN */
CanTxMsg TxMessage;
...
switch(CAN_TransmitStatus(CAN_Transmit(&TxMessage))
{
case CANTXOK: ...;break;
...
}
```

6.2.8 CAN_CancelTransmit function

Table 90 describes the CAN_CancelTransmit function.

Table 90. CAN_CancelTransmit function

Function name	CAN_CancelTransmit
Function prototype	void CAN_CancelTransmit(u8 Mailbox)
Behavior description	Cancels a transmission request
Input parameter	Mailbox number
Output parameter	None
Return parameter	None
Required preconditions	Transmission pending in a mailbox
Called functions	None

Example:

```
/* Cancel a CAN transmit initiates by CANTransmit */
u8 MBNumber;
CanTxMsg TxMessage;
MBNumber = CAN_Transmit(&TxMessage);
if (CAN_TransmitStatus(MBNumber) == CANTXPENDING)
{
  CAN_CancelTransmit(MBNumber);
}
```

6.2.9 CAN_FIFORelease function

Table 91 describes the CAN_FIFORelease function.

Table 91. CAN_FIFORelease function

Function name	CAN_FIFORelease
Function prototype	void CAN_FIFORelease(u8 FIFONumber)
Behavior description	Releases a FIFO
Input parameter	FIFO number: FIFO to release, CANFIFO0 or CANFIFO1.
Output parameter	None
Return parameter	None
Required preconditions	none
Called functions	None

Example:

```
/* Release FIFO 0*/
CAN_FIFORelease(CANFIFO0);
```

6.2.10 CAN_MessagePending function

Table 92 describes the CAN_MessagePending function.

Table 92. CAN_MessagePending function

Function name	CAN_MessagePending
Function prototype	u8 CAN_MessagePending(u8 FIFONumber)
Behavior description	Return the number of pending messages.
Input parameter	FIFONumber: receive FIFO number, CANFIFO0 or CANFIFO1.
Output parameter	None
Return parameter	NbMessage which is the number of pending messages
Required preconditions	none
Called functions	None

Example:

```
/* Check the number of pending messages for FIFO 0*/
u8 MessagePending = 0;
MessagePending = CAN_MessagePending(CANFIFO0);
```

6.2.11 CAN_Receive function

Table 93 describes the CAN_Receive function.

Table 93. CAN_Receive function

Function name	CAN_Receive
Function prototype	void CAN_Receive(u8 FIFONumber, CanRxMsg* RxMessage)
Behavior description	Receives a message.
Input parameter	FIFONumber: receive FIFO number, CANFIFO0 or CANFIFO1.
Output parameter	RxMessage: pointer to a structure which contains CAN Id, CAN DLC and CAN data.
Return parameter	None
Required preconditions	None
Called functions	None

CanRxMsg structure

The CanRxMsg structure is defined in the *stm32f10x_can.h* file:

```
typedef struct
{
  u32 StdId;
  u32 ExtId;
  u8 IDE;
  u8 RTR;
  u8 DLC;
  u8 Data[8];
  u8 FMI;
} CanRxMsg;
```

StdId

StdId is used to configure the standard identifier. This member ranges from 0 to 0x7FF.

ExtId

ExtId is used to configure the extended identifier. This member ranges from 0 to 0x1FFF FFFF.

IDE

IDE is used to configure the type of identifier for the message that will be received. See *Table 87* for the values taken by this member.

Table 94. IDE values

IDE	Description
CAN_ID_STD	standard ID used
CAN_ID_EXT	extended ID used

RTR

RTR is used to select the type of frame for the received message. It can be set either to data frame or remote frame.

Table 95. RTR values

RTR	Description
CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame

DLC

DLC is used to configure the length of the frame that will be transmitted. It ranges from 0 to 0x8.

Data[8]

Data[8] contains the data to be received. It ranges from 0 to 0xFF.

FMI

FMI configures the index of the filter the message stored in the mailbox passes through. FMI ranges from 0 to 0xFF.

Example:

```
/* Receive a message with the CAN */
CanRxMsg RxMessage;
CAN_Receive(&RxMessage);
```

6.2.12 CAN_Sleep function

Table 96 describes the CAN_Sleep function.

Table 96. CAN_Sleep function

Function name	CAN_Sleep
Function prototype	u8 CAN_Sleep(void)
Behavior description	Put the CAN in low power mode.
Input parameter	None
Output parameter	None
Return parameter	CANSLEEPOK if sleep entered, CANSLEEPFAILED otherwisedata.
Required preconditions	None
Called functions	None

Example:

```
/* Enter the CAN sleep mode*/
CAN_Sleep();
```

6.2.13 CAN_WakeUp function

Table 97 describes the CAN_Wakeup function.

Table 97. CAN_Wakeup function

Function name	CAN_WakeUp
Function prototype	u8 CAN_WakeUp(void)
Behavior description	Wakes up the CAN.
Input parameter	None
Output parameter	None
Return parameter	CANWAKEUPOK if sleep mode left, CANWAKEUPFAILED otherwise.
Required preconditions	None
Called functions	None

Example:

```
/* CAN waking up */
CAN_WakeUp();
```

6.2.14 CAN_GetFlagStatus function

Table 98 describes the CAN_GetFlagStatus function.

Table 98. CAN_GetFlagStatus function

Function name	CAN_GetFlagStatus
Function prototype	FlagStatus CAN_GetFlagStatus(u32 CAN_FLAG)
Behavior description	Checks whether the specified CAN flag is set or not.
Input parameter	CAN_FLAG: it specifies the flag to be checked. Refer to *CAN_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of CAN_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

CAN_FLAG

The CAN_FLAG is used to define the type of flag that will be checked. See *Table 99* for a description of CAN_FLAG values.

Table 99. CAN_FLAG definition

CAN_FLAG	Description
CAN_FLAG_EWG	Error Warning Flag
CAN_FLAG_EPV	Error Passive Flag
CAN_FLAG_BOF	Bus-Off Flag

Example:

```
/* Test if the CAN warning limit has been reached */
FlagStatus Status;
Status = CAN_GetFlagStatus(CAN_FLAG_EWG);
```

6.2.15 CAN_ClearFlag function

Table 100 describes the CAN_ClearFlag function.

Table 100. CAN_ClearFlag function

Function name	CAN_ClearFlag
Function prototype	void CAN_ClearFlag(u32 CAN_Flag)
Behavior description	Clears the CAN's pending flags.
Input parameter	CAN_FLAG specifies the flag to clear. Refer to *CAN_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the CAN bus-off state flag */
CAN_ClearFlag(CAN_FLAG_BOF);
```

6.2.16 CAN_GetITStatus function

Table 101 describes the CAN_GetITStatus function.

Table 101. CAN_GetITStatus function

Function name	CAN_GetITStatus
Function prototype	`ITStatus CAN_GetITStatus(u32 CAN_IT)`
Behavior description	Checks whether the specified CAN interrupt has occurred or not.
Input parameter	CAN_IT: CAN interrupt source to check. Refer to *CAN_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of CAN_IT (SET or RESET).
Required preconditions	None
Called functions	None

CAN_IT

The CAN_IT input parameter selects the interrupt that will be checked. See *Table 102* for a description of CAN_IT values.

Table 102. CAN_IT values

CAN_IT	Description
CAN_IT_RQCP0	Request completed mailbox 0
CAN_IT_RQCP1	Request completed mailbox 1
CAN_IT_RQCP2	Request completed mailbox 2
CAN_IT_FMP0	FIFO 0 Message Pending
CAN_IT_FULL0	FIFO 0 three messages stored
CAN_IT_FOVR0	FIFO 0 Overrun
CAN_IT_FMP1	FIFO 1 Message Pending
CAN_IT_FULL1	FIFO 1 three messages stored
CAN_IT_FOVR1	FIFO 1 Overrun
CAN_IT_EWGF	Warning limit reached
CAN_IT_EPVF	Error passive limit reached
CAN_IT_BOFF	Bus-of state entered
CAN_IT_WKUI	SOF detected whilst in sleep mode

Example:

```
/* Test if the CAN FIFO 0 overrun interrupt has occurred or not */
ITStatus Status;
Status = CAN_GetITStatus(CAN_IT_FOVR0);
```

6.2.17 CAN_ClearITPendingBit function

Table 103 describes the CAN_ClearITPendingBit function.

Table 103. CAN_ClearITPendingBit function

Function name	CAN_ClearITPendingBit
Function prototype	void CAN_ClearITPendingBit(u32 CAN_IT)
Behavior description	Clears the CAN pending interrupt bits.
Input parameter	CAN_IT: pending interrupt bit to clear. Refer to *CAN_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the CAN error passive overflow interrupt pending bit */
CAN_ClearITPendingBit(CAN_IT_EPVF);
```

7 DMA controller (DMA)

The DMA controller provides access to twelve data channels. Since peripherals are memory mapped, data transfers from/to peripherals are managed like memory/memory data transfers.

Section 7.1: DMA register structures describes the data structures used in the DMA Firmware Library. *Section 7.2: Firmware library functions* presents the Firmware Library functions.

7.1 DMA register structures

The DMA register structures, *DMA_Channel_TypeDef and DMA_TypeDef,* are defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 CCR;
  vu32 CNDTR;
  vu32 CPAR;
  vu32 CMAR;
} DMA_Channel_TypeDef;

typedef struct
{
  vu32 ISR;
  vu32 IFCR;
} DMA_TypeDef;
```

Table 104 shows the list of all DMA registers.

Table 104. DMA registers

Register	Description
ISR	DMA Interrupt Status register
IFCR	DMA Interrupt Flag Clear Register
CCRx	DMA Channelx Configuration register
CNDTRx	DMA Channelx Number of Data to Transfer register
CPARx	DMA Channelx Peripheral Address Register
CMARx	DMA Channelx Memory0 Address Register

The DMA and its seven channels are also declared in *stm32f10x_map*:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
....
#define DMA1_BASE              (AHBPERIPH_BASE + 0x0000)
```

```
#define DMA2_BASE             (AHBPERIPH_BASE + 0x0400)

#define DMA1_Channel1_BASE      (AHBPERIPH_BASE + 0x0008)
#define DMA1_Channel2_BASE      (AHBPERIPH_BASE + 0x001C)
#define DMA1_Channel3_BASE      (AHBPERIPH_BASE + 0x0030)
#define DMA1_Channel4_BASE      (AHBPERIPH_BASE + 0x0044)
#define DMA1_Channel5_BASE      (AHBPERIPH_BASE + 0x0058)
#define DMA1_Channel6_BASE      (AHBPERIPH_BASE + 0x006C)
#define DMA1_Channel7_BASE      (AHBPERIPH_BASE + 0x0080)
#define DMA2_Channel1_BASE      (AHBPERIPH_BASE + 0x0408)
#define DMA2_Channel2_BASE      (AHBPERIPH_BASE + 0x041C)
#define DMA2_Channel3_BASE      (AHBPERIPH_BASE + 0x0430)
#define DMA2_Channel4_BASE      (AHBPERIPH_BASE + 0x0444)
#define DMA2_Channel5_BASE      (AHBPERIPH_BASE + 0x0458)
....
#ifndef DEBUG
...
#ifdef _DMA
   #define DMA1               ((DMA_TypeDef *) DMA1_BASE)
   #define DMA2               ((DMA_TypeDef *) DMA2_BASE)
#endif /*_DMA */

#ifdef _DMA1_Channel1
  #define DMA1_Channel1         ((DMA_Channel_TypeDef *)
DMA1_Channel1_BASE)
#endif /*_DMA1_Channel1 */

#ifdef _DMA1_Channel2
  #define DMA1_Channel2         ((DMA_Channel_TypeDef *)
DMA1_Channel2_BASE)
#endif /*_DMA1_Channel2 */

#ifdef _DMA1_Channel3
  #define DMA1_Channel3         ((DMA_Channel_TypeDef *)
DMA1_Channel3_BASE)
#endif /*_DMA1_Channel3 */

#ifdef _DMA1_Channel4
  #define DMA1_Channel4         ((DMA_Channel_TypeDef *)
DMA1_Channel4_BASE)
#endif /*_DMA1_Channel4 */

#ifdef _DMA1_Channel5
  #define DMA1_Channel5         ((DMA_Channel_TypeDef *)
DMA1_Channel5_BASE)
#endif /*_DMA1_Channel5 */

#ifdef _DMA1_Channel6
  #define DMA1_Channel6         ((DMA_Channel_TypeDef *)
DMA1_Channel6_BASE)
#endif /*_DMA1_Channel6 */
```

```
#ifdef _DMA1_Channel7
  #define DMA1_Channel7        ((DMA_Channel_TypeDef *)
DMA1_Channel7_BASE)
#endif /*_DMA1_Channel7 */

#ifdef _DMA2_Channel1
  #define DMA2_Channel1        ((DMA_Channel_TypeDef *)
DMA2_Channel1_BASE)
#endif /*_DMA2_Channel1 */

#ifdef _DMA2_Channel2
  #define DMA2_Channel2        ((DMA_Channel_TypeDef *)
DMA12_Channel2_BASE)
#endif /*_DMA2_Channel2 */

#ifdef _DMA2_Channel3
  #define DMA2_Channel3        ((DMA_Channel_TypeDef *)
DMA2_Channel3_BASE)
#endif /*_DMA2_Channel3 */

#ifdef _DMA2_Channel4
  #define DMA2_Channel4        ((DMA_Channel_TypeDef *)
DMA2_Channel4_BASE)
#endif /*_DMA2_Channel4 */

#ifdef _DMA2_Channel5
  #define DMA2_Channel5        ((DMA_Channel_TypeDef *)
DMA2_Channel5_BASE)
#endif /*_DMA2_Channel5 */

...
#else   /* DEBUG */
...
#ifdef _DMA
   EXT DMA_TypeDef             *DMA1;
   EXT DMA_TypeDef             *DMA2;
#endif /*_DMA */

#ifdef _DMA1_Channel1
  EXT DMA1_Channel_TypeDef     *DMA1_Channel1;
#endif /*_DMA1_Channel1 */

#ifdef _DMA1_Channel2
  EXT DMA1_Channel_TypeDef     *DMA1_Channel2;
#endif /*_DMA1_Channel2 */

#ifdef _DMA1_Channel3
  EXT DMA1_Channel_TypeDef     *DMA1_Channel3;
#endif /*_DMA1_Channel3 */

#ifdef _DMA1_Channel4
  EXT DMA1_Channel_TypeDef     *DMA1_Channel4;
```

```
#endif /*_DMA1_Channel4 */

#ifdef _DMA1_Channel5
  EXT DMA1_Channel_TypeDef     *DMA1_Channel5;
#endif /*_DMA1_Channel5 */

#ifdef _DMA1_Channel6
  EXT DMA1_Channel_TypeDef     *DMA1_Channel6;
#endif /*_DMA1_Channel6 */

#ifdef _DMA1_Channel7
  EXT DMA1_Channel_TypeDef     *DMA1_Channel7;
#endif /*_DMA1_Channel7 */

#ifdef _DMA2_Channel1
  EXT DMA2_Channel_TypeDef     *DMA2_Channel1;
#endif /*_DMA2_Channel1 */

#ifdef _DMA2_Channel2
  EXT DMA2_Channel_TypeDef     *DMA2_Channel2;
#endif /*_DMA2_Channel2 */

#ifdef _DMA2_Channel3
  EXT DMA2_Channel_TypeDef     *DMA2_Channel3;
#endif /*_DMA2_Channel3 */

#ifdef _DMA2_Channel4
  EXT DMA2_Channel_TypeDef     *DMA2_Channel4;
#endif /*_DMA2_Channel4 */

#ifdef _DMA2_Channel5
  EXT DMA2_Channel_TypeDef     *DMA2_Channel5;
#endif /*_DMA2_Channel5 */
...
#endif
```

When using the Debug mode, _DMA, _DMA1_Channel1, _DMA1_Channel2, ..., _DMA1_Channel7, _DMA2_Channel1, _DMA2_Channel2, ..., _DMA2_Channel5 pointers are initialized in *stm32f10x_lib.c* file:

```
...
#ifdef _DMA
   DMA1 = (DMA_TypeDef *)  DMA1_BASE;
   DMA2 = (DMA_TypeDef *)  DMA2_BASE;
#endif /*_DMA */

#ifdef _DMA1_Channel1
  DMA1_Channel1 = (DMA_Channel_TypeDef *)  DMA1_Channel1_BASE;
#endif /*_DMA1_Channel1 */

#ifdef _DMA1_Channel2
  DMA1_Channel2 = (DMA_Channel_TypeDef *)  DMA1_Channel2_BASE;
#endif /*_DMA1_Channel2 */
```

```
#ifdef _DMA1_Channel3
  DMA1_Channel3 = (DMA_Channel_TypeDef *)  DMA1_Channel3_BASE;
#endif /*_DMA1_Channel3 */

#ifdef _DMA1_Channel4
  DMA1_Channel4 = (DMA_Channel_TypeDef *)  DMA1_Channel4_BASE;
#endif /*_DMA1_Channel4 */

#ifdef _DMA1_Channel5
  DMA1_Channel5 = (DMA_Channel_TypeDef *)  DMA1_Channel5_BASE;
#endif /*_DMA1_Channel5 */

#ifdef _DMA1_Channel6
  DMA1_Channel6 = (DMA_Channel_TypeDef *)  DMA1_Channel6_BASE;
#endif /*_DMA1_Channel6 */

#ifdef _DMA1_Channel7
  DMA1_Channel7 = (DMA_Channel_TypeDef *)  DMA1_Channel7_BASE;
#endif /*_DMA1_Channel7 */

#ifdef _DMA2_Channel1
  DMA2_Channel1 = (DMA_Channel_TypeDef *)  DMA2_Channel1_BASE;
#endif /*_DMA2_Channel1 */

#ifdef _DMA2_Channel2
  DMA2_Channel2 = (DMA_Channel_TypeDef *)  DMA2_Channel2_BASE;
#endif /*_DMA2_Channel2 */

#ifdef _DMA2_Channel3
  DMA2_Channel3 = (DMA_Channel_TypeDef *)  DMA2_Channel3_BASE;
#endif /*_DMA2_Channel3 */

#ifdef _DMA2_Channel4
  DMA2_Channel4 = (DMA_Channel_TypeDef *)  DMA2_Channel4_BASE;
#endif /*_DMA2_Channel4 */

#ifdef _DMA2_Channel5
  DMA2_Channel5 = (DMA_Channel_TypeDef *)  DMA2_Channel5_BASE;
#endif /*_DMA2_Channel5 */

...
To access the DMA registers, _DMA, _DMA1_Channel1 to _DMA1_Channel7
and _DMA2_Channel1 to _DMA2_Channel5 must be defined in
stm32f10x_conf.h as follows:
...
#define _DMA
#define _DMA1_Channel1
#define _DMA1_Channel2
#define _DMA1_Channel3
#define _DMA1_Channel4
#define _DMA1_Channel5
```

```
#define _DMA1_Channel6
#define _DMA1_Channel7
#define _DMA2_Channel1
#define _DMA2_Channel2
#define _DMA2_Channel3
#define _DMA2_Channel4
#define _DMA2_Channel5

...
```

7.2 Firmware library functions

Table 105 lists the various functions of the DMA firmware library.

Table 105. DMA firmware library functions

Function name	Description
DMA_DeInit	Resets the DMAy Channelx registers to their default reset values.
DMA_Init	Initializes the DMAy Channelx according to the specified parameters in the DMA_InitStruct.
DMA_StructInit	Fills each DMA_InitStruct member with its default value.
DMA_Cmd	Enables or disables the specified DMAy Channelx.
DMA_ITConfig	Enables or disables the specified DMAy Channelx interrupts.
DMA_GetCurrDataCounter	Returns the number of remaining data units in the current DMAy Channelx transfer.
DMA_GetFlagStatus	Checks whether the specified DMAy Channelx flag is set or not.
DMA_ClearFlag	Clears the DMAy Channelx pending flags.
DMA_GetITStatus	Checks whether the specified DMAy Channelx interrupt has occurred or not.
DMA_ClearITPendingBit	Clears the DMAy Channelx interrupt pending bits.

7.2.1 DMA_DeInit function

Table 106 describes the DMA_DeInit function.

Table 106. DMA_DeInit function

Function name	DMA_DeInit
Function prototype	void DMA_DeInit(DMA_Channel_TypeDef* DMAy_Channelx)
Behavior description	Resets the DMAy Channelx registers to their default reset values.
Input parameter	DMAy_Channel*x*: where y selects the DMA (y = 1 for DMA1, y = 2 for DMA2) and x selects the DMA Channel (x = 1 to 7 for DMA1 or x = 1 to 5 for DMA2).
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_AHBPeriphClockCmd().

Example:

```
/* Deinitialize the DMA1 Channel2 */
DMA_DeInit(DMA1_Channel2);
```

7.2.2 DMA_Init function

Table 107 describes the DMA_Init function.

Table 107. DMA_Init function

Function name	DMA_Init
Function prototype	void DMA_Init(DMA_Channel_TypeDef* DMAy_Channelx, DMA_InitTypeDef* DMA_InitStruct)
Behavior description	Initializes the DMAy Channelx according to the parameters specified in the DMA_InitStruct.
Input parameter1	DMAy_Channel*x*: where y selects the DMA (y = 1 for DMA1, y = 2 for DMA2) and x selects the DMA Channel (x = 1 to 7 for DMA1 or x = 1 to 5 for DMA2).
Input parameter2	DMA_InitStruct: pointer to a DMA_InitTypeDef structure that contains the configuration information for the specified DMAy Channelx. Refer to *DMA_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

DMA_InitTypeDef structure

The DMA_InitTypeDef structure is defined in the *stm32f10x_dma.h* file:

```
typedef struct
{
  u32 DMA_PeripheralBaseAddr;
  u32 DMA_MemoryBaseAddr;
  u32 DMA_DIR;
  u32 DMA_BufferSize;
  u32 DMA_PeripheralInc;
  u32 DMA_MemoryInc;
  u32 DMA_PeripheralDataSize;
  u32 DMA_MemoryDataSize;
  u32 DMA_Mode;
  u32 DMA_Priority;
  u32 DMA_M2M;
} DMA_InitTypeDef;
```

DMA_PeripheralBaseAddr

This member is used to define the peripheral base address for DMAy Channelx.

DMA_MemoryBaseAddr

This member is used to define the memory base address for DMAy Channelx.

DMA_DIR

DMA_DIR specifies if the peripheral is the source or destination. The values taken by this member are given in *Table 108*.

Table 108. DMA_DIR definition

DMA_DIR	Description
DMA_DIR_PeripheralDST	Peripheral is the destination
DMA_DIR_PeripheralSRC	Peripheral is the source

DMA_BufferSize

DMA_BufferSize is used to define 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.

DMA_PeripheralInc

DMA_Peripherallinc specifies whether the Peripheral address register is incremented or not. The values taken by this member are given in *Table 109*.

Table 109. DMA_PeripheralInc definition

DMA_PeripheralInc	Description
DMA_PeripheralInc_Enable	Current peripheral register incremented
DMA_PeripheralInc_Disable	Current peripheral register unchanged

DMA_MemoryInc

DMA_MemoryInc specifies whether the memory address register is incremented or not. The values taken by this member are given in *Table 110*.

Table 110. DMA_MemoryInc definition

DMA_MemoryInc	Description
DMA_MemoryInc_Enable	Current memory register incremented
DMA_MemoryInc_Disable	Current memory register unchanged

DMA_PeripheralDataSize

DMA_PeripheralDataSize configures the Peripheral data width. The values taken by this member are given in *Table 111*.

Table 111. DMA_PeripheralDataSize definition

DMA_PeripheralDataSize	Description
DMA_PeripheralDataSize_Byte	Data width = 8 bits
DMA_PeripheralDataSize_HalfWord	Data width = 16 bits
DMA_PeripheralDataSize_Word	Data width = 32 bits

DMA_MemoryDataSize

DMA_MemoryDataSize defines the Memory data width. The values taken by this member are given in *Table 112*.

Table 112. DMA_MemoryDataSize definition

DMA_MemoryDataSize	Description
DMA_MemoryDataSize_Byte	Data width = 8 bits
DMA_MemoryDataSize_HalfWord	Data width = 16 bits
DMA_MemoryDataSize_Word	Data width = 32 bits

DMA_Mode

DMA_Mode configures the operation mode of the DMAy Channelx. The values taken by this member are given in *Table 113*.

Table 113. DMA_Mode definition

DMA_Mode	Description
DMA_Mode_Circular	Circular buffer mode is used
DMA_Mode_Normal	Normal buffer mode is used

Note: The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel (see DMA_M2M).

DMA_Priority

DMA_Priority configures the software priority for the DMAy Channelx. The values taken by this member are given in *Table 114*.

Table 114. DMA_Priority definition

DMA_Priority	Description
DMA_Priority_VeryHigh	DMAy Channelx has a very high priority
DMA_Priority_High	DMAy Channelx has a high priority
DMA_Priority_Medium	DMAy Channelx has a medium priority
DMA_Priority_Low	DMAy Channelx has a low priority

DMA_M2M

DMA_M2M enables the DMAy Channelx memory- to-memory transfer. The values taken by this member are given in *Table 115*.

Table 115. DMA_M2M definition

DMA_M2M	Description
DMA_M2M_Enable	DMAy Channelx configured for memory-to-memory transfer
DMA_M2M_Disable	DMAy Channelx not configured for memory-to-memory transfer

Example:

```
/* Initialize the DMA1 Channel1 according to the DMA_InitStructure
members */
  DMA_InitTypeDef  DMA_InitStructure;

  DMA_InitStructure.DMA_PeripheralBaseAddr = 0x40005400;
  DMA_InitStructure.DMA_MemoryBaseAddr = 0x20000100;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_InitStructure.DMA_BufferSize = 256;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructure.DMA_PeripheralDataSize =
  DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize =
  DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
  DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
  DMA_Init(DMA1_Channel1, &DMA_InitStructure);
```

7.2.3 DMA_StructInit function

Table 116 describes the DMA_Init function.

Table 116. DMA_StructInit function

Function name	DMA_StructInit
Function prototype	void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct)
Behavior description	Fills each DMA_InitStruct member with its default value.
Input parameter	DMA_InitStruct: pointer to the DMA_InitTypeDef structure to be initialized
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The DMA_InitStruct members have the following default values:

Table 117. DMA_InitStruct default values

Member	Default value
DMA_PeripheralBaseAddr	0
DMA_MemoryBaseAddr	0
DMA_DIR	DMA_DIR_PeripheralSRC
DMA_BufferSize	0
DMA_PeripheralInc	DMA_PeripheralInc_Disable
DMA_MemoryInc	DMA_MemoryInc_Disable
DMA_PeripheralDataSize	DMA_PeripheralDataSize_Byte
DMA_MemoryDataSize	DMA_MemoryDataSize_Byte
DMA_Mode	DMA_Mode_Normal
DMA_Priority	DMA_Priority_Low
DMA_M2M	DMA_M2M_Disable

Example:

```
/* Initialize a DMA_InitTypeDef structure */
DMA_InitTypeDef  DMA_InitStructure;
DMA_StructInit(&DMA_InitStructure);
```

7.2.4 DMA_Cmd function

Table 118 describes DMA_Cmd function.

Table 118. DMA_Cmd function

Function name	DMA_Cmd
Function prototype	void DMA_Cmd(DMA_Channel_TypeDef* DMAy_Channelx, FunctionalState NewState)
Behavior description	Enables or disables the specified DMAy Channelx.
Input parameter1	DMAy_Channel*x*: where y selects the DMA (y = 1 for DMA1, y = 2 for DMA2) and x selects the DMA Channel (x = 1 to 7 for DMA1 or x = 1 to 5 for DMA2).
Input parameter2	NewState: new state of the DMAy Channelx. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable DMA1 Channel7 */
DMA_Cmd(DMA1_Channel7, ENABLE);
```

7.2.5 DMA_ITConfig function

Table 119 describes DMA_ITConfig function.

Table 119. DMA_ITConfig function

Function name	DMA_ITConfig
Function prototype	void DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, u32 DMA_IT, FunctionalState NewState)
Behavior description	Enables or disables the specified DMAy Channelx interrupts.
Input parameter1	DMAy_Channel*x*: where y selects the DMA (y = 1 for DMA1, y = 2 for DMA2) and x selects the DMA Channel (x = 1 to 7 for DMA1 or x = 1 to 5 for DMA2).
Input parameter2	DMA_IT: specifies the DMAy Channelx interrupt sources to be enabled or disabled. More than one interrupt can be selected using the “|” operator. Refer to *DMA_IT* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the specified DMAy Channelx interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

DMA_IT

The DMA_IT input parameter enables or disables DMAy Channelx interrupts. One or a combination of the following values can be used.

Table 120. DMA_IT values

DMA_IT	Description
DMA_IT_TC	Transfer complete interrupt mask
DMA_IT_HT	Half transfer interrupt mask
DMA_IT_TE	Transfer error interrupt mask

Example:

```
/* Enable DMA1 Channel5 complete transfer interrupt */
DMA_ITConfig(DMA1_Channel5, DMA_IT_TC, ENABLE);
```

7.2.6 DMA_GetCurrDataCounter function

Table 121 describes DMA_GetCurrDataCounter function.

Table 121. DMA_GetCurrDataCounter function

Function name	DMA_GetCurrDataCounter
Function prototype	u16 DMA_GetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx)
Behavior description	Returns the number of remaining data units in the current DMAy Channelx transfer.
Input parameter	DMAy_Channelx: where y selects the DMA (y = 1 for DMA1, y = 2 for DMA2) and x selects the DMA Channel (x = 1 to 7 for DMA1 or x = 1 to 5 for DMA2).
Output parameter	None
Return parameter	The number of remaining data units in the current DMAy Channelx transfer.
Required preconditions	None
Called functions	None

Example:

```
/* Get  the number of remaining data units in the current DMA1
Channel2 transfer */
u16 CurrDataCount;
CurrDataCount = DMA_GetCurrDataCounter(DMA1_Channel2);
```

7.2.7 DMA_GetFlagStatus function

Table 122 describes DMA_GetFlagStatus function.

Table 122. DMA_GetFlagStatus function

Function name	DMA_GetFlagStatus
Function prototype	FlagStatus DMA_GetFlagStatus(u32 DMA_FLAG)
Behavior description	Checks whether the specified DMAy Channelx flag is set or not.
Input parameter	DMA_FLAG: specifies the flag to check. Refer to *DMA_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	New state of DMA_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

DMA_FLAG

The DMA_FLAG is used to define the type of flag that will be checked. See *Table 123* for a description of this input parameter.

Table 123. DMA_FLAG definition

DMA_FLAG	Description
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

Table 123. DMA_FLAG definition (continued)

DMA_FLAG	Description
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

Example:

```
/* Test if the DMA1 Channel6 half transfer interrupt flag is set or
not */
FlagStatus Status;
Status = DMA_GetFlagStatus(DMA1_FLAG_HT6);
```

7.2.8 DMA_ClearFlag function

Table 124 describes DMA_ClearFlag function.

Table 124. DMA_ClearFlag function

Function name	DMA_ClearFlag
Function prototype	void DMA_ClearFlag(u32 DMA_FLAG)
Behavior description	Clears the DMAy Channelx's pending flags.
Input parameter	DMA_FLAG: flag to be cleared. More than one flag can be cleared using the “|” operator. Refer to *DMA_FLAG* for more details on the allowed values for this parameter. The user can select more than one flag, by ‘ORing’ them.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the DMA1 Channel3 transfer error interrupt pending bit */
DMA_ClearFlag(DMA1_FLAG_TE3);
```

7.2.9 DMA_GetITStatus function

Table 125 describes DMA_GetITStatus function.

Table 125. DMA_GetITStatus function

Function name	DMA_GetITStatus
Function prototype	ITStatus DMA_GetITStatus(u32 DMA_IT)
Behavior description	Checks whether the specified DMAy Channelx interrupt has occurred or not.
Input parameter	DMA_IT: DMAy Channelx interrupt source to check. Refer to *DMA_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of DMA_IT (SET or RESET).
Required preconditions	None
Called functions	None

DMA_IT

The DMA_IT selects the interrupt that will be checked. See *Table 126* for a description of this input parameter.

Table 126. DMA_IT values

DMA_IT	Description
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

Table 126. DMA_IT values (continued)

DMA_IT	Description
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

Example:

```
/* Test if the DMA1 Channel7 transfer complete interrupt has
occurred or not */
ITStatus Status;
Status = DMA_GetITStatus(DMA1_IT_TC7);
```

7.2.10 DMA_ClearITPendingBit function

Table 127 describes DMA_ClearITPendingBit function.

Table 127. DMA_ClearITPendingBit function

Function name	DMA_ClearITPending Bit
Function prototype	void DMA_ClearITPendingBit(u32 DMA_IT)
Behavior description	Clears the DMAy Channelx's interrupt pending bits.
Input parameter	DMA_IT: DMAy Channelx interrupt pending bit to clear.More than one interrupt can be cleared using the "|" operator. Refer to *DMA_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the DMA1 Channel5 global interrupt pending bit */
DMA_ClearITPendingBit(DMA1_IT_GL5);
```

8 External interrupt/event controller (EXTI)

The External interrupt/event controller (EXTI) consists of up to 19-edge detectors which are used to generate event/interrupt requests. Each input line can be independently configured to select the type (pulse or pending) and the corresponding trigger event (rising, falling or both). Each line can be masked independently. A pending register maintains the status of the interrupt requests.

Section 8.1: EXTI register structure describes the data structures used in the EXTI firmware library. *Section 8.2: Firmware library functions* presents the firmware library functions.

8.1 EXTI register structure

The EXTI register structure, EXTI_*TypeDef, is* defined in the *stm32f10xstm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 IMR;
  vu32 EMR;
  vu32 RTSR;
  vu32 FTSR;
  vu32 SWIER;
  vu32 PR;
} EXTI_TypeDef;
```

Table 128 shows the list of all EXTI registers.

Table 128. EXTI registers

Register	Description
IMR	Interrupt Mask Register
EMR	Event Mask Register
RTSR	Rising Trigger Selection Register
FTSR	Falling Trigger Selection Register
SWIR	Software Interrupt Event Register
PR	Pending Register

The EXTI peripheral is declared in the same file, as follows:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
...
#define EXTI_BASE             (APB2PERIPH_BASE + 0x0400)

#ifndef DEBUG
...
#ifdef _EXTI
  #define EXTI                ((EXTI_TypeDef *) EXTI_BASE)
#endif /*_EXTI */
```

```
...
#else   /* DEBUG */
...
#ifdef _EXTI
  EXT EXTI_TypeDef            *EXTI;
#endif /*_EXTI */
...
#endif
```

When using the Debug mode, EXTI pointer is initialized in *stm32f10x_lib.c* file:

```
#ifdef _EXTI
EXTI = (EXTI_TypeDef *)  EXTI_BASE;
#endif /*_EXTI */
```

To access the EXTI registers, _EXTI must be defined in *stm32f10x_conf.h* as follows:

```
#define _EXTI
```

8.2 Firmware library functions

Table 129 lists the various functions of the EXTI firmware library.

Table 129. EXTI Firmware library functions

Function name	Description
EXTI_DeInit	Resets the EXTI peripheral registers to their default reset values.
EXTI_Init	Initializes the EXTI peripheral according to the specified parameters in the EXTI_InitStruct.
EXTI_StructInit	Fills each EXTI_InitStruct member with its default value.
EXTI_GenerateSWInterrupt	Generates a software interrupt.
EXTI_GetFlagStatus	Checks whether the specified EXTI line flag is set or not.
EXTI_ClearFlag	Clears the EXTI's line pending flags.
EXTI_GetITStatus	Checks whether the specified EXTI line is asserted or not.
EXTI_ClearITPendingBit	Clears the EXTI's line pending bits.

8.2.1 EXTI_DeInit function

Table 130 describes the EXTI_DeInit function.

Table 130. EXTI_DeInit function

Function name	EXTI_DeInit
Function prototype	void EXTI_DeInit(void)
Behavior description	Resets the EXTI peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Resets the EXTI registers to their default reset value */
EXTI_DeInit();
```

8.2.2 EXTI_Init function

Table 131 describes the EXTI_DeInit function.

Table 131. EXTI_DeInit function

Function name	EXTI_Init
Function prototype	void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct)
Behavior description	Initializes the EXTI peripheral according to the parameters specified in the EXTI_InitStruct.
Input parameter	EXTI_InitStruct: pointer to a EXTI_InitTypeDef structure that contains the configuration information for the specified EXTI peripheral. Refer to *EXTI_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

EXTI_InitTypeDef structure

The EXTI_InitTypeDef structure is defined in *stm32f10x_exti.h*:

```
typedef struct
{
  u32 EXTI_Line;
  EXTIMode_TypeDef EXTI_Mode;
  EXTIrigger_TypeDef EXTI_Trigger;
  FunctionalState EXTI_LineCmd;
} EXTI_InitTypeDef;
```

EXTI_Line

EXTI_Line selects the external lines to be enabled or disabled. The values taken by this member are given in *Table 132*.

Table 132. EXTI_Line values

EXTI_Line	Description
EXTI_Line0	External interrupt line 0
EXTI_Line1	External interrupt line 1
EXTI_Line2	External interrupt line 2
EXTI_Line3	External interrupt line 3
EXTI_Line4	External interrupt line 4
EXTI_Line5	External interrupt line 5
EXTI_Line6	External interrupt line 6
EXTI_Line7	External interrupt line 7
EXTI_Line8	External interrupt line 8
EXTI_Line9	External interrupt line 9
EXTI_Line10	External interrupt line 10
EXTI_Line11	External interrupt line 11
EXTI_Line12	External interrupt line 12
EXTI_Line13	External interrupt line 13
EXTI_Line14	External interrupt line 14
EXTI_Line15	External interrupt line 15
EXTI_Line16	External interrupt line 16
EXTI_Line17	External interrupt line 17
EXTI_Line18	External interrupt line 18

EXTI_Mode

EXTI_Mode configures the mode for the enabled lines. The values taken by this member are given in *Table 133*.

Table 133. EXTI_Mode values

EXTI_Mode	Description
EXTI_Mode_Event	EXTI lines configured as event request
EXTI_Mode_Interrupt	EXTI lines configured as interrupt request

EXTI_Trigger

EXTI configures the trigger signal active edge for the enabled lines.The values taken by this member are given in *Table 134*.

Table 134. EXT_Trigger values

EXTI_Trigger	Description
EXTI_Trigger_Falling	Interrupt request configured on falling edge of the input line
EXTI_Trigger_Rising	Interrupt request configured on rising edge of the input line
EXTI_Trigger_Rising_Falling	Interrupt request configured on rising and falling edge of the input line

EXTI_LineCmd

This member is used to define the new state of the selected line. It can be set either to ENABLE or DISABLE.

Example:

```
/* Enables external lines 12 and 14 interrupt generation on falling
edge */
EXTI_InitTypeDef EXTI_InitStructure;
EXTI_InitStructure.EXTI_Line = EXTI_Line12 | EXTI_Line14;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
```

8.2.3 EXTI_Struct function

Table 135 describes the EXTI_StructInit function.

Table 135. EXTI_StructInit function

Function name	EXTI_StructInit
Function prototype	void EXTI_StructInit(EXTI_InitTypeDef*EXTI_InitStruct)
Behavior description	Fills each EXTI_InitStruct member with its default value.
Input parameter	EXTI_InitStruct: pointer to a EXTI_InitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 136 gives the EXTI_InitStruct members default values:

Table 136. EXTI_InitStruct default values

Member	Default value
EXTI_Line	EXTI_LineNone
EXTI_Mode	EXTI_Mode_Interrupt
EXTI_Trigger	EXTI_Trigger_Falling
EXTI_LineCmd	DISABLE

Example:

```
/* Initialize the EXTI Init Structure parameters */
EXTI_InitTypeDef EXTI_InitStructure;
EXTI_StructInit(&EXTI_InitStructure);
```

8.2.4 EXTI_GenerateSWInterrupt function

Table 137 describes the EXTI_GenerateSWInterrupt function.

Table 137. EXTI_GenerateSWInterrupt function

Function name	EXTI_GenerateSWInterrupt
Function prototype	void EXTI_GenerateSWInterrupt(u32 EXTI_Line)
Behavior description	Generates a software interrupt.
Input parameter	EXTI_Line: EXTI lines to be enabled or disabled. Refer to *EXTI_Line* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a software interrupt request */
EXTI_GenerateSWInterrupt(EXTI_Line6);
```

8.2.5 EXTI_GetFlagStatus function

Table 138 describes the EXTI_GetFlagStatus function.

Table 138. EXTI_GetFlagStatus function

Function name	EXTI_GetFlagStatus
Function prototype	FlagStatus EXTI_GetFlagStatus(u32 EXTI_Line)
Behavior description	Checks whether the specified EXTI line flag is set or not.
Input parameter	EXTI_Line: EXTI lines flag to check. Refer to *EXTI_Line* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of EXTI_Line (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the status of EXTI line 8 */
FlagStatus EXTIStatus;
EXTIStatus = EXTI_GetFlagStatus(EXTI_Line8);
```

8.2.6 EXTI_ClearFlag function

Table 139 describes the EXTI_ClearFlag function.

Table 139. EXTI_ClearFlag function

Function name	EXTI_ClearFlag
Function prototype	`void EXTI_ClearFlag(u32 EXTI_Line)`
Behavior description	Clears the EXTI line pending flags.
Input parameter	EXTI_Line: EXTI lines flags to clear. Refer to *EXTI_Line* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the EXTI line 2 pending flag */
EXTI_ClearFlag(EXTI_Line2);
```

8.2.7 EXTI_GetITStatus function

Table 140 describes the EXTI_GetITStatus function.

Table 140. EXTI_GetITStatus function

Function name	EXTI_GetITStatus
Function prototype	ITStatus EXTI_GetITStatus(u32 EXTI_Line)
Behavior description	Checks whether the specified EXTI line is asserted or not.
Input parameter	EXTI_Line: EXTI lines pending bits to check. Refer to *EXTI_Line* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of EXTI_Line (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the status of EXTI line 8 */
ITStatus EXTIStatus;
EXTIStatus = EXTI_GetITStatus(EXTI_Line8);
```

8.2.8 EXTI_ClearITPendingBit function

Table 141 describes the EXTI_ClearITPendingBit function.

Table 141. EXTI_ClearITPendingBit function

Function name	EXTI_ClearITPendingBit
Function prototype	void EXTI_ClearITPendingBit(u32 EXTI_Line)
Behavior description	Clears the EXTI's line pending bits.
Input parameter	EXTI_Line: EXTI lines pending bits to clear. Refer to *EXTI_Line* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clears the EXTI line 2 interrupt pending bit */
EXTI_ClearITpendingBit(EXTI_Line2);
```

9 Flash memory (FLASH)

Section 9.1: FLASH register structures describes the data structures used in the FLASH Firmware Library. *Section 9.2: Firmware library functions* presents the Firmware Library functions.

9.1 FLASH register structures

The FLASH register structures, *FLASH_TypeDef* and OB_TypeDef, are defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 ACR;
  vu32 KEYR;
  vu32 OPTKEYR;
  vu32 SR;
  vu32 CR;
  vu32 AR;
  vu32 RESERVED;
  vu32 OBR;
  vu32 WRPR;
} FLASH_TypeDef;

typedef struct
{
  vu16 RDP;
  vu16 USER;
  vu16 Data0;
  vu16 Data1;
  vu16 WRP0;
  vu16 WRP1;
  vu16 WRP2;
  vu16 WRP3;
} OB_TypeDef;
```

Table 142 and *Table 143* give the list of the FLASH registers and Option Byte registers (OB), respectively.

Table 142. FLASH registers

Register	Description
ACR	Flash Access Control Register
KEYR	FPEC Key Register
OPTKEYR	Option Byte Key Register
SR	Flash Status Register
CR	Flash Control Register
AR	Flash Address Register
OBR	Option Byte and Status Register
WRPR	Option Byte write protection Register

Table 143. Option Bytes registers (OB)

Register	Description
RDP	Read Out Option Byte
USER	User Option Byte
Data0	Data0 Option Byte
Data1	Data1 Option Byte
WRP0	Write Protection 0 Option Byte
WRP1	Write Protection 1 Option Byte
WRP2	Write Protection 2 Option Byte
WRP3	Write Protection 3 Option Byte

The FLASH peripheral is declared in *stm32f10x_map.h*:

```
/* Flash registers base address */
#define FLASH_BASE            ((u32)0x40022000)

/* Flash Option Bytes base address */
#define OB_BASE               ((u32)0x1FFFF800)
#ifndef DEBUG
...
#ifdef _FLASH
  #define FLASH               ((FLASH_TypeDef *) FLASH_BASE)
  #define OB                  ((OB_TypeDef *) OB_BASE)
#endif /*_FLASH */
...
#else   /* DEBUG */
...
#ifdef _FLASH
  EXT FLASH_TypeDef            *FLASH;
  EXT OB_TypeDef               *OB;
#endif /*_FLASH */
...
#endif
```

When using the Debug mode, FLASH and OB pointers are initialized in *stm32f10x_lib.c* file:

```
#ifdef _FLASH
FLASH = (FLASH_TypeDef *)  FLASH_BASE;
OB = (OB_TypeDef *)  OB_BASE;
#endif /*_FLASH */
```

To access the FLASH registers, _FLASH must be defined in *stm32f10x_conf.h* as follows:

```
#define _FLASH
```

By default only the functions performing FLASH configuration (latency, prefetch, half cycle) are enabled (see *Table 144*).

To enable FLASH program/erase/protections functions, `_FLASH_PROG` must be defined in *stm32f10x_conf.h* as follows:

```
#define _FLASH_PROG
```

9.2 Firmware library functions

Table 144 lists the various functions of the FLASH library.

Table 144. FLASH library function

Function name	Description
FLASH_SetLatency	Sets the code latency value.
FLASH_HalfCycleAccessCmd	Enables or disables the Half cycle FLASH access.
FLASH_PrefetchBufferCmd	Enables or disables the Prefetch Buffer.
FLASH_Unlock	Unlocks the FLASH Program Erase Controller.
FLASH_Lock	Locks the Flash Program Erase Controller.
FLASH_ErasePage	Erases a specified FLASH page.
FLASH_EraseAllPages	Erases all FLASH pages.
FLASH_EraseOptionBytes	Erases the FLASH option bytes.
FLASH_ProgramWord	Programs a word at a specified address.
FLASH_ProgramHalfWord	Programs a half word at a specified address.
FLASH_ProgramOptionByteData	Programs a half word at a specified Option Byte Data address.
FLASH_EnableWriteProtection	Write protects the desired pages
FLASH_ReadOutProtection	Enables or disables the read out protection.
FLASH_UserOptionByteConfig	Programs the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY.
FLASH_GetUserOptionByte	Returns the FLASH User Option Bytes values.
FLASH_GetWriteProtectionOptionByte	Returns the FLASH Write Protection Option Bytes Register value.
FLASH_GetReadOutProtectionStatus	Checks whether the FLASH Read Out Protection Status is set or not.
FLASH_GetPrefetchBufferStatus	Checks whether the FLASH Prefetch Buffer status is set or not.
FLASH_ITConfig	Enables or disables the specified FLASH interrupts.
FLASH_GetFlagStatus	Checks whether the specified FLASH flag is set or not.
FLASH_ClearFlag	Clears the FLASH pending flags.
FLASH_GetStatus	Returns the FLASH Status.
FLASH_WaitForLastOperation	Waits for a Flash operation to complete or a TIMEOUT to occur.

9.2.1 FLASH_SetLatency function

Table 145 describes the FLASH_SetLatency function.

Table 145. FLASH_SetLatency function

Function name	FLASH_SetLatency
Function prototype	void FLASH_SetLatency(u32 FLASH_Latency)
Behavior description	Sets the code latency value.
Input parameter	FLASH_Latency specifies the FLASH Latency value. Refer to *FLASH_Latency* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FLASH_Latency

FLASH_Latency is used to configure the FLASH Latency value. See *Table 146* for the values of this parameter.

Table 146. FLASH_Latency values

FLASH_Latency	Description
FLASH_Latency_0	Zero Latency cycle.
FLASH_Latency_1	One Latency cycle.
FLASH_Latency_2	Two Latency cycles.

Example:

```
/* Configure the Latency cycle: Set 2 Latency cycles */
FLASH_SetLatency(FLASH_Latency_2);
```

9.2.2 FLASH_HalfCycleAccessCmd function

Table 147 describes the FLASH_HalfCycleAccessCmd function.

Table 147. FLASH_HalfCycleAccessCmd function

Function name	FLASH_HalfCycleAccessCmd
Function prototype	void FLASH_HalfCycleAccessCmd(u32 FLASH_HalfCycleAccess)
Behavior description	Enables or disables the Half cycle Flash access.
Input parameter	FLASH_HalfCycle: FLASH Half cycle mode. Refer to *FLASH_HalfCycleAccess* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FLASH_HalfCycleAccess

FLASH_HalfCycleAccess is used to select the FLASH Half Cycle access mode. See *Table 148* for the values of this parameter.

Table 148. FLASH_HalfCycleAccess values

FLASH_HalfCycleAccess	Description
FLASH_HalfCycleAccess_Enable	Half Cycle Access Enable
FLASH_HalfCycleAccess_Disable	Half Cycle Access Disable

Example:

```
/* Enable the Half Cycle Flash access */
FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Enable);
```

9.2.3 FLASH_PrefetchBufferCmd function

Table 149 describes the FLASH_PrefetchBufferCmd function.

Table 149. FLASH_PrefetchBufferCmd function

Function name	FLASH_PrefetchBufferCmd
Function prototype	void FLASH_PrefetchBufferCmd(u32 FLASH_PrefetchBuffer)
Behavior description	Enables or disables the Prefetch Buffer.
Input parameter	FLASH_PrefetchBuffer: Prefetch buffer status. Refer to *FLASH_PrefetchBuffer* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FLASH_PrefetchBuffer

FLASH_PrefetchBuffer is used to select the FLASH Prefetch Buffer status. See *Table 150* for the values of this parameter.

Table 150. FLASH_PrefetchBuffer values

FLASH_PrefetchBuffer	Description
FLASH_PrefetchBuffer_Enable	Prefetch Buffer Enable
FLASH_PrefetchBuffer_Disable	Prefetch Buffer Disable

Example:

```
/* Enable The Prefetch Buffer */
FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
```

9.2.4 FLASH_Unlock function

Table 151 describes the FLASH_Unlock function.

Table 151. FLASH_Unlock function

Function name	FLASH_Unlock
Function prototype	void FLASH_Unlock(void)
Behavior description	Unlocks the FLASH Program Erase Controller.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Unlocks the Flash */
FLASH_Unlock();
```

9.2.5 FLASH_Lock function

Table 152 describes the FLASH_Lock function.

Table 152. FLASH_Lock function

Function name	FLASH_Lock
Function prototype	void FLASH_Lock(void)
Behavior description	Locks the FLASH Program Erase Controller.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Locks the Flash */
FLASH_Lock();
```

9.2.6 FLASH_ErasePage function

Table 153 describes the FLASH_ErasePage function.

Table 153. FLASH_ErasePage function

Function name	FLASH_ErasePage
Function prototype	FLASH_Status FLASH_ErasePage(u32 Page_Address)
Behavior description	Erases a FLASH page.
Input parameter	FLASH_Page: page to be erased
Output parameter	None
Return parameter	The Erase operation Status.
Required preconditions	None
Called functions	None

Example:

```
/* Erases the Flash Page 0 */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_ErasePage(0x08000000);
```

9.2.7 FLASH_EraseAllPages function

Table 154 describes FLASH_EraseAllPages function.

Table 154. FLASH_EraseAllPages function

Function name	FLASH_EraseAllPages
Function prototype	FLASH_Status FLASH_EraseAllPages(void)
Behavior description	Erases all FLASH pages.
Input parameter	None
Output parameter	None
Return parameter	The Erase operation Status
Required preconditions	None
Called functions	None

Example:

```
/* Erases the Flash */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_EraseAllPages();
```

9.2.8 FLASH_EraseOptionBytes function

Table 155 describes the FLASH_EraseOptionBytes function.

Table 155. FLASH_EraseOptionBytes function

Function name	FLASH_EraseOptionBytes
Function prototype	FLASH_Status FLASH_EraseOptionBytes(void)
Behavior description	Erases the FLASH option bytes.
Input parameter	None
Output parameter	None
Return parameter	The Erase operation Status
Required preconditions	None
Called functions	None

Example:

```
/* Erases the Flash Option Bytes */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_EraseOptionBytes();
```

9.2.9 FLASH_ProgramWord function

Table 156 describes the FLASH_ProgramWord function.

Table 156. FLASH_ProgramWord function

Function name	FLASH_ProgramWord
Function prototype	FLASH_Status FLASH_ProgramWord(u32 Address, u32 Data)
Behavior description	Programs a word at a specified address.
Input parameter1	Address: address to be programmed.
Input parameter2	Data: specifies the data to be programmed.
Output parameter	None
Return parameter	The Program operation Status.
Required preconditions	None
Called functions	None

Example:

```
/* Writes the Data1 at the Address1 */
FLASH_Status status = FLASH_COMPLETE;
u32 Data1 = 0x1234567;
u32 Address1 = 0x8000000;
status = FLASH_ProgramWord(Address1, Data1);
```

9.2.10 FLASH_ProgramHalfWord function

Table 157 describes the FLASH_ProgramHalfWord function.

Table 157. FLASH_ProgramHalfWord function

Function name	FLASH_ProgramHalfWord
Function prototype	FLASH_Status FLASH_ProgramHalfWord(u32 Address, u16 Data)
Behavior description	Programs a half word at a specified address.
Input parameter1	Address: address to be programmed.
Input parameter2	Data: half-word data to be programmed.
Output parameter	None
Return parameter	The Program operation Status.
Required preconditions	None
Called functions	None

Example:

```
/* Writes the Data1 at the Address1 */
FLASH_Status status = FLASH_COMPLETE;
u16 Data1 = 0x1234;
u32 Address1 = 0x8000004;
status = FLASH_ProgramHalfWord(Address1, Data1);
```

9.2.11 FLASH_ProgramOptionByteData function

Table 158 describes the FLASH_ProgramOptionByteData function.

Table 158. FLASH_ProgramOptionByteData function

Function name	FLASH_ProgramOptionByteData
Function prototype	FLASH_Status FLASH_ProgramOptionByteData(u32 Address, u8 Data)
Behavior description	Programs a half word at a specified Option Byte Data address.
Input parameter1	Address: address to be programmed. This parameter can be 0x1FFFF804 or 0x1FFFF806.
Input parameter2	Data: specifies the data to be programmed.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Writes the Data1 at the Address1 */
FLASH_Status status = FLASH_COMPLETE;
u8 Data1 = 0x12;
u32 Address1 = 0x1FFFF804;
status = FLASH_ProgramOptionByteData(Address1, Data1);
```

9.2.12 FLASH_EnableWriteProtection function

Table 159 describes the FLASH_EnableWriteProtection function.

Table 159. FLASH_EnableWriteProtection function

Function name	FLASH_EnableWriteProtection
Function prototype	FLASH_Status FLASH_EnableWriteProtection(u32 FLASH_Pages)
Behavior description	Write protects the desired pages.
Input parameter	FLASH_Pages: address of the pages to be write protected. Refer to *FLASH_Pages* for more details on the values of this parameter.
Output parameter	None
Return parameter	The write protection operation Status.
Required preconditions	None
Called functions	None

FLASH_Pages

FLASH_Pages is used to configure the FLASH write protection pages. *Table 160* and *Table 161* give the values taken by this parameter for Medium-density STM32F10xxx devices (FLASH page size equal to 1 KB) and High-density STM32F10xxx devices (FLASH page size equal to 2 KB), respectively.

Table 160. FLASH_Pages values for Medium-density devices

FLASH_Pages	Description
FLASH_WRProt_Pages0to3	Write protection of page 0 to 3.
FLASH_WRProt_Pages4to7	Write protection of page 4 to 7.
FLASH_WRProt_Pages8to11	Write protection of page 8 to 11.
FLASH_WRProt_Pages12to15	Write protection of page 12 to 15.
FLASH_WRProt_Pages16to19	Write protection of page 16 to 19.
FLASH_WRProt_Pages20to23	Write protection of page 20 to 23.
FLASH_WRProt_Pages24to27	Write protection of page 24 to 27.
FLASH_WRProt_Pages28to31	Write protection of page 28 to 31.
FLASH_WRProt_Pages32to35	Write protection of page 32 to 35.
FLASH_WRProt_Pages36to39	Write protection of page 36 to 39.
FLASH_WRProt_Pages40to43	Write protection of page 40 to 43.
FLASH_WRProt_Pages44to47	Write protection of page 44 to 47.
FLASH_WRProt_Pages48to51	Write protection of page 48 to 51.
FLASH_WRProt_Pages52to55	Write protection of page 52 to 55.
FLASH_WRProt_Pages56to59	Write protection of page 56 to 59.
FLASH_WRProt_Pages60to63	Write protection of page 60 to 63.
FLASH_WRProt_Pages64to67	Write protection of page 64 to 67.

Table 160. FLASH_Pages values for Medium-density devices (continued)

FLASH_Pages	Description
FLASH_WRProt_Pages68to71	Write protection of page 68 to 71.
FLASH_WRProt_Pages72to75	Write protection of page 72 to 75.
FLASH_WRProt_Pages76to79	Write protection of page 76 to 79.
FLASH_WRProt_Pages80to83	Write protection of page 80 to 83.
FLASH_WRProt_Pages84to87	Write protection of page 84 to 87.
FLASH_WRProt_Pages88to91	Write protection of page 88 to 91.
FLASH_WRProt_Pages92to95	Write protection of page 92 to 95.
FLASH_WRProt_Pages96to99	Write protection of page 96 to 99.
FLASH_WRProt_Pages100to103	Write protection of page 100 to 103.
FLASH_WRProt_Pages104to107	Write protection of page 104 to 107.
FLASH_WRProt_Pages108to111	Write protection of page 108 to 111.
FLASH_WRProt_Pages112to115	Write protection of page 112 to 115.
FLASH_WRProt_Pages116to119	Write protection of page 115 to 119.
FLASH_WRProt_Pages120to123	Write protection of page 120 to 123.
FLASH_WRProt_Pages124to127	Write protection of page 124 to 127.
FLASH_WRProt_AllPages	Write protection all Pages.

Table 161. FLASH_Pages values for High-density devices

FLASH_Pages	Description
FLASH_WRProt_Pages0to1	Write protection of page 0 to 1.
FLASH_WRProt_Pages2to3	Write protection of page 2 to 3.
FLASH_WRProt_Pages4to5	Write protection of page 4 to 5.
FLASH_WRProt_Pages6to7	Write protection of page 6 to 7.
FLASH_WRProt_Pages8to9	Write protection of page 8 to 9.
FLASH_WRProt_Pages10to11	Write protection of page 10 to 11.
FLASH_WRProt_Pages12to13	Write protection of page 12 to 13.
FLASH_WRProt_Pages14to15	Write protection of page 14 to 15.
FLASH_WRProt_Pages16to17	Write protection of page 16 to 17.
FLASH_WRProt_Pages18to19	Write protection of page 18 to 19.
FLASH_WRProt_Pages20to21	Write protection of page 20 to 21.
FLASH_WRProt_Pages22to23	Write protection of page 22 to 23.
FLASH_WRProt_Pages24to25	Write protection of page 24 to 25.
FLASH_WRProt_Pages26to27	Write protection of page 26 to 27.
FLASH_WRProt_Pages28to29	Write protection of page 28 to 29.
FLASH_WRProt_Pages30to31	Write protection of page 30 to 31.

Table 161. FLASH_Pages values for High-density devices (continued)

FLASH_Pages	Description
FLASH_WRProt_Pages32to33	Write protection of page 32 to 33.
FLASH_WRProt_Pages34to35	Write protection of page 34 to 35.
FLASH_WRProt_Pages36to37	Write protection of page 36 to 37.
FLASH_WRProt_Pages38to39	Write protection of page 38 to 39.
FLASH_WRProt_Pages40to41	Write protection of page 40 to 41.
FLASH_WRProt_Pages42to43	Write protection of page 42 to 43.
FLASH_WRProt_Pages44to45	Write protection of page 44 to 45.
FLASH_WRProt_Pages46to47	Write protection of page 46 to 47.
FLASH_WRProt_Pages48to49	Write protection of page 48 to 49.
FLASH_WRProt_Pages50to51	Write protection of page 50 to 51.
FLASH_WRProt_Pages52to53	Write protection of page 52 to 53.
FLASH_WRProt_Pages54to55	Write protection of page 54 to 55.
FLASH_WRProt_Pages56to57	Write protection of page 56 to 57.
FLASH_WRProt_Pages58to59	Write protection of page 58 to 59.
FLASH_WRProt_Pages60to61	Write protection of page 60 to 61.
FLASH_WRProt_Pages62to255	Write protection of page 62 to 255.
FLASH_WRProt_AllPages	Write protection all Pages.

Example:

```
/* Protects the Pages0to3 and Pages108to111 */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_EnableWriteProtection
(FLASH_WRProt_Pages0to3|FLASH_WRProt_Pages108to111);
```

9.2.13 FLASH_ReadOutProtection function

Table 162 describes the FLASH_ReadOutProtection function.

Table 162. FLASH_ReadOutProtection function

Function name	FLASH_ReadOutProtection
Function prototype	FLASH_Status FLASH_ReadOutProtection(FunctionalState NewState)
Behavior description	Enables or disables the read out protection.
Input parameter	NewState: new state of the Read Out protection. This parameter can be set either to ENABLE or DISABLE.
Output parameter	None
Return parameter	The protection operation Status.
Required preconditions	If the user has already programmed the other option bytes before calling this function, he must re-program them since this function erases all option bytes.
Called functions	None

Example:

```
/* Disables the ReadOut Protection */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_ReadOutProtection(DISABLE);
```

Note: To safely program the option bytes, the user has to follow the order of the operations described below:

1. *Call the FLASH_ReadOutProtection function, if there is a need to read-protect the Flash memory*
2. *Call the FLASH_EnableWriteProtection function in order to write-protect some pages or all the Flash memory*
3. *Call the FLASH_UserOptionByteConfig to program the user option byte: IWDG_SW / RST_STOP /RST_STDBY*
4. *Call the FLASH_ProgramOptionByteData to program a half-word to the specified option byte data addresses*
5. *Generate a reset to load the new option bytes*

9.2.14 FLASH_UserOptionByteConfig function

Table 163 describes the FLASH_UserOptionByteConfig function.

Table 163. FLASH_UserOptionByteConfig function

Function name	FLASH_UserOptionByteConfig
Function prototype	FLASH_Status FLASH_UserOptionByteConfig(u16 OB_IWDG, u16 OB_STOP, u16 OB_STDBY)
Behavior description	Programs the FLASH User Option Byte: IWDG_SW / RST_STOP RST_STDBY.
Input parameter1	OB_IWDG: Selects the IWDG mode. Refer to *OB_IWDG* for more details on the values of this parameter.
Input parameter2	OB_STOP: Reset event when entering Stop mode. Refer to *OB_STOP* for more details on the values of this parameter.
Input parameter3	OB_STDBY: Reset event when entering Standby mode. Refer to *OB_STDBY* for more details on the values of this parameter.
Output parameter	None
Return parameter	The Option Byte program Status.
Required preconditions	None
Called functions	None

OB_IWDG

This parameter configures the IWDG mode. See *Table 164* for the values taken by OB_IWDG.

Table 164. OB_IWDG values

OB_IWDG	Description
OB_IWDG_SW	Software IWDG selected.
OB_IWDG_HW	Hardware IWDG selected.

OB_STOP

This parameter specifies if a Reset is generated or not when entering Stop mode. See *Table 165* for the values taken by OB_STOP.

Table 165. OB_STOP values

OB_STOP	Description
OB_STOP_NoRST	No reset generated when entering Stop mode
OB_STOP_RST	Reset generated when entering Stop mode

OB_STDBY

This parameter specifies if a Reset is generated or not when entering Standby mode. See *Table 166* for the values taken by OB_STBY.

Table 166. OB_STDBY values

OB_STDBY	Description
OB_STDBY_NoRST	No reset generated when entering Standby mode
OB_STDBY_RST	Reset generated when entering Standby mode

Example:

```
/* Option Bytes Configuration: software watchdog, Reset generation
when entering in Stop and No reset generation when entering in
Standby */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_UserOptionByteConfig(OB_IWDG_SW, OB_STOP_RST,
OB_STDBY_NoRST);
```

9.2.15 FLASH_GetUserOptionByte function

Table 167 describes the FLASH_GetUserOptionByte function.

Table 167. FLASH_GetUserOptionByte function

Function name	FLASH_GetUserOptionByte
Function prototype	u32 FLASH_GetUserOptionByte(void)
Behavior description	Returns the FLASH User Option Bytes values.
Input parameter	None
Output parameter	None
Return parameter	The FLASH User Option Bytes values:IWDG_SW(Bit0), RST_STOP(Bit1) and RST_STDBY(Bit2).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the user option byte values */
u32 UserByteValue = 0x0;
u32 IWDGValue = 0x0, RST_STOPValue = 0x0, RST_STDBYValue = 0x0;
UserByteValue = FLASH_GetUserOptionByte();
IWDGValue = UserByteValue & 0x0001;
RST_STOPValue = UserByteValue & 0x0002;
RST_STDBYValue = UserByteValue & 0x0004;
```

9.2.16 FLASH_GetWriteProtectionOptionByte function

Table 168 describes the FLASH_GetWriteProtectionOptionByte function.

Table 168. FLASH_GetWriteProtectionOptionByte function

Function name	FLASH_GetWriteProtectionOptionByte
Function prototype	u32 FLASH_GetWriteProtectionOptionByte(void)
Behavior description	Returns the FLASH Write Protection Option Bytes Register value.
Input parameter	None
Output parameter	None
Return parameter	The FLASH Write Protection Option Bytes Register value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the Write Protection option byte values */
u32 WriteProtectionValue = 0x0;
WriteProtectionValue = FLASH_GetWriteProtectionOptionByte();
```

9.2.17 FLASH_GetReadOutProtectionStatus function

Table 169 describes the FLASH_GetReadOutProtectionStatus function.

Table 169. FLASH_GetReadOutProtectionStatus function

Function name	FLASH_GetReadOutProtectionStatus
Function prototype	FlagStatus FLASH_GetReadOutProtectionStatus(void)
Behavior description	Checks whether the FLASH Read Out Protection Status is set or not.
Input parameter	None
Output parameter	None
Return parameter	FLASH ReadOut Protection Status (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the ReadOut Protection status */
FlagStatus status = RESET;
status = FLASH_GetReadOutProtectionStatus();
```

9.2.18 FLASH_GetPrefetchBufferStatus function

Table 170 describes the FLASH_GetPrefetchBufferStatus function.

Table 170. FLASH_GetPrefetchBufferStatus function

Function name	FLASH_GetPrefetchBufferStatus
Function prototype	FlagStatus FLASH_GetPrefetchBufferStatus(void)
Behavior description	Checks whether the FLASH Prefetch Buffer status is set or not.
Input parameter	None
Output parameter	None
Return parameter	FLASH Prefetch Buffer Status (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the Prefetch Buffer status */
FlagStatus status = RESET;
status = FLASH_GetPrefetchBufferStatus();
```

9.2.19 FLASH_ITConfig function

Table 171 describes the FLASH_ITConfig function.

Table 171. FLASH_ITConfig function

Function name	FLASH_ITConfig
Function prototype	void FLASH_ITConfig(u16 FLASH_IT, FunctionalState NewState)
Behavior description	Enables or disables the specified FLASH interrupts.
Input parameter1	FLASH_IT: FLASH interrupt sources to be enabled or disabled. Refer to *FLASH_IT* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified FLASH interrupts. This parameter can be set to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FLASH_IT

This parameter is used to enable or disable FLASH interrupts. One or a combination of the following values can be used:

Table 172. FLASH_IT values

FLASH_IT	Description
FLASH_IT_ERROR	FPEC error interrupt source
FLASH_IT_EOP	End of FLASH Operation Interrupt source

Example:

```
/* Enables the EOP Interrupt source */
FLASH_ITConfig(FLASH_IT_EOP, ENABLE);
```

9.2.20 FLASH_GetFlagStatus function

Table 173 describes the FLASH_GetFlagStatus function.

Table 173. Flah_GetFlagStatus function

Function name	FLASH_GetFlagStatus
Function prototype	FlagStatus FLASH_GetFlagStatus(u16 FLASH_FLAG)
Behavior description	Checks whether the specified FLASH flag is set or not.
Input parameter	None
Input parameter	FLASH_FLAG: flag to be checked. Refer to *FLASH_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FLASH_FLAG

The FLASH flags which can be checked by issuing the FLASH_GetFlagStatus function are listed in the following table:

Table 174. FLASH_FLAG definition

FLASH_FLAG	Description
FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_EOP	FLASH end of operation flag
FLASH_FLAG_PGERR	FLASH Program error flag
FLASH_FLAG_WRPRTERR	FLASH Page Write protected error flag
FLASH_FLAG_OPTERR	FLASH Option Byte error flag

Example:

```
/* Checks whether the EOP Flag Status is SET or not */
FlagStatus status = RESET;
status = FLASH_GetFlagStatus(FLASH_FLAG_EOP);
```

9.2.21 FLASH_ClearFlag function

Table 175 describes the FLASH_ClearFlag function.

Table 175. FLASH_ClearFlag function

Function name	FLASH_ClearFlag
Function prototype	void FLASH_ClearFlag(u16 FLASH_Flag)
Behavior description	Clears the FLASH pending flags
Input parameter	FLASH_FLAG: flag to be cleared Refer to *FLASH_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FLASH_FLAG

The FLASH flags that can be cleared by issuing the FLASH_ClearFlag function are listed in the following table:

Table 176. FLASH_FLAG definition

FLASH_FLAG	Description
FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_EOP	FLASH end of operation flag
FLASH_FLAG_PGERR	FLASH Program error flag
FLASH_FLAG_WRPRTERR	FLASH Page Write protected error flag

Example:

```
/* Clears all flags */
FLASH_ClearFlag(FLASH_FLAG_BSY|FLASH_FLAG_EOP|FLASH_FLAG_PGER
|FLASH_FLAG_WRPRTERR);
```

9.2.22 FLASH_GetStatus function

Table 177 describes the FLASH_GetStatus function.

Table 177. FLASH_GetStatus function

Function name	FLASH_GetStatus
Function prototype	FLASH_Status FLASH_GetStatus(void)
Behavior description	Returns the FLASH Status.
Input parameter	None
Output parameter	None
Return parameter	FLASH Status: The returned value can be: FLASH_BUSY, FLASH_ERROR_PG or FLASH_ERROR_WRP or FLASH_COMPLETE
Required preconditions	None
Called functions	None

Example:

```
/* Check for the Flash status */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_GetStatus();
```

9.2.23 FLASH_WaitForLastOperation function

Table 178 describes the FLASH_WaitForLastOperation function.

Table 178. FLASH_WaitForLastOperation function

Function name	FLASH_WaitForLastOperation
Function prototype	FLASH_Status FLASH_WaitForLastOperation(u32 Timeout)
Behavior description	Waits for a Flash operation to complete or a TIMEOUT to occur.
Input parameter	None
Output parameter	None
Return parameter	Return the appropriate operation Status. This parameter can be FLASH_BUSY, FLASH_ERROR_PG or FLASH_ERROR_WRP or FLASH_COMPLETE or FLASH_TIMEOUT
Required preconditions	None
Called functions	None

Example:

```
/* Waits for the Flash operation to be completed */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_WaitForLastOperation();
```

10 General purpose I/O (GPIO)

The GPIO driver can be used for several purposes, including pin configuration, single bit set/reset, lock mechanism, reading from a port pin, and writing data into a port pin.

Section 10.1: GPIO register structure describes the data structures used in the GPIO Firmware Library. *Section 10.2: Firmware library functions* presents the Firmware Library functions.

10.1 GPIO register structure

The GPIO register structure, GPIO_*TypeDef,* is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 CRL;
  vu32 CRH;
  vu32 IDR;
  vu32 ODR;
  vu32 BSRR;
  vu32 BRR;
  vu32 LCKR;
} GPIO_TypeDef;

typedef struct
{
  vu32 EVCR;
  vu32 MAPR;
  vu32 EXTICR[4];
} AFIO_TypeDef;
```

Table 179 gives the list of the GPIO registers:

Table 179. GPIO registers

Register	Description
CRL	Port Control Register low
CRH	Port Control Register High
IDR	Input Data Register
ODR	Output Data Register
BSRR	Bit Set Reset Register
BRR	Bit Reset Register
LCKR	Lock Register
EVCR	Event Control Register
MAPR	Remap Debug and AF Register
EXTICR	EXTI Line 0 to Line 15 Configuration Register

The five GPIO peripherals are declared in *stm32f10x_map*.h:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
...
#define AFIO_BASE             (APB2PERIPH_BASE + 0x0000)
#define GPIOA_BASE            (APB2PERIPH_BASE + 0x0800)
#define GPIOB_BASE            (APB2PERIPH_BASE + 0x0C00)
#define GPIOC_BASE            (APB2PERIPH_BASE + 0x1000)
#define GPIOD_BASE            (APB2PERIPH_BASE + 0x1400)
#define GPIOE_BASE            (APB2PERIPH_BASE + 0x1800)
#define GPIOF_BASE            (APB2PERIPH_BASE + 0x1C00)
#define GPIOG_BASE            (APB2PERIPH_BASE + 0x2000)

#ifndef DEBUG
...
#ifdef _AFIO
  #define AFIO                  ((AFIO_TypeDef *) AFIO_BASE)
#endif /*_AFIO */

#ifdef _GPIOA
  #define GPIOA                 ((GPIO_TypeDef *) GPIOA_BASE)
#endif /*_GPIOA */

#ifdef _GPIOB
  #define GPIOB                 ((GPIO_TypeDef *) GPIOB_BASE)
#endif /*_GPIOB */

#ifdef _GPIOC
  #define GPIOC                 ((GPIO_TypeDef *) GPIOC_BASE)
#endif /*_GPIOC */

#ifdef _GPIOD
  #define GPIOD                 ((GPIO_TypeDef *) GPIOD_BASE)
#endif /*_GPIOD */

#ifdef _GPIOE
  #define GPIOE                 ((GPIO_TypeDef *) GPIOE_BASE)
#endif /*_GPIOE */
#ifdef _GPIOF
  #define GPIOF                 ((GPIO_TypeDef *) GPIOF_BASE)
#endif /*_GPIOF */
#ifdef _GPIOG
  #define GPIOG                 ((GPIO_TypeDef *) GPIOG_BASE)
#endif /*_GPIOG */
...
#else   /* DEBUG */
...
#ifdef _AFIO
  EXT AFIO_TypeDef            *AFIO;
```

```
#endif /*_AFIO */

#ifdef _GPIOA
  EXT GPIO_TypeDef            *GPIOA;
#endif /*_GPIOA */

#ifdef _GPIOB
  EXT GPIO_TypeDef            *GPIOB;
#endif /*_GPIOB */

#ifdef _GPIOC
  EXT GPIO_TypeDef            *GPIOC;
#endif /*_GPIOC */

#ifdef _GPIOD
  EXT GPIO_TypeDef            *GPIOD;
#endif /*_GPIOD */

#ifdef _GPIOE
  EXT GPIO_TypeDef            *GPIOE;
#endif /*_GPIOE */
#ifdef _GPIOF
  EXT GPIO_TypeDef            *GPIOF;
#endif /*_GPIOF */
#ifdef _GPIOG
  EXT GPIO_TypeDef            *GPIOG;
#endif /*_GPIOG */
...
#endif
```

When using the Debug mode, _AFIO, _GPIOA, _GPIOB, _GPIOC, _GPIOD, _GPIOE, _GPIOF and _GPIOG pointers are initialized in *stm32f10x_lib.c* file:

```
#ifdef _GPIOA
  GPIOA = (GPIO_TypeDef *)  GPIOA_BASE;
#endif /*_GPIOA */

#ifdef _GPIOB
  GPIOB = (GPIO_TypeDef *)  GPIOB_BASE;
#endif /*_GPIOB */

#ifdef _GPIOC
  GPIOC = (GPIO_TypeDef *)  GPIOC_BASE;
#endif /*_GPIOC */

#ifdef _GPIOD
  GPIOD = (GPIO_TypeDef *)  GPIOD_BASE;
#endif /*_GPIOD */

#ifdef _GPIOE
  GPIOE = (GPIO_TypeDef *)  GPIOE_BASE;
#endif /*_GPIOE */
```

```
#ifdef _GPIOF
  GPIOF = (GPIO_TypeDef *)  GPIOG_BASE;
#endif /*_GPIOF */

#ifdef _GPIOG
  GPIOG = (GPIO_TypeDef *)  GPIOG_BASE;
#endif /*_GPIOG */

#ifdef _AFIO
  AFIO = (AFIO_TypeDef *)  AFIO_BASE;
#endif /*_AFIO */
```

To access the GPIO registers, _GPIO, _AFIO, _GPIOA, _GPIOB, _GPIOC, _GPIOD, _GPIOE, _GPIOF and _GPIOG must be defined in stm32f10x_conf.h:

```
#define _GPIO
#define _GPIOA
#define _GPIOB
#define _GPIOC
#define _GPIOD
#define _GPIOE
#define _GPIOF
#define _GPIOG
#define _AFIO
```

10.2 Firmware library functions

Table 180 gives the list of the GPIO firmware library functions.

Table 180. GPIO firmware library functions

Function name	Description
GPIO_DeInit	Resets the GPIOx peripheral registers to their default reset values.
GPIO_AFIODeInit	Resets the Alternate Functions (remap, event control and EXTI configuration) registers to their default reset values.
GPIO_Init	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
GPIO_StructInit	Fills each GPIO_InitStruct member with its default value.
GPIO_ReadInputDataBit	Reads the specified input port pin
GPIO_ReadInputData	Reads the specified GPIO input data port
GPIO_ReadOutputDataBit	Reads the specified output data port bit
GPIO_ReadOutputData	Reads the specified GPIO output data port
GPIO_SetBits	Sets the selected data port bits
GPIO_ResetBits	Clears the selected data port bits
GPIO_WriteBit	Sets or clears the selected data port bit
GPIO_Write	Writes data to the specified GPIO data port
GPIO_PinLockConfig	Locks GPIO Pins configuration registers
GPIO_EventOutputConfig	Selects the GPIO pin used as Event output.
GPIO_EventOutputCmd	Enables or disables the Event Output.
GPIO_PinRemapConfig	Changes the mapping of the specified pin.
GPIO_EXTILineConfig	Selects the GPIO pin used as EXTI Line.

10.2.1 GPIO_DeInit function

Table 181 describes the GPIO_DeInit function.

Table 181. GPIO_DeInit function

Function name	GPIO_DeInit
Function prototype	void GPIO_DeInit(GPIO_TypeDef* GPIOx)
Behavior description	Resets the GPIOx peripheral registers to their default reset values.
Input parameter	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB2PeriphResetCmd()

Example:

```
/* Resets the GPIOA peripheral registers to their default reset
values */
GPIO_DeInit(GPIOA);
```

10.2.2 GPIO_AFIODeInit function

Table 182 describes the GPIO_AFIODeInit function.

Table 182. GPIO_AFIODeInit function

Function name	GPIO_AFIODeInit
Function prototype	void GPIO_AFIODeInit(void)
Behavior description	Resets the Alternate functions registers (remap, event control and EXTI configuration) to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB2PeriphResetCmd()

Example:

```
/* Resets the Alternate functions registers to their default reset
values */
GPIO_AFIODeInit();
```

10.2.3 GPIO_Init function

Table 183 describes the GPIO_Init function.

Table 183. GPIO_Init function

Function name	GPIO_Init
Function prototype	void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
Behavior description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	GPIO_InitStruct: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral. Refer to *GPIO_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

GPIO_InitTypeDef structure

The GPIO_InitTypeDef structure is defined in the *stm32f10x_gpio.h* file:

```
typedef struct
{
  u16 GPIO_Pin;
  GPIOSpeed_TypeDef GPIO_Speed;
  GPIOMode_TypeDef GPIO_Mode;
} GPIO_InitTypeDef;
```

GPIO_Pin

This member selects the GPIO pins to configure. Multiple-pin configuration can be performed by using the '|' operator. Any combination of the following values can be used:

Table 184. GPIO_Pin values

GPIO_Pin	Description
GPIO_Pin_None	No pin selected
GPIO_Pin_0	Pin 0 Selected
GPIO_Pin_1	Pin 1 Selected
GPIO_Pin_2	Pin 2 Selected
GPIO_Pin_3	Pin 3 Selected
GPIO_Pin_4	Pin 4 Selected
GPIO_Pin_5	Pin 5 Selected
GPIO_Pin_6	Pin 6 Selected
GPIO_Pin_7	Pin 7 Selected
GPIO_Pin_8	Pin 8 Selected
GPIO_Pin_9	Pin 9 Selected
GPIO_Pin_10	Pin 10 Selected
GPIO_Pin_11	Pin 11 Selected
GPIO_Pin_12	Pin 12 Selected
GPIO_Pin_13	Pin 13 Selected
GPIO_Pin_14	Pin 14 Selected
GPIO_Pin_15	Pin 15 Selected
GPIO_Pin_All	All Pins Selected

GPIO_Speed

GPIO_Speed is used to configure the speed for the selected pins.See *Table 185* for the values taken by this member.

Table 185. GPIO_Speed values

GPIO_Speed	Description
GPIO_Speed_10MHz	Output Maximum Frequency = 10 MHz
GPIO_Speed_2MHz	Output Maximum Frequency = 2 MHz
GPIO_Speed_50MHz	Output Maximum Frequency = 50 MHz

GPIO_Mode

GPIO_Mode configures the operating mode for the selected pins.See *Table 186* for the values taken by this member.

Table 186. GPIO_Mode values

GPIO_Mode	Description
GPIO_Mode_AIN	Analog Input
GPIO_Mode_IN_FLOATING	Input Floating
GPIO_Mode_IPD	Input Pull-Down
GPIO_Mode_IPU	Input Pull-up
GPIO_Mode_Out_OD	Open Drain Output
GPIO_Mode_Out_PP	Push-Pull Output
GPIO_Mode_AF_OD	Open Drain Output Alternate-Function
GPIO_Mode_AF_PP	Push-Pull Output Alternate-Function

Note: 1 When a pin is configured in input pull-up or pull-down mode, the Px_BSRR and Px_BRR registers are used.

2 GPIO_Mode allows to configure both the GPIO direction (Input/Output) and the corresponding input/output configuration: bits[7:4] GPIO_Mode configure the GPIO direction, while bits [4:0] define the configuration. The GPIO direction have the following indexes:

- *GPIO in input mode = 0x00*
- *GPIO in output mode = 0x01*

Table 187 shows all the GPIO_Mode indexes and codes.

Table 187. GPIO_Mode indexes and codes

GPIO Direction	Index	Mode	Configuration	Mode Code
GPIO Input	0x00	GPIO_Mode_AIN	0x00	0x00
		GPIO_Mode_IN_FLOATING	0x04	0x04
		GPIO_Mode_IPD	0x08	0x28
		GPIO_Mode_IPU	0x08	0x48
GPIO Output	0x01	GPIO_Mode_Out_OD	0x04	0x14
		GPIO_Mode_Out_PP	0x00	0x10
		GPIO_Mode_AF_OD	0x0C	0x1C
		GPIO_Mode_AF_PP	0x08	0x18

Example:

```
/* Configure all the GPIOA in Input Floating mode */
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

10.2.4 GPIO_StructInit function

Table 188 describes the GPIO_StructInit function.

Table 188. GPIO_StructInit function

Function name	GPIO_StructInit
Function prototype	void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)
Behavior description	Fills each GPIO_InitStruct member with its default value.
Input parameter	GPIO_InitStruct: pointer to a GPIO_InitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The GPIO_InitStruct default values are given in *Table 189*.

Table 189. GPIO_InitStruct default values

Member	Default value
GPIO_Pin	GPIO_Pin_All
GPIO_Speed	GPIO_Speed_2MHz
GPIO_Mode	GPIO_Mode_IN_FLOATING

Example:

```
/* Initialize the GPIO Init Structure parameters */
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_StructInit(&GPIO_InitStructure);
```

10.2.5 GPIO_ReadInputDataBit function

Table 190 describes the GPIO_ReadInputDataBit function.

Table 190. GPIO_ReadInputDataBit function

Function name	GPIO_ReadInputDataBit
Function prototype	u8 GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
Behavior description	Reads the specified input port pin.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	GPIO_Pin: port bit to be read. Refer to *GPIO_Pin* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The input port pin value.
Required preconditions	None
Called functions	None

Example:

```
/* Reads the seventh pin of the GPIOB and store it in ReadValue
variable */
u8 ReadValue;
ReadValue = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_7);
```

10.2.6 GPIO_ReadInputData function

Table 191 describes the GPIO_ReadInputData function.

Table 191. GPIO_ReadInputData function

Function name	GPIO_ReadInputData
Function prototype	u16 GPIO_ReadInputData(GPIO_TypeDef* GPIOx)
Behavior description	Reads the specified GPIO input data port.
Input parameter	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Output parameter	None
Return parameter	GPIO input data port value.
Required preconditions	None
Called functions	None

Example:

```
/*Read the GPIOC input data port and store it in ReadValue
variable*/
```

```
u16 ReadValue;
ReadValue = GPIO_ReadInputData(GPIOC);
```

10.2.7 GPIO_ReadOutputDataBit function

Table 192 describes the GPIO_ReadOutputDataBit function.

Table 192. GPIO_ReadOutputDataBit function

Function name	GPIO_ReadOutputDataBit
Function prototype	u8 GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
Behavior description	Reads the specified output data port bit.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	GPIO_Pin: port bit to read. Refer to *GPIO_Pin* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The output port pin value.
Required preconditions	None
Called functions	None

Example:

```
/* Reads the seventh pin of the GPIOB and store it in ReadValue
variable */
u8 ReadValue;
ReadValue = GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_7);
```

10.2.8 GPIO_ReadOutputData function

Table 193 describes the GPIO_ReadOutputData function.

Table 193. GPIO_ReadOutputData function

Function name	GPIO_ReadOutputData
Function prototype	u16 GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)
Behavior description	Reads the specified GPIO output data port.
Input parameter	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Output parameter	None
Return parameter	GPIO output data port value.
Required preconditions	None
Called functions	None

Example:

```
/* Read the GPIOC output data port and store it in ReadValue
variable */
u16 ReadValue;
ReadValue = GPIO_ReadOutputData(GPIOC);
```

10.2.9 GPIO_SetBits

Table 193 describes the GPIO_SetBits function.

Table 194. GPIO_SetBits function

Function name	GPIO_SetBits
Function prototype	void GPIO_SetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
Behavior description	Sets the selected data port bits.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	GPIO_Pin: specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15). Refer to *GPIO_Pin* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the GPIOA port pin 10 and pin 15 */
GPIO_SetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

10.2.10 GPIO_ResetBits

Table 195 describes the GPIO_ResetBits function.

Table 195. GPIO_ResetBits function

Function name	GPIO_ResetBits
Function prototype	void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
Behavior description	Clears the selected data port bits.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	GPIO_Pin: specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15). Refer to *GPIO_Pin* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clears the GPIOA port pin 10 and pin 15 */
GPIO_ResetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

10.2.11 GPIO_WriteBit function

Table 196 describes the GPIO_WriteBit function.

Table 196. GPIO_WriteBit function

Function name	GPIO_WriteBit
Function prototype	void GPIO_WriteBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin, BitAction BitVal)
Behavior description	Sets or clears the selected data port bit.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	GPIO_Pin: port bit to be written. Refer to *GPIO_Pin* for more details on the allowed values for this parameter.
Input parameter3	BitVal: this parameter specifies the value to be written to the selected bit. BitVal must be one of the BitAction enum values: Bit_RESET: to clear the port pin. Bit_SET: to set the port pin.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the GPIOA port pin 15 */
GPIO_WriteBit(GPIOA, GPIO_Pin_15, Bit_SET);
```

10.2.12 GPIO_Write function

Table 197 describes the GPIO_Write function.

Table 197. GPIO_Write function

Function name	GPIO_Write
Function prototype	void GPIO_Write(GPIO_TypeDef* GPIOx, u16 PortVal)
Behavior description	Writes the passed value in the selected data GPIOx port register.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	PortVal: the value to be written to the data port register.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Write data to GPIOA data port */
GPIO_Write(GPIOA, 0x1101);
```

10.2.13 GPIO_PinLockConfig function

Table 198 describes the GPIO_PinLockConfig function.

Table 198. GPIO_PinLockConfig function

Function name	GPIO_PinLockConfig
Function prototype	void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
Behavior description	Locks GPIO pins configuration registers.
Input parameter1	GPIOx: where x can be (A..G) to select the GPIO peripheral.
Input parameter2	GPIO_Pin: port bit to be written. Refer to *GPIO_Pin* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Lock GPIOA Pin0 and Pin1 */
GPIO_PinLockConfig(GPIOA, GPIO_Pin_0 | GPIO_Pin_1);
```

10.2.14 GPIO_EventOutputConfig function

Table 199 describes the GPIO_EventOutputConfig function.

Table 199. GPIO_EventOutputConfig function

Function name	GPIO_EventOuputConfig
Function prototype	void GPIO_EventOutputConfig(u8 GPIO_PortSource, u8 GPIO_PinSource)
Behavior description	Selects the GPIO pin used as Event output.
Input parameter1	GPIO_PortSource: selects the GPIO port to be used as source for Event output. Refer to *GPIO_PortSource* for more details on the allowed values for this parameter.
Input parameter2	GPIO_PinSource: pin for the Event output. This parameter can be GPIO_PinSourcex where x can be (0..15).
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

GPIO_PortSource

This parameter is used to select the GPIO port source used as Event output. See *Table 200* for the values taken by GPIO_PortSource.

Table 200. GPIO_PortSource values

GPIO_PortSource	Description
GPIO_PortSourceGPIOA	GPIOA Selected
GPIO_PortSourceGPIOB	GPIOB Selected
GPIO_PortSourceGPIOC	GPIOC Selected
GPIO_PortSourceGPIOD	GPIOD Selected
GPIO_PortSourceGPIOE	GPIOE Selected

Example:

```
/* Selects the GPIOE pin 5 for EVENT output */
GPIO_EventOutputConfig(GPIO_PortSourceGPIOE, GPIO_PinSource5);
```

10.2.15 GPIO_EventOutputCmd function

Table 201 describes the GPIO_EventOutputCmd function.

Table 201. GPIO_EventOutputCmd function

Function name	GPIO_EventOuputCmd
Function prototype	void GPIO_EventOutputCmd(FunctionalState NewState)
Behavior description	Enables or disables the Event Output.
Input parameter	NewState: new state of the Event output. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable Event Ouput to the GPIOC pin 6 */
GPIO_EventOutputConfig(GPIO_PortSourceGPIOC, GPIO_PinSource6);
GPIO_EventOutputCmd(ENABLE);
```

10.2.16 GPIO_PinRemapConfig function

Table 202 describes the GPIO_PinRemapConfig function.

Table 202. GPIO_PinRemapConfig function

Function name	GPIO_PinRemapConfig
Function prototype	void GPIO_PinRemapConfig(u32 GPIO_Remap, FunctionalState NewState)
Behavior description	Changes the mapping of the specified pin.
Input parameter1	GPIO_Remap: selects the pin to remap. Refer to *GPIO_Remap* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the port pin remapping. This parameter can be set to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

GPIO_Remap

GPIO_Remap parameter is used to change the alternate function mapping. See *Table 203* for the values taken by this parameter.

Table 203. GPIO_Remap values

GPIO_Remap	Description
GPIO_Remap_SPI1	SPI1 Alternate Function mapping
GPIO_Remap_I2C1	I2C1 Alternate Function mapping
GPIO_Remap_USART1	USART1 Alternate Function mapping
GPIO_Remap_USART2	USART2 Alternate Function mapping
GPIO_PartialRemap_USART3	USART3 Partial Alternate Function mapping
GPIO_FullRemap_USART3	USART3 Full Alternate Function mapping
GPIO_PartialRemap_TIM1	TIM1 Partial Alternate Function mapping
GPIO_FullRemap_TIM1	TIM1 Full Alternate Function mapping
GPIO_PartialRemap1_TIM2	TIM2 Partial1 Alternate Function mapping
GPIO_PartialRemap2_TIM2	TIM2 Partial2 Alternate Function mapping
GPIO_FullRemap_TIM2	TIM2 Full Alternate Function mapping
GPIO_PartialRemap_TIM3	TIM3 Partial Alternate Function mapping
GPIO_FullRemap_TIM3	TIM3 Full Alternate Function mapping
GPIO_Remap_TIM4	TIM4 Alternate Function mapping
GPIO_Remap1_CAN	CAN Alternate Function mapping
GPIO_Remap2_CAN	CAN Alternate Function mapping

Table 203. GPIO_Remap values (continued)

GPIO_Remap	Description
GPIO_Remap_PD01	PD01 Alternate Function mapping
GPIO_Remap_TIM5CH4_LSI	LSI connected to TIM5 Channel4 input capture for calibration
GPIO_Remap_ADC1_ETRGINJ	ADC1 External Trigger Injected Conversion mapping
GPIO_Remap_ADC1_ETRGREG	ADC1 External Trigger Regular Conversion mapping
GPIO_Remap_ADC2_ETRGINJ	ADC2 External Trigger Injected Conversion mapping
GPIO_Remap_ADC2_ETRGREG	ADC2 External Trigger Regular Conversion mapping
GPIO_Remap_SWJ_NoJTRST	Full SWJ Enabled (JTAG-DP + SW-DP) but without JTRST
GPIO_Remap_SWJ_JTAGDisable	JTAG-DP Disabled and SW-DP Enabled
GPIO_Remap_SWJ_Disable	Full SWJ Disabled (JTAG-DP + SW-DP)

Example:

```
/* I2C1_SCL on PB.08, I2C1_SDA on PB.09 */
GPIO_PinRemapConfig(GPIO_Remap_I2C1, ENABLE);
```

10.2.17 GPIO_EXTILineConfig function

Table 204 describes the GPIO_EXTILineConfig function.

Table 204. GPIO_EXTILineConfig function

Function name	GPIO_EXTILineConfig
Function prototype	void GPIO_EXTILineConfig(u8 GPIO_PortSource, u8 GPIO_PinSource)
Behavior description	Selects the GPIO pin used as EXTI Line.
Input parameter1	GPIO_PortSource: selects the GPIO port to be used as source for EXTI lines. Refer to *GPIO_PortSource* for more details on the allowed values for this parameter.
Input parameter2	GPIO_PinSource: EXTI line to be configured. This parameter can be GPIO_PinSourcex where x can be (0..15).
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

GPIO_PortSource

This parameter is used to select the GPIO port source used as Event output. See *Table 205* for the values taken by GPIO_PortSource.

Table 205. GPIO_PortSource values

GPIO_PortSource	Description
GPIO_PortSourceGPIOA	GPIOA Selected
GPIO_PortSourceGPIOB	GPIOB Selected
GPIO_PortSourceGPIOC	GPIOC Selected
GPIO_PortSourceGPIOD	GPIOD Selected
GPIO_PortSourceGPIOE	GPIOE Selected
GPIO_PortSourceGPIOF	GPIOF Selected
GPIO_PortSourceGPIOG	GPIOG Selected

Example:

```
/* Selects PB.08 as EXTI Line 8 */
GPIO_EXTILineConfig(GPIO_PortSource_GPIOB, GPIO_PinSource8);
```

11 Inter-integrated circuit (I2C)

The I2C bus interface module is the interface between the microcontroller and the serial I2C bus. It provides both multi-master and slave functions. It controls all I2C bus specific sequencing, protocol, arbitration and timing. It can also perform additional functions such as CRC generation and checking, SMBus and PMBus.

The I2C driver can be used to transmit and receive data through the I2C interface. The status of the executed action is returned by the I2C driver.

Section 11.1: I2C register structure describes the register structure used in the I2C Firmware Library. *Section 11.2: Firmware library functions* presents the Firmware Library functions.

11.1 I2C register structure

The I2C register structure, *I2C_TypeDef,* is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu16 CR1;
  u16 RESERVED0;
  vu16 CR2;
  u16 RESERVED1;
  vu16 OAR1;
  u16 RESERVED2;
  vu16 OAR2;
  u16 RESERVED3;
  vu16 DR;
  u16 RESERVED4;
  vu16 SR1;
  u16 RESERVED5;
  vu16 SR2;
  u16 RESERVED6;
  vu16 CCR;
  u16 RESERVED7;
  vu16 TRISE;
  u16 RESERVED8;
} I2C_TypeDef;
```

Table 206 gives the list of I²C registers:

Table 206. I²C registers

Register	Description
CR1	I²C Control Register1
CR2	I²C Control Register2
OAR1	I²C Own Address Register1
OAR2	I²C Own Address Register2 (Dual Address)
DR	I²C Data Register
SR1	I²C Status Register1
SR2	I²C Status Register2
CCR	I²C Clock Control Register
TRISE	I²C Rise Time Register

The two I²C peripherals are declared in *stm32f10x_map*.h:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
....
#define I2C1_BASE             (APB1PERIPH_BASE + 0x5400)
#define I2C2_BASE             (APB1PERIPH_BASE + 0x5800)
....
#ifndef DEBUG
...
#ifdef _I2C1
  #define I2C1                ((I2C_TypeDef *) I2C1_BASE)
#endif /*_I2C1 */

#ifdef _I2C2
  #define I2C2                ((I2C_TypeDef *) I2C2_BASE)
#endif /*_I2C2 */
...
#else   /* DEBUG */
...
#ifdef _I2C1
  EXT I2C_TypeDef             *I2C1;
#endif /*_I2C1 */

#ifdef _I2C2
  EXT I2C_TypeDef             *I2C2;
#endif /*_I2C2 */
...
#endif
```

When using the Debug mode, _I2C1 and _I2C2 pointers are initialized in *stm32f10x_lib.c* file.

```
...
#ifdef _I2C1
  I2C1 = (I2C_TypeDef *)  I2C1_BASE;
#endif /*_I2C1 */

#ifdef _I2C2
  I2C2 = (I2C_TypeDef *)  I2C2_BASE;
#endif /*_I2C2 */
...
```

To access the I^2C registers, _I2C, _I2C1 and _I2C2 must be defined in *stm32f10x_conf.h* as follows:

```
...
#define _I2C
#define _I2C1
#define _I2C2
...
```

11.2 Firmware library functions

Table 207 gives the list of the I^2C firmware library functions.

Table 207. I^2C firmware library functions

Function name	Description
I2C_DeInit	Resets the I2Cx peripheral registers to their default reset values.
I2C_Init	Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
I2C_StructInit	Fills each I2C_InitStruct member with its default value.
I2C_Cmd	Enables or disables the specified I^2C peripheral.
I2C_DMACmd	Enables or disables the specified I^2C DMA requests.
I2C_DMALastTransferCmd	Specifies that the next DMA transfer is the last one.
I2C_GenerateSTART	Generates I2Cx communication Start condition.
I2C_GenerateSTOP	Generates I2Cx communication Stop condition.
I2C_AcknowledgeConfig	Enables or disables the specified I^2C acknowledge feature.
I2C_OwnAddress2Config	Configures the specified I^2C own address2.
I2C_DualAddressCmd	Enables or disables the specified I^2C dual addressing mode.
I2C_GeneralCallCmd	Enables or disables the specified I^2C general call feature.
I2C_ITConfig	Enables or disables the specified I^2C interrupts.
I2C_SendData	Sends a data byte through the I2Cx peripheral.
I2C_ReceiveData	Returns the most recent received data by the I2Cx peripheral.

Table 207. I^2C firmware library functions (continued)

Function name	Description
I2C_Send7bitAddress	Transmits the address byte to select the slave device.
I2C_ReadRegister	Reads the specified I^2C register and returns its value.
I2C_SoftwareResetCmd	Enables or disables the specified I^2C software reset.
I2C_SMBusAlertConfig	Drives the SMBAlert pin high or low for the specified I^2C.
I2C_TransmitPEC	Enables or disables the specified I^2C PEC transfer.
I2C_PECPositionConfig	Selects the specified I^2C PEC position.
I2C_CalculatePEC	Enables or disables the PEC value calculation of the transferred bytes.
I2C_GetPEC	Returns the PEC value for the specified I^2C.
I2C_ARPCmd	Enables or disables the specified I^2C ARP.
I2C_StretchClockCmd	Enables or disables the specified I^2C clock stretching.
I2C_FastModeDutyCycleConfig	Selects the specified I^2C fast mode duty cycle.
I2C_GetLastEvent	Returns the last I2Cx event
I2C_CheckEvent	Checks whether the last I2Cx event is equal to the one passed as parameter.
I2C_GetFlagStatus	Checks whether the specified I^2C flag is set or not.
I2C_ClearFlag	Clears the I2Cx's pending flags.
I2C_GetITStatus	Checks whether the specified I^2C interrupt has occurred or not.
I2C_ClearITPendingBit	Clears the I2Cx's interrupt pending bits.

11.2.1 I2C_DeInit function

Table 208 describes the I2C_DeInit function.

Table 208. I2C_DeInit function

Function name	I2C_DeInit
Function prototype	void I2C_DeInit(I2C_TypeDef* I2Cx)
Behavior description	Resets the I2Cx peripheral registers to their default reset values.
Input parameter	I2C*x*: where x can be 1 or 2 to select the I2C peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB1PeriphClockCmd().

Example:

```
/* Deinitialize I2C2 interface*/
I2C_DeInit(I2C2);
```

11.2.2 I2C_Init function

Table 209 describes the I2C_Init function.

Table 209. I2C_Init function

Function name	I2C_Init
Function prototype	void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)
Behavior description	Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
Input parameter1	I2C*x*: where x can be 1 or 2 to select the I2C peripheral.
Input parameter2	I2C_InitStruct: pointer to a I2C_InitTypeDef structure that contains the configuration information for the specified I2C peripheral. Refer to the *I2C_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_InitTypeDef structure

The I2C_InitTypeDef structure is defined in the *stm32f10x_i2c.h* file:

```
typedef struct
{
  u16 I2C_Mode;
  u16 I2C_DutyCycle;
  u16 I2C_OwnAddress1;
  u16 I2C_Ack;
  u16 I2C_AcknowledgedAddress;
  u32 I2C_ClockSpeed;
} I2C_InitTypeDef;
```

I2C_Mode

I2C_Mode is used to configure the I2C mode. See *Table 214* for the values taken by this member.

Table 210. I2C_Mode definition

I2C_Mode	Description
I2C_Mode_I2C	I2C is configured in I2C mode
I2C_Mode_SMBusDevice	I2C is configured in SMBus device mode
I2C_Mode_SMBusHost	I2C is configured in SMBus host mode

I2C_DutyCycle

I2C_DutyCycle is used to select the I2C fast mode duty cycle. See *Table 211* for the values taken by this member.

Table 211. I2C_DutyCycle definition

I2C_DutyCycle	Description
I2C_DutyCycle_16_9	I^2C fast mode Tlow/Thigh=16/9
I2C_DutyCycle_2	I^2C fast mode Tlow/Thigh=2

Note: This member is meaningful only when the I^2C operates in Fast mode (working clock speed greater than 100 kHz).

I2C_OwnAddress1

This member is used to configure the first device own address. It can be a 7-bit or 10-bit address.

I2C_Ack

I2C_Ack enables or disables the acknowledgement. See *Table 212* for the values taken by this member.

Table 212. I2C_Ack definition

I2C_Ack	Description
I2C_Ack_Enable	Enables the acknowledgement
I2C_Ack_Disable	Disables the acknowledgement

I2C_AcknowledgedAddress

I2C_AcknowledgedAddress defines whether if 7-bit or 10-bit address is acknowledged. See *Table 213* for the values taken by this member.

Table 213. I2C_AcknowledgedAddress defines

I2C_AcknowledgedAddress	Description
I2C_AcknowledgedAddress_7bit	Acknowledge 7-bit address
I2C_AcknowledgedAddress_10bit	Acknowledge 10-bit address

I2C_ClockSpeed

This member is used to configure the clock frequency. It must be set to a value lower than 400 kHz.

Example:

```
/* Initialize the I2C1 according to the I2C_InitStructure members */
I2C_InitTypeDef  I2C_InitStructure;
I2C_InitStructure.I2C_Mode = I2C_Mode_SMBusHost;
I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStructure.I2C_OwnAddress1 = 0x03A2;
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_InitStructure.I2C_AcknowledgedAddress =
I2C_AcknowledgedAddress_7bit;
I2C_InitStructure.I2C_ClockSpeed = 200000;
I2C_Init(I2C1, &I2C_InitStructure);
```

11.2.3 I2C_StructInit function

Table 214 describes the I2C_StructInit function.

Table 214. I2C_StructInit function

Function name	I2C_StructInit
Function prototype	void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)
Behavior description	Fills each I2C_InitStruct member with its default value.
Input parameter	I2C_InitStruct: pointer to the I2C_InitTypeDef structure to be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The I2C_InitStruct members have the following default values:

Table 215. I2C_InitStruct default values

Member	Default value
I2C_Mode	I2C_Mode_I2C
I2C_DutyCycle	I2C_DutyCycle_2
I2C_OwnAddress1	0
I2C_Ack	I2C_Ack_Disable
I2C_AcknowledgedAddress	I2C_AcknowledgedAddress_7bit
I2C_ClockSpeed	5000

Example:

```
/* Initialize an I2C_InitTypeDef structure */
I2C_InitTypeDef  I2C_InitStructure;
I2C_StructInit(&I2C_InitStructure);
```

11.2.4 I2C_Cmd function

Table 216 describes the I2C_Cmd function.

Table 216. I2C_Cmd function

Function name	I2C_Cmd
Function prototype	void I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I2C peripheral.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I2Cx peripheral. This parameter can be set to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable I2C1 peripheral */
I2C_Cmd(I2C1, ENABLE);
```

11.2.5 I2C_DMACmd function

Table 217 describes the I2C_DMACmd function.

Table 217. I2C_DMACmd function

Function name	I2C_DMACmd
Function prototype	I2C_DMACmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I2C DMA requests.
Input parameter1	I2C*x*: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I^2C DMA transfer. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable I2C2 DMA transfer */
I2C_DMACmd(I2C2, ENABLE);
```

11.2.6 I2C_DMALastTransferCmd function

Table 218 describes the I2C_DMALastTransferCmd function.

Table 218. I2C_DMALastTransferCmd function

Function name	I2C_DMALastTransferCmd
Function prototype	I2C_DMALastTransferCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Specifies that the next DMA transfer is the last one.
Input parameter1	I2C*x*: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I^2C DMA last transfer. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Specify that the next I2C2 DMA transfer is the last one */
I2C_DMALastTransferCmd(I2C2, ENABLE);
```

11.2.7 I2C_GenerateSTART function

Table 219 describes the I2C_GenerateSTART function.

Table 219. I2C_GenerateSTART function

Function name	I2C_GenerateSTART
Function prototype	void I2C_GenerateSTART(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Generates I2Cx communication Start condition.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I2C Start condition generation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a Start condition on I2C1 */
I2C_GenerateSTART(I2C1, ENABLE);
```

11.2.8 I2C_GenerateSTOP function

Table 220 describes the I2C_GenerateSTOP function.

Table 220. I2C_GenerateSTOP function

Function name	I2C_GenerateSTOP
Function prototype	void I2C_GenerateSTOP(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Generates I2Cx communication Stop condition.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I^2C Stop condition generation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a Stop condition on I2C2 */
I2C_GenerateSTOP(I2C2, ENABLE);
```

11.2.9 I2C_AcknowledgeConfig function

Table 221 describes the I2C_AcknowledgeConfig function.

Table 221. I2C_AcknowledgeConfig function

Function name	I2C_AcknowledgeConfig
Function prototype	void I2C_AcknowledgeConfig(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I^2C acknowledge feature.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I^2C Acknowledgement. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the I2C1 Acknowledgement */
I2C_AcknowledgeConfig(I2C1, ENABLE);
```

11.2.10 I2C_OwnAddress2Config function

Table 222 describes the I2C_OwnAddress2Config function.

Table 222. I2C_OwnAddress2Config function

Function name	I2C_OwnAddress2Config
Function prototype	void I2C_OwnAddress2Config(I2C_TypeDef* I2Cx, u8 Address)
Behavior description	Configures the specified I^2C own address2.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	Address: specifies the 7-bit I^2C own address2.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the I2C1 own address2 to 0x38 */
I2C_OwnAddress2Config(I2C1, 0x38);
```

11.2.11 I2C_DualAddressCmd function

Table 223 describes the I2C_DualAddressCmd function.

Table 223. I2C_DualAddressCmd function

Function name	I2C_DualAddressCmd
Function prototype	void I2C_DualAddressCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I^2C dual addressing mode.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I^2C dual addressing mode. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the I2C2 dual addressing mode*/
I2C_DualAdressCmd(I2C2, ENABLE);
```

11.2.12 I2C_GeneralCallCmd function

Table 224 describes the I2C_GeneralCallCmd function.

Table 224. I2C_GeneralCallCmd function

Function name	I2C_GeneralCallCmd
Function prototype	void I2C_GeneralCallCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I^2C general call feature.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I^2C general call. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the I2C1 general call feature */
I2C_GeneralCallCmd(I2C1, ENABLE);
```

11.2.13 I2C_ITConfig function

Table 225 describes the I2C_ITConfig function.

Table 225. I2C_ITConfig function

Function name	I2C_ITConfig
Function prototype	void I2C_ITConfig(I2C_TypeDef* I2Cx, u16 I2C_IT, FunctionalState NewState)
Behavior description	Enables or disables the specified I^2C interrupts.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_IT: I^2C interrupts sources to be enabled or disabled. Refer to *I2C_IT* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the specified I^2C interrupts. This parameter can be set to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_IT

This parameter enables or disables I^2C interrupts. One or a combination of the following values can be used:

Table 226. I2C_IT values

I2C_IT	Description
I2C_IT_BUF	Buffer interrupt mask
I2C_IT_EVT	Event interrupt mask
I2C_IT_ERR	Error interrupt mask

Example:

```
/* Enable I2C2 event and buffer interrupts */
I2C_ITConfig(I2C2, I2C_IT_BUF | I2C_IT_EVT, ENABLE);
```

11.2.14 I2C_SendData function

Table 227 describes the I2C_SendData function.

Table 227. I2C_SendData function

Function name	I2C_SendData
Function prototype	void I2C_SendData(I2C_TypeDef* I2Cx, u8 Data)
Behavior description	Sends a data byte through the I2Cx peripheral.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	Data: byte to be transmitted.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Transmit 0x5D byte on I2C2 */
I2C_SendData(I2C2, 0x5D);
```

11.2.15 I2C_ReceiveData function

Table 228 describes the I2C_ReceiveData function.

Table 228. I2C_ReceiveData function

Function name	I2C_ReceiveData
Function prototype	u8 I2C_ReceiveData(I2C_TypeDef* I2Cx)
Behavior description	Returns the most recent received data by the I2Cx peripheral.
Input parameter	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Output parameter	None
Return parameter	Received byte.
Required preconditions	None
Called functions	None

Example:

```
/* Read the received byte on I2C1 */
u8 ReceivedData;
ReceivedData = I2C_ReceiveData(I2C1);
```

11.2.16 I2C_Send7bitAddress function

Table 229 describes the I2C_Send7bitAddress function.

Table 229. I2C_Send7bitAddress function

Function name	I2C_Send7bitAddress
Function prototype	void I2C_Send7bitAddress(I2C_TypeDef* I2Cx, u8 Address, u8 I2C_Direction)
Behavior description	Transmits the address byte to select the slave device.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	Address: slave address to be transmitted.
Input parameter3	I2C_Direction: specifies whether the I^2C device will act as a transmitter or a receiver. Refer to *I2C_Direction* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_Direction

This parameter configures the I^2C interface in transmitter or receiver mode (see *Table 230*).

Table 230. I2C_Direction

I2C_Direction	Description
I2C_Direction_Transmitter	Selects transmission direction
I2C_Direction_Receiver	Selects receive direction

Example:

```
/* Send, as transmitter, the Slave device address 0xA8 in 7-bit
addressing mode in I2C1 */
I2C_Send7bitAddress(I2C1, 0xA8, I2C_Direction_Transmitter);
```

11.2.17 I2C_ReadRegister function

Table 231 describes the I2C_ReadRegister function.

Table 231. I2C_ReadRegister function

Function name	I2C_ReadRegister
Function prototype	u16 I2C_ReadRegister(I2C_TypeDef* I2Cx, u8 I2C_Register)
Behavior description	Reads the specified I2C register and returns its value.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_Register: register to be read. Refer to *I2C_Register* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The value of the read register. (1)
Required preconditions	None
Called functions	None

1. Some flags could be cleared when the register is read.

I2C_Register

The list of the I^2C registers that can be read by issuing a I2C_ReadRegister function are listed in *Table 232*.

Table 232. Readable I2C registers

I2C_Register	Description
I2C_Register_CR1	I2C_CR1 register selected for read.
I2C_Register_CR2	I2C_CR2 register selected for read.
I2C_Register_OAR1	I2C_OAR1 register selected for read.
I2C_Register_OAR2	I2C_OAR2 register selected for read.
I2C_Register_DR	I2C_DR register selected for read.
I2C_Register_SR1	I2C_SR1 register selected for read.
I2C_Register_SR2	I2C_SR2 register selected for read.
I2C_Register_CCR	I2C_CCR register selected for read.
I2C_Register_TRISE	I2C_TRISE register selected for read.

Example:

```
/* Return the I2C_CR1 register value of I2C2 peripheral */
u16 RegisterValue;
RegisterValue = I2C_ReadRegister(I2C2, I2C_Register_CR1);
```

11.2.18 I2C_SoftwareResetCmd function

Table 233 describes the I2C_SoftwareResetCmd function.

Table 233. I2C_SoftwareResetCmd function

Function name	I2C_SoftwareResetCmd
Function prototype	I2C_SoftwareResetCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I^2C software reset.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I^2C software reset. This parameter can be set to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Put under reset the I2C1 peripheral */
I2C_SoftwareResetCmd(I2C1, ENABLE);
```

11.2.19 I2C_SMBusAlertConfig function

Table 234 describes the I2C_SMBusAlertConfig function.

Table 234. I2C_SMBusAlertConfig function

Function name	I2C_SMBusAlertConfig
Function prototype	void I2C_SMBusAlertConfig(I2C_TypeDef* I2Cx, u16 I2C_SMBusAlert)
Behavior description	Drives the SMBusAlert pin High or Low for the specified I^2C.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_SMBusAlert: SMBAlert pin level. Refer to *I2C_SMBusAlert* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_SMBusAlert

This parameter selects the SMBusAlert pin active level (see *Table 235*).

Table 235. I2C_SMBusAlert values

I2C_SMBusAlert	Description
I2C_SMBusAlert_Low	SMBAlert pin driven Low
I2C_SMBusAlert_High	SMBAlert pin driven High

Example:

```
/* Let the I2C2 SMBusAlert pin High */
I2C_SMBusAlertConfig(I2C2, I2C_SMBusAlert_High);
```

11.2.20 I2C_TransmitPEC function

Table 236 describes the I2C_TransmitPEC function.

Table 236. I2C_TransmitPEC function

Function name	I2C_TransmitPEC
Function prototype	I2C_TransmitPEC(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I^2C PEC transfer.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I2C PEC transfer. This parameter can be set to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the I2C1 PEC transfer */
I2C_TransmitPEC(I2C1, ENABLE);
```

11.2.21 I2C_PECPositionConfig function

Table 237 describes the I2C_PECPositionConfig function.

Table 237. I2C_PECPositionConfig function

Function name	I2C_PECPositionConfig
Function prototype	void I2C_PECPositionConfig(I2C_TypeDef* I2Cx, u16 I2C_PECPosition)
Behavior description	Selects the specified I2C PEC position.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_PECPosition: PEC position. Refer to *I2C_PECPosition* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_PECPosition

This parameter selects the PEC position (see *Table 238*).

Table 238. I2C_PECPosition values

I2C_PECPosition	Description
I2C_PECPosition_Next	PEC bit indicates that the next byte is PEC
I2C_PECPosition_Current	PEC bit indicates that current byte is PEC

Example:

```
/* Configure the PEC bit to indicvates that the next byte in shift
register is PEC for I2C2 */
I2C_PECPositionConfig(I2C2, I2C_PECPosition_Next);
```

11.2.22 I2C_CalculatePEC function

Table 239 describes the I2C_CalculatePEC function.

Table 239. I2C_CalculatePEC function

Function name	I2C_CalculatePEC
Function prototype	void I2C_CalculatePEC(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the PEC calculation for the transferred bytes.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the PEC value calculation. This parameter can be ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the PEC calculation for the transfered bytes from I2C2 */
I2C_CalculatePEC(I2C2, ENABLE);
```

11.2.23 I2C_GetPEC function

Table 240 describes the I2C_GetPEC function.

Table 240. I2C_GetPEC function

Function name	I2C_GetPEC
Function prototype	u8 I2C_GetPEC(I2C_TypeDef* I2Cx)
Behavior description	Returns the PEC value for the specified I^2C interface
Input parameter	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Output parameter	None
Return parameter	The PEC value.
Required preconditions	None
Called functions	None

Example:

```
/* Returns the I2C2 PEC value */
u8 PECValue;
PECValue = I2C_GetPEC(I2C2);
```

11.2.24 I2C_ARPCmd function

Table 241 describes the I2C_ARPCmd function.

Table 241. I2C_ARPCmd function

Function name	I2C_ARPCmd
Function prototype	void I2C_ARPCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I^2C ARP.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the I2C xARP. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the I2C1 ARP feature */
I2C_ARPCmd(I2C1, ENABLE);
```

11.2.25 I2C_StretchClockCmd function

Table 242 describes the I2C_StretchClockCmd function.

Table 242. I2C_StretchClockCmd function

Function name	I2C_StretchClockCmd
Function prototype	void I2C_StretchClockCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
Behavior description	Enables or disables the specified I2C Clock stretching.
Input parameter1	I2Cx: where x can be 1or 2 to select the I^2C peripheral.
Input parameter2	NewState: new state of the Clock stretching. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the I2C2 clock stretching */
I2C_StretchClockCmd(I2C2, ENABLE);
```

11.2.26 I2C_FastModeDutyCycleConfig function

Table 243 describes the I2C_FastModeDutyCycleConfig function.

Table 243. I2C_FastModeDutyCycleConfig function

Function name	I2C_FastModeDutyCycleConfig
Function prototype	void I2C_FastModeDutyCycleConfig(I2C_TypeDef* I2Cx, u16 I2C_DutyCycle)
Behavior description	Selects the specified I^2C fast mode duty cycle.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_DutyCycle: fast mode duty cycle. Refer to *I2C_DutyCycle* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_DutyCycle

This parameter configures the I2C fast mode duty cycle (see *Table 244*).

Table 244. I2C_DutyCycle

I2C_DutyCycle	Description
I2C_DutyCycle_2	I2C fast mode Tlow/Thigh=2
I2C_DutyCycle_16_9	I2C fast mode Tlow/Thigh=16/9

Example:

```
/* Set the fast mode duty cyle to 16/9 for I2C2 */
I2C_FastModeDutyCycleConfig(I2C2, I2C_DutyCycle_16_9);
```

11.2.27 I2C_GetLastEvent function

Table 245 describes the I2C_GetLastEvent function.

Table 245. I2C_GetLastEvent function

Function name	I2C_GetLastEvent
Function prototype	u32 I2C_GetLastEvent(I2C_TypeDef* I2Cx)
Behavior description	Returns the last I2Cx event
Input parameter	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Output parameter	None
Return parameter	last I2Cx event
Required preconditions	None
Called functions	None

Example:

```
/* Get last I2C1 event */
u32 Event;
Event = I2C_GetLastEvent(I2C1);
```

11.2.28 I2C_CheckEvent function

Table 246 describes the I2C_CheckEvent function.

Table 246. I2C_CheckEvent function

Function name	I2C_CheckEvent
Function prototype	ErrorStatus I2C_CheckEvent(I2C_TypeDef* I2Cx, u32 I2C_EVENT)
Behavior description	Checks whether the last I2Cx event is equal to the one passed as parameter.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_EVENT: specifies the event to be checked. Refer to *I2C_EVENT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	An ErrorStatus enumeration value: SUCCESS: Last event is equal to the I2C_EVENT ERROR: Last event is different from the I2C_EVENT
Required preconditions	None
Called functions	None

I2C_EVENT

The events that can be checked by issuing an I2C_CheckEvent function are listed in *Table 247*.

Table 247. I2C_Event

I2C_EVENT	Description
I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_BYTE_RECEIVED	EV2
I2C_EVENT_SLAVE_BYTE_TRANSMITTED	EV3
I2C_EVENT_SLAVE_ACK_FAILURE	EV3-1
I2C_EVENT_SLAVE_STOP_DETECTED	EV4
I2C_EVENT_MASTER_MODE_SELECT	EV5
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED	EV6
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED	EV6
I2C_EVENT_MASTER_BYTE_RECEIVED	EV7
I2C_EVENT_MASTER_BYTE_TRANSMITTING	EV8
I2C_EVENT_MASTER_BYTE_TRANSMITTED	EV8-2
I2C_EVENT_MASTER_MODE_ADDRESS10	EV9

Example:

```
/* Check if the event happen on I2C1 is equal to
I2C_EVENT_MASTER_BYTE_RECEIVED */
ErrorStatus  Status;
Status = I2C_CheckEvent(I2C1, I2C_EVENT_MSTER_BYTE_RECEIVED);
```

11.2.29 I2C_GetFlagStatus function

Table 248 describes the I2C_GetFlagStatus function.

Table 248. I2C_GetFlagStatus function

Function name	I2C_GetFlagStatus
Function prototype	FlagStatus I2C_GetFlagStatus(I2C_TypeDef* I2Cx, u32 I2C_FLAG)
Behavior description	Checks whether the specified I2C flag is set or not.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I2C peripheral.
Input parameter2	I2C_FLAG: specifies the flag to be checked Refer to *I2C_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of I2C_FLAG (SET or RESET).(1)
Required preconditions	None
Called functions	None

1. Some flags could be cleared when the register is read.

I2C_FLAG

The I2C flags that can be checked by issuing an I2C_GetFlagStatus function are listed in *Table 249*.

Table 249. I2C_FLAG definition

I2C_FLAG	Description
I2C_FLAG_DUALF	Dual flag (Slave mode)
I2C_FLAG_SMBHOST	SMBus host header (Slave mode)
I2C_FLAG_SMBDEFAULT	SMBus default header (Slave mode)
I2C_FLAG_GENCALL	General call header flag (Slave mode)
I2C_FLAG_TRA	Transmitter/Receiver flag
I2C_FLAG_BUSY	Bus busy flag
I2C_FLAG_MSL	Master/Slave flag
I2C_FLAG_SMBALERT	SMBus Alert flag
I2C_FLAG_TIMEOUT	Timeout or Tlow error flag
I2C_FLAG_PECERR	PEC error in reception flag
I2C_FLAG_OVR	Overrun/Underrun flag (Slave mode)
I2C_FLAG_AF	Acknowledge failure flag

Table 249. I2C_FLAG definition (continued)

I2C_FLAG	Description
I2C_FLAG_ARLO	Arbitration lost flag (Master mode)
I2C_FLAG_BERR	Bus error flag
I2C_FLAG_TXE	Data register empty flag (Transmitter)
I2C_FLAG_RXNE	Data register not empty (Receiver) flag
I2C_FLAG_STOPF	Stop detection flag (Slave mode)
I2C_FLAG_ADD10	10-bit header sent flag (Master mode)
I2C_FLAG_BTF	Byte transfer finished flag
I2C_FLAG_ADDR	Address sent flag (Master mode) “ADSL” Address matched flag (Slave mode)”ENDAD”
I2C_FLAG_SB	Start bit flag (Master mode)

Note: Only bits[27:0] are used by the I2C_GetFlagStatus function to return the selected flag status. This value corresponds to the flag position in the calculated register which contains the two I2C status register I2C_SR1 and I2C_SR2.

Example:

```
/* Return the I2C_FLAG_AF flag state of I2C2 peripheral */
Flagstatus Status;
Status = I2C_GetFlagStatus(I2C2, I2C_FLAG_AF);
```

11.2.30 I2C_ClearFlag function

Table 250 describes the I2C_ClearFlag function.

Table 250. I2C_ClearFlag function

Function name	I2C_ClearFlag
Function prototype	void I2C_ClearFlag(I2C_TypeDef* I2Cx, u32 I2C_FLAG)
Behavior description	Clears the I2Cx’s pending flags.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_FLAG: specifies the flag to clear. This parameter can be any combination of the values defined in *I2C_FLAG*. Refer to *I2C_FLAG* for more details on the allowed values for this parameter. Note: DUALF, SMBHOST, SMBDEFAULT, GENCALL, TRA, BUSY, MSL, TXE and RXNE flags cannot be cleared by issuing this function
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_FLAG

The I2C flags that can be cleared by issuing an I2C_ClearFlag function are listed in *Table 251*.

Table 251. I2C_FLAG definition

I2C_FLAG	Description
I2C_FLAG_SMBALERT	SMBus Alert flag
I2C_FLAG_TIMEOUT	Timeout or Tlow error flag
I2C_FLAG_PECERR	PEC error in reception flag
I2C_FLAG_OVR	Overrun/Underrun flag (Slave mode)
I2C_FLAG_AF	Acknowledge failure flag
I2C_FLAG_ARLO	Arbitration lost flag (Master mode)
I2C_FLAG_BERR	Bus error flag

Note: 1 STOPF (STOP detection) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetFlagStatus()`) followed by a write operation to the I2C_CR1 register (`I2C_Cmd()` to re-enable the I2C peripheral).

2 ADD10 (10-bit header sent) is cleared by a software sequence: a read operation to the I2C_SR1 (`I2C_GetFlagStatus()`) followed by writing the second byte of the address to the DR register.

3 BTF (Byte Transfer Finished) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetFlagStatus()`) followed by a read/write to I2C_DR register (`I2C_SendData()`).

4 ADDR (Address sent) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetFlagStatus()`) followed by a read operation to the I2C_SR2 register ((void)(I2Cx->SR2)).

5 SB (Start Bit) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetFlagStatus()`) followed by a write operation to the I2C_DR reigister (`I2C_SendData()`).

Example:

```
/* Clear the SMBus Alert flag on I2C2 */
I2C_ClearFlag(I2C2, I2C_FLAG_SMBALERT);
```

11.2.31 I2C_GetITStatus function

Table 252 describes the I2C_GetITStatus function.

Table 252. I2C_GetITStatus function

Function name	I2C_GetITStatus
Function prototype	ITStatus I2C_GetITStatus(I2C_TypeDef* I2Cx, u32 I2C_IT)
Behavior description	Checks whether the specified I^2C interrupt has occurred or not.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I^2C peripheral.
Input parameter2	I2C_IT: specifies the interrupt source to check. Refer to *I2C_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	New state of I2C_IT (SET or RESET)(1)
Required preconditions	None
Called functions	None

1. Some flags could be cleared when the register is read.

I2C_IT

The I2C_IT parameter is used to select the I^2C interrupt flags that can be checked by issuing an I2C_GetITStatus function (see *Table 253*).

Table 253. I2C_IT definition

I2C_IT	Description
I2C_IT_SMBALERT	SMBus Alert flag
I2C_IT_TIMEOUT	Timeout or Tlow error flag
I2C_IT_PECERR	PEC error in reception flag
I2C_IT_OVR	Overrun/Underrun flag (Slave mode)
I2C_IT_AF	Acknowledge failure flag
I2C_IT_ARLO	Arbitration lost flag (Master mode)
I2C_IT_BERR	Bus error flag
I2C_IT_TXE	Data register empty flag (Transmitter)
I2C_IT_RXNE	Data register not empty (Receiver) flag
I2C_IT_STOPF	Stop detection flag (Slave mode)
I2C_IT_ADD10	10-bit header sent flag (Master mode)
I2C_IT_BTF	Byte transfer finished flag
I2C_IT_ADDR	Address sent flag (Master mode) “ADSL” Address matched flag (Slave mode) ”ENDAD”
I2C_IT_SB	Start bit flag (Master mode)

Example:

```
/* Return the I2C_IT_OVR flag state of I2C1 peripheral */
ITstatus Status;
Status = I2C_GetITStatus(I2C1, I2C_IT_OVR);
```

11.2.32 I2C_ClearITPendingBit function

Table 254 describes the I2C_ClearITPendingBit function.

Table 254. I2C_ClearITPendingBit function

Function name	I2C_ClearITPendingBit
Function prototype	void I2C_ClearITPendingBit(I2C_TypeDef* I2Cx, u32 I2C_IT)
Behavior description	Clears the I2Cx's interrupt pending bits.
Input parameter1	I2Cx: where x can be 1 or 2 to select the I2C peripheral.
Input parameter2	I2C_IT: specifies the interrupt pending bit to clear. This parameter can be any combination of the values defined in *I2C_IT*. Refer to *I2C_IT* for more details on the allowed values for this parameter. Note: DUALF, SMBHOST, SMBDEFAULT, GENCALL, TRA, BUSY, MSL, TXE and RXNE flags cannot be cleared by issuing this function.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

I2C_IT

The I2C_IT parameter is used to select the I2C interrupt pending flags that can be cleared by issuing an I2C_ClearITPendingBit function (see *Table 255*).

Table 255. I2C_IT definition

I2C_IT	Description
I2C_IT_SMBALERT	SMBus Alert flag
I2C_IT_TIMEOUT	Timeout or Tlow error flag
I2C_IT_PECERR	PEC error in reception flag
I2C_IT_OVR	Overrun/Underrun flag (Slave mode)
I2C_IT_AF	Acknowledge failure flag
I2C_IT_ARLO	Arbitration lost flag (Master mode)
I2C_IT_BERR	Bus error flag

Note: 1 STOPF (STOP detection) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetITStatus()`) followed by a write operation to the I2C_CR1 register (`I2C_Cmd()` to re-enable the I2C peripheral).

2 ADD10 (10-bit header sent) is cleared by a software sequence: a read operation to the I2C_SR1 (`I2C_GetITStatus()`) followed by writing the second byte of the address into the I2C_DR register.

3 BTF (Byte Transfer Finished) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetITStatus()`) followed by a read/write to the I2C_DR register (`I2C_SendData()`).

4 ADDR (Address sent) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetITStatus()`) followed by a read operation to the I2C_SR2 register ((void)(I2Cx->SR2)).

5 SB (Start Bit) is cleared by a software sequence: a read operation to the I2C_SR1 register (`I2C_GetITStatus()`) followed by a write operation to the I2C_DR reigister (`I2C_SendData()`).

Example:

```
/* Clear the Timeout interrupt opending bit on I2C2 */
I2C_ClearITPendingBit(I2C2, I2C_IT_TIMEOUT);
```

12 Independent watchdog (IWDG)

The Independent watchdog (IWDG) can be used to resolve processor malfunctions due to hardware or software failures. It can operate either in stop or in standby mode.

Section 12.1: IWDG register structure describes the data structures used in the IWDG Firmware Library. *Section 12.2: Firmware library functions* presents the Firmware Library functions.

12.1 IWDG register structure

The IWDG register structure, *IWDG_TypeDef,* is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 KR;
  vu32 PR;
  vu32 RLR;
  vu32 SR;
} IWDG_TypeDef;
```

Table 256 gives the list of IWDG registers.

Table 256. IWDG registers

Register	Description
KR	IWDG Key Register
PR	IWDG Prescaler Register
RLR	IWDG Reload Register
SR	IWDG Status Register

The IWDG peripheral is declared in *stm32f10x_map.h*:

```
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)

#define IWDG_BASE             (APB1PERIPH_BASE + 0x3000)

#ifndef DEBUG
...
#ifdef _IWDG
  #define IWDG                  ((IWDG_TypeDef *) IWDG_BASE)
#endif /*_IWDG */
...
#else   /* DEBUG */
...
#ifdef _IWDG
  EXT IWDG_TypeDef            *IWDG;
#endif /*_IWDG */
...
#endif
```

When using the Debug mode, the IWDG pointer is initialized in *stm32f10x_lib.c*:

```
#ifdef _IWDG
IWDG = (IWDG_TypeDef *) IWDG_BASE;
#endif /*_IWDG */
```

To access the independent watchdog registers, _IWDG must be defined in *stm32f10x_conf.h* as follows:

```
#define _IWDG
```

12.2 Firmware library functions

Table 257 gives the list of IWDG firmware library functions.

Table 257. IWDG firmware library functions

Function name	Description
IWDG_WriteAccessCmd	Enables or disables write access to IWDG_PR and IWDG_RLR registers.
IWDG_SetPrescaler	Sets IWDG Prescaler value
IWDG_SetReload	Sets IWDG Reload value
IWDG_ReloadCounter	Reloads IWDG counter with value defined in the reload register
IWDG_Enable	Enables IWDG
IWDG_GetFlagStatus	Checks whether the specified IWDG flag is set or not

12.2.1 IWDG_WriteAccessCmd function

Table 258 describes the IWDG_WriteAccessCmd function.

Table 258. IWDG_WriteAccessCmd function

Function name	IWDG_WriteAccessCmd
Function prototype	void IWDG_WriteAccessCmd(u16 IWDG_WriteAccess)
Behavior description	Enables or disables write access to IWDG_PR and IWDG_RLR registers.
Input parameter	IWDG_WriteAccess: new state of write access to IWDG_PR and IWDG_RLR registers. Refer to *IWDG_WriteAccess* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

IWDG_WriteAccess

This parameter enables or disables write access to IWDG_PR and IWDG_RLR registers (see *Table 259*).

Table 259. IWDG_WriteAccess definition

IWDG_WriteAccess	Description
IWDG_WriteAccess_Enable	Write access to IWDG_PR and IWDG_RLR registers enabled
IWDG_WriteAccess_Disable	Write access to IWDG_PR and IWDG_RLR registers disabled

Example:

```
/* Enable write access to IWDG_PR and IWDG_RLR registers */
IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
```

12.2.2 IWDG_SetPrescaler function

Table 260 describes the IWDG_SetPrescaler function.

Table 260. IWDG_SetPrescaler function

Function name	IWDG_SetPrescaler
Function prototype	void IWDG_SetPrescaler(u8 IWDG_Prescaler)
Behavior description	Sets IWDG Prescaler value.
Input parameter	IWDG_Prescaler: IWDG Prescaler value. Refer to *IWDG_Prescaler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

IWDG_Prescaler

This parameter selects the IWDG prescaler (see *Table 261*).

Table 261. IWDG_Prescaler definition

IWDG_Prescaler	Description
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

Example:

```
/* Set IWDG prescaler to 8 */
IWDG_SetPrescaler(IWDG_Prescaler_8);
```

12.2.3 IWDG_SetReload function

Table 262 describes the IWDG_SetReload function.

Table 262. IWDG_SetReload function

Function name	IWDG_SetReload
Function prototype	void IWDG_SetReload(u16 Reload)
Behavior description	Sets IWDG Reload value.
Input parameter	Reload: IWDG Reload value. This parameter must be a number between 0 and 0x0FFF.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set IWDG reload value to 0xFFF */
IWDG_SetReload(0xFFF);
```

12.2.4 IWDG_ReloadCounter function

Table 263 describes the IWDG_ReloadCounter function.

Table 263. IWDG_ReloadCounter function

Function name	IWDG_ReloadCounter
Function prototype	void IWDG_ReloadCounter(void)
Behavior description	Reloads IWDG counter with the value defined in the reload register (write access to IWDG_PR and IWDG_RLR registers disabled).
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Reload IWDG counter */
IWDG_ReloadCounter();
```

12.2.5 IWDG_Enable function

Table 264 describes the IWDG_Enable function.

Table 264. IWDG_Enable function

Function name	IWDG_Enable
Function prototype	void IWDG_Enable(void)
Behavior description	Enables IWDG (write access to IWDG_PR and IWDG_RLR registers disabled).
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable IWDG */
IWDG_Enable();
```

12.2.6 IWDG_GetFlagStatus function

Table 265 describes the IWDG_GetFlagStatus function.

Table 265. IWDG_GetFlagStatus function

Function name	IWDG_GetFlagStatus
Function prototype	`FlagStatus IWDG_GetFlagStatus(u16 IWDG_FLAG)`
Behavior description	Checks whether the specified IWDG flag is set or not.
Input parameter	IWDG_FLAG: flag to be checked. Refer to *IWDG_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of IWDG_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

IWDG_FLAG

The IWDG flags that can be checked by issuing an IWDG_GetFlagStatus function are listed in *Table 266*.

Table 266. IWDG_FLAG definition

IWDG_FLAG	Description
IWDG_FLAG_PVU	Prescaler Value Update on going
IWDG_FLAG_RVU	Reload Value Update on going

Example:

```
/* Test if a prescaler value update is on going */
FlagStatus Status;
Status = IWDG_GetFlagStatus(IWDG_FLAG_PVU);
if(Status == RESET)
{
...
}
else
{
...
}
```

13 Nested vectored interrupt controller (NVIC)

The NVIC driver can be used for several purposes, such as enabling and disabling IRQ interrupts, enabling and disabling individual IRQ channels, and changing IRQ channel priorities.

Section 13.1: NVIC register structure describes the data structures used in the NVIC Firmware Library.*Section 13.2: Firmware library functions* presents the Firmware Library functions.

13.1 NVIC register structure

The NVIC register structure, *NVIC_TypeDef*, is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 ISER[2];
  u32 RESERVED0[30];
  vu32 ICER[2];
  u32 RSERVED1[30];
  vu32 ISPR[2];
  u32 RESERVED2[30];
  vu32 ICPR[2];
  u32 RESERVED3[30];
  vu32 IABR[2];
  u32 RESERVED4[62];
  vu32 IPR[11];
} NVIC_TypeDef;

typedef struct
{
  vuc32 CPUID;
  vu32 ICSR;
  vu32 VTOR;
  vu32 AIRCR;
  vu32 SCR;
  vu32 CCR;
  vu32 SHPR[3];
  vu32 SHCSR;
  vu32 CFSR;
  vu32 HFSR;
  vu32 DFSR;
  vu32 MMFAR;
  vu32 BFAR;
  vu32 AFSR;
} SCB_TypeDef; /* System Control Block Structure */
```

Table 267 gives the list of the NVIC registers.

Table 267. NVIC registers

Register	Description
Enable	Interrupt Set Enable Register
Disable	Interrupt Clear Enable Register
Set	Interrupt Set Pending Register
Clear	Interrupt Clear Pending Register
Active	Interrupt Active Bit Register
Priority	Interrupt Priority Register
CPUID	CPUID Base Register
ICSR	Interrupt Control State Register
VTOR	Vector Table Offset Register
AIRCR	Application Interrupt/Reset Control Register
SCR	System Control Register
CCR	Configuration Control Register
SHPR	System Handlers Priority Register
SHCSR	System Handler Control and State Register
CFSR	Configurable Fault Status Registers
HFSR	Hard Fault Status Register
DFSR	Debug Fault Register
MMFAR	Memory Manage Fault Address Register
BFAR	Bus Fault Address Register

The NVIC peripheral is declared in *stm32f10x_map*.h:

```
...
#define SCS_BASE                ((u32)0xE000E000)

#define NVIC_BASE               (SCS_BASE + 0x0100)
#define SCB_BASE                (SCS_BASE + 0x0D00)
...

#ifndef DEBUG
...
#ifdef _NVIC
  #define NVIC                  ((NVIC_TypeDef *) NVIC_BASE)
  #define SCB                   ((SCB_TypeDef *) SCB_BASE)
#endif /*_NVIC */
...
#else   /* DEBUG */
...
#ifdef _NVIC
  EXT NVIC_TypeDef              *NVIC;
  EXT SCB_TypeDef               *SCB;
#endif /*_NVIC */
...
#endif
```

When using the Debug mode, NVIC and SCB pointers are initialized in *stm32f10x_lib.c* file:

```
#ifdef _NVIC
  NVIC = (NVIC_TypeDef *)  NVIC_BASE;
  SCB = (SCB_TypeDef *)  SCB_BASE;
#endif /*_NVIC */
```

To access the NVIC registers, _NVIC must be defined in stm32f10x_conf.h, as follows:

```
#define _NVIC
```

13.2 Firmware library functions

Table 268 gives the list of the NVIC firmware library functions.

Table 268. NVIC firmware library functions

Function name	Description
NVIC_DeInit	Resets the NVIC peripheral registers to their default reset values.
NVIC_SCBDeInit	Resets the SCB peripheral registers to their default reset values.
NVIC_PriorityGroupConfig	Configures the priority grouping: pre-emption priority and subpriority.
NVIC_Init	Initializes the NVIC peripheral according to the specified parameters in the NVIC_InitStruct.
NVIC_StructInit	Fills each NVIC_InitStruct member with its default value.
NVIC_SETPRIMASK	Enables the PRIMASK priority: Raises the execution priority to 0.
NVIC_RESETPRIMASK	Disables the PRIMASK priority.
NVIC_SETFAULTMASK	Enables the FAULTMASK priority: Raises the execution priority to -1.
NVIC_RESETFAULTMASK	Disables the FAULTMASK priority.
NVIC_BASEPRICONFIG	The execution priority can be changed from 15 (lowest configurable priority) to 1. Writing 0 will disable the execution priority mask.
NVIC_GetBASEPRI	Returns the BASEPRI mask value.
NVIC_GetCurrentPendingIRQChannel	Returns the current pending served IRQ channel identifier.
NVIC_GetIRQChannelPendingBitStatus	Checks whether the specified IRQ Channel pending bit is set or not.
NVIC_SetIRQChannelPendingBit	Sets the NVIC interrupt pending bits.
NVIC_ClearIRQChannelPendingBit	Clears the NVIC interrupt pending bits.
NVIC_GetCurrentActiveHandler	Returns the current active Handler (IRQ Channel and SystemHandler) identifier.

Table 268. NVIC firmware library functions (continued)

Function name	Description
NVIC_GetIRQChannelActiveBitStatus	Checks whether the specified IRQ Channel active bit is set or not.
NVIC_GetCPUID	Returns the ID number, the version number and the implementation details of the Cortex-M3 core.
NVIC_SetVectorTable	Sets the vector table location and offset.
NVIC_GenerateSystemReset	Generate a system reset.
NVIC_GenerateCoreReset	Generate a Core (Core + NVIC) reset.
NVIC_SystemLPConfig	Selects the condition for the system to enter low power mode.
NVIC_SystemHandlerConfig	Enables or disables the specified System Handlers.
NVIC_SystemHandlerPriorityConfig	Configures the specified System Handlers priority.
NVIC_GetSystemHandlerPendingBitStatus	Checks whether the specified System handlers pending bit is set or not.
NVIC_SetSystemHandlerPendingBit	Sets System Handler pending bit.
NVIC_ClearSystemHandlerPendingBit	Clears System Handler pending bit.
NVIC_GetSystemHandlerActiveBitStatus	Checks whether the specified System handlers active bit is set or not.
NVIC_GetFaultHandlerSources	Returns the system fault handlers sources.
NVIC_GetFaultAddress	Returns the address of the location that generated a fault handler.

13.2.1 NVIC_DeInit function

Table 269 describes the NVIC_DeInit function.

Table 269. NVIC_DeInit function

Function name	NVIC_DeInit
Function prototype	void NVIC_DeInit(void)
Behavior description	Resets the NVIC peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Resets the NVIC registers to their default reset value */
NVIC_DeInit();
```

13.2.2 NVIC_SCBDeInit function

Table 270 describes the NVIC_SCBDeInit function.

Table 270. NVIC_SCBDeInit function

Function name	NVIC_SCBDeInit
Function prototype	void NVIC_SCBDeInit(void)
Behavior description	Resets the SCB peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Resets the SCB registers to their default reset value */
NVIC_SCBDeInit();
```

13.2.3 NVIC_PriorityGoupConfig function

Table 271 describes the NVIC_PriorityGoupConfig function.

Table 271. NVIC_PriorityGoupConfig function

Function name	NVIC_PriorityGroupConfig
Function prototype	void NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup)
Behavior description	Configures the priority grouping: pre-emption priority and subpriority.
Input parameter	NVIC_PriorityGroup: priority grouping bits length. Refer to *NVIC_PriorityGroup* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	Priority grouping should be configured only once.
Called functions	None

NVIC_PriorityGroup

This parameter configures the priority grouping bit length (see *Table 272*).

Table 272. NVIC_PriorityGroup

NVIC_PriorityGroup	Description
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

Example:

```
/* Configure the Priority Grouping with 1 bit */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
```

13.2.4 NVIC_Init function

Table 273 describes the NVIC_Init function.

Table 273. NVIC_Init function

Function name	NVIC_Init
Function prototype	void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)
Behavior description	Initializes the NVIC peripheral according to the parameters specified in the NVIC_InitStruct.
Input parameter	NVIC_InitStruct: pointer to a NVIC_InitTypeDef structure that contains the configuration information for the specified NVIC peripheral. Refer to *NVIC_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

NVIC_InitTypeDef structure

The NVIC_InitTypeDef structure is defined in the *stm32f10x_nvic.h* file:

```
typedef struct
{
  u8 NVIC_IRQChannel;
  u8 NVIC_IRQChannelPreemptionPriority;
  u8 NVIC_IRQChannelSubPriority;
  FunctionalState NVIC_IRQChannelCmd;
} NVIC_InitTypeDef;
```

NVIC_IRQChannel

This member specifies the IRQ channel to be enabled or disabled. The list of the IRQ channels is given in *Table 274*.

Table 274. NVIC_IRQChannels

NVIC_IRQChannel	Description
WWDG_IRQChannel	Window WatchDog Interrupt
PVD_IRQChannel	PVD through EXTI Line detection Interrupt
TAMPER_IRQChannel	Tamper Interrupt
RTC_IRQChannel	RTC global Interrupt
FlashItf_IRQChannel	FLASH global Interrupt
RCC_IRQChannel	RCC global Interrupt
EXTI0_IRQChannel	EXTI Line0 Interrupt
EXTI1_IRQChannel	EXTI Line1 Interrupt
EXTI2_IRQChannel	EXTI Line2 Interrupt
EXTI3_IRQChannel	EXTI Line3 Interrupt
EXTI4_IRQChannel	EXTI Line4 Interrupt
DMAChannel1_IRQChannel	DMA Channel1 global Interrupt
DMAChannel2_IRQChannel	DMA Channel2 global Interrupt
DMAChannel3_IRQChannel	DMA Channel3 global Interrupt
DMAChannel4_IRQChannel	DMA Channel4 global Interrupt
DMAChannel5_IRQChannel	DMA Channel5 global Interrupt
DMAChannel6_IRQChannel	DMA Channel6 global Interrupt
DMAChannel7_IRQChannel	DMA Channel7 global Interrupt
ADC_IRQChannel	ADC global Interrupt
USB_HP_CANTX_IRQChannel	USB High Priority or CAN TX Interrupts
USB_LP_CAN_RX0_IRQChannel	USB Low Priority or CAN RX0 Interrupts
CAN_RX1_IRQChannel	CAN RX1 Interrupt
CAN_SCE_IRQChannel	CAN SCE Interrupt
EXTI9_5_IRQChannel	EXTI Line[9:5] Interrupts
TIM1_BRK_IRQChannel	TIM1 Break Interrupt
TIM1_UP_IRQChannel	TIM1 UP Interrupt

Table 274. NVIC_IRQChannels (continued)

NVIC_IRQChannel	Description
TIM1_TRG_COM_IRQChannel	TIM1 Trigger and Commutation Interrupts
TIM1_CC_IRQChannel	TIM1 Capture Compare Interrupt
TIM2_IRQChannel	TIM2 global Interrupt
TIM3_IRQChannel	TIM3 global Interrupt
TIM4_IRQChannel	TIM4 global Interrupt
I2C1_EV_IRQChannel	I2C1 Event Interrupt
I2C1_ER_IRQChannel	I2C1 Error Interrupt
I2C2_EV_IRQChannel	I2C2 Event Interrupt
I2C2_ER_IRQChannel	I2C2 Error Interrupt
SPI1_IRQChannel	SPI1 global Interrupt
SPI2_IRQChannel	SPI2 global Interrupt
USART1_IRQChannel	USART1 global Interrupt
USART2_IRQChannel	USART2 global Interrupt
USART3_IRQChannel	USART3 global Interrupt
EXTI15_10_IRQChannel	EXTI Line[15:10] Interrupts
RTCAlarm_IRQChannel	RTC Alarm through EXTI Line Interrupt
USBWakeUp_IRQChannel	USB WakeUp from suspend through EXTI Line Interrupt
TIM8_BRK_IRQChannel	TIM8 Break Interrupt
TIM8_UP_IRQChannel	TIM8 Update Interrupt
TIM8_TRG_COM_IRQChannel	TIM8 Trigger and Commutation Interrupt
TIM8_CC_IRQChannel	TIM8 Capture Compare Interrupt
ADC3_IRQChannel	ADC3 global Interrupt
FSMC_IRQChannel	FSMC global Interrupt
SDIO_IRQChannel	SDIO global Interrupt
TIM5_IRQChannel	TIM5 global Interrupt
SPI3_IRQChannel	SPI3 global Interrupt
UART4_IRQChannel	UART4 global Interrupt
UART5_IRQChannel	UART5 global Interrupt
TIM6_IRQChannel	TIM6 global Interrupt
TIM7_IRQChannel	TIM7 global Interrupt
DMA2_Channel1_IRQChannel	DMA2 Channel 1 global Interrupt
DMA2_Channel2_IRQChannel	DMA2 Channel 2 global Interrupt
DMA2_Channel3_IRQChannel	DMA2 Channel 3 global Interrupt
DMA2_Channel4_5_IRQChannel	DMA2 Channel 4 and DMA2 Channel 5 global Interrupt

NVIC_IRQChannelPreemptionPriority

This member configures the pre-emption priority for the IRQ channel specified in the NVIC_IRQChannel member. The values taken by this member are listed in *Table 275*.

NVIC_IRQChannelSubPriority

This member configures the subpriority level for the IRQ channel specified in the NVIC_IRQChannel member. The values taken by this member are listed in *Table 275*.

Table 275 gives the allowed values of the pre-emption priority and subpriority according to the Priority Grouping configuration performed by NVIC_PriorityGroupConfig function:

Table 275. Pre-emption priority and subpriority values(1)(2)

NVIC_PriorityGroup	NVIC_IRQChannel PreemptionPriority	NVIC_IRQChannel SubPriority	Description
NVIC_PriorityGroup_0	0	0-15	0 bits for pre-emption priority 4 bits for subpriority
NVIC_PriorityGroup_1	0-1	0-7	1 bits for pre-emption priority 3 bits for subpriority
NVIC_PriorityGroup_2	0-3	0-3	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PriorityGroup_3	0-7	0-1	3 bits for pre-emption priority 1 bits for subpriority
NVIC_PriorityGroup_4	0-15	0	4 bits for pre-emption priority 0 bits for subpriority

1. When PriorityGroup_0 is selected, NVIC_IRQChannelPreemptionPriority member has no effect on the interrupt channel configuration.
2. When PriorityGroup_4 is selected, NVIC_IRQChannelSubPriority member has no effect on the interrupt channel configuration.

NVIC_IRQChannelCmd

This member specifies whether the IRQ channel defined in the *NVIC_IRQChannel* member will be enabled or disabled. This member can be set either to ENABLE or DISABLE.

Example:

```
NVIC_InitTypeDef NVIC_InitStructure;
/* Configure the Priority Grouping with 1 bit */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

/* Enable TIM3 global interrupt with Preemption Priority 0 and Sub
Priority as 2 */
NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStructure(&NVIC_InitStructure);

/* Enable USART1 global interrupt with Preemption Priority 1 and Sub
Priority as 5 */
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQChannel;
```

```
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 5;
NVIC_InitStructure(&NVIC_InitStructure);

/* Enable RTC global interrupt with Preemption Priority 1 and Sub
Priority as 7 */
NVIC_InitStructure.NVIC_IRQChannel = RTC_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 7;
NVIC_InitStructure(&NVIC_InitStructure);

/* Enable EXTI4 interrupt with Preemption Priority 1 and Sub
Priority as 7 */
NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 7;
NVIC_InitStructure(&NVIC_InitStructure);

/* TIM3 interrupt priority is higher than USART1, RTC and EXTI4
interrupts priorities. USART1 interrupt priority is higher than RTC
and EXTI4 interrupts priorities. RTC interrupt priority is higher
than EXTI4 interrupt prioriy. */
```

13.2.5 NVIC_StructInit function

Table 276 describes the NVIC_StructInit function.

Table 276. NVIC_StructInit function

Function name	NVIC_StructInit
Function prototype	void NVIC_StructInit (NVIC_InitTypeDef* NVIC_InitStruct)
Behavior description	Fills each NVIC_InitStruct member with its default value.
Input parameter	NVIC_InitStruct: pointer to a NVIC_InitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The NVIC_InitStruct members have the following default values:

Table 277. NVIC_InitStruct default values

Member	Default value
NVIC_IRQChannel	0x0
NVIC_IRQChannelPreemptionPriority	0
NVIC_IRQChannelSubPriority	0
NVIC_IRQChannelCmd	DISABLE

Example:

```
/* The following example illustrates how to initialize a
NVIC_InitTypeDef structure */
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_StructInit(&NVIC_InitStructure);
```

13.2.6 NVIC_SETPRIMASK function

Table 278 describes the NVIC_SETPRIMASK function.

Table 278. NVIC_SETPRIMASK function(1)(2)(3)

Function name	NVIC_SETPRIMASK
Function prototype	void NVIC_SETPRIMASK(void)
Behavior description	Enables the PRIMASK priority: raises the execution priority to 0.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__SETPRIMASK()

1. This function is coded in assembler.
2. This function only affects the group priority. It has no effect on the sub-priority.
3. Before setting the PRIMASK register, it is recommended to clear it when returning from exception to enable other exceptions.

Example:

```
/* Enable the PRIMASK priority */
NVIC_SETPRIMASK();
```

13.2.7 NVIC_RESETPRIMASK function

Table 279 describes the NVIC_RESETPRIMASK function.

Table 279. NVIC_RESETPRIMASK function(1)

Function name	NVIC_RESETPRIMASK
Function prototype	void NVIC_RESETPRIMASK(void)
Behavior description	Disables the PRIMASK priority.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__RESETPRIMASK()

1. This function is coded in assembler.

Example:

```
/* Enable the PRIMASK priority */
NVIC_RESETPRIMASK();
```

13.2.8 NVIC_SETFAULTMASK function

Table 280 describes the NVIC_SETFAULTMASK function.

Table 280. NVIC_SETFAULTMASK function(1)(2)(3)

Function name	NVIC_SETFAULTMASK
Function prototype	void NVIC_SETFAULTMASK(void)
Behavior description	Enables the FAULTMASK priority: raises the execution priority to -1.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__SETFAULTMASK()

1. This function is coded in assembler.
2. This function only affects the group priority. It has no effect on the sub-priority.
3. FAULTMASK can only be set when the execution priority is lower than -1. Setting the FaultMask raises the priority of the exception handler to the level of a HardFault. FAULTMASK is cleared automatically on all exception returns except a return from NMI.

Example:

```
/* Enable the FAULTMASK priority */
NVIC_SETFAULTMASK();
```

13.2.9 NVIC_RESETFAULTMASK function

Table 281 describes the NVIC_RESETFAULTMASK function.

Table 281. NVIC_RESETFAULTMASK function(1)

Function name	NVIC_RESETFAULTMASK
Function prototype	void NVIC_SETFAULTMASK(void)
Behavior description	Disables the FAULTMASK priority.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__RESETFAULTMASK()

1. This function is coded in assembler.

Example:

```
/* Disable the FAULTMASK priority */
NVIC_RESETFAULTMASK();
```

13.2.10 NVIC_BASEPRICONFIG function

Table 282 describes the NVIC_BASEPRICONFIG function.

Table 282. NVIC_BASEPRICONFIG function(1)(2)(3)

Function name	NVIC_BASEPRICONFIG
Function prototype	void NVIC_BASEPRICONFIG(u32 NewPriority)
Behavior description	The execution priority can be changed from 15 (lowest configurable priority) to 1. Writing 0 will disable the execution priority mask.
Input parameter	NewPriority: new priority value of the execution priority.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__BASEPRICONFIG()

1. This function is coded in assembler.
2. This function only affects the group priority. It has no effect on the sub-priority.
3. BASEPRI value can be changed from N (lowest configurable priority) to 1. Clearing this register to '0' has no effect on the current priority. A non-zero value will act as a priority mask, affecting the execution priority when the priority defined by BASEPRI is higher than the current executing priority.

Example:

```
/* Mask the execution priority to 10 */
__BASEPRICONFIG(10);
```

13.2.11 NVIC_GetBASEPRI function

Table 283 describes the NVIC_GetBASEPRI function.

Table 283. NVIC_GetBASEPRI function(1)

Function name	NVIC_GetBASEPRI
Function prototype	u32 NVIC_GetBASEPRI(void)
Behavior description	Returns the BASEPRI mask value.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__GetBASEPRI()

1. This function is coded in assembler.

Example:

```
/* Get the execution priority to value */
u32 BASEPRI_Mask = 0;
BASEPRI_Mask = NVIC_GetBASEPRI();
```

13.2.12 NVIC_GetCurrentPendingIRQChannel function

Table 284 describes the NVIC_GetCurrentPendingIRQChannel function.

Table 284. NVIC_GetCurrentPendingIRQChannel function

Function name	NVIC_GetCurrentPendingIRQChannel
Function prototype	u16 NVIC_GetCurrentPendingIRQChannel(void)
Behavior description	Returns the current pending IRQ channel identifier.
Input parameter	None
Output parameter	None
Return parameter	Pending IRQ Channel Identifier.
Required preconditions	None
Called functions	None

Example:

```
/* Get the current pending IRQ channel identifier */
u16 CurrentPendingIRQChannel;
CurrentPendingIRQChannel = NVIC_GetCurrentPendingIRQChannel();
```

13.2.13 NVIC_GetIRQChannelPendingBitStatus function

Table 285 describes the NVIC_GetIRQChannelPendingBitStatus function.

Table 285. NVIC_GetIRQChannelPendingBitStatus function

Function name	NVIC_GetIRQChannelPendingBitStatus
Function prototype	ITStatus NVIC_GetIRQChannelPendingBitStatus(u8 NVIC_IRQChannel)
Behavior description	Checks whether the specified IRQ Channel pending bit is set or not.
Input parameter	NVIC_IRQChannel: interrupt pending bit to check. Refer to *NVIC_IRQChannel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of IRQ Channel pending bit (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the IRQ channel pending bit status of the ADC_IRQChannel */
ITStatus IRQChannelPendingBitStatus;
IRQChannelPendingBitStatus =
NVIC_GetIRQChannelPendingBitStatus(ADC_IRQChannel);
```

13.2.14 NVIC_SetIRQChannelPendingBit function

Table 286 describes the NVIC_SetIRQChannelPendingBitStatus function.

Table 286. NVIC_SetIRQChannelPendingBitStatus function

Function name	NVIC_SetIRQChannelPendingBit
Function prototype	void NVIC_SetIRQChannelPendingBit(u8 NVIC_IRQChannel)
Behavior description	Sets the NVIC interrupt pending bit.
Input parameter	NVIC_IRQChannel: specifies the interrupt pending bit to Set. Refer to *NVIC_IRQChannel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set SPI1 Global interrupt pending bit */
NVIC_SetIRQChannelPendingBit(SPI1_IRQChannel);
```

13.2.15 NVIC_ClearIRQChannelPendingBit function

Table 287 describes the NVIC_ClearIRQChannelPendingBit function.

Table 287. NVIC_ClearIRQChannelPendingBit function

Function name	NVIC_ClearIRQChannelPendingBit
Function prototype	void NVIC_ClearIRQChannelPendingBit(u8 NVIC_IRQChannel)
Behavior description	Clears the NVIC interrupt pending bit.
Input parameter	NVIC_IRQChannel: specifies the interrupt pending bit to clear. Refer to *NVIC_IRQChannel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear ADC IRQ Channel Pending bit */
NVIC_ClearIRQChannelPendingBit(ADC_IRQChannel);
```

13.2.16 NVIC_GetCurrentActiveHandler function

Table 288 describes the NVIC_GetCurrentActiveHandler function.

Table 288. NVIC_GetCurrentActiveHandler function

Function name	NVIC_GetCurrentActiveHandler
Function prototype	u16 NVIC_GetCurrentActiveHandler(void)
Behavior description	Returns the current active Handler (IRQ Channel and SystemHandler) identifier.
Input parameter	None
Output parameter	None
Return parameter	Active Handler Identifier.
Required preconditions	None
Called functions	None

Example:

```
/* Get the current active Handler identifier */
u16 CurrentActiveHandler;
CurrentActiveHandler = NVIC_GetCurrentActiveHandler();
```

13.2.17 NVIC_GetIRQChannelActiveBitStatus function

Table 289 describes the NVIC_GetIRQChannelActiveBitStatus function.

Table 289. NVIC_GetIRQChannelActiveBitStatus function

Function name	NVIC_GetIRQChannelActiveBitStatus
Function prototype	ITStatus NVIC_GetIRQChannelActiveBitStatus(u8 NVIC_IRQChannel)
Behavior description	Checks whether the specified IRQ Channel active bit is set or not.
Input parameter	NVIC_IRQChannel: specifies the interrupt active bit to check. Refer to *NVIC_IRQChannel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of IRQ Channel active bit (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the active IRQ channel status of the ADC_IRQChannel */
ITStatus IRQChannelActiveBitStatus;
IRQChannelActiveBitStatus =
NVIC_GetIRQChannelActiveBitStatus(ADC_IRQChannel);
```

13.2.18 NVIC_GetCPUID function

Table 290 describes the NVIC_GetCPUID function.

Table 290. NVIC_GetCPUID function

Function name	NVIC_GetCPUID
Function prototype	u32 NVIC_GetCPUID(void)
Behavior description	Returns the ID number, version number and the implementation details of the Cortex-M3 core.
Input parameter	None
Output parameter	None
Return parameter	CPU ID.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the CPU ID */
u32 CM3_CPUID;
CM3_CPUID = NVIC_GetCPUID();
```

13.2.19 NVIC_SetVectorTable function

Table 291 describes the NVIC_SetVectorTable function.

Table 291. NVIC_SetVectorTable function

Function name	NVIC_SetVectorTable
Function prototype	void NVIC_SetVectorTable(u32 NVIC_VectTab, u32 Offset)
Behavior description	Sets the vector table location and Offset.
Input parameter1	NVIC_VectTab: specifies if the vector table is in RAM or code memory. Refer to *NVIC_VectTab* for more details on the allowed values for this parameter.
Input parameter2	Offset: Vector Table base offset field. This value must be a multiple of 0x100.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

NVIC_VectTab

This parameter defines the table base address (see *Table 292*).

Table 292. NVIC_VectTab values

NewTableBase	Description
NVIC_VectTab_FLASH	Vector Table is in FLASH
NVIC_VectTab_RAM	Vector Table is in RAM

Example:

```
/* Vector Table is in FLASH at 0x0 */
NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
```

13.2.20 NVIC_GenerateSystemReset function

Table 293 describes the NVIC_GenerateSystemReset function.

Table 293. NVIC_GenerateSystemReset function

Function name	NVIC_GenerateSystemReset
Function prototype	void NVIC_GenerateSystemReset(void)
Behavior description	Generate a system reset.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a system reset */
NVIC_GenerateSystemReset();
```

13.2.21 NVIC_GenerateCoreReset function

Table 294 describes the NVIC_GenerateCoreReset function.

Table 294. NVIC_GenerateCoreReset function

Function name	NVIC_GenerateCoreReset
Function prototype	void NVIC_GenerateCoreReset(void)
Behavior description	Generate a core (core + NVIC) reset.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a core reset */
NVIC_GenerateCoreReset();
```

13.2.22 NVIC_SystemLPConfig function

Table 295 describes the NVIC_SystemLPConfig function.

Table 295. NVIC_SystemLPConfig function

Function name	NVIC_SystemLPConfig
Function prototype	void NVIC_SystemLPConfig(u8 LowPowerMode, FunctionalState NewState)
Behavior description	Selects the condition for the system to enter low power mode.
Input parameter1	LowPowerMode: new mode for the system to enter low power mode. Refer to *LowPowerMode* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the LP condition. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

LowPowerMode

This parameter configures the low power mode of the device (see *Table 296*).

Table 296. LowerPowerMode definition

LowPowerMode	Description
NVIC_LP_SEVONPEND	Wake-up on Pend
NVIC_LP_SLEEPDEEP	Deep Sleep Enable
NVIC_LP_SLEEPONEXIT	Sleep on ISR exit

Example:

```
/* wakeup the system on interrupt pending */
NVIC_SystemLPConfig(SEVONPEND, ENABLE);
```

13.2.23 NVIC_SystemHandlerConfig function

Table 297 describes the NVIC_SystemHandlerConfig function.

Table 297. NVIC_SystemHandlerConfig function

Function name	NVIC_SystemHandlerConfig
Function prototype	void NVIC_SystemHandlerConfig(u32 SystemHandler, FunctionalState NewState)
Behavior description	Enables or disables the specified System Handlers.
Input parameter1	SystemHandler: system handler to be enabled or disabled. Refer to *SystemHandler* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified System Handlers. This parameter can be set to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler to be enabled or disabled (see *Table 298*).

Table 298. SystemHandler types

SystemHandler	Description
SystemHandler_MemoryManage	Memory Manage Handler
SystemHandler_BusFault	Bus Fault Handler
SystemHandler_UsageFault	Usage Fault Handler

The SystemHandler parameter values allow to configure at the same time the NVIC register, the SCB register, and the index bits. The SystemHandler is coded on 23 bits as shown in *Table 299*, *Table 300*, *Table 301*, *Table 302*, *Table 303*, *Table 304*, *Table 305*, *Table 306*, *Table 307*, and *Table 308*.

Example:

```
/* Enable the Memory Manage Handler */
NVIC_SystemHandlerConfig(SystemHandler_MemoryManage, ENABLE);
```

Table 299. SystemHandler definition

System Handler	Bits																					Value
	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
SystemHandler_ NMI (see *Table 300*)	Reserved																		0x1F					0x1F
SystemHandler_ HardFault (see *Table 301*)	Reserved			0	Reserved																			0x0
SystemHandler_ MemoryManage (see *Table 302*)	0	0	1	0x0						0xD				0	0	Res		0x10						0x43430
SystemHandler_ BusFault (see *Table 303*)	1	1	1	1						0xE				1	0	Res		0x11						0x547931
SystemHandler_ UsageFault (see *Table 304*)	-	2	1	0x3						Reserved				2	0	Res		0x12						0x24C232
SystemHandler_ SVCall (see *Table 305*)	Reserved							0x7				0xF		3	1	Reserved								0x1FF40
SystemHandler_ DebugMonitor (see *Table 306*)	Reserved			2				0x8				Reserved		0	2	Reserved								0xA0080
SystemHandler_ PSV (see *Table 307*)	Reserved							0xA				Reserved		2	2			0x1C						0x2829C
SystemHandler_ SysTick (see *Table 308*)	Reserved							0xB				Reserved		3	2			0x1A						0x2C39A

Table 300. SystemHandler_NMI definition

Bits	NMI	
	Registers/Bits	Functions
[4:0]	– IRQControlState – NMIPENDSET[31]	NVIC_SetSystemHandlerPendingBit
5	Not Used	
[7:6]	Not Used	
[9:8]	Not Used	
[13:10]	Not Used	
[17:14]	Not Used	
[19:18]	Not Used	
[21:20]	Not Used	
22	Not Used	

Table 301. SystemHandler_HardFault definition

Bits	Hard Fault	
	Registers/Bits	**Functions**
[4:0]	Not Used	
5	Not Used	
[7:6]	Not Used	
[9:8]	Not Used	
[13:10]	Not Used	
[17:14]	Not Used	
[19:18]	– HardFaultStatus	NVIC_GetFaultHandlerSources
[21:20]		
22	Not Used	

Table 302. SystemHandler_MemoryManage definition

Bits	Memory Manage	
	Registers/Bits	**Functions**
[4:0]	– SysHandlerCtrl – MEMFAULTENA[16]	NVIC_SystemHandlerConfig
5	Not Used	
[7:6]	– SystemPriority[0] – PRI_4[7:0]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	– SysHandlerCtrl – MEMFAULTPENDED[13]	NVIC_GetSystemHandlerPendingBitStatus
[17:14]	– SysHandlerCtrl – MEMFAULTACT[0]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	– ConfigFaultStatus – [7:0]	NVIC_GetFaultHandlerSources
[21:20]		
22	– MemoryManageFaultAddr	NVIC_GetFaultAddress

Table 303. SystemHandler_BusFault definition

Bits	Bus Fault	
	Registers/Bits	Functions
[4:0]	– SysHandlerCtrl – BUSFAULTENA[17]	NVIC_SystemHandlerConfig
5	Not Used	
[7:6] [9:8]	– SystemPriority[0] – PRI_5[15:8]	NVIC_SystemHandlerPriorityConfig
[13:10]	– SysHandlerCtrl – BUSFAULTPENDED[14]	NVIC_GetSystemHandlerPendingBitStatus
[17:14]	– SysHandlerCtrl – BUSFAULTACT[1]	NVIC_GetSystemHandlerActiveBitStatus
[19:18] [21:20]	– ConfigFaultStatus – [15:8]	NVIC_GetFaultHandlerSources
22	– BusFaultAddr	NVIC_GetFaultAddress

Table 304. SystemHandler_UsageFault definition

Bits	Usage Fault	
	Registers/Bits	Functions
[4:0]	– SysHandlerCtrl – USGFAULTENA[18]	NVIC_SystemHandlerConfig
5	Not Used	
[7:6] [9:8]	– SystemPriority[0] – PRI_6[23:16]	NVIC_SystemHandlerPriorityConfig
[13:10]	Not Used	
[17:14]	– SysHandlerCtrl – USGFAULTACT[3]	NVIC_GetSystemHandlerActiveBitStatus
[19:18] [21:20]	– ConfigFaultStatus – [31:16]	NVIC_GetFaultHandlerSources
22	Not Used	

Table 305. SystemHandler_SVCall definition

Bits	SVCall	
	Registers/Bits	Functions
[4:0]	Not Used	
5	Not Used	
[7:6]	– SystemPriority[1] – PRI_11[31:24]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	– SysHandlerCtrl – SVCALLPENDED[15]	NVIC_GetSystemHandlerPendingBitStatus
[17:14]	– SysHandlerCtrl – SVCALLACT[7]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	Not Used	
[21:20]	Not Used	
22	Not Used	

Table 306. SystemHandler_DebugMonitor definition

Bits	Debug Monitor	
	Registers/Bits	Functions
[4:0]	Not Used	
5	Not Used	
[7:6]	– SystemPriority[2] – PRI_12[7:0]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	Not Used	
[17:14]	– SysHandlerCtrl – MONITORACT[8]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	– DebugFaultStatus	NVIC_GetFaultHandlerSources
[21:20]		
22	Not Used	

Table 307. SystemHandler_PSV definition

Bits	PSV	
	Registers/Bits	Functions
[4:0]	– IRQControlState – PENDSVSET[28]	NVIC_SetSystemHandlerPendingBit
	– IRQControlState – PENDSVCLR[27]	NVIC_ClearSystemHandlerPendingBit
5	Not Used	
[7:6]	– SystemPriority[2] – PRI_14[23:16]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	Not Used	
[17:14]	– SysHandlerCtrl – PENDSVACT[10]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	Not Used	
[21:20]	Not Used	
22	Not Used	

Table 308. SystemHandler_SysTick definition

Bits	SysTick	
	Registers/Bits	Functions
[4:0]	– IRQControlState – PENDSTSET[26]	NVIC_SetSystemHandlerPendingBit
	– IRQControlState – PENDSVCLR[25]	NVIC_ClearSystemHandlerPendingBit
5	Not Used	
[7:6]	– SystemPriority[2] – PRI_15[31:24]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	Not Used	
[17:14]	– SysHandlerCtrl – SYSTICKACT[11]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	Not Used	
[21:20]	Not Used	
22	Not Used	

13.2.24 NVIC_SystemHandlerPriorityConfig function

Table 309 describes the NVIC_SystemHandlerPriorityConfig function.

Table 309. NVIC_SystemHandlerPriorityConfig function

Function name	NVIC_SystemHandlerPriorityConfig
Function prototype	void NVIC_SystemHandlerPriorityConfig(u32 SystemHandler, u8 SystemHandlerPreemptionPriority, u8 SystemHandlerSubPriority)
Behavior description	Configures the specified System Handlers priority.
Input parameter1	SystemHandler: system handler to be enabled or disabled. Refer to *SystemHandler* for more details on the allowed values for this parameter.
Input parameter2	SystemHandlerPreemptionPriority: new priority group of the specified system handlers. Refer to *NVIC_IRQChannelPreemptionPriority* for more details on the allowed values for this parameter.
Input parameter3	SystemHandlerSubPriority: new sub priority of the specified system handlers. Refer to *NVIC_IRQChannelSubPriority* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler which will be configured (see *Table 310*).

Table 310. SystemHandler types

SystemHandler	Description
SystemHandler_MemoryManage	Memory Manage Handler
SystemHandler_BusFault	Bus Fault Handler
SystemHandler_UsageFault	Usage Fault Handler
SystemHandler_SVCall	SVCall Handler
SystemHandler_DebugMonitor	Debug Monitor Handler
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	SysTick Handler

Example:

```
/* Enable the Memory Manage Handler */
NVIC_SystemHandlerPriorityConfig(SystemHandler_MemoryManage, 2, 8);
```

13.2.25 NVIC_GetSystemHandlerPendingBitStatus function

Table 311 describes the NVIC_GetSystemHandlerPendingBitStatus function.

Table 311. NVIC_GetSystemHandlerPendingBitStatus function

Function name	NVIC_GetSystemHandlerPendingBitStatus
Function prototype	ITStatus NVIC_GetSystemHandlerPendingBitStatus(u32 SystemHandler)
Behavior description	Checks whether the specified System handlers pending bit is set or not.
Input parameter	SystemHandler: system handler pending bit to check. Refer to *SystemHandler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of System Handler pending bit (SET or RESET).
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler (see *Table 312*).

Table 312. systemHandler types

SystemHandler	Description
SystemHandler_MemoryManage	Memory Manage Handler
SystemHandler_BusFault	Bus Fault Handler
SystemHandler_SVCall	SVCall Handler

Example:

```
/* Check if the Memory Manage Fault has occured */
ITStatus MemoryHandlerStatus;
MemoryHandlerStatus
=NVIC_GetSystemHandlerPendingBitStatus(SystemHandler_MemoryManage);
```

13.2.26 NVIC_SetSystemHandlerPendingBit function

Table 313 describes the NVIC_SetSystemHandlerPendingBit function.

Table 313. NVIC_SetSystemHandlerPendingBit function

Function name	NVIC_SetSystemHandlerPendingBit
Function prototype	void NVIC_SetSystemHandlerPendingBit(u32 SystemHandler)
Behavior description	Sets System Handler pending bit.
Input parameter	SystemHandler: system handler pending bit to be set. Refer to *SystemHandler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler (see *Table 314*).

Table 314. systemHandler types

SystemHandler	Description
SystemHandler_NMI	NMI Handler
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	SysTick Handler

Example:

```
/* Set NMI Pending Bit */
NVIC_SetSystemHandlerPendingBit(SystemHandler_NMI);
```

13.2.27 NVIC_ClearSystemHandlerPendingBit function

Table 315 describes the NVIC_ClearSystemHandlerPendingBit function.

Table 315. NVIC_ClearSystemHandlerPendingBit function

Function name	NVIC_ClearSystemHandlerPendingBit
Function prototype	void NVIC_ClearSystemHandlerPendingBit(u32 SystemHandler)
Behavior description	Clears System Handler pending bit.
Input parameter	SystemHandler: system handler pending bit to be reset. Refer to *SystemHandler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler (see *Table 316*).

Table 316. systemHandler types

SystemHandler	Description
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	SysTick Handler

Example:

```
/* Clear SysTick Pending Bit */
NVIC_ClearSystemHandlerPendingBit(SystemHandler_SysTick);
```

13.2.28 NVIC_GetSystemHandlerActiveBitStatus function

Table 317 describes the NVIC_GetSystemHandlerActiveBitStatus function.

Table 317. NVIC_GetSystemHandlerActiveBitStatus function

Function name	NVIC_GetSystemHandlerActiveBitStatus
Function prototype	ITStatus NVIC_GetSystemHandlerActiveBitStatus(u32 SystemHandler)
Behavior description	Checks whether the specified System handlers active bit is set or not.
Input parameter	SystemHandler: system handler active bit to be checked. Refer to *SystemHandler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of System Handler active bit (SET or RESET).
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler (see *Table 318*).

Table 318. systemHandler types

SystemHandler	Description
SystemHandler_MemoryManage	Memory Manage Handler
SystemHandler_BusFault	Bus Fault Handler
SystemHandler_UsageFault	Usage Fault Handler
SystemHandler_SVCall	SVCall Handler
SystemHandler_DebugMonitor	Debug Monitor Handler
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	SysTick Handler

Example:

```
/* Check if the Bus Fault is active or stacked */
ITStatus BusFaultHandlerStatus;
BusFaultHandlerStatus =
NVIC_GetSystemHandlerActiveBitStatus(SystemHandler_BusFault);
```

13.2.29 NVIC_GetFaultHandlerSources function

Table 319 describes the NVIC_GetFaultHandlerSources function.

Table 319. NVIC_GetFaultHandlerSources function

Function name	NVIC_GetFaultHandlerSources
Function prototype	u32 NVIC_GetFaultHandlerSources(u32 SystemHandler)
Behavior description	Returns the system handler fault sources.
Input parameter	SystemHandler: system handler of which the fault sources will be returned. Refer to *SystemHandler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	Source of the fault handler.
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler (see *Table 320*).

Table 320. systemHandler types

SystemHandler	Description
SystemHandler_HardFault	Hard Fault Handler
SystemHandler_MemoryManage	Memory Manage Handler
SystemHandler_BusFault	Bus Fault Handler
SystemHandler_UsageFault	Usage Fault Handler
SystemHandler_DebugMonitor	Debug Monitor Handler

Example:

```
/* Gets the sources of the Bus Fault Handler */
u32 BusFaultHandlerSource;
BusFaultHandlerSource
=NVIC_GetFaultHandlerSources(SystemHandler_BusFault);
```

13.2.30 NVIC_GetFaultAddress function

Table 321 describes the NVIC_GetFaultAddress function

Table 321. NVIC_GetFaultAddress function

Function name	NVIC_GetFaultAddress
Function prototype	u32 NVIC_GetFaultAddress(u32 SystemHandler)
Behavior description	Returns the address of the location that generated a fault handler.
Input parameter	SystemHandler: system handler of which the fault address will be returned Refer to *SystemHandler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	Fault address.
Required preconditions	None
Called functions	None

SystemHandler

This parameter selects the system handler (see *Table 322*).

Table 322. SystemHandler types

SystemHandler	Description
SystemHandler_MemoryManage	Memory Manage Handler
SystemHandler_BusFault	Bus Fault Handler

Example:

```
/* Gets the address of the Bus Fault Handler */
u32 BusFaultHandlerAddress;
BusFaultHandlerAddress =
NVIC_GetFaultAddress(SystemHandler_BusFault);
```

14 Power control (PWR)

The PWR is used for a variety of purposes including power management and low power mode selection.

Section 14.1: PWR register structure describes the data structures used in the PWR Firmware Library. *Section 14.2: Firmware library functions* presents the Firmware Library functions.

14.1 PWR register structure

The PWR register structure, *PWR_TypeDef,* is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 CR;
  vu32 CSR;
} PWR_TypeDef;
```

Table 323 gives the list of PWR registers.

Table 323. PWR registers

Register	Description
CR	Power Control Register
CSR	Power Control Status Register

The PWR peripheral is declared in *stm32f10x_map.h*:

```
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)

#define PWR_BASE              (APB1PERIPH_BASE + 0x7000)

#ifndef DEBUG
...
#ifdef _PWR
  #define PWR                   ((PWR_TypeDef *) PWR_BASE)
#endif /*_PWR */
...
#else   /* DEBUG */
...
#ifdef _PWR
  EXT PWR_TypeDef             *PWR;
#endif /*_PWR */
...
#endif
```

When using the Debug mode, PWR pointer is initialized in *stm32f10x_lib.c* file:

```
#ifdef _PWR
PWR = (PWR_TypeDef *)  PWR_BASE;
#endif /*_PWR */
```

To access the PWR registers, _PWR must be defined in *stm32f10x_conf.h* as follows:

```
#define _PWR
```

14.2 Firmware library functions

Table 324 gives the list of the various PWR library functions.

Table 324. PWR firmware library functions

Function name	Description
PWR_DeInit	Resets the PWR peripheral registers to their default reset values.
PWR_BackupAccessCmd	Enables or disables access to the RTC and backup registers.
PWR_PVDCmd	Enables or disables the Power Voltage Detector(PVD).
PWR_PVDLevelConfig	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
PWR_WakeUpPinCmd	Enables or disables the WakeUp Pin functionality.
PWR_EnterSTOPMode	Enters Stop mode.
PWR_EnterSTANDBYMode	Enters Standby mode.
PWR_GetFlagStatus	Checks whether the specified PWR flag is set or not.
PWR_ClearFlag	Clears the PWR's pending flags.

14.2.1 PWR_DeInit function

Table 325 describes the PWR_DeInit function.

Table 325. PWR_DeInit function

Function name	PWR_DeInit
Function prototype	void PWR_DeInit(void)
Behavior description	Resets the PWR peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB1PeriphResetCmd

Example:

```
/* Deinitialize the PWR registers */
PWR_DeInit();
```

14.2.2 PWR_BackupAccessCmd function

Table 326 describes the PWR_BackupAccessCmd function.

Table 326. PWR_BackupAccessCmd function

Function name	PWR_BackupAccessCmd
Function prototype	void PWR_BackupAccessCmd(FunctionalState NewState)
Behavior description	Enables or disables access to the RTC and backup registers.
Input parameter	NewState: new state of the access to the RTC and backup registers. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable access to the RTC and backup registers */
PWR_BackupAccessCmd(ENABLE);
```

14.2.3 PWR_PVDCmd function

Table 327 describes the PWR_PVDCmd function.

Table 327. PWR_PVDCmd function

Function name	PWR_PVDCmd
Function prototype	void PWR_PVDCmd(FunctionalState NewState)
Behavior description	Enables or disables the Power Voltage Detector(PVD).
Input parameter	NewState: new state of the PVD. This parameter can be set either to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the Power Voltage Detector(PVD) */
PWR_PVDCmd(ENABLE);
```

14.2.4 PWR_PVDLevelConfig function

Table 328 describes the PWR_PVDLevelConfig function.

Table 328. PWR_PVDLevelConfig function

Function name	PWR_PVDLevelConfig
Function prototype	void PWR_PVDLevelConfig(u32 PWR_PVDLevel)
Behavior description	Configures the voltage threshold detected by the Power Voltage Detector (PVD).
Input parameter	PWR_PVDLevel: PVD detection level Refer to *PWR_PVDLevel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

PWR_PVDLevel

This parameter configures the PVD detection level value (see *Table 329*).

Table 329. PWR_PVDLevel values

PWR_PVDLevel	Description
PWR_PVDLevel_2V2	PVD detection level set to 2.2V
PWR_PVDLevel_2V3	PVD detection level set to 2.3V
PWR_PVDLevel_2V4	PVD detection level set to 2.4V
PWR_PVDLevel_2V5	PVD detection level set to 2.5V
PWR_PVDLevel_2V6	PVD detection level set to 2.6V
PWR_PVDLevel_2V7	PVD detection level set to 2.7V
PWR_PVDLevel_2V8	PVD detection level set to 2.8V
PWR_PVDLevel_2V9	PVD detection level set to 2.9V

Example:

```
/* Set PVD detection level to 2.5V */
PWR_PVDLevelConfig(PWR_PVDLevel_2V5);
```

14.2.5 PWR_WakeUpPinCmd function

Table 330 describes the PWR_WakeUpPinCmd function.

Table 330. PWR_WakeUpPinCmd function

Function name	PWR_WakeUpPinCmd
Function prototype	void PWR_WakeUpPinCmd(FunctionalState NewState)
Behavior description	Enables or disables the WakeUp Pin functionality.
Input parameter	NewState: new state of the WakeUp Pin functionality. This parameter can be set either to ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* WakeUp pin used for wake-up function */
PWR_WakeUpPinCmd(ENABLE);
```

14.2.6 PWR_EnterSTOPMode function

Table 331 describes the PWR_EnterSTOPMode function.

Table 331. PWR_EnterSTOPMode function

Function name	PWR_EnterSTOPMode
Function prototype	void PWR_EnterSTOPMode(u32 PWR_Regulator, u8 PWR_STOPEntry)
Behavior description	Enters Stop mode.
Input parameter1	PWR_Regulator: regulator state in Stop mode. Refer to *PWR_Regulator* for more details on the allowed values for this parameter.
Input parameter2	PWR_STOPEntry: specifies if Stop mode in entered with WFI or WFE instruction. Refer to *PWR_STOPEntry* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__WFI(), __WFE()

PWR_Regulator

This parameter configures the regulator state in Stop mode. See *Table 332* for the possible values of PWR_Regulator.

Table 332. PWR_Regulator definition

PWR_Regulator	Description
PWR_Regulator_ON	Stop mode with regulator ON
PWR_Regulator_LowPower	Stop mode with regulator in low power mode

PWR_STOPEntry

This parameter defines the Stop mode entry.

Table 333. PWR_STOPEntry definition

PWR_Regulator	Description
PWR_STOPEntry_WFI	Enter Stop mode with WFI instruction
PWR_STOPEntry_WFE	Enter Stop mode with WFE instruction

Example:

```
/* Put the system in Stop mode with regulator on */
PWR_EnterSTOPMode(PWR_Regulator_ON, PWR_STOPEntry_WFE);
```

14.2.7 PWR_EnterSTANDBYMode function

Table 334 describes the PWR_EnterSTANDBYMode function.

Table 334. PWR_EnterSTANDBYMode function

Function name	PWR_EnterSTANDBYMode
Function prototype	void PWR_EnterSTANDBYMode(void)
Behavior description	Enters Standby mode.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	__WFI()

Example:

```
/* Put the system in Standby mode */
PWR_EnterSTANDBYMode();
```

14.2.8 PWR_GetFlagStatus function

Table 335 describes the PWR_GetFlagStatus function.

Table 335. PWR_GetFlagStatus function

Function name	PWR_GetFlagStatus
Function prototype	FlagStatus PWR_GetFlagStatus(u32 PWR_FLAG)
Behavior description	Checks whether the specified PWR flag is set or not.
Input parameter	PWR_FLAG: flag to be checked. Refer to *PWR_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of PWR_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

PWR_FLAG

The PWR flags that can be checked by issuing a PWR_GetFlagStatus function are listed in *Table 336*.

Table 336. PWR_Flag values

PWR_FLAG	Description
PWR_FLAG_WU	Wake-up flag
PWR_FLAG_SB	StandBy flag
PWR_FLAG_PVDO	PVD Output(1)

1. This flag is read only. It cannot be cleared.

Example:

```
/* Test if the StandBy flag is set or not */
FlagStatus Status;
Status = PWR_GetFlagStatus(PWR_FLAG_SB);
if(Status == RESET)
{
...
}
else
{
...
}
```

14.2.9 PWR_ClearFlag function

Table 337 describes the PWR_ClearFlag function.

Table 337. PWR_ClearFlag function

Function name	PWR_ClearFlag
Function prototype	void PWR_ClearFlag(u32 PWR_FLAG)
Behavior description	Clears the PWR's pending flags.
Input parameter	PWR_FLAG: flag to be cleared. Refer to *PWR_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the StandBy pending flag */
PWR_ClearFlag(PWR_FLAG_SB);
```

15 Reset and clock control (RCC)

The RCC can be used for a variety of purposes, including clock configuration, peripheral reset and clock management.

Section 15.1: RCC register structure describes the data structures used in the RCC Firmware Library. *Section 15.2: Firmware library functions* presents the Firmware Library functions.

15.1 RCC register structure

The RCC register structure, *RCC_TypeDef, is* defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 CR;
  vu32 CFGR;
  vu32 CIR;
  vu32 APB2RSTR;
  vu32 APB1RSTR;
  vu32 AHBENR;
  vu32 APB2ENR;
  vu32 APB1ENR;
  vu32 BDCR;
  vu32 CSR;
} RCC_TypeDef;
```

Table 338 gives the list of RCC registers.

Table 338. RCC registers

Register	Description
CR	Clock control register
CFGR	Clock configuration register
CIR	Clock interrupt register
APB2RSTR	APB2 Peripheral reset register
APB1RSTR	APB1 Peripheral reset register
AHBENR	AHB Peripheral Clock enable register
APB2ENR	APB2 Peripheral Clock enable register
APB1ENR	APB1 Peripheral Clock enable register
BDCR	Backup domain control register
CSR	Control/status register

The RCC peripheral is declared in the same file:

```
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
#define RCC_BASE              (AHBPERIPH_BASE + 0x1000)

#ifndef DEBUG
...
#ifdef _RCC
  #define RCC                 ((RCC_TypeDef *) RCC_BASE)
#endif /*_RCC */
...
#else   /* DEBUG */
...
#ifdef _RCC
  EXT RCC_TypeDef             *RCC;
#endif /*_RCC */
...
#endif
```

When using the Debug mode, RCC pointer is initialized in *stm32f10x_lib.c* file:

```
#ifdef _RCC
  RCC = (RCC_TypeDef *)  RCC_BASE;
#endif /*_RCC */
```

To access the reset and clock control registers, _RCC must be defined in *stm32f10x_conf.h* as follows:

```
#define _RCC
```

15.2 Firmware library functions

Table 339 gives the list of the various functions of the RCC library.

Table 339. RCC firmware library functions

Function name	Description
RCC_DeInit	Resets the RCC clock configuration to the default reset state.
RCC_HSEConfig	Configures the External High Speed oscillator (HSE).
RCC_WaitForHSEStartUp	Waits for HSE start-up.
RCC_AdjustHSICalibrationValue	Adjusts the Internal High Speed oscillator (HSI) calibration value.
RCC_HSICmd	Enables or disables the Internal High Speed oscillator (HSI).
RCC_PLLConfig	Configures the PLL clock source and multiplication factor.
RCC_PLLCmd	Enables or disables the PLL.
RCC_SYSCLKConfig	Configures the system clock (SYSCLK).
RCC_GetSYSCLKSource	Returns the clock source used as system clock.
RCC_HCLKConfig	Configures the AHB clock (HCLK).
RCC_PCLK1Config	Configures the Low Speed APB clock (PCLK1).

Table 339. RCC firmware library functions (continued)

Function name	Description
RCC_PCLK2Config	Configures the High Speed APB clock (PCLK2).
RCC_ITConfig	Enables or disables the specified RCC interrupts.
RCC_USBCLKConfig	Configures the USB clock (USBCLK).
RCC_ADCCLKConfig	Configures the ADC clock (ADCCLK).
RCC_LSEConfig	Configures the External Low Speed oscillator (LSE).
RCC_LSICmd	Enables or disables the Internal Low Speed oscillator (LSI).
RCC_RTCCLKConfig	Configures the RTC clock (RTCCLK).
RCC_RTCCLKCmd	Enables or disables the RTC clock.
RCC_GetClocksFreq	Returns the frequencies of different on chip clocks.
RCC_AHBPeriphClockCmd	Enables or disables the AHB peripheral clock.
RCC_APB2PeriphClockCmd	Enables or disables the High Speed APB (APB2) peripheral clock.
RCC_APB1PeriphClockCmd	Enables or disables the Low Speed APB (APB1) peripheral clock.
RCC_APB2PeriphResetCmd	Forces or releases High Speed APB (APB2) peripheral reset.
RCC_APB1PeriphResetCmd	Forces or releases Low Speed APB (APB1) peripheral reset.
RCC_BackupResetCmd	Forces or releases the Backup domain reset.
RCC_ClockSecuritySystemCmd	Enables or disables the Clock Security System.
RCC_MCOConfig	Selects the clock source to output on MCO pin.
RCC_GetFlagStatus	Checks whether the specified RCC flag is set or not.
RCC_ClearFlag	Clears the RCC reset flags.
RCC_GetITStatus	Checks whether the specified RCC interrupt has occurred or not.
RCC_ClearITPendingBit	Clears the RCC’s interrupt pending bits.

15.2.1 RCC_DeInit function

Table 340 describes the RCC_DeInit function.

Table 340. RCC_DeInit function(1)

Function name	RCC_DeInit
Function prototype	void RCC_DeInit(void)
Behavior description	Resets the RCC clock configuration to the default reset state.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

1. The default reset state of the clock configuration is given below:
   - HSI on
   - HSI used as the system clock source
   - HSE and PLL off
   - AHB, APB1 and APB2 prescaler set to 1 and ADC prescaler set to 2.

Example:

```
/* Reset the RCC clock configuration to the default reset state */
RCC_DeInit();
```

15.2.2 RCC_HSEConfig function

Table 341 describes the RCC_HSEConfig function.

Table 341. RCC_HSEConfig function

Function name	RCC_HSEConfig
Function prototype	void RCC_HSEConfig(u32 RCC_HSE)
Behavior description	Configures the External High Speed oscillator (HSE).
Input parameter	RCC_HSE: new state of the HSE. Refer to *RCC_HSE* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	HSE can not be stopped if it is used directly or through the PLL as system clock.
Called functions	None

RCC_HSE

This parameter configures the HSE state (see *Table 342*).

Table 342. RCC_HSE definition

RCC_HSE	Description
RCC_HSE_OFF	HSE oscillator OFF
RCC_HSE_ON	HSE oscillator ON
RCC_HSE_Bypass	HSE oscillator bypassed with external clock

Example:

```
/* Enable the HSE */
RCC_HSEConfig(RCC_HSE_ON);
```

15.2.3 RCC_WaitForHSEStartUp function

Table 343 describes the RCC_WaitForHSEStartUp function.

Table 343. RCC_WaitForHSEStartUp function

Function name	RCC_WaitForHSEStartUp
Function prototype	ErrorStatus RCC_WaitForHSEStartUp(void)
Behavior description	Waits for HSE startup. This functions waits till HSE is ready and exits if the timeout is reached.
Input parameter	None
Output parameter	None
Return parameter	An ErrorStatus enumuration value: – SUCCESS: HSE oscillator is stable and ready to use – ERROR: HSE oscillator not yet ready
Required preconditions	None
Called functions	None

Note: This function uses a predefined Timeout value. This value can be adjusted in the stm32f10x_conf.h file.

```
#define HSEStartUp_TimeOut    ((u16)0x0500) /* Timeout for HSE
startup */
```

Example:

```
ErrorStatus HSEStartUpStatus;

/* Enable HSE */
RCC_HSEConfig(RCC_HSE_ON);

/* Wait till HSE is ready and if Time out is reached exit */
HSEStartUpStatus = RCC_WaitForHSEStartUp();

  if(HSEStartUpStatus == SUCCESS)
  {
    /* Add here PLL and system clock config */
```

```
}
else
{
  /* Add here some code to deal with this error */
}
```

15.2.4 RCC_AdjustHSICalibrationValue function

Table 344 describes the RCC_AdjustHSICalibrationValue function.

Table 344. RCC_AdjustHSICalibrationValue function

Function name	RCC_AdjustHSICalibrationValue
Function prototype	void RCC_AdjustHSICalibrationValue(u8 HSICalibrationValue)
Behavior description	Adjusts the Internal High Speed oscillator (HSI) calibration value.
Input parameter	HSICalibrationValue: calibration trimming value. This parameter must be a number between 0 and 0x1F.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set HSI calibration value to 0x1F (maximum) */
RCC_AdjustHSICalibrationValue(0x1F);
```

15.2.5 RCC_HSICmd function

Table 345 describes the RCC_HSICmd function.

Table 345. RCC_HSICmd function

Function name	RCC_HSICmd
Function prototype	void RCC_HSICmd(FunctionalState NewState)
Behavior description	Enables or disables the Internal High Speed oscillator (HSI).
Input parameter	NewState: new state of the HSI. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	HSI can not be stopped if it is used directly or through the PLL as system clock, or if a Flash program operation is ongoing.
Called functions	None

Example:

```
/* Enable Internal High Speed oscillator */
RCC_HSICmd(ENABLE);
```

15.2.6 RCC_PLLConfig function

Table 346 describes the RCC_PLLConfig function.

Table 346. RCC_PLLConfig function

Function name	RCC_PLLConfig
Function prototype	void RCC_PLLConfig(u32 RCC_PLLSource, u32 RCC_PLLMul)
Behavior description	Configures the PLL clock source and multiplication factor.
Input parameter1	RCC_PLLSource: PLL entry clock source. Refer to *RCC_PLLSource* for more details on the allowed values for this parameter.
Input parameter2	RCC_PLLMul: PLL multiplication factor. Refer to *RCC_PLLMul* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	This function must be used only when the PLL is disabled.
Called functions	None

RCC_PLLSource

This parameter selects the PLL entry clock source (see *Table 347*).

Table 347. RCC_PLLSource definition

RCC_PLLSource	Description
RCC_PLLSource_HSI_Div2	PLL clock entry = HSI oscillator clock divided by 2
RCC_PLLSource_HSE_Div1	PLL clock entry = HSE oscillator clock
RCC_PLLSource_HSE_Div2PLL clock entry	PLL clock entry = HSE oscillator clock divided by 2

RCC_PLLMul

This parameter selects the PLL multiplication factor (see *Table 348*).

Table 348. RCC_PLLMul definition

RCC_PLLMul	Description
RCC_PLLMul_2	PLL clock entry x 2
RCC_PLLMul_3	PLL clock entry x 3
RCC_PLLMul_4	PLL clock entry x 4
RCC_PLLMul_5	PLL clock entry x 5
RCC_PLLMul_6	PLL clock entry x 6
RCC_PLLMul_7	PLL clock entry x 7
RCC_PLLMul_8	PLL clock entry x 8
RCC_PLLMul_9	PLL clock entry x 9

Table 348. RCC_PLLMul definition (continued)

RCC_PLLMul	Description
RCC_PLLMul_10	PLL clock entry x 10
RCC_PLLMul_11	PLL clock entry x 11
RCC_PLLMul_12	PLL clock entry x12
RCC_PLLMul_13	PLL clock entry x 13
RCC_PLLMul_14	PLL clock entry x 14
RCC_PLLMul_15	PLL clock entry x 15
RCC_PLLMul_16	PLL clock entry x 16

Warning: The software must configure correctly the PLL to generate a PLL output frequency that does not exceed 72 MHz.

Example:

```
/* Set PLL clock output to 72MHz using HSE (8MHz) as entry clock */
RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
```

15.2.7 RCC_PLLCmd function

Table 349 describes the RCC_PLLCmd function.

Table 349. RCC_PLLCmd function

Function name	RCC_PLLCmd
Function prototype	void RCC_PLLCmd(FunctionalState NewState)
Behavior description	Enables or disables the PLL.
Input parameter	NewState: new state of the PLL. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	The PLL can not be disabled if it is used as system clock.
Called functions	None

Example:

```
/* Enable the PLL */
RCC_PLLCmd(ENABLE);
```

15.2.8 RCC_SYSCLKConfig function

Table 350 describes the RCC_SYSCLKConfig function.

Table 350. RCC_SYSCLKConfig function

Function name	RCC_SYSCLKConfig
Function prototype	void RCC_SYSCLKConfig(u32 RCC_SYSCLKSource)
Behavior description	Configures the system clock (SYSCLK).
Input parameter	RCC_SYSCLKSource: clock source used as system clock. Refer to *RCC_SYSCLKSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_SYSCLKSource

This parameter selects the system clock source (see *Table 351*).

Table 351. RCC_SYSCLKSource definition

RCC_SYSCLKSource	Description
RCC_SYSCLKSource_HSI	HSI selected as system clock
RCC_SYSCLKSource_HSE	HSE selected as system clock
RCC_SYSCLKSource_PLLCLK	PLL selected as system clock

Example:

```
/* Select the PLL as system clock source */
RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
```

15.2.9 RCC_GetSYSCLKSource function

Table 352 describes the RCC_GetSYSCLKSource function.

Table 352. RCC_GetSYSCLKSource function

Function name	RCC_GetSYSCLKSource
Function prototype	u8 RCC_GetSYSCLKSource(void)
Behavior description	Returns the clock source used as system clock.
Input parameter	None
Output parameter	None
Return parameter	The clock source used as system clock.The returned value can be one of the following: – 0x00: HSI used as system clock – 0x04: HSE used as system clock – 0x08: PLL used as system clock
Required preconditions	None
Called functions	None

Example:

```
/* Test if HSE is used as system clock */
if(RCC_GetSYSCLKSource() != 0x04)
{
}
else
{
}
```

15.2.10 RCC_HCLKConfig function

Table 353 describes the RCC_HCLKConfig function.

Table 353. RCC_HCLKConfig function

Function name	RCC_HCLKConfig
Function prototype	void RCC_HCLKConfig(u32 RCC_HCLK)
Behavior description	Configures the AHB clock(HCLK).
Input parameter	RCC_HCLK: defines the AHB clock. This clock is derived from the system clock (SYSCLK). Refer to *RCC_HCLK* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_HCLK

RCC_HCLK configures the AHB clock. Refer to *Table 354* for the values taken by this parameter.

Table 354. RCC_HCLK values

RCC_HCLK	Description
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

Example:

```
/* Configure HCLK such as HCLK = SYSCLK */
RCC_HCLKConfig(RCC_SYSCLK_Div1);
```

15.2.11 RCC_PCLK1Config function

Table 355 describes the RCC_PCLK1Config function.

Table 355. RCC_PCLK1Config function

Function name	RCC_PCLK1Config
Function prototype	`void RCC_PCLK1Config(u32 RCC_PCLK1)`
Behavior description	Configures the Low Speed APB clock (PCLK1).
Input parameter	RCC_PCLK1: defines the APB1 clock. This clock is derived from the AHB clock (HCLK). Refer to *RCC_PCLK1* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_PCLK1

RCC_PCLK1 configures the APB1 clock. Refer to *Table 356* for the values taken by this parameter.

Table 356. RCC_PCLK1 values

RCC_PCLK1	Description
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

Example:

```
/* Configure PCLK1 such as PCLK1 = HCLK/2 */
RCC_PCLK1Config(RCC_HCLK_Div2);
```

15.2.12 RCC_PCLK2Config function

Table 357 describes the RCC_PCLK2Config function.

Table 357. RCC_PCLK2Config function

Function name	RCC_PCLK2Config
Function prototype	`void RCC_PCLK2Config(u32 RCC_PCLK2)`
Behavior description	Configures the High Speed APB clock (PCLK2).
Input parameter	RCC_PCLK2: defines the APB2 clock. This clock is derived from the AHB clock (HCLK). Refer to *RCC_PCLK2* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_PCLK2

RCC_PCLK2 configures the APB2 clock. Refer to *Table 358* for the values taken by this parameter.

Table 358. RCC_PCLK2 values

RCC_PCLK2	Description
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

Example:

```
/* Configure PCLK2 such as PCLK2 = HCLK */
RCC_PCLK2Config(RCC_HCLK_Div1);
```

15.2.13 RCC_ITConfig function

Table 359 describes the RCC_ITConfig function.

Table 359. RCC_ITConfig function

Function name	RCC_ITConfig
Function prototype	`void RCC_ITConfig(u8 RCC_IT, FunctionalState NewState)`
Behavior description	Enables or disables the specified RCC interrupts.
Input parameter1	RCC_IT: specifies the RCC interrupt sources to be enabled or disabled. Refer to *RCC_IT* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified RCC interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_IT

RCC_IT enables or disables RCC interrupts. One or a combination of the following values can be used:

Table 360. RCC_IT values

RCC_IT	Description
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

Example:

```
/* Enable PLL Ready interrupt */
RCC_ITConfig(RCC_IT_PLLRDY, ENABLE);
```

15.2.14 RCC_USBCLKConfig function

Table 361 describes the RCC_USBCLKConfig function.

Table 361. RCC_USBCLKConfig function

Function name	RCC_USBCLKConfig
Function prototype	`void RCC_USBCLKConfig(u32 RCC_USBCLKSource)`
Behavior description	Configures the USB clock (USBCLK).
Input parameter	RCC_USBCLKSource specifies the USB clock source. This clock is derived from the PLL output. Refer to *RCC_USBCLKSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	The USB needs a 48 MHz clock to operate correctly. The user must select the USB division factor according to the PLL multiplication factor and PLL clock source frequency in order to obtain a 48 MHz frequency. Once the USB clock is enabled, the USB division factor cannot be modified.
Called functions	None

RCC_USBCLKSource

This parameter selects the USB clock source (see *Table 362*).

Table 362. RCC_USBCLKSource values

RCC_USBCLKSource	Description
RCC_USBCLKSource_PLLCLK_1Div5	USB clock source = PLL clock divided by 1.5 selected
RCC_USBCLKSource_PLLCLK_Div1	USB clock source = PLL clock selected

Example:

```
/* PLL clock divided by 1.5 used as USB clock source */
RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5);
```

15.2.15 RCC_ADCCLKConfig function

Table 363 describes the RCC_ADCCLKConfig function.

Table 363. RCC_ADCCLKConfig function

Function name	RCC_ADCCLKConfig
Function prototype	`void RCC_ADCCLKConfig(u32 RCC_ADCCLK)`
Behavior description	Configures the ADC clock (ADCCLK).
Input parameter	RCC_ADCCLK defines the ADC clock. This clock is derived from the APB2 clock (PCLK2). Refer to *RCC_ADCCLK* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_ADCCLK

RCC_ADCCLK configures the ADC clock. Refer to *Table 364* for the values taken by this parameter.

Table 364. RCC_ADCCLK values

RCC_ADCCLK	Description
RCC_PCLK2_Div2	ADC clock = PCLK2/2
RCC_PCLK2_Div4	ADC clock = PCLK2/4
RCC_PCLK2_Div6	ADC clock = PCLK2/6
RCC_PCLK2_Div8	ADC clock = PCLK2/8

Example:

```
/* Configure ADCCLK such as ADCCLK = PCLK2/2 */
RCC_ADCCLKConfig(RCC_PCLK2_Div2);
```

15.2.16 RCC_LSEConfig function

Table 365 describes the RCC_LSEConfig function.

Table 365. RCC_LSEConfig function

Function name	RCC_LSEConfig
Function prototype	`void RCC_LSEConfig(u32 RCC_LSE)`
Behavior description	Configures the External Low Speed oscillator (LSE).
Input parameter	RCC_LSE: new state of the LSE. Refer to *RCC_LSE* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_LSE

This parameter configures the LSE state (see *Table 366*).

Table 366. RCC_LSE values

RCC_LSE	Description
RCC_LSE_OFF	LSE oscillator OFF
RCC_LSE_ON	LSE oscillator ON
RCC_LSE_Bypass	LSE oscillator bypassed with external clock

Example:

```
/* Enable the LSE */
RCC_LSEConfig(RCC_LSE_ON);
```

15.2.17 RCC_LSICmd function

Table 367 describes the RCC_LSICmd function.

Table 367. RCC_LSICmd function

Function name	RCC_LSICmd
Function prototype	`void RCC_LSICmd(FunctionalState NewState)`
Behavior description	Enables or disables the Internal Low Speed oscillator (LSI).
Input parameter	NewState: new state of the LSI. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	LSI can not be disabled if the IWDG is running.
Called functions	None

Example:

```
/* Enable the Internal Low Speed oscillator */
RCC_LSICmd(ENABLE);
```

15.2.18 RCC_RTCCLKConfig function

Table 368 describes the RCC_RTCCLKConfig function.

Table 368. RCC_RTCCLKConfig function

Function name	RCC_RTCCLKConfig
Function prototype	`void RCC_RTCCLKConfig(u32 RCC_RTCCLKSource)`
Behavior description	Configures the RTC clock (RTCCLK).
Input parameter	RCC_RTCCLKSource: RTC clock source. Refer to *RCC_RTCCLKSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	Once the RTC clock is selected it cannot be changed unless the Backup domain is reset.
Called functions	None

RCC_RTCCLKSource

This parameter selects the RTC clock source (see *Table 369*).

Table 369. RCC_RTCCLKSource values

RCC_RTCCLKSource	Description
RCC_RTCCLKSource_LSE	LSE selected as RTC clock
RCC_RTCCLKSource_LSI	LSI selected as RTC clock
RCC_RTCCLKSource_HSE_Div128	HSE clock divided by 128 selected as RTC clock

Example:

```
/* Select the LSE as RTC clock source */
RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
```

15.2.19 RCC_RTCCLKCmd function

Table 370 describes the RCC_RTCCLKCmd function.

Table 370. RCC_RTCCLKCmd function

Function name	RCC_RTCCLKCmd
Function prototype	`void RCC_RTCCLKCmd(FunctionalState NewState)`
Behavior description	Enables or disables the RTC clock.
Input parameter	NewState: new state of the RTC clock. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	This function must be used only after the RTC clock was selected using the RCC_RTCCLKConfig function.
Called functions	None

Example:

```
/* Enable the RTC clock */
RCC_RTCCLKCmd(ENABLE);
```

15.2.20 RCC_GetClocksFreq function

Table 371 describes the RCC_GetClocksFreq function.

Table 371. RCC_GetClocksFreq function

Function name	RCC_GetClocksFreq
Function prototype	`void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks)`
Behavior description	Returns the frequencies of different on chip clocks.
Input parameter	RCC_Clocks: pointer to an RCC_ClocksTypeDef structure which contains the clock frequencies. Refer to the *RCC_ClocksTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_ClocksTypeDef structure

The RCC_ClocksTypeDef structure is defined in the *stm32f10x_rcc.h* file:

```
typedef struct
{
  u32 SYSCLK_Frequency;
  u32 HCLK_Frequency;
  u32 PCLK1_Frequency;
  u32 PCLK2_Frequency;
  u32 ADCCLK_Frequency;
}RCC_ClocksTypeDef;
```

SYSCLK_Frequency

This member returns SYSCLK clock frequency expressed in Hz.

HCLK_Frequency

This member returns HCLK clock frequency expressed in Hz.

PCLK1_Frequency

This member returns PCLK1 clock frequency expressed in Hz.

PCLK2_Frequency

This member returns PCLK2 clock frequency expressed in Hz.

ADCCLK_Frequency

This member returns ADCCLK clock frequency expressed in Hz.

Example:

```
/* Get the frequencies of different on chip clocks */
RCC_ClocksTypeDef RCC_Clocks;
RCC_GetClocksFreq(&RCC_Clocks);
```

15.2.21 RCC_AHBPeriphClockCmd function

Table 372 describes the RCC_AHBPeriphClockCmd function.

Table 372. RCC_AHBPeriphClockCmd function

Function name	RCC_AHBPeriphClockCmd
Function prototype	void RCC_AHBPeriphClockCmd(u32 RCC_AHBPeriph, FunctionalState NewState)
Behavior description	Enables or disables the AHB peripheral clock.
Input parameter1	RCC_AHBPeriph: AHB peripheral to gate the clock. Refer to *RCC_AHBPeriph* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_AHBPeriph

This parameter selects the AHB peripheral that gates the clock. One or a combination of the following values can be used:

Table 373. RCC_AHBPeriph values(1)

RCC_AHBPeriph	Description
RCC_AHBPeriph_DMA1	DMA1 clock
RCC_AHBPeriph_DMA2	DMA2 clock
RCC_AHBPeriph_SRAM	SRAM clock
RCC_AHBPeriph_FLITF	FLITF clock
RCC_AHBPeriph_CRC	CRC clock
RCC_AHBPeriph_FSMC	FSMC clock
RCC_AHBPeriph_SDIO	SDIO clock

1. SRAM and FLITF clock can be disabled only during sleep mode.

Example:

```
/* Enable DMA1 clock */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1);
```

15.2.22 RCC_APB2PeriphClockCmd function

Table 374 describes the RCC_APB2PeriphClockCmd function.

Table 374. RCC_APB2PeriphClockCmd function

Function name	RCC_APB2PeriphClockCmd
Function prototype	void RCC_APB2PeriphClockCmd(u32 RCC_APB2Periph, FunctionalState NewState)
Behavior description	Enables or disables the High Speed APB (APB2) peripheral clock.
Input parameter1	RCC_APB2Periph: APB2 peripheral to gate the clock. Refer to *RCC_APB2Periph* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_APB2Periph

This parameter selects the APB2 peripheral that gates the clock. One or a combination of the following values can be used:

Table 375. RCC_APB2Periph values

RCC_APB2Periph	Description
RCC_APB2Periph_AFIO	Alternate Function I/O clock
RCC_APB2Periph_GPIOA	IO port A clock
RCC_APB2Periph_GPIOB	IO port B clock
RCC_APB2Periph_GPIOC	IO port C clock
RCC_APB2Periph_GPIOD	IO port D clock
RCC_APB2Periph_GPIOE	IO port E clock
RCC_APB2Periph_GPIOF	IO port F clock
RCC_APB2Periph_GPIOG	IO port G clock
RCC_APB2Periph_ADC1	ADC 1 interface clock
RCC_APB2Periph_ADC2	ADC 2 interface clock
RCC_APB2Periph_TIM1	TIM1 clock
RCC_APB2Periph_SPI1	SPI1 clock
RCC_APB2Periph_TIM8	TIM8 clock
RCC_APB2Periph_USART1	USART1 clock
RCC_APB2Periph_ADC3	ADC3 interface clock
RCC_APB2Periph_ALL	All APB2 peripheral clock

Example:

```
/* Enable GPIOA, GPIOB and SPI1 clocks */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB
|
                       RCC_APB2Periph_SPI1, ENABLE);
```

15.2.23 RCC_APB1PeriphClockCmd function

Table 376 describes the RCC_APB1PeriphClockCmd function.

Table 376. RCC_APB1PeriphClockCmd function

Function name	RCC_APB1PeriphClockCmd
Function prototype	void RCC_APB1PeriphClockCmd(u32 RCC_APB1Periph, FunctionalState NewState)
Behavior description	Enables or disables the Low Speed APB (APB1) peripheral clock.
Input parameter1	RCC_APB1Periph: APB1 peripheral to gates its clock. Refer to *RCC_APB1Periph* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_APB1Periph

This parameter selects the APB1 peripheral that gates the clock. One or a combination of the following values can be used:

Table 377. RCC_APB1Periph values

RCC_APB1Periph	Description
RCC_APB1Periph_TIM2	TIM2 clock
RCC_APB1Periph_TIM3	TIM3 clock
RCC_APB1Periph_TIM4	TIM4 clock
RCC_APB1Periph_TIM5	TIM5 clock
RCC_APB1Periph_TIM6	TIM6 clock
RCC_APB1Periph_TIM7	TIM7 clock
RCC_APB1Periph_WWDG	Window Watchdog clock
RCC_APB1Periph_SPI2	SPI2 clock
RCC_APB1Periph_SPI3	SPI3 clock
RCC_APB1Periph_USART2	USART2 clock
RCC_APB1Periph_USART3	USART3 clock

Table 377. RCC_APB1Periph values (continued)

RCC_APB1Periph	Description
RCC_APB1Periph_UART4	UART4 clock
RCC_APB1Periph_UART5	UART5 clock
RCC_APB1Periph_I2C1	I2C1 clock
RCC_APB1Periph_I2C2	I2C2 clock
RCC_APB1Periph_USB	USB clock
RCC_APB1Periph_CAN	CAN clock
RCC_APB1Periph_BKP	Backup interface clock
RCC_APB1Periph_PWR	Power Controller interface clock
RCC_APB1Periph_DAC	DAC interface clock
RCC_APB1Periph_ALL	All APB1 peripheral clock

Example:

```
/* Enable BKP and PWR clocks */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_BKP | RCC_APB1Periph_PWR,
ENABLE);
```

15.2.24 RCC_APB2PeriphResetCmd function

Table 378 describes the RCC_APB2PeriphResetCmd function.

Table 378. RCC_APB2PeriphResetCmd function

Function name	RCC_APB2PeriphResetCmd
Function prototype	void RCC_APB2PeriphResetCmd(u32 RCC_APB2Periph, FunctionalState NewState)
Behavior description	Forces or releases High Speed APB (APB2) peripheral reset.
Input parameter1	RCC_APB2Periph: APB2 peripheral to reset. Refer to *RCC_APB2Periph* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enter the SPI1 peripheral to reset */
RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1, ENABLE);

/* Exit the SPI1 peripheral from reset */
RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1, DISABLE);
```

15.2.25 RCC_APB1PeriphResetCmd function

Table 379 describes the RCC_APB1PeriphResetCmd function.

Table 379. RCC_APB1PeriphResetCmd function

Function name	RCC_APB1PeriphResetCmd
Function prototype	void RCC_APB1PeriphResetCmd(u32 RCC_APB1Periph, FunctionalState NewState)
Behavior description	Forces or releases Low Speed APB (APB1) peripheral reset.
Input parameter1	RCC_APB1Periph: specifies the APB1 peripheral to reset. Refer to *RCC_APB1Periph* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enter the SPI2 peripheral to reset */
RCC_APB1PeriphResetCmd(RCC_APB1Periph_SPI2, ENABLE);

/* Exit the SPI2 peripheral from reset */
RCC_APB1PeriphResetCmd(RCC_APB1Periph_SPI2, DISABLE);
```

15.2.26 RCC_BackupResetCmd function

Table 380 describes the RCC_BackupResetCmd function.

Table 380. RCC_BackupResetCmd function

Function name	RCC_BackupResetCmd
Function prototype	`void RCC_BackupResetCmd(FunctionalState NewState)`
Behavior description	Forces or releases the Backup domain reset.
Input parameter	NewState: new state of the Backup domain reset. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Reset the entire Backup domain */
RCC_BackupResetCmd(ENABLE);
```

15.2.27 RCC_ClockSecuritySystemCmd function

Table 381 describes the RCC_ClockSecuritySystemCmd function.

Table 381. RCC_ClockSecuritySystemCmd function

Function name	RCC_ClockSecuritySystemCmd
Function prototype	void RCC_ClockSecuritySystemCmd(FunctionalState NewState)
Behavior description	Enables or disables the Clock Security System.
Input parameter	NewState: new state of the Clock Security System. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the Clock Security System */
RCC_ClockSecuritySystemCmd(ENABLE);
```

15.2.28 RCC_MCOConfig function

Table 382 describes the RCC_MCOConfig function.

Table 382. RCC_MCOConfig function

Function name	RCC_MCOConfig
Function prototype	`void RCC_MCOConfig(u8 RCC_MCO)`
Behavior description	Selects the clock source to output on MCO pin.
Input parameter	RCC_MCO: specifies the clock source to output. Refer to *RCC_MCO* or more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_MCO

RCC_MCO selects the clock source to output on MCO pin. Refer to *Table 383* for the values taken by this parameter.

Table 383. RCC_MCO values

RCC_MCO	Description
RCC_MCO_NoClock	No clock selected
RCC_MCO_SYSCLK	System clock selected
RCC_MCO_HSI	HSI oscillator clock selected
RCC_MCO_HSE	HSE oscillator clock selected
RCC_MCO_PLLCLK_Div2	PLL clock divided by 2 selected

Warning: When selecting the System Clock to be output onto MCO, make sure that its frequency does not exceed 50 MHz (the maximum I/O speed).

Example:

```
/* Output PLL clock divided by 2 on MCO pin */
RCC_MCOConfig(RCC_MCO_PLLCLK_Div2);
```

15.2.29 RCC_GetFlagStatus function

Table 384 describes the RCC_GetFlagStatus function.

Table 384. RCC_GetFlagStatus function

Function name	RCC_GetFlagStatus
Function prototype	FlagStatus RCC_GetFlagStatus(u8 RCC_FLAG)
Behavior description	Checks whether the specified RCC flag is set or not.
Input parameter	RCC_FLAG: the flag to check. Refer to *RCC_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of RCC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

RCC_FLAG

The RCC flags that can be checked by issuing an RCC_GetFlagStatus function are listed in *Table 385*.

Table 385. RCC_FLAG values

RCC_FLAG	Description
RCC_FLAG_HSIRDY	HSI oscillator clock ready
RCC_FLAG_HSERDY	HSE oscillator clock ready
RCC_FLAG_PLLRDY	PLL clock ready
RCC_FLAG_LSERDY	LSE oscillator clock ready
RCC_FLAG_LSIRDY	LSI oscillator clock ready
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

Example:

```
/* Test if the PLL clock is ready or not */
FlagStatus Status;
Status = RCC_GetFlagStatus(RCC_FLAG_PLLRDY);
if(Status == RESET)
{
...
}
else
{
...
}
```

15.2.30 RCC_ClearFlag function

Table 386 describes the RCC_ClearFlag function.

Table 386. RCC_ClearFlag function

Function name	RCC_ClearFlag
Function prototype	`void RCC_ClearFlag(void)`
Behavior description	Clears the RCC reset flags. The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRRST
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the reset flags */
RCC_ClearFlag();
```

15.2.31 RCC_GetITStatus function

Table 387 describes the RCC_GetITStatus function.

Table 387. RCC_GetITStatus function

Function name	RCC_GetITStatus
Function prototype	ITStatus RCC_GetITStatus(u8 RCC_IT)
Behavior description	Checks whether the specified RCC interrupt has occurred or not.
Input parameter	RCC_IT: RCC interrupt source to check. Refer to *RCC_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of RCC_IT (SET or RESET).
Required preconditions	None
Called functions	None

RCC_IT

RCC_IT enables or disables RCC interrupts. One or a combination of the following values can be used:

Table 388. RCC_IT values

RCC_IT	Description
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

Example:

```
/* Test if the PLL Ready interrupt has occurred or not */
ITStatus Status;
Status = RCC_GetITStatus(RCC_IT_PLLRDY);
if(Status == RESET)
{
...
}
else
{
...
}
```

15.2.32 RCC_ClearITPendingBit function

Table 389 describes the RCC_ClearITPendingBit function.

Table 389. RCC_ClearITPendingBit function

Function name	RCC_ClearITPendingBit
Function prototype	void RCC_ClearITPendingBit(u8 RCC_IT)
Behavior description	Clears the RCC's interrupt pending bits.
Input parameter	RCC_IT: specifies the interrupt pending bit to clear. Refer to *RCC_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

RCC_IT

RCC_IT enables or disables RCC interrupts. One or a combination of the following values can be used:

Table 390. RCC_IT values

RCC_IT	Description
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

Example:

```
/* Clear the PLL Ready interrupt pending bit */
RCC_ClearITPendingBit(RCC_IT_PLLRDY);
```

16 Real-time clock (RTC)

The RTC provides a set of continuously running counters which can be used, with suitable software, to provide a clock-calendar function. The counter values can be written to set the current time/date of the system.

Section 16.1: RTC register structure describes the data structures used in the RTC Firmware Library. *Section 16.2: Firmware library functions* presents the Firmware Library functions.

16.1 RTC register structure

The RTC register structure, RTC_*TypeDef*, is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu16 CRH;
  u16 RESERVED1;
  vu16 CRL;
  u16 RESERVED2;
  vu16 PRLH;
  u16 RESERVED3;
  vu16 PRLL;
  u16 RESERVED4;
  vu16 DIVH;
  u16 RESERVED5;
  vu16 DIVL;
  u16 RESERVED6;
  vu16 CNTH;
  u16 RESERVED7;
  vu16 CNTL;
  u16 RESERVED8;
  vu16 ALRH;
  u16 RESERVED9;
  vu16 ALRL;
  u16 RESERVED10;
} RTC_TypeDef;
```

Table 391 gives the list of the RTC registers.

Table 391. RTC registers

Register	Description
CRH	Control Register High
CRL	Control Register Low
PRLH	Prescaler Load Register High
PRLL	Prescaler Load Register Low
DIVH	Divider Register High
DIVL	Divider Register Low
CNTH	Counter Register High
CNTL	Counter Register Low
ALRH	Alarm Register High
ALRL	Alarm Register Low

The RTC peripheral is declared in *stm32f10x_map.h*:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
...
#define RTC_BASE              (APB1PERIPH_BASE + 0x2800)

#ifndef DEBUG
...
#ifdef _RTC
  #define RTC                 ((RTC_TypeDef *) RTC_BASE)
#endif /*_RTC */
...
#else   /* DEBUG */
...
#ifdef _RTC
  EXT RTC_TypeDef           *RTC;
#endif /*_RTC */
...
#endif
```

When using the Debug mode, RTC pointer is initialized in *stm32f10x_lib.c* file:

```
#ifdef _RTC
  RTC = (RTC_TypeDef *)  RTC_BASE;
#endif /*_RTC */
```

To access the RTC registers, _RTC must be defined in *stm32f10x*_conf.h as follows:

```
#define _RTC
```

16.2 Firmware library functions

Table 392 gives the list of the various RTC library functions.

Table 392. RTC firmware library functions

Function name	Description
RTC_ITConfig	Enables or disables the specified RTC interrupts.
RTC_EnterConfigMode	Enters the RTC configuration mode.
RTC_ExitConfigMode	Exits from the RTC configuration mode.
RTC_GetCounter	Gets the RTC counter value.
RTC_SetCounter	Sets the RTC counter value.
RTC_SetPrescaler	Sets the RTC prescaler value.
RTC_SetAlarm	Sets the RTC Alarm value.
RTC_GetDivider	Gets the RTC Divider value.
RTC_WaitForLastTask	Waits until last write operation on RTC registers is completed
RTC_WaitForSynchro	Waits until the RTC registers (RTC_CNT, RTC_ALR and RTC_PRL) are synchronized with RTC APB clock.
RTC_GetFlagStatus	Checks whether the specified RTC flag is set or not.
RTC_ClearFlag	Clears the RTC pending flags.
RTC_GetITStatus	Checks whether the specified RTC interrupt has occurred or not.
RTC_ClearITPendingBit	Clears the RTC interrupt pending bits.

16.2.1 RTC_ITConfig function

Table 393 describes the RTC_ITConfig function.

Table 393. RTC_ITConfig function

Function name	RTC_ITConfig
Function prototype	void RTC_ITConfig(u16 RTC_IT, FunctionalState NewState)
Behavior description	Enables or disables the specified RTC interrupts.
Input parameter1	RTC_IT: RTC interrupts sources to be enabled or disabled. Refer to *RTC_IT* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified RTC interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	Before using this function, you must call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	None

RTC_IT

RTC_IT enables or disables RTC interrupts. One or a combination of the following values can be used:

Table 394. RTC_IT values

RTC_IT	Description
RTC_IT_OW	Overflow interrupt enabled
RTC_IT_ALR	Alarm interrupt enabled
RTC_IT_SEC	Second interrupt enabled

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Alarm interrupt enabled */
RTC_ITConfig(RTC_IT_ALR, ENABLE);
```

16.2.2 RTC_EnterConfigMode function

Table 395 describes RTC_EnterConfigMode function.

Table 395. RTC_EnterConfigMode function

Function name	RTC_EnterConfigMode
Function prototype	`void RTC_EnterConfigMode(void)`
Behavior description	Enters the RTC configuration mode.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the configuration mode */
RTC_EnterConfigMode();
```

16.2.3 RTC_ExitConfigMode function

Table 396 describes the RTC_ExitConfigMode function.

Table 396. RTC_ExitConfigMode function

Function name	RTC_ExitConfigMode
Function prototype	`void RTC_ExitConfigMode(void)`
Behavior description	Exits from the RTC configuration mode.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Exit the configuration mode */
RTC_ExitConfigMode();
```

16.2.4 RTC_GetCounter function

Table 397 describes the RTC_GetCounter function.

Table 397. RTC_GetCounter function

Function name	RTC_GetCounter
Function prototype	`u32 RTC_GetCounter(void)`
Behavior description	Gets the RTC counter value.
Output parameter	None
Return parameter	RTC counter value
Required preconditions	None
Called functions	None

Example:

```
/* Gets the counter value */
u32 RTCCounterValue;
RTCCounterValue = RTC_GetCounter();
```

16.2.5 RTC_SetCounter function

Table 398 describes RTC_SetCounter function.

Table 398. RTC_SetCounter function

Function name	RTC_SetCounter
Function prototype	`void RTC_SetCounter(u32 CounterValue)`
Behavior description	Sets the RTC counter value.
Input parameter	CounterValue: RTC counter new value.
Output parameter	None
Return parameter	None
Required preconditions	Before issuing this function, call RTC_WaitForLastTask() function (wait until RTOFF flag is set)
Called functions	RTC_EnterConfigMode() RTC_ExitConfigMode()

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Sets Counter value to 0xFFFF5555 */
RTC_SetCounter(0xFFFF5555);
```

16.2.6 RTC_SetPrescaler function

Table 399 describes the RTC_SetPrescaler function.

Table 399. RTC_SetPrescaler function

Function name	RTC_SetPrescaler
Function prototype	`void RTC_SetPrescaler(u32 PrescalerValue)`
Behavior description	Sets the RTC prescaler value.
Input parameter	PrescalerValue: RTC prescaler new value.
Output parameter	None
Return parameter	None
Required preconditions	Before using this function, call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	RTC_EnterConfigMode() RTC_ExitConfigMode()

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Sets Prescaler value to 0x7A12 */
RTC_SetPrescaler(0x7A12);
```

16.2.7 RTC_SetAlarm function

Table 400 describes the RTC_SetAlarm function.

Table 400. RTC_SetAlarm function

Function name	RTC_SetAlarm
Function prototype	`void RTC_SetAlarm(u32 AlarmValue)`
Behavior description	Sets the RTC alarm value.
Input parameter	AlarmValue: RTC alarm new value.
Output parameter	None
Return parameter	None
Required preconditions	Before using this function, call *RTC_WaitForLastTask()* function (wait until RTOFF flag is set).
Called functions	RTC_EnterConfigMode() RTC_ExitConfigMode()

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Sets Alarm value to 0xFFFFFFFA */
RTC_SetAlarm(0xFFFFFFFA);
```

16.2.8 RTC_GetDivider function

Table 401 describes RTC_GetDivider function.

Table 401. RTC_GetDivider function

Function name	RTC_GetDivider
Function prototype	`u32 RTC_GetDivider(void)`
Behavior description	Gets the RTC Divider value.
Output parameter	None
Return parameter	RTC divider value
Required preconditions	None
Called functions	None

Example:

```
/* Gets the current RTC Divider value */
u32 RTCDividerValue;
RTCDividerValue = RTC_GetDivider();
```

16.2.9 RTC_WaitForLastTask function

Table 402 describes RTC_WaitForLastTask function.

Table 402. RTC_WaitForLastTask function

Function name	RTC_WaitForLastTask
Function prototype	`void RTC_WaitForLastTask(void)`
Behavior description	Waits until last write operation on RTC registers is completed. This function must be called before any write operation to an RTC register.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Sets Alarm value to 0x10 */
RTC_SetAlarm(0x10);
```

16.2.10 RTC_WaitForSynchro function

Table 403 describes RTC_WaitForSynchro function.

Table 403. RTC_WaitForSynchro function

Function name	RTC_WaitForSynchro
Function prototype	`void RTC_WaitForSynchro(void)`
Behavior description	Waits until the RTC registers (RTC_CNT, RTC_ALR and RTC_PRL) are synchronized with RTC APB clock. This function must be called before any read operation after an APB reset or an APB clock stop.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Wait until the RTC registers are synchronized with RTC APB clock
*/
RTC_WaitForSynchro();
```

16.2.11 RTC_GetFlagStatus function

Table 404 describes RTC_GetFlagStatus function

Table 404. RTC_GetFlagStatus function

Function name	RTC_GetFlagStatus
Function prototype	`FlagStatus RTC_GetFlagStatus(u16 RTC_FLAG)`
Behavior description	Checks whether the specified RTC flag is set or not.
Input parameter	RTC_FLAG: specifies the flag to check. Refer to *RTC_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of RTC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

RTC_FLAG

The RTC flags that can be checked by issuing an RTC_GetFlagStatus function are listed in *Table 405*.

Table 405. RTC_FLAG values

RTC_FLAG	Description
RTC_FLAG_RTOFF	RTC operation OFF Flag
RTC_FLAG_RSF	Registers Synchronized Flag
RTC_FLAG_OW	Overflow interrupt Flag
RTC_FLAG_ALR	Alarm interrupt Flag
RTC_FLAG_SEC	Second interrupt Flag

Example:

```
/* Gets the RTC overflow interrupt status */
FlagStatus OverrunFlagStatus;
OverrunFlagStatus = RTC_GetFlagStatus(RTC_Flag_OW);
```

16.2.12 RTC_ClearFlag function

Table 406 describes RTC_ClearFlag function.

Table 406. RTC_ClearFlag function

Function name	RTC_ClearFlag
Function prototype	`void RTC_ClearFlag(u16 RTC_FLAG)`
Behavior description	Clears the RTC's pending flags.
Input parameter	RTC_FLAG: flag to be cleared. Refer to *RTC_FLAG* for more details on the allowed values for this parameter. The RTC_FLAG_RTOFF cannot be cleared by software. The RTC_FLAG_RSF is cleared only after an APB reset or an APB clock stop.
Output parameter	None
Return parameter	None
Required preconditions	Before using this function, call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	None

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Clears the RTC overflow flag */
RTC_ClearFlag(RTC_FLAG_OW);
```

16.2.13 RTC_GetITStatus function

Table 407 describes the RTC_GetITStatus function.

Table 407. RTC_GetITStatus function

Function name	RTC_GetITStatus
Function prototype	`ITStatus RTC_GetITStatus(u16 RTC_IT)`
Behavior description	Checks whether the specified RTC interrupt has occurred or not.
Input parameter	RTC_IT: RTC interrupt source to check. Refer to *RTC_IT* or more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of the RTC_IT(SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the RTC Second interrupt status */
ITStatus SecondITStatus;
SecondITStatus = RTC_GetITStatus(RTC_IT_SEC);
```

16.2.14 RTC_ClearITPendingBit function

Table 408 describes the RTC_ClearITPendingBit function.

Table 408. RTC_ClearITPendingBit function

Function name	RTC_ClearITPendingBit
Function prototype	`void RTC_ClearITPendingBit(u16 RTC_IT)`
Behavior description	Clears the RTC's interrupt pending bits.
Input parameter	RTC_IT: interrupt pending bit to clear. Refer to *RTC_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	Before using this function, call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	None

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Clears the RTC Second interrupt */
RTC_ClearITPendingBit(RTC_IT_SEC);
```

17 Serial peripheral interface (SPI)

The Serial Peripheral Interface (SPI) allows synchronous serial communication with external devices. The interface can be configured to operate in master or slave mode.

Section 17.1: SPI register structure describes the data structures used in the SPI Firmware Library. *Section 17.2: Firmware library functions* presents the Firmware Library functions.

17.1 SPI register structure

The SPI register structure, *SPI_TypeDef*, is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu16 CR1;
  u16 RESERVED0;
  vu16 CR2;
  u16 RESERVED1;
  vu16 SR;
  u16  RESERVED2;
  vu16 DR;
  u16  RESERVED3;
  vu16 CRCPR;
  u16 RESERVED4;
  vu16 RXCRCR;
  u16  RESERVED5;
  vu16 TXCRCR;
  u16  RESERVED6;
  vu16 I2SCFGR;
  u16 RESERVED7;
  vu16 I2SPR;
  u16 RESERVED8;
} SPI_TypeDef;
```

Table 409 gives the list of SPI registers.

Table 409. SPI registers

Register	Description
CR1	SPI Control Register1
CR2	SPI Control Register2
SR	SPI Status Register
DR	SPI Data Register
CRCPR	SPI CRC Polynomial Register
RxCRCR	SPI Rx CRC Register
TxCRCR	SPI Tx CRC Register
I2SCFGR	I^2S Configuration Register
I2SPR	I^2S Prescaler Register

The two SPI peripherals are declared in *stm32f10x_map.h*:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
....
#define SPI1_BASE             (APB2PERIPH_BASE + 0x3000)
#define SPI2_BASE             (APB1PERIPH_BASE + 0x3800)
#define SPI3_BASE             (APB1PERIPH_BASE + 0x3C00)
....
#ifndef DEBUG
...
#ifdef _SPI1
  #define SPI1                ((SPI_TypeDef *) SPI1_BASE)
#endif /*_SPI1 */

#ifdef _SPI2
  #define SPI2                ((SPI_TypeDef *) SPI2_BASE)
#endif /*_SPI2 */
#ifdef _SPI3
  #define SPI3 ((SPI_TypeDef *) SPI3_BASE)
#endif /*_SPI3 */
...
#else   /* DEBUG */
...
#ifdef _SPI1
  EXT SPI_TypeDef             *SPI1;
#endif /*_SPI1 */

#ifdef _SPI2
  EXT SPI_TypeDef             *SPI2;
#endif /*_SPI2 */

#ifdef _SPI3
  EXT SPI_TypeDef             *SPI3;
#endif /*_SPI3 */
...
#endif
```

When using the Debug mode, _SPI1, _SPI2 and _SPI3 pointers are initialized in *stm32f10x_lib.c* file:

```
...
#ifdef _SPI1
  SPI1 = (SPI_TypeDef *)  SPI1_BASE;
#endif /*_SPI1 */

#ifdef _SPI2
  SPI2 = (SPI_TypeDef *)  SPI2_BASE;
#endif /*_SPI2 */

#ifdef _SPI3
  SPI3 = (SPI_TypeDef *)  SPI3_BASE;
#endif /*_SPI3 */
...
```

To access the SPI registers, _SPI, _SPI1 and _SPI2 must be defined in *stm32f10x_conf.h* as follows:

```
...
#define _SPI
#define _SPI1
#define _SPI2
#define _SPI3
...
```

17.2 Firmware library functions

Table 410 lists the various functions of the SPI library.

Table 410. SPI firmware library functions

Function name	Description
SPI_I2S_DeInit	Re-initializes the SPIx peripheral registers to their default reset values.
SPI_Init	Initializes the SPIx peripheral according to the specified parameters in the SPI_InitStruct.
I2S_Init	Initializes the SPIx peripheral according to the specified parameters in the I2S_InitStruct.
SPI_StructInit	Fills each SPI_InitStruct member with its default value.
I2S_StructInit	Fills each I2S_InitStruct member with its default value.
SPI_Cmd	Enables or disables the specified SPI peripheral.
I2S_Cmd	Enables or disables the specified SPI peripheral (in I^2S mode).
SPI_I2S_ITConfig	Enables or disables the specified SPI/I2S interrupts.
SPI_I2S_DMACmd	Enables or disables the SPIx/I2Sx DMA interface.
SPI_I2S_SendData	Transmits data through the SPIx/I2Sx peripheral.
SPI_I2S_ReceiveData	Returns the most recent received data through the SPIx/I2Sx peripheral.
SPI_NSSInternalSoftwareConfig	Configures internally by software the NSS pin for the selected SPI.
SPI_SSOutputCmd	Enables or disables the SS output for the selected SPI.
SPI_DataSizeConfig	Configures the data size for the selected SPI.
SPI_TransmitCRC	Transmits the SPIx CRC value
SPI_CalculateCRC	Enables or disables the CRC value calculation of the transferred bytes.
SPI_GetCRC	Returns the transmit or the receive CRC register value for the specified SPI.
SPI_GetCRCPolynomial	Returns the CRC Polynomial register value for the specified SPI.
SPI_BiDirectionalLineConfig	Selects the data transfer direction in bidirectional mode for the specified SPI.
SPI_I2S_GetFlagStatus	Checks whether the specified SPI/I2S flag is set or not.
SPI_I2S_ClearFlag	Clears the SPIx/I2Sx pending flags.
SPI_I2S_GetITStatus	Checks whether the specified SPI/I2S interrupt has occurred or not.
SPI_I2S_ClearITPendingBit	Clears the SPIx/I2Sx interrupt pending bits.

17.2.1 SPI_I2S_DeInit function

Table 411 describes the SPI_I2S_DeInit function.

Table 411. SPI_DeInit function

Function name	SPI_I2S_DeInit
Function prototype	void SPI_I2S_DeInit(SPI_TypeDef* SPIx)
Behavior description	Resets the SPIx/I2Sx peripheral registers to their default reset values.
Input parameter	SPI*x*: where x can be 1 or 2 to select the SPI peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB2PeriphClockCmd() for SPI1 RCC_APB1PeriphClockCmd() for SPI2 and SPI3

Example:

```
/* Deinitialize the SPI2 */
SPI_DeInit(SPI2);
/* Deinitialize the I2S3 */
SPI_DeInit(SPI3);
```

17.2.2 SPI_Init function

Table 412 describes the SPI_Init function.

Table 412. SPI_Init function

Function name	SPI_Init
Function prototype	void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)
Behavior description	Initializes the SPIx peripheral according to the parameters specified in the SPI_InitStruct.
Input parameter1	SPI*x*: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_InitStruct: pointer to a SPI_InitTypeDef structure that contains the configuration information for the specified SPI peripheral. Refer to the *SPI_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SPI_InitTypeDef structure

The SPI_InitTypeDef structure is defined in the *stm32f10x_spi.h* file:

```
typedef struct
{
u16 SPI_Direction;
  u16 SPI_Mode;
u16 SPI_DataSize;
  u16 SPI_CPOL;
  u16 SPI_CPHA;
  u16 SPI_NSS;
  u16 SPI_BaudRatePrescaler;
  u16 SPI_FirstBit;
  u16 SPI_CRCPolynomial;
} SPI_InitTypeDef;
```

SPI_Direction

SPI_Direction configures the SPI unidirectional or bidirectional data mode. Refer to *Table 413* for the values taken by this member.

Table 413. SPI_Direction definition

SPI_Direction	Description
SPI_Direction_2Lines_FullDuplex	SPI configured as 2 lines unidirectional full duplex
SPI_Direction_2Lines_RxOnly	SPI configured as 2 lines unidirectional Rx only
SPI_Direction_1Line_Rx	SPI configured as 1 line bidirectional Rx only
SPI_Direction_1Line_Tx	SPI configured as 1 line bidirectional Tx only

SPI_Mode

SPI_Mode configures the SPI operating mode. Refer to *Table 414* for the values taken by this member.

Table 414. SPI_Mode definition

SPI_Mode	Description
SPI_Mode_Master	SPI configured as a master
SPI_Mode_Slave	SPI configured as a slave

SPI_DataSize

SPI_DataSize configures the SPI data size. Refer to *Table 415* for the values taken by this member.

Table 415. SPI_DataSize definition

SPI_DataSize	Description
SPI_DataSize_16b	SPI 16-bit data frame format for transmission and reception
SPI_DataSize_8b	SPI 8-bit data frame format for transmission and reception

SPI_CPOL

SPI_CPOL selects the serial clock steady state. Refer to *Table 416* for the values taken by this member.

Table 416. SPI_CPOL definition

SPI_CPOL	Description
SPI_CPOL_High	Clock idle high
SPI_CPOL_Low	Clock idle low

SPI_CPHA

SPI_CPHA configures the clock active edge for the bit capture. Refer to *Table 417* for the values taken by this member.

Table 417. SPI_CPHA definition

SPI_CPHA	Description
SPI_CPHA_2Edge	Data is captured on the second edge
SPI_CPHA_1Edge	Data is captured on the first edge

SPI_NSS

SPI_NSS specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. Refer to *Table 418* for the values taken by this member.

Table 418. SPI_NSS definition

SPI_NSS	Description
SPI_NSS_Hard	NSS managed by external pin
SPI_NSS_Soft	Internal NSS signal controlled by SSI bit

SPI_BaudRatePrescaler

SPI_BaudRatePrescaler is used to define the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. Refer to *Table 419* for the values taken by this member.

Table 419. SPI_BaudRatePrescaler definition

SPI_BaudratePrescaler	Description
SPI_BaudRatePrescaler2	Baud Rate Prescaler equal to 2
SPI_BaudRatePrescaler4	Baud Rate Prescaler equal to 4
SPI_BaudRatePrescaler8	Baud Rate Prescaler equal to 8
SPI_BaudRatePrescaler16	Baud Rate Prescaler equal to 16
SPI_BaudRatePrescaler32	Baud Rate Prescaler equal to 32
SPI_BaudRatePrescaler64	Baud Rate Prescaler equal to 64
SPI_BaudRatePrescaler128	Baud Rate Prescaler equal to 128
SPI_BaudRatePrescaler256	Baud Rate Prescaler equal to 256

Note: The communication clock is derived from the master clock. The slave clock does not need to be set.

SPI_FirstBit

SPI_FirstBit specifies whether data transfers start from MSB or LSB bit. Refer to *Table 420* for the values taken by this member.

Table 420. SPI_FirstBit definition

SPI_FirstBit	Description
SPI_FisrtBit_MSB	First bit to transfer is the MSB
SPI_FisrtBit_LSB	First bit to transfer is the LSB

SPI_CRCPolynomial

SPI_CRCPolynomial defines the polynomial used for the CRC calculation.

Example:

```
/* Initialize the SPI1 according to the SPI_InitStructure members */
SPI_InitTypeDef  SPI_InitStructure;
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_DatSize = SPI_DatSize_16b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler =
SPI_BaudRatePrescaler_128;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_Init(SPI1, &SPI_InitStructure);
```

17.2.3 I2S_Init function

Table 421 describes the I2S_Init function.

Table 421. I2S_Init function

Function name	I2S_Init
Function prototype	void I2S_Init(SPI_TypeDef* SPIx, I2S_InitTypeDef* I2S_InitStruct)
Behavior description	Initializes the SPIx peripheral (in I^2S mode) according to the parameters specified in I2S_InitStruct.
Input parameter1	SPI*x*: where x can be 2 or 3 to select the SPI peripheral (in I^2S mode).
Input parameter2	I2S_InitStruct: pointer to an I2S_InitTypeDef structure that contains the configuration information for the specified SPI peripheral (in I^2S mode). Refer to *I2S_InitTypeDef* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_GetClocksFreq()

I2S_InitTypeDef

The I2S_InitTypeDef structure is defined in the *stm32f10x_spi.h* file:

```
typedef struct
{
u16 I2S_Mode;
u16 I2S_Standard;
u16 I2S_DataFormat;
u16 I2S_MCLKOutput;
u16 I2S_AudioFreq;
u16 I2S_CPOL;
} I2S_InitTypeDef;
```

- **I2S_Mode**

  Specifies the I^2S peripheral Master/Slave and Transmitter/Receiver configuration as shown in *Table 422*.

Table 422. I^2S peripheral configuration

I2S_Mode	Description
I2S_Mode_SlaveTx	I^2S peripheral is configured as Slave and Transmitter
I2S_Mode_SlaveRx	I^2S peripheral is configured as Slave and Receiver
I2S_Mode_MasterTx	I^2S peripheral is configured as Master and Transmitter
I2S_Mode_MasterRx	I^2S peripheral is configured as Master and Receiver

- **I2S_Standard**
  Specifies the standard used for the I²S communication as shown in *Table 423*.

Table 423. Used standard

I2S_Standard	Description
I2S_Standardd_Phillips	Uses the Phillips I²S standard
I2S_Standard_MSB	Uses the MSB standard
I2S_Standard_LSB	Uses the LSB standard
I2S_Standard_PCMShort	Uses PCM mode with short frame
I2S_Standard_PCMLong	Uses PCM mode with long frame

- **I2S_DataFormat**
  Specifies the data format for the I²S communication as shown in *Table 424*.

Table 424. Used data format

I2S_DataFormat	Description
I2S_DataFormat_16b	Data are 16 bits long in 16 bits packet frame
I2S_DataFormat_16bextended	Data are 16 bits long in 32 bits packet frame
I2S_DataFormat_24b	Data are 24 bits long in 32 bits packet frame
I2S_DataFormat_32b	Data are 32 bits long in 32 bits packet frame

- **I2S_MCLKOutput**
  Specifies whether the I²S MCLK output is enabled or not as shown in *Table 425*.

Table 425. I²S MCLK output

I2S_MCLKOutput	Description
I2S_MCLKOutput_Enable	I²S MCLK output is enabled
I2S_MCLKOutput_Disable	I²S MCLK output is disabled

- **I2S_AudioFreq**
  Specifies the frequency selected for the I²S communication as shown in *Table 426*.

Table 426. Selecting the I²S frequency

I2S_AudioFreq	Description
I2S_AudioFreq_48k	Configures the I²S baud rate to 48 kHz.
I2S_AudioFreq_44k	Configures the I²S baud rate to 44.1 kHz.
I2S_AudioFreq_22k	Configures the I²S baud rate to 22.05 kHz.
I2S_AudioFreq_16k	Configures the I²S baud rate to 16 kHz.
I2S_AudioFreq_8k	Configures the I²S baud rate to 8 kHz.
I2S_AudioFreq_Default	Configures I2SDIV and ODD to their default values: 0x02 and 0x00, respectively.

- **I2S_CPOL**
  Specifies the idle state of the I2S clock as shown in *Table 427*.

Table 427. I^2S clock idle state

I2S_CPOL	Description
I2S_CPOL_Low	I2S clock's idle state is low
I2S_CPOL_High	I2S clock's idle state is high

Example:

```
/* Initialize the SPI2 according to the I2S_InitStructure members */
  I2S_InitTypeDef  I2S_InitStructure;

  I2S_InitStructure.I2S_Mode = I2S_Mode_MasterTx;
  I2S_InitStructure.I2S_Standard = I2S_Standard_Phillips;
  I2S_InitStructure.I2S_DataFormat = I2S__DataFormat_16bextended;
  I2S_InitStructure.I2S_MCLKOutput = I2S_MCLKOutput_Enable;
  I2S_InitStructure.I2S_AudioFreq = I2S_AudioFreq_16K;
  I2S_InitStructure.I2S_CPOL = SPI_CPOL_Low;

  I2S_Init(SPI2, &I2S_InitStructure);
```

17.2.4 SPI_StructInit function

Table 428 describes the SPI_StructInit function.

Table 428. SPI_StructInit function

Function name	SPI_StructInit
Function prototype	`void SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)`
Behavior description	Fills each SPI_InitStruct member with its default value.
Input parameter	SPI_InitStruct: pointer to a SPI_InitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Refer to *Table 429* for the SPI_InitStruct member default values.

Table 429. SPI_InitStruct default values

Member	Default value
SPI_Direction	SPI_Direction_2Lines_FullDuplex
SPI_Mode	SPI_Mode_Slave
SPI_DataSize	SPI_DataSize_8b
SPI_CPOL	SPI_CPOL_Low
SPI_CPHA	SPI_CPHA_1Edge
SPI_NSS	SPI_NSS_Hard
SPI_BaudRatePrescaler	SPI_BaudRatePrescaler_2
SPI_FirstBit	SPI_FirstBit_MSB
SPI_CRCPolynomial	7

Example:

```
/* Initialize an SPI_InitTypeDef structure */
SPI_InitTypeDef  SPI_InitStructure;
SPI_StructInit(&SPI_InitStructure);
```

17.2.5 I2S_StructInit function

Table 430 describes the I2S_StructInit function.

Table 430. I2S_StructInit function

Function name	I2S_StructInit
Function prototype	void I2S_StructInit(I2S_InitTypeDef* I2S_InitStruct)
Behavior description	Fills each I2S_InitStruct member with its default value.
Input parameter	I2S_InitStruct: pointer to an I2S_InitTypeDef structure that will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The I2S_InitStruct members have the default values given in *Table 431*.

Table 431. Default `I2S_InitStruct` values

Member	Default value
I2S_Mode	I2S_Mode_SlaveTx
I2S_Standard	I2S_Standard_Phillips
I2S_DataFormat	I2S_DataFormat_16b
I2S_MCLKOutput	I2S_MCLKOutput_Disable
I2S_AudioFreq	I2S_AudioFreq_Default
I2S_CPOL	I2S_CPOL_Low

Example:

```
/* Initialize an I2S_InitTypeDef structure */
I2S_InitTypeDef  I2S_InitStructure;
I2S_StructInit(&I2S_InitStructure);
```

17.2.6 SPI_Cmd function

Table 432 describes the SPI_Cmd function.

Table 432. SPI_Cmd function

Function name	SPI_Cmd
Function prototype	`void SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)`
Behavior description	Enables or disables the specified SPI peripheral.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	NewState: new state of the SPIx peripheral. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable SPI3 */
SPI_Cmd(SPI3, ENABLE);
```

17.2.7 I2S_Cmd

Table 430 describes the I2S_Cmd function.

Table 433. I2S_Cmd function

Function name	I2S_Cmd
Function prototype	void I2S_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)
Behavior description	Enables or disables the specified SPI peripheral in I^2S mode.
Input parameter1	SPIx: where x can be 2 or 3 to select the SPI peripheral (in I^2S mode).
Input parameter2	NewState: new state of the SPIx peripheral. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable I2S3 */
I2S_Cmd(SPI3, ENABLE);
```

17.2.8 SPI_I2S_ITConfig function

Table 434 describes the SPI_I2S_ITConfig function.

Table 434. SPI_I2S_ITConfig function

Function name	SPI_I2S_ITConfig
Function prototype	`void SPI_I2S_ITConfig(SPI_TypeDef* SPIx, u8 SPI_I2S_IT, FunctionalState NewState)`
Behavior description	Enables or disables the specified SPI/I2S interrupts.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_I2S_IT: SPI interrupt source to be enabled or disabled. Refer to *SPI_I2S_IT* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the specified SPI/I2S interrupt. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SPI_I2S_IT

SPI_I2S_IT enables or disables SPI/I2S interrupts. See *Table 435* for the values taken by this parameter.

Table 435. SPI_I2S_IT flags

SPI_I2S_IT	Description
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

Example:

```
/* Enable SPI2 Tx buffer empty interrupt */
SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_TXE, ENABLE);
```

17.2.9 SPI_I2S_DMACmd function

Table 436 describes the SPI_I2S_DMACmd function.

Table 436. SPI_I2S_DMACmd function

Function name	SPI_I2S_DMACmd
Function prototype	`void SPI_I2S_DMACmd(SPI_TypeDef* SPIx, u16 SPI_I2S_DMAReq, FunctionalState NewState)`
Behavior description	Enables or disables the SPIx/I2Sx DMA interface.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_I2S_DMAReq: SPI DMA transfer request to be enabled or disabled. Refer to *SPI_I2S_DMAReq* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the selected SPI/I2S DMA transfer request. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SPI_I2S_DMAReq

SPI_I2S_DMAReq enables or disables SPI Tx or/and Rx DMA transfer requests. See *Table 437* for the values taken by this parameter.

Table 437. SPI_I2S_DMAReq values

SPI_I2S_DMAReq	Description
SPI_I2S_DMAReq_Tx	Selects Tx buffer DMA transfer request
SPI_I2S_DMAReq_Rx	Selects Rx buffer DMA transfer request

Example:

```
/* Enable SPI2 Rx buffer DMA transfer request */
SPI_I2S_DMACmd(SPI2, SPI_I2S_DMAReq_Rx, ENABLE);

/* Enable I2S3 Rx buffer DMA transfer request */
SPI_I2S_DMACmd(SPI3, SPI_I2S_DMAReq_Rx, ENABLE);
```

17.2.10 SPI_I2S_SendData function

Table 438 describes the SPI_I2S_SendData function.

Table 438. SPI_I2S_SendData function

Function name	SPI_I2S_SendData
Function prototype	`void SPI_I2S_SendData(SPI_TypeDef* SPIx, u16 Data)`
Behavior description	Transmits data through the SPIx/I2Sx peripheral.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter 2	Data: Byte or half word (in SPI mode), or half word (in I^2S mode) to be transmitted.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send 0xA5 through the SPI1 peripheral */
SPI_I2S_SendData(SPI1, 0xA5);
```

17.2.11 SPI_I2S_ReceiveData function

Table 439 describes the SPI_I2S_ReceiveData function.

Table 439. SPI_I2S_ReceiveData function

Function name	SPI_I2S_ReceiveData
Function prototype	`u16 SPI_I2S_ReceiveData(SPI_TypeDef* SPIx)`
Behavior description	Returns the most recent data received through the SPIx/I2Sx peripheral.
Input parameter	SPIx: where x can be 1, 2 or 3 to select the SPI/I2S? peripheral.
Output parameter	None
Return parameter	The value of the received data.
Required preconditions	None
Called functions	None

Example:

```
/* Read the most recent data received by the SPI2 peripheral */
u16 ReceivedData;
ReceivedData = SPI_I2S_ReceiveData(SPI2);
```

17.2.12 SPI_NSSInternalSoftwareConfig function

Table 440 describes the SPI_NSSInternalSoftwareConfig function.

Table 440. SPI_NSSInternalSoftwareConfig function

Function name	SPI_NSSInternalSoftwareConfig
Function prototype	`void SPI_NSSInternalSoftwareConfig(SPI_TypeDef* SPIx, u16 SPI_NSSInternalSoft)`
Behavior description	Internally configures by software the NSS pin for the specified SPIx interface.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_NSSInternalSoft: SPI NSS internal state. Refer to *SPI_NSSInternalSoft* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SPI_NSSInternalSoft

SPI_NSSInternalSoft internally sets or resets the NSS pin. See *Table 441* for the values taken by this parameter.

Table 441. SPI_NSSInternalSoft values

SPI_NSSInternalSoft	Description
SPI_NSSInternalSoft_Set	Set NSS pin internally
SPI_NSSInternalSoft_Reset	Reset NSS pin internally

Example:

```
/* Set internally by software the SPI1 NSS pin */
SPI_NSSInternalSoftwareConfig(SPI1, SPI_NSSInternalSoft_Set);

/* Reset internally by sofwtare the SPI2 NSS pin */
SPI_NSSInternalSoftwareConfig(SPI2, SPI_NSSInternalSoft_Reset);
```

17.2.13 SPI_SSOutputCmd function

Table 442 describes the SPI_SSOutputCmd function.

Table 442. SPI_SSOutputCmd function

Function name	SPI_SSOutputCmd
Function prototype	`void SPI_SSOutputCmd(SPI_TypeDef* SPIx, FunctionalState NewState)`
Behavior description	Enables or disables the SS output for the selected SPI.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	NewState: new state of the SPIx SS output. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the SPI1 SS output: single master mode */
SPI_SSOutputCmd(SPI1, ENABLE);
```

17.2.14 SPI_DataSizeConfig function

Table 443 describes the SPI_DataSizeConfig function.

Table 443. SPI_DataSizeConfig function

Function name	SPI_DataSizeConfig
Function prototype	`void SPI_DataSizeConfig(SPI_TypeDef* SPIx, u16 SPI_DatSize)`
Behavior description	Configures the data size for the selected SPI peripheral
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_DataSize: SPI data size. Refer to *SPI_DataSize* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SPI_DataSize

SPI_DataSize sets 8-bit or 16-bit data frame format. See *Table 444* for the values taken by this parameter.

Table 444. SPI_DataSize values

SPI_NSSInternalSoft	Description
SPI_DataSize_8b	Set 8-bit data size
SPI_DataSize_16b	Set 16-bit data size

Example:

```
/* Set 8bit data frame format for SPI1 */
SPI_DataSizeConfig(SPI1, SPI_DataSize_8b);

/* Set 16bit data frame format for SPI2 */
SPI_DataSizeConfig(SPI2, SPI_DataSize_16b);
```

17.2.15 SPI_TransmitCRC function

Table 445 describes the SPI_TransmitCRC function.

Table 445. SPI_TransmitCRC function

Function name	SPI_TransmitCRC
Function prototype	`void SPI_TransmitCRC(SPI_TypeDef* SPIx)`
Behavior description	Transmit of the SPIx CRC value.
Input parameter	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the CRC transfer for SPI1 */
SPI_TransmitCRC(SPI1);
```

17.2.16 SPI_CalculateCRC function

Table 446 describes the SPI_CalculateCRC function.

Table 446. SPI_CalculateCRC function

Function name	SPI_CalculateCRC
Function prototype	`void SPI_CalculateCRC(SPI_TypeDef* SPIx, FunctionalState NewState)`
Behavior description	Enables or disables the CRC value calculation of the transferred bytes.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	NewState: new state of the SPIx CRC value calculation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the CRC calculation for the transfered bytes from SPI2 */
SPI_CalculateCRC(SPI2, ENABLE);
```

17.2.17 SPI_GetCRC function

Table 447 describes the SPI_GetCRC function.

Table 447. SPI_GetCRC function

Function name	SPI_GetCRC
Function prototype	`u16 SPI_GetCRC(SPI_TypeDef* SPIx, u8 SPI_CRC)`
Behavior description	Returns the transmit or the receive CRC register value for the specified SPI peripheral.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_CRC: CRC register to be read. Refer to *SPI_CRC* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The selected CRC register value.
Required preconditions	None
Called functions	None

SPI_CRC

SPI_CRC selects the SPI Rx or SPI Tx CRC register. See *Table 448* for the values taken by this parameter.

Table 448. SPI_CRC values

SPI_CRC	Description
SPI_CRC_Tx	Selects Tx CRC register
SPI_CRC_Rx	Selects Rx CRC register

Example:

```
/* Returns the SPI1 transmit CRC register */
u16 CRCValue;
CRCValue = SPI_GetCRC(SPI1, SPI_CRC_Tx);
```

17.2.18 SPI_GetCRCPolynomial function

Table 449 describes the SPI_GetCRCPolynomial function.

Table 449. SPI_GetCRCPolynomial function

Function name	SPI_GetCRCPolynomial
Function prototype	`u16 SPI_GetCRCPolynomial(SPI_TypeDef* SPIx)`
Behavior description	Returns the CRC Polynomial register value for the specified SPI.
Input parameter	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Output parameter	None
Return parameter	The CRC Polynomial register value.
Required preconditions	None
Called functions	None

Example:

```
/* Returns the SPI2 CRC polynomial register */
u16 CRCPolyValue;
CRCPolyValue = SPI_GetCRCPolynomial(SPI2);
```

17.2.19 SPI_BiDirectionalLineConfig function

Table 450 describes the SPI_BiDirectionalLineConfig function.

Table 450. SPI_BiDirectionalLineConfig function

Function name	SPI_BiDirectionalLineConfig
Function prototype	`SPI_BiDirectionalLineConfig(SPI_TypeDef* SPIx, u16 SPI_Direction)`
Behavior description	Selects the data transfer direction in bidirectional mode for the specified SPI.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_Direction: data transfer direction in bidirectional mode. Refer to *SPI_Direction* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SPI_Direction

SPI_Direction configures data transfer direction in bidirectional mode. See *Table 451* for the values taken by this parameter.

Table 451. SPI_Direction values

SPI_Direction	Description
SPI_Direction_Tx	Selects Tx transmission direction
SPI_Direction_Rx	Selects Rx receive direction

Example:

```
/* Set the SPI2 in bidirectional transmit only mode */
SPI_BiDirectionalLineConfig(SPI_Direction_Tx);
```

17.2.20 SPI_I2S_GetFlagStatus function

Table 452 describes the SPI_I2S_GetFlagStatus function.

Table 452. SPI_I2S_GetFlagStatus function

Function name	SPI_I2S_GetFlagStatus
Function prototype	`FlagStatus SPI_I2S_GetFlagStatus(SPI_TypeDef* SPIx, u16 SPI_I2S_FLAG)`
Behavior description	Checks whether the specified SPI/I2S flag is set or not.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI/I2S peripheral.
Input parameter2	SPI_I2S_FLAG: flag to be checked. Refer to *SPI_I2S_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of SPI_I2S_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

SPI_I2S_FLAG

The SPI/I2S flags that can be checked by issuing an SPI_I2S_GetFlagStatus function are listed in *Table 453*.

Table 453. SPI_I2S_FLAG flags

SPI_I2S_FLAG	Description
SPI_I2S_FLAG_BSY	Busy flag
SPI_I2S_FLAG_OVR	Overrun flag
SPI_FLAG_MODF	Mode fault flag
SPI_FLAG_CRCERR	CRC error flag
I2S_FLAG_UDR	Underrun flag
I2S_FLAG_CHSIDE	Channel side flag
SPI_I2S_FLAG_TXE	Transmit buffer empty flag
SPI_I2S_FLAG_RXNE	Receive buffer not empty flag

Example:

```
/* Test if the SPI1 transmit buffer empty flag is set or not */
FlagStatus Status;
Status = SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE);

/* Get the I2S3 received (or to be transmitted) data channel side
(left or right) */
FlagStatus Status;
Status = SPI_I2S_GetFlagStatus(SPI3, I2S_FLAG_CHSIDE);
```

17.2.21 SPI_I2S_ClearFlag function

Table 454 describes the SPI_I2S_ClearFlag function.

Table 454. SPI_I2S_ClearFlag function

Function name	SPI_I2S_ClearFlag
Function prototype	`void SPI_I2S_ClearFlag(SPI_TypeDef* SPIx, u16 SPI_I2S_FLAG)`
Behavior description	Clears the SPIx CRC error (CRCERR) flag.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI peripheral.
Input parameter2	SPI_I2S_FLAG: specifies the SPI flag to clear. This function clears only the CRCERR flag. Notes: – BSY, TXE and RXNE flags are reset by hardware – OVR (OverRun error) flag is cleared by a software sequence: a read operation to the SPI_DR register (`SPI_I2S_ReceiveData()`) followed by a read operation to the SPI_SR register (`SPI_I2S_GetFlagStatus()`). – UDR (UnderRun error) flag is cleared by a read operation to the SPI_SR register (`SPI_I2S_GetFlagStatus()`). – MODF (Mode Fault) flag is cleared by a software sequence: a read/write operation to the SPI_SR register (`SPI_I2S_GetFlagStatus()`) followed by a write operation to the SPI_CR1 register (`SPI_Cmd()` to enable the SPI).
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the SPI2 CRCERR pending bit */
SPI_I2S_ClearFlag(SPI2, SPI_FLAG_CRCERR);
```

17.2.22 SPI_I2S_GetITStatus function

Table 455 describes the SPI_I2S_GetITStatus function.

Table 455. SPI_I2S_GetITStatus function

Function name	SPI_I2S_GetITStatus
Function prototype	`ITStatus SPI_I2S_GetITStatus(SPI_TypeDef* SPIx, u8 SPI_I2S_IT)`
Behavior description	Checks whether the specified SPI interrupt has occurred or not.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPI/I2S peripheral.
Input parameter2	SPI_I2S_IT: SPI/I2S interrupt source to be checked. Refer to *SPI_I2S_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of SPI_I2S_IT (SET or RESET).
Required preconditions	None
Called functions	None

SPI_I2S_IT

The SPI/I2S interrupt that can be checked by issuing an SPI_I2S_GetITStatus function are listed in *Table 456*.

Table 456. SPI_I2S_IT flags

SPI_I2S_IT	Description
I2S_IT_UDR	I2S Underrun Error interrupt
SPI_I2S_IT_OVR	Overrun interrupt flag
SPI_IT_MODF	Mode Fault interrupt flag
SPI_IT_CRCERR	CRC Error interrupt flag
SPI_I2S_IT_TXE	Transmit buffer empty interrupt flag
SPI_I2S_IT_RXNE	Receive buffer not empty interrupt flag

Example:

```
/* Test if the SPI1 Overrun interrupt has occurred or not */
ITStatus Status;
Status = SPI_I2S_GetITStatus(SPI1, SPI_I2S_IT_OVR);

/* Test if the I2S2 Underrun interrupt has occurred or not */
ITStatus Status;
Status = SPI_I2S_GetITStatus(SPI2, I2S_IT_UDR);
```

17.2.23 SPI_I2S_ClearITPendingBit function

Table 457 describes the SPI_I2S_ClearITPendingBit function.

Table 457. SPI_I2S_ClearITPendingBit function

Function name	SPI_I2S_ClearITPendingBit
Function prototype	`void SPI_I2S_ClearITPendingBit(SPI_TypeDef* SPIx, u8 SPI_I2S_IT)`
Behavior description	Clears the SPIx CRC Error (CRCERR) interrupt pending bit.
Input parameter1	SPIx: where x can be 1, 2 or 3 to select the SPIx peripheral.
Input parameter2	SPI_I2S_IT: specifies the SPI interrupt pending bit to clear. This function clears only the CRCERR interrupt pending bit. Notes: – TXE and RXNE interrupt flags are reset by hardware – OVR (OverRun Error) interrupt pending bit is cleared by a software sequence: a read operation to the SPI_DR register (`SPI_I2S_ReceiveData()`) followed by a read operation to the SPI_SR register (`SPI_I2S_GetITStatus()`). – UDR (UnderRun Error) interrupt pending bit is cleared by a read operation to the SPI_SR register (`SPI_I2S_GetITStatus()`). – MODF (Mode Fault) interrupt pending bit is cleared by a software sequence: a read/write operation to the SPI_SR register (`SPI_I2S_GetITStatus()`) followed by a write operation to the SPI_CR1 register (`SPI_Cmd()` to enable the SPI).
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the SPI2 CRC error interrupt pending bit */
SPI_I2S_ClearITPendingBit(SPI2, SPI_IT_CRCERR);
```

18 Cortex system timer (SysTick)

The Systick provides a simple 24-bit decrementing wrap-on-zero clear-on-write counter with a flexible control mechanism.

Section 18.1: SysTick register structure describes the data structures used in the SysTick Firmware Library. *Section 18.2: Firmware library functions* presents the Firmware Library functions.

18.1 SysTick register structure

The SysTick register structure, *SysTick_TypeDef, is* defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 CTRL;
  vu32 LOAD;
  vu32 VAL;
  vuc32 CALIB;
} SysTick_TypeDef;
```

Table 458 gives the list of the SysTick registers.

Table 458. SysTick registers

Register	Description
CTRL	SysTick Control and Status Register
LOAD	SysTick Reload value Register
VAL	SysTick Current value Register
CALIB	SysTick Calibration value Register

The SysTick peripheral is declared in *stm32f10x_map*.h:

```
#define SCS_BASE                ((u32)0xE000E000)
#define SysTick_BASE            (SCS_BASE + 0x0010)
#ifndef DEBUG
...
#ifdef _SysTick
  #define SysTick               ((SysTick_TypeDef *) SysTick_BASE)
#endif /*_SysTick */
...
#else   /* DEBUG */
...
#ifdef _SysTick
  EXT SysTick_TypeDef           *SysTick;
#endif /*_SysTick */
...
#endif
```

When using the Debug mode, SysTick pointer is initialized in *stm32f10x_lib.c* file:

```
#ifdef _SysTick
SysTick = (SysTick_TypeDef *)  SysTick_BASE;
#endif /*_SysTick */
```

To access the SysTick registers, _SysTick must be defined in *stm32f10x_conf.h* as follows:

```
#define _SysTick
```

18.2 Firmware library functions

Table 459 gives the list of the various functions of the SysTick library.

Table 459. SysTick firmware library functions

Function name	Description
SysTick_CLKSourceConfig	Configures the SysTick clock source.
SysTick_SetReload	Sets SysTick Reload value.
SysTick_CounterCmd	Enables or disables the SysTick counter.
SysTick_ITConfig	Enables or disables the SysTick Interrupt.
SysTick_GetCounter	Gets SysTick counter value.
SysTick_GetFlagStatus	Checks whether the specified SysTick flag is set or not.

18.2.1 SysTick_CLKSourceConfig function

Table 460 describes the SysTick_CLKSourceConfig function.

Table 460. SysTick_CLKSourceConfig function

Function name	SysTick_CLKSourceConfig
Function prototype	`void SysTick_CLKSourceConfig(u32 SysTick_CLKSource)`
Behavior description	Configures the SysTick clock source.
Input parameter	SysTick_CLKSource: SysTick clock source. Refer to *SysTick_CLKSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SysTick_CLKSource

SysTick_CLKSource selects the SysTick clock source. Refer to *Table 461* for the values taken by this parameter.

Table 461. SysTick_CLKSource values

SysTick_CLKSource	Description
SysTick_CLKSource_HCLK_Div8	SysTick clock source = AHB clock divided by 8
SysTick_CLKSource_HCLK	SysTick clock source = AHB clock

Example:

```
/* AHB clock selected as SysTick clock source */
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
```

18.2.2 SysTick_SetReload function

Table 462 describes the SysTick_SetReload function.

Table 462. SysTick_SetReload function

Function name	SysTick_SetReload
Function prototype	`void SysTick_SetReload(u32 Reload)`
Behavior description	Sets SysTick Reload value.
Input parameter	Reload: SysTick Reload new value. This parameter must be a number between 1 and 0x00FFFFFF.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set SysTick reload value to 0xFFFF */
SysTick_SetReload(0xFFFF);
```

18.2.3 SysTick_CounterCmd function

Table 463 describes the SysTick_CounterCmd function.

Table 463. SysTick_CounterCmd function

Function name	SysTick_CounterCmd
Function prototype	`void SysTick_CounterCmd(u32 SysTick_Counter)`
Behavior description	Enables or disables the SysTick counter.
Input parameter	SysTick_Counter: new state of the SysTick counter. Refer to *SysTick_Counter* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SysTick_Counter

SysTick_Counter selects the SysTick counter state. Refer to *Table 464* for the values taken by this parameter.

Table 464. SysTick_Counter values

SysTick_Counter	Description
SysTick_Counter_Disable	Disable counter
SysTick_Counter_Enable	Enable counter
SysTick_Counter_Clear	Clear counter value to 0

Example:

```
/* Enable SysTick counter */
SysTick_CounterCmd(SysTick_Counter_Enable);
```

18.2.4 SysTick_ITConfig function

Table 465 describes the SysTick_ITConfig function.

Table 465. SysTick_ITConfig function

Function name	SysTick_ITConfig
Function prototype	`void SysTick_ITConfig(FunctionalState NewState)`
Behavior description	Enables or disables the SysTick Interrupt.
Input parameter	NewState: new state of the SysTick Interrupt. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable SysTick interrupt */
SysTick_ITConfig(ENABLE);
```

18.2.5 SysTick_GetCounter function

Table 466 describes the SysTick_GetCounter function.

Table 466. SysTick_GetCounter function

Function name	SysTick_GetCounter
Function prototype	`u32 SysTick_GetCounter(void)`
Behavior description	Gets SysTick counter value.
Input parameter	None
Output parameter	None
Return parameter	SysTick current value.
Required preconditions	None
Called functions	None

Example:

```
/* Get SysTick current counter value */
u32 SysTickCurrentCounterValue;
SysTickCurrentCounterValue = SysTick_GetCounter();
```

18.2.6 SysTick_GetFlagStatus function

Table 467 describes the SysTick_GetFlagStatus function.

Table 467. SysTick_GetFlagStatus function

Function name	SysTick_GetFlagStatus
Function prototype	`FlagStatus SysTick_GetFlagStatus(u8 SysTick_FLAG)`
Behavior description	Checks whether the specified SysTick flag is set or not.
Input parameter	SysTick_FLAG: flag to be checked. Refer to *SysTick_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of SysTick_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

SysTick_FLAG

The SysTick flags that can be checked by issuing a SysTick_GetFlagStatus function are listed in the following table:

Table 468. SysTick flags

SysTick_FLAG	Description
SysTick_FLAG_COUNT	1 = timer counted to 0 since last time this was read.
SysTick_FLAG_SKEW	1 = the calibration value is not exactly 10ms because of clock frequency.
SysTick_FLAG_NOREF	1 = the reference clock is not provided

Example:

```
/* Test if the Count flag is set or not */
FlagStatus Status;
Status = SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
if(Status == RESET)
{
...
}
else
{
...
}
```

19 Advanced-control timer, general-purpose timer and basic timer (TIM)

Each TIM timer consists of a 16-bit auto-reload counter driven by a programmable prescaler.

The advanced-control and general-purpose timers can be used for a variety of purposes, including the measurement of input signal pulse lengths (input capture) or the generation of output waveforms (output compare, PWM, complementary PWM with dead-time insertion for advanced-control timers, etc.).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the CPU clock prescaler.

The basic timers may be used as generic timers for time-base generation but they are also specifically used to drive the digital-to-analog converter (DAC). The basic timers are internally connected to the DAC and are able to drive it through their trigger outputs.

Section 18.1: SysTick register structure describes the data structures used in the TIM firmware library. *Section 18.2: Firmware library functions* presents the firmware library functions.

19.1 TIM register structure

The TIM register structure, TIM_*TypeDef*, is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
vu16 CR1;
  u16 RESERVED0;
  vu16 CR2;
  u16 RESERVED1;
  vu16 SMCR;
  u16 RESERVED2;
  vu16 DIER;
  u16 RESERVED3;
  vu16 SR;
  u16 RESERVED4;
  vu16 EGR;
  u16 RESERVED5;
  vu16 CCMR1;
  u16 RESERVED6;
  vu16 CCMR2;
  u16 RESERVED7;
  vu16 CCER;
  u16 RESERVED8;
  vu16 CNT;
  u16 RESERVED9;
  vu16 PSC;
  u16 RESERVED10;
  vu16 ARR;
  u16 RESERVED11;
```

```
  vu16 RCR;
  u16 RESERVED12;
  vu16 CCR1;
  u16 RESERVED13;
  vu16 CCR2;
  u16 RESERVED14;
  vu16 CCR3;
  u16 RESERVED15;
  vu16 CCR4;
  u16 RESERVED16;
  vu16 BDTR;
  u16 RESERVED17;
  vu16 DCR;
  u16 RESERVED18;
  vu16 DMAR;
  u16 RESERVED19;
} TIM_TypeDef;
```

Table 469 gives the list of TIM registers.

Table 469. TIM registers

Register	Description
CR1	Control Register1
CR2	Control Register2
SMCR	Slave Mode Control Register
DIER	DMA and Interrupt Enable Register
SR	Status Register
EGR	Event Generation Register
CCMR1	Capture/Compare Mode Register 1
CCMR2	Capture/Compare Mode Register 2
CCER	Capture/Compare Enable Register
CNT	Counter Register
PSC	Prescaler Register
ARR	Auto-Reload Register
RCR	Repetition Counter Register
CCR1	Capture/Compare Register 1
CCR2	Capture/Compare Register 2
CCR3	Capture/Compare Register 3
CCR4	Capture/Compare Register 4
BDTR	Break and Dead Time Register
DCR	DMA Control Register
DMAR	DMA Address for Burst mode Register

The TIM peripheral is declared in the *stm32f10x_map* file:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define APB3PERIPH_BASE       (PERIPH_BASE + 0x18000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
...
#define TIM2_BASE             (APB1PERIPH_BASE + 0x0000)
#define TIM3_BASE             (APB1PERIPH_BASE + 0x0400)
#define TIM4_BASE             (APB1PERIPH_BASE + 0x0800)
#define TIM5_BASE             (APB1PERIPH_BASE + 0x0C00)
#define TIM6_BASE             (APB1PERIPH_BASE + 0x1000)
#define TIM7_BASE             (APB1PERIPH_BASE + 0x1400)
...
#define TIM1_BASE             (APB2PERIPH_BASE + 0x2C00)
....
#define TIM8_BASE             (APB2PERIPH_BASE + 0x3400)
...
#ifndef DEBUG
...
#ifdef _TIM2
  #define TIM2 ((TIM_TypeDef *) TIM2_BASE)
#endif /*_TIM2 */
#ifdef _TIM3
  #define TIM3 ((TIM_TypeDef *) TIM3_BASE)
#endif /*_TIM3 */
#ifdef _TIM4
  #define TIM4 ((TIM_TypeDef *) TIM4_BASE)
#endif /*_TIM4 */
#ifdef _TIM
  #define TIM5 ((TIM_TypeDef *) TIM5_BASE)
#endif /*_TIM5 */
#ifdef _TIM6
  #define TIM6 ((TIM_TypeDef *) TIM6_BASE)
#endif /*_TIM6 */
#ifdef _TIM7
  #define TIM7 ((TIM_TypeDef *) TIM7_BASE)
#endif /*_TIM7 */
....
#ifdef _TIM1
  #define TIM1 ((TIM_TypeDef *) TIM1_BASE)
#endif /*_TIM */
....
#ifdef _TIM8
  #define TIM8 ((TIM_TypeDef *) TIM8_BASE)
#endif /*_TIM8 */
...
#else   /* DEBUG */
....
#ifdef _TIM2
  EXT TIM_TypeDef            *TIM2;
#endif /*_TIM2 */
```

```
#ifdef _TIM3
  EXT TIM_TypeDef            *TIM3;
#endif /*_TIM3 */
#ifdef _TIM4
  EXT TIM_TypeDef            *TIM4;
#endif /*_TIM4 */

#ifdef _TIM5
  EXT TIM_TypeDef            *TIM5;
#endif /*_TIM5 */
#ifdef _TIM6
  EXT TIM_TypeDef            *TIM6;
#endif /*_TIM6 */
#ifdef _TIM7
  EXT TIM_TypeDef            *TIM7;
#endif /*_TIM7 */
...
#ifdef _TIM1
  EXT TIM_TypeDef            *TIM1;
#endif /*_TIM */
..
#ifdef _TIM8
  EXT TIM_TypeDef            *TIM8;
#endif /*_TIM8 */
..
#endif
```

When using the Debug mode, _TIM pointer is initialized in the *stm32f10x_lib.c* file:

```
...
#ifdef _TIM1
  TIM1 = (TIM_TypeDef *)  TIM1_BASE;
#endif /*_TIM */

#ifdef _TIM2
  TIM2 = (TIM_TypeDef *)  TIM2_BASE;
#endif /*_TIM2 */

#ifdef _TIM3
  TIM3 = (TIM_TypeDef *)  TIM3_BASE;
#endif /*_TIM3 */

#ifdef _TIM4
  TIM4 = (TIM_TypeDef *)  TIM4_BASE;
#endif /*_TIM4 */

#ifdef _TIM5
  TIM5 = (TIM_TypeDef *)  TIM5_BASE;
#endif /*_TIM5 */

#ifdef _TIM6
  TIM6 = (TIM_TypeDef *)  TIM6_BASE;
#endif /*_TIM6 */
```

```
#ifdef _TIM7
  TIM7 = (TIM_TypeDef *)  TIM7_BASE;
#endif /*_TIM7 */

#ifdef _TIM8
  TIM8 = (TIM_TypeDef *)  TIM8_BASE;
#endif /*_TIM8 */
...
To access TIM registers, _TIM must be defined in stm32f10x_conf.h as
follows:
...
#define _TIM
#define _TIM1
#define _TIM2
#define _TIM3
#define _TIM4
#define _TIM5
#define _TIM6
#define _TIM7
#define _TIM8
...
```

19.2 Firmware library functions

Table 470 gives the list of the various functions of the TIM library.

Table 470. TIM firmware library functions

Function name	Description
TIM_DeInit	Resets the TIM peripheral registers to their default reset values.
TIM_TimeBaseInit	Initializes the TIM Time Base Unit according to the specified parameters in the TIM_TimeBaseInitStruct.
TIM_OC1Init	Initializes the TIM Channel1 according to the specified parameters in the TIM_OCInitStruct.
TIM_OC2Init	Initializes the TIM Channel2 according to the specified parameters in the TIM_OCInitStruct.
TIM_OC3Init	Initializes the TIM Channel3 according to the specified parameters in the TIM_OCInitStruct.
TIM_OC4Init	Initializes the TIM Channel4 according to the specified parameters in the TIM_OCInitStruct.
TIM_ICInit	Initializes the TIM peripheral according to the specified parameters in the TIM_ICInitStruct.
TIM_PWMIConfig	Configures the TIM peripheral in PWM Input Mode according to the specified parameters in the TIM_ICInitStruct.
TIM_BDTRConfig	Configures the: Break feature, dead time, Lock level, OSSI, OSSR State and AOE (automatic output enable).
TIM_TimeBaseStructInit	Fills each TIM_TimeBaseInitStruct member with its default value.
TIM_OCStructInit	Fills each TIM_OCInitStruct member with its default value.

Table 470. TIM firmware library functions (continued)

Function name	Description
TIM_ICStructInit	Fills each TIM_ICInitStruct member with its default value.
TIM_BDTRStructInit	Fills each TIM_BDTRInitStruct member with its default value.
TIM_Cmd	Enables or disables the specified TIM peripheral.
TIM_CtrlPWMOutputs	Enables or disables the TIM peripheral main outputs.
TIM_ITConfig	Enables or disables the specified TIM interrupts.
TIM_GenerateEvent	Configures the TIM event to be generated by software.
TIM_DMAConfig	Configures the TIM DMA interface.
TIM_DMACmd	Enables or disables the TIM DMA requests.
TIM_InternalClockConfig	Configures the TIM's internal Clock.
TIM_ITRxExternalClockConfig	Configures the TIM's internal trigger as external clock.
TIM_TIxExternalClockConfig	Configures the TIM trigger as external clock.
TIM_ETRClockMode1Config	Configures the TIM's external clock Mode1.
TIM_ETRClockMode2Config	Configures the TIM's external clock Mode2.
TIM_ETRConfig	Configures the TIM's external trigger (ETR).
TIM_PrescalerConfig	Configures the TIM prescaler.
TIM_CounterModeConfig	Specifies the TIM counter mode to be used.
TIM_SelectInputTrigger	Selects the TIM input trigger source.
TIM_EncoderInterfaceConfig	Configures the TIM encoder interface.
TIM_ForcedOC1Config	Forces the TIM Channel1 output waveform to active or inactive level.
TIM_ForcedOC2Config	Forces the TIM Channel2 output waveform to active or inactive level.
TIM_ForcedOC3Config	Forces the TIM Channel3 output waveform to active or inactive level.
TIM_ForcedOC4Config	Forces the TIM Channel4 output waveform to active or inactive level.
TIM_ARRPreloadConfig	Enables or disables the TIM peripheral Preload register on ARR.
TIM_SelectCOM	Selects the TIM peripheral Commutation event.
TIM_SelectCCDMA	Selects the TIM peripheral Capture Compare DMA source.
TIM_CCPreloadControl	Sets or Resets the TIM peripheral Capture Compare Preload Control bit.
TIM_OC1PreloadConfig	Enables or disables the TIM peripheral Preload Register on CCR1.
TIM_OC2PreloadConfig	Enables or disables the TIM peripheral Preload Register on CCR2.
TIM_OC3PreloadConfig	Enables or disables the TIM peripheral Preload Register on CCR3.
TIM_OC4PreloadConfig	Enables or disables the TIM peripheral Preload Register on CCR4.
TIM_OC1FastConfig	Configures the TIM Capture Compare 1 Fast feature.
TIM_OC2FastConfig	Configures the TIM Capture Compare 2 Fast feature.
TIM_OC3FastConfig	Configures the TIM Capture Compare 3 Fast feature.
TIM_OC4FastConfig	Configures the TIM Capture Compare 4 Fast feature.
TIM_ClearOC1Ref	Clears or safeguards the OCREF1 signal on an external event

Table 470. TIM firmware library functions (continued)

Function name	Description
TIM_ClearOC2Ref	Clears or safeguards the OCREF2 signal on an external event
TIM_ClearOC3Ref	Clears or safeguards the OCREF3 signal on an external event
TIM_ClearOC4Ref	Clears or safeguards the OCREF4 signal on an external event
TIM_OC1PolarityConfig	Configures the TIM Channel 1 polarity.
TIM_OC1NPolarityConfig	Configures the TIM Channel 1N polarity.
TIM_OC2PolarityConfig	Configures the TIM Channel 2 polarity.
TIM_OC2NPolarityConfig	Configures the TIM Channel 2N polarity.
TIM_OC3PolarityConfig	Configures the TIM Channel 3 polarity.
TIM_OC3NPolarityConfig	Configures the TIM Channel 3N polarity.
TIM_OC4PolarityConfig	Configures the TIM Channel 4 polarity.
TIM_CCxCmd	Enables or disables the TIM Capture Compare Channel x.
TIM_CCxNCmd	Enables or disables the TIM Capture Compare Channel xN.
TIM_SelectOCxM	Selects the TIM Output Compare mode. This function disables the selected channel before changing the Output Compare mode. User has to enable this channel using TIM_CCxCmd and TIM_CCxNCmd functions.
TIM_UpdateDisableConfig	Enables or Disables the TIM update event.
TIM_UpdateRequestConfig	Selects the TIM update request interrupt source.
TIM_SelectHallSensor	Enables or disables the TIM's hall sensor interface.
TIM_SelectOnePulseMode	Enables or disables the TIM's one-pulse mode.
TIM_SelectOutputTrigger	Selects the TIM trigger output mode.
TIM_SelectSlaveMode	Selects the TIM slave mode.
TIM_SelectMasterSlaveMode	Sets or resets the TIM master/slave mode.
TIM_SetCounter	Sets the TIM Counter Register value.
TIM_SetAutoreload	Sets the TIM Autoreload Register value.
TIM_SetCompare1	Sets the TIM Capture Compare1 Register value.
TIM_SetCompare2	Sets the TIM Capture Compare2 Register value.
TIM_SetCompare3	Sets the TIM Capture Compare3 Register value.
TIM_SetCompare4	Sets the TIM Capture Compare4 Register value.
TIM_SetIC1Prescaler	Sets the TIM Input Capture 1 prescaler.
TIM_SetIC2Prescaler	Sets the TIM Input Capture 2 prescaler.
TIM_SetIC3Prescaler	Sets the TIM Input Capture 3 prescaler.
TIM_SetIC4Prescaler	Sets the TIM Input Capture 4 prescaler.
TIM_SetClockDivision	Sets the TIM clock division value.
TIM_GetCapture1	Gets the TIM Input Capture 1 value.
TIM_GetCapture2	Gets the TIM Input Capture 2 value.

Table 470. TIM firmware library functions (continued)

Function name	Description
TIM_GetCapture3	Gets the TIM Input Capture 3 value.
TIM_GetCapture4	Gets the TIM Input Capture 4 value.
TIM_GetCounter	Gets the TIM counter value.
TIM_GetPrescaler	Gets the Prescaler value.
TIM_GetFlagStatus	Checks whether the specified TIM flag is set or not.
TIM_ClearFlag	Clears the TIM's pending flags.
TIM_GetITStatus	Checks whether the specified TIM interrupt has occurred or not.
TIM_ClearITPendingBit	Clears the TIM's interrupt pending bits.

19.2.1 TIM_DeInit function

Table 471 describes the TIM_DeInit function.

Table 471. TIM_DeInit function

Function name	TIM_DeInit
Function prototype	`void TIM_DeInit(TIM_TypeDef* TIMx)`
Behavior description	Resets the TIM peripheral registers to their default reset values.
Input parameter	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB2PeriphResetCmd and RCC_APB1PeriphResetCmd

Example:

```
/* Resets TIM1 */
TIM_DeInit(TIM1);
```

19.2.2 TIM_TimeBaseInit function

Table 472 describes the TIM_TimeBaseInit function.

Table 472. TIM_TimeBaseInit function

Function name	TIM_TimeBaseInit
Function prototype	`void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)`
Behavior description	Initializes the TIM Time Base Unit according to the parameters specified in the TIM_TimeBaseInitStruct.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_TimeBaseInitStruct: pointer to a TIM_TimeBaseInitTypeDef structure that contains the configuration information for the specified TIM Time Base Unit. Refer to *TIM_TimeBaseInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_TimeBaseInitTypeDef structure

The TIM_TimeBaseInitTypeDef structure is defined in the *stm32f10x_TIM.h* file:

```
typedef struct
{
  u16 TIM_Period;
  u16 TIM_Prescaler;
  u16 TIM_ClockDivision;
  u16 TIM_CounterMode;
  u8 TIM_RepetitionCounter;
} TIM_TimeBaseInitTypeDef;
```

This sturcture is used with all TIMx except for TIM6 and TIM7.

TIM_Period

TIM_Period configures the period value to be loaded into the active Auto-Reload Register at the next update event. This member must be a number between 0x0000 and 0xFFFF.

TIM_Prescaler

TIM_Prescaler configures the prescaler value used to divide the TIM clock. This member must be a number between 0x0000 and 0xFFFF.

TIM_ClockDivision

TIM_ClockDivision configures the clock division. This member can be set to one of the following values:

Table 473. TIM_ClockDivision definition

TIM_ClockDivision	Description
TIM_CKD_DIV1	$T_{DTS} = T_{ck\_tim}$
TIM_CKD_DIV2	$T_{DTS} = 2 \times T_{ck\_tim}$
TIM_CKD_DIV4	$T_{DTS} = 4 \times T_{ck\_tim}$

TIM_CounterMode

TIM_CounterMode selects the counter mode. This member can be set to one of the following values:

Table 474. TIM_CounterMode definition

TIM_CounterMode	Description
TIM_Counter_Up	TIM Upcounting mode.
TIM_Counter_Down	TIM Downcounting mode.
TIM_Counter_CenterAligned1	TIM CenterAligned Mode1 Counting mode.
TIM_Counter_CenterAligned2	TIM CenterAligned Mode2 Counting mode.
TIM_Counter_CenterAligned3	TIM CenterAligned Mode3 Counting mode.

TIM_RepetitionCounter

TIM_RepetitionCounter configures the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N).

This means in PWM mode that (N+1) corresponds to:

- the number of PWM periods in edge-aligned mode
- the number of half PWM period in center-aligned mode

This member must be a number between 0x00 and 0xFF. This parameter is valid only for TIM1 and TIM8.

19.2.3 TIM_OC1Init function

Table 475 describes the TIM_OC1Init function.

Table 475. TIM_OC1Init function

Function name	TIM_OC1Init
Function prototype	`void TIM_OC1Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)`
Behavior description	Initializes the TIM Channel 1 according to the parameters specified in the TIM_OCInitStruct.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCInitStruct: pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to *TIM_OCInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_OCInitTypeDef structure

The TIM_OCInitTypeDef structure is defined in the *stm32f10x_tim.h* file:

```
typedef struct
{
u16 TIM_OCMode;
  u16 TIM_OutputState;
  u16 TIM_OutputNState;
  u16 TIM_Pulse;
  u16 TIM_OCPolarity;
  u16 TIM_OCNPolarity;
  u16 TIM_OCIdleState;
  u16 TIM_OCNIdleState;
} TIM_OCInitTypeDef;
```

TIM_OutputNState, TIM_OCNPolarity, TIM_OCIdleState and TIM_OCNIdleState parameters are used only with TIM1 and TIM8, which can generate complementary signals.

TIM_OCMode

TIM_OCMode selects the TIM mode. This member can be set to one of the following values:

Table 476. TIM_OCMode definition

TIM_OCMode	Description
TIM_OCMode_Timing	TIM Output Compare Timing mode.
TIM_OCMode_Active	TIM Output Compare Active mode.
TIM_OCMode_Inactive	TIM Output Compare Inactive mode.

Table 476. TIM_OCMode definition (continued)

TIM_OCMode	Description
TIM_OCMode_Toggle	TIM Output Compare Toggle mode.
TIM_OCMode_PWM1	TIM Pulse Width Modulation mode1.
TIM_OCMode_PWM2	TIM Pulse Width Modulation mode2.

TIM_OutputState

TIM_OutputState selects the TIM Output Compare state. This member can be set to one of the following values:

Table 477. TIM_OutputState definition

TIM_OutputState	Description
TIM_OutputState_Disable	Disables the TIM Output Compare state.
TIM_OutputState_Enable	Enables the TIM Output Compare state.

TIM_OutputNState

TIM_OutputNState selects the TIM complementary Output Compare state. This member can be set to one of the following values:

Table 478. TIM_OutputNState definition

TIM_OutputNState	Description
TIM_OutputNState_Disable	Disables the TIM Output N Compare state.
TIM_OutputNState_Enable	Enables the TIM Output N Compare state.

TIM_Pulse

TIM_Pulse configures the pulse value to be loaded into the Capture Compare Register. This member must be a number between 0x0000 and 0xFFFF.

TIM_OCPolarity

TIM_OCPolarity configures the output polarity. This member can be set to one of the following values:

Table 479. TIM_OCPolarity definition

TIM_OCPolarity	Description
TIM_OCPolarity_High	Sets the TIM Output Compare polarity to high.
TIM_OCPolarity_Low	Sets the TIM Output Compare polarity to low.

TIM_OCNPolarity

TIM_OCNPolarity configures the complementary output polarity. This member can be set to one of the following values:

Table 480. TIM_OCNPolarity definition

TIM_OCNPolarity	Description
TIM_OCNPolarity_High	Sets the Output Compare N Polarity to high.
TIM_OCNPolarity_Low	Sets the Output Compare N Polarity to low.

TIM_OCIdleState

TIM_OCIdleState selects the TIM Output Compare pin state during Idle state. This member can be set to one of the following values:

Table 481. TIM_OCIdleState definition

TIM_OCIdleState	Description
TIM_OCIdleState_Set	TIM Output OC Idle state set when MOE = 0
TIM_OCIdleState_Reset	TIM Output OC Idle state reset when MOE = 0

TIM_OCNIdleState

TIM_OCNIdleState selects the TIM Output Compare pin state during Idle state. This member can be one of the following values:

Table 482. TIM_OCNIdleState definition

TIM_OCNIdleState	Description
TIM_OCNIdleState_Set	TIM Output OCN Idle state set when MOE = 0
TIM_OCNIdleState_Reset	TIM Output OCN Idle state reset when MOE = 0

Example:

```
/* Configures the TIM1 Channel1 in PWM Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;
TIM_OC1Init(TIM1, &TIM_OCInitStructure);


/* Configures the TIM3 Channel1 in Toggle Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Toggle;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;

TIM_OC1Init(TIM3, &TIM_OCInitStructure);
```

19.2.4 TIM_OC2Init function

Table 483 describes the TIM_OC2Init function.

Table 483. TIM_OC2Init function

Function name	TIM_OC2Init
Function prototype	`void TIM_OC2Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)`
Behavior description	Initializes the TIM Channel 2 according to the parameters specified in the TIM_OCInitStruct.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCInitStruct: pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to *TIM_OCInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Configures the TIM1 Channel1 in PWM Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;
TIM_OC2Init(TIM1, &TIM_OCInitStructure);

/* Configures the TIM3 Channel1 in Toggle Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Toggle;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OC2Init(TIM3, &TIM_OCInitStructure);
```

19.2.5 TIM_OC3Init function

Table 484 describes the TIM_OC3Init function.

Table 484. TIM_OC3Init function

Function name	TIM_OC3Init
Function prototype	`void TIM_OC3Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)`
Behavior description	Initializes the TIM Channel 3 according to the parameters specified in the TIM_OCInitStruct.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCInitStruct: pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to *TIM_OCInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Configures the TIM1 Channel1 in PWM Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;

TIM_OC3Init(TIM1, &TIM_OCInitStructure);

/* Configures the TIM3 Channel1 in Toggle Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Toggle;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;

TIM_OC3Init(TIM3, &TIM_OCInitStructure);
```

19.2.6 TIM_OC4Init function

Table 485 describes the TIM_OC4Init function.

Table 485. TIM_OC14nit function

Function name	TIM_OC4Init
Function prototype	`void TIM_OC4Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)`
Behavior description	Initializes the TIM Channel 4 according to the specified parameters in the TIM_OCInitStruct.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCInitStruct: pointer to a TIM_OCInitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to *TIM_OCInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Configures the TIM1 Channel4 in PWM Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;

TIM_OC4Init(TIM1, &TIM_OCInitStructure);


/* Configures the TIM3 Channel4 in PWM Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0x7FF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;

TIM_OC4Init(TIM3, &TIM_OCInitStructure);
```

19.2.7 TIM_ICInit function

Table 486 describes the TIM_ICInit function.

Table 486. TIM_ICInit function

Function name	TIM_ICInit
Function prototype	`void TIM_ICInit(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)`
Behavior description	Initializes the TIM according to the parameters specified in the TIM_ICInitStruct.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ICInitStruct: pointer to a TIM_ICInitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to *TIM_ICInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_ICInitTypeDef structure

The TIM_ICInitTypeDef structure is defined in the *stm32f10x_tim.h* file:

```
typedef struct
{
u16 TIM_Channel;
  u16 TIM_ICPolarity;
  u16 TIM_ICSelection;
  u16 TIM_ICPrescaler;
  u8 TIM_ICFilter;
} TIM_ICInitTypeDef;
```

TIM_Channel

TIM_Channel selects the TIM channel. This member can be set to one of the following values:

Table 487. TIM_Channel definition

TIM_Channel	Description
TIM_Channel_1	TIM Channel 1 is used.
TIM_Channel_2	TIM Channel 2 is used.
TIM_Channel_3	TIM Channel 3 is used.
TIM_Channel_4	TIM Channel 4 is used.

TIM_ICPolarity

TIM_ICPolarity selects the active edge of the input signal. This member can be set to one of the following values:

Table 488. TIM_ICPolarity definition

TIM_ICPolarity	Description
TIM_ICPolarity_Rising	The active edge is the TIM Input Capture rising edge.
TIM_ICPolarity_Falling	The active edge is the TIM Input Capture falling edge.

TIM_ICSelection

TIM_ICSelection selects the input. This member can be set to one of the following values:

Table 489. TIM_ICSelection definition

TIM_ICSelection	Description
TIM_ICSelection_DirectTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively.
TIM_ICSelection_IndirectTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively.
TIM_ICSelection_TRC	TIM Input 1, 2, 3 or 4 is selected to be connected to TRC.

TIM_ICPrescaler

TIM_ICPrescaler configures the Input Capture Prescaler. This member can be set to one of the following value:

Table 490. TIM_ICPrescaler definition

TIM_ICPrescaler	Description
TIM_ICPSC_DIV1	Capture performed each time an edge is detected on the capture input.
TIM_ICPSC_DIV2	Capture performed once every 2 events.
TIM_ICPSC_DIV4	Capture performed once every 4 events.
TIM_ICPSC_DIV8	Capture performed once every 8 events.

TIM_ICFilter

TIM_ICFilter specifies the input capture filter. This member can be set to a value between 0x0 and 0xF.

Example:

```
/* TIM3 Input Capture Channel 1 mode Configuration */

TIM_ICInitTypeDef TIM_ICInitStructure;

TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
```

```
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV2;
TIM_ICInitStructure.TIM_ICFilter = 0x0;

TIM_ICInit(TIM3, &TIM_ICInitStructure);
```

19.2.8 TIM_PWMIConfig function

Table 491 describes the TIM_PWMIConfig function.

Table 491. TIM_PWMIConfig function

Function name	TIM_PWMIConfig
Function prototype	`TIM_PWMIConfig(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)`
Behavior description	Configures the TIM peripheral in PWM Input mode according to the parameters specified in the TIM_ICInitStruct.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ICInitStruct: pointer to a TIM_ICInitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to *TIM_OCInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* TIM1 PWM Input Channel 1 mode Configuration */

TIM_ICInitTypeDef TIM_ICInitStructure;

TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0x0;

TIM_PWMIConfig(TIM1, &TIM_ICInitStructure);
```

19.2.9 TIM_BDTRConfig function

Table 492 describes the TIM_BDTRConfig function.

Table 492. TIM_BDTRConfig function

Function name	TIM_BDTRConfig
Function prototype	`void TIM_BDTRConfig(TIM_TypeDef* TIMx, TIM_BDTRInitTypeDef *TIM_BDTRInitStruct)`
Behavior description	Configure the break feature, dead time, Lock level, OSSI, OSSR State and AOE (automatic output enable).
Input parameter1	TIMx: where x can be 1 or 8 to select the TIM peripheral.
Input parameter2	TIM_BDTRInitStruct: pointer to a TIM_BDTRInitTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral. Refer to *TIM_BDTRInitStruct structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_BDTRInitStruct structure

The TIM_BDTRInitStruct structure is defined in the *stm32f10x_tim.h* file:

```
typedef struct
{
  u16 TIM_OSSRState;
  u16 TIM_OSSIState;
  u16 TIM_LOCKLevel;
  u16 TIM_DeadTime;
  u16 TIM_Break;
  u16 TIM_BreakPolarity;
  u16 TIM_AutomaticOutput;
} TIM_BDTRInitTypeDef;
```

TIM_OSSRState

TIM_OSSRState configures the Off-State selection used in Run mode. This member can be set to one of the following values:

Table 493. TIM_OSSRState definition

TIM_OSSRState	Description
TIM_OSSRState_Enable	TIM OSSR State is enabled
TIM_OSSRState_Disable	TIM OSSR State is disabled

TIM_OSSIState

TIM_OSSIState selects the Off-State used in Idle state. This member can be set to one of the following values:

Table 494. TIM_OSSIState definition

TIM_OSSIState	Description
TIM_OSSIState_Enable	TIM OSSI State is enabled
TIM_OSSIState_Disable	TIM OSSI State is disabled

TIM_LOCKLevel

TIM_LOCKLevel configures the LOCK level parameters. This member can be set to one of the following values:

Table 495. TIM_LOCKLevel definition

TIM_LOCKLevel	Description
TIM_LOCKLevel_OFF	No bit is locked.
TIM_LOCKLevel_1	LOCK level 1 is used.
TIM_LOCKLevel_2	LOCK level 2 is used.
TIM_LOCKLevel_3	LOCK level 3 is used.

TIM_DeadTime

TIM_DeadTime specifies the delay time between the switching-off and the switching-on of the outputs.

TIM_Break

TIM_Break enables or disables the TIM Break input. This member can be set to one of the following values:

Table 496. TIM_Break definition

TIM_Break	Description
TIM_Break_Enable	TIM Break Input is enabled
TIM_Break_Disable	TIM Break Input is disabled

TIM_BreakPolarity

TIM_BreakPolaritiy configures the TIM Break Input pin polarity. This member can be set to one of the following values:

Table 497. TIM_BreakPolarity definition

TIM_BreakPolarity	Description
TIM_BreakPolarity_Low	Sets the TIM Break input pin polarity to low.
TIM_BreakPolarity_High	Sets the TIM Break Input pin polarity to high.

TIM_AutomaticOutput

TIM_AutomaticOutput enables or disables the Automatic Output feature. This member can be set to one of the following values:

Table 498. TIM_AutomaticOutput definition

TIM_AutomaticOutput	Description
TIM_AutomaticOutput_Enable	Enables the TIM Automatic Output.
TIM_AutomaticOutput_Disable	Disables the TIM Automatic Output.

Example:

```
/* OSSR, OSSI, Automatic Output enable, Break, dead time and Lock
Level configuration*/
TIM_BDTRInitTypeDef TIM_BDTRInitStructure;

TIM_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;
TIM_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;
TIM_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_1;
TIM_BDTRInitStructure.TIM_DeadTime = 0x05;
TIM_BDTRInitStructure.TIM_Break = TIM_Break_Enable;
TIM_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_High;
TIM_BDTRInitStructure.TIM_AutomaticOutput =
TIM_AutomaticOutput_Enable;

TIM_BDTRConfig(TIM1, &TIM_BDTRInitStructure);
```

19.2.10 TIM_TimeBaseStructInit function

Table 499 describes the TIM_TimeBaseStructInit function.

Table 499. TIM_TimeBaseStructInit function

Function name	TIM_TimeBaseStructInit
Function prototype	`void TIM_TimeBaseStructInit(TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)`
Behavior description	Fills each TIM_TimeBaseInitStruct member with its default value.
Input parameter	TIM_TimeBaseInitStruct: pointer to a TIM_TimeBaseInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The TIM_TimeBaseInitStruct members have the following default values:

Table 500. TIM_TimeBaseInitStruct default values

Member	Default value
TIM_Period	0xFFFF
TIM_Prescaler	0x0000
TIM_CKD	TIM_CKD_DIV1
TIM_CounterMode	TIM_CounterMode_Up
TIM_RepetitionCounter	0x0000

Example:

```
/* The following example illustrates how to initialize a
TIM_TimeBaseInitTypeDef structure */
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_TimeBaseStructInit(& TIM_TimeBaseInitStructure);
```

19.2.11 TIM_OCStructInit function

Table 501 describes the TIM_OCStructInit function.

Table 501. TIM_OCStructInit function

Function name	TIM_OCStructInit
Function prototype	`void TIM_OCStructInit(TIM_OCInitTypeDef* TIM_OCInitStruct)`
Behavior description	Fills each TIM_OCInitStruct member with its default value.
Input parameter	TIM_OCInitStruct: pointer to a TIM_OCInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The TIM_OCInitStruct members have the following default values:

Table 502. TIM_OCInitStruct default values

Member	Default value
TIM_OCMode	TIM_OCMode_Timing
TIM_OutputState	TIM_OutputState_Disable
TIM_OutputNState	TIM_OutputNState_Disable
TIM_Pulse	0x0000
TIM_OCPolarity	TIM_OCPolarity_High
TIM_OCNPolarity	TIM_OCPolarity_High
TIM_OCIdleState	TIM_OCIdleState_Reset
TIM_OCNIdleState	TIM_OCNIdleState_Reset

Example:

```
/* The following example illustrates how to initialize a
TIM_OCInitTypeDef structure */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCStructInit(& TIM_OCInitStructure);
```

19.2.12 TIM_ICStructInit function

Table 503 describes the TIM_ICStructInit function.

Table 503. TIM_ICStructInit function

Function name	TIM_ICStructInit
Function prototype	`void TIM_ICStructInit(TIM_ICInitTypeDef* TIM_ICInitStruct)`
Behavior description	Fills each TIM_ICInitStruct member with its default value.
Input parameter	TIM_ICInitStruct: pointer to a TIM_ICInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The TIM_ICInitStruct members have the following default values:

Table 504. TIM_ICInitStruct default values

Member	Default value
TIM_Channel	TIM_Channel_1
TIM_ICSelection	TIM_ICSelection_DirectTI
TIM_ICPrescaler	TIM_ICPSC_DIV1
TIM_ICPolarity	TIM_ICPolarity_Rising
TIM_ICFilter	0x00

Example:

```
/* The following example illustrates how to initialize a
TIM_ICInitTypeDef structure */
TIM_ICInitTypeDef TIM_ICInitStructure;
TIM_ICStructInit(& TIM_ICInitStructure);
```

19.2.13 TIM_BDTRStructInit function

Table 505 describes the TIM_BDTRStructInit function.

Table 505. TIM_BDTRStructInit function

Function name	TIM_BDTRStructInit
Function prototype	`void TIM_BDTRStructInit(TIM_BDTRInitTypeDef* TIM_BDTRInitStruct)`
Behavior description	Fills each TIM_BDTRInitStruct member with its default value.
Input parameter	TIM_BDTRInitStruct: pointer to a TIM_BDTRInitStruct structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The TIM_BDTRInitStruct members have the following default values:

Table 506. TIM_BDTRInitStruct default values

Member	Default value
TIM_OSSRState	TIM_OSSRState_Disable
TIM_OSSIState	TIM_OSSIState_Disable
TIM_LOCKLevel	TIM_LOCKLevel_OFF
TIM_DeadTime	0x00
TIM_Break	TIM_Break_Disable
TIM_BreakPolarity	TIM_BreakPolarity_Low
TIM_AutomaticOutput	TIM_AutomaticOutput_Disable

Example:

```
/* The following example illustrates how to initialize a
TIM_BDTRInitTypeDef structure */
TIM_BDTRInitTypeDef TIM_BDTRInitStructure;
TIM_BDTRStructInit(& TIM_BDTRInitStructure);
```

19.2.14 TIM_Cmd function

Table 507 describes the TIM_Cmd function.

Table 507. TIM_Cmd function

Function name	TIM_Cmd
Function prototype	`void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)`
Behavior description	Enables or disables the specified TIM peripheral.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	NewState: new state of the TIM peripheral. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM counter */
TIM_Cmd(ENABLE);
```

19.2.15 TIM_CtrlPWMOutputs function

Table 508 describes the TIM_CtrlPWMOutputs function.

Table 508. TIM_CtrlPWMOutputs function

Function name	TIM_CtrlPWMOutputs
Function prototype	`void TIM_CtrlPWMOutputs(TIM_TypeDef* TIMx, FunctionalState Newstate)`
Behavior description	Enables or disables the TIM peripheral's main outputs.
Input parameter1	TIMx: where x can be 1 or 8 to select the TIM peripheral.
Input parameter2	NewState: new state of the TIM peripheral's main outputs. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM8 peripheral Main Outputs. */
TIM_CtrlPWMOutputs(TIM8, ENABLE);
```

19.2.16 TIM_ITConfig function

Table 509 describes the TIM_ITConfig function.

Table 509. TIM_ITConfig function

Function name	TIM_ITConfig
Function prototype	`void TIM_ITConfig(TIM_TypeDef* TIMx, u16 TIM_IT, FunctionalState NewState)`
Behavior description	Enables or disables the specified TIM interrupts.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_IT: TIMx interrupt sources to be enabled or disabled. Refer to *TIM_IT* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the specified TIM interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_IT

TIM_IT enables or disables TIM interrupts. One or a combination of the following values can be used:

Note: TIM6 and TIM7 can only generate an update interrupt. TIM_IT_COM and TIM_IT_Break are used only with TIM1 and TIM8.

Table 510. TIM_IT values

TIM_IT	Description
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 COM Interrupt source
TIM_IT_Trigger	TIM Trigger Interrupt source
TIM_IT_Break	TIM Break Interrupt source

Example:

```
/* Enables the TIM5 Capture Compare channel 1 Interrupt source */
TIM_ITConfig(TIM5, TIM_IT_CC1, ENABLE );
```

19.2.17 TIM_GenerateEvent function

Table 511 describes the TIM_GenerateEvent function.

Table 511. TIM_GenerateEvent function

Function name	TIM_GenerateEvent
Function prototype	`void TIM_GenerateEvent(TIM_TypeDef* TIMx, u16 TIM_EventSource)`
Behavior description	Configures the TIM event to be generated by software.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_EventSource: specifies the TIM software event sources. Refer to *TIM_EventSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_EventSource

The TIM event software source can be selected by using one or a combination of the values provided in *Table 512*.

Note: TIM6 and TIM7 can only generate an update event. TIM_EventSource_COM and TIM_EventSource_Break are used only with TIM1 and TIM8.

Table 512. TIM_EventSource values

TIM_EventSource	Description
TIM_EventSource_Update	TIM update event source
TIM_EventSource_CC1	TIM Capture/Compare 1 event source
TIM_EventSource_CC2	TIM Capture/Compare 2 event source
TIM_EventSource_CC3	TIM Capture/Compare 3 event source
TIM_EventSource_CC4	TIM Capture/Compare 4 event source
TIM_EventSource_COM	TIM COM event source
TIM_EventSource_Trigger	TIM Trigger event source
TIM_EventSource_Break	TIM Break event source

Example:

```
/* Selects the Capture compare4 software event generation for TIM4
*/
TIM_GenerateEvent(TIM4, TIM_EventSource_CC4);
```

19.2.18 TIM_DMAConfig function

Table 513 describes the TIM_DMAConfig function.

Table 513. TIM_DMAConfig function

Function name	TIM_DMAConfig
Function prototype	`void TIM_DMAConfig(TIM_TypeDef* TIMx,u8 TIM_DMABase, u16 TIM_DMABurstLength)`
Behavior description	Configures the TIM's DMA interface.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_DMABase: DMA Base address. Refer to *TIM_DMABase* for more details on the allowed values for this parameter.
Input parameter3	TIM_DMABurstLength: DMA Burst length. Refer to *TIM_DMABurstLength* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_DMABase

TIM_DMABase selects the TIM DMA's base address (see *Table 514*).

Table 514. TIM_DMABase values

TIM_DMABase	Description
TIM_DMABase_CR1	CR1 register used as DMA Base
TIM_DMABase_CR2	CR2 register used as DMA Base
TIM_DMABase_SMCR	SMCR register used as DMA Base
TIM_DMABase_DIER	DIER register used as DMA Base
TIM_DMABase_SR	SR register used as DMA Base
TIM_DMABase_EGR	EGR register used as DMA Base
TIM_DMABase_CCMR1	CCMR1 register used as DMA Base
TIM_DMABase_CCMR2	CCMR2 register used as DMA Base
TIM_DMABase_CCER	CCER register used as DMA Base
TIM_DMABase_CNT	CNT register used as DMA Base
TIM_DMABase_PSC	PSC register used as DMA Base
TIM_DMABase_ARR	ARR register used as DMA Base
TIM_DMABase_RCR	RCR register used as DMA Base
TIM_DMABase_CCR1	CCR1 register used as DMA Base
TIM_DMABase_CCR2	CCR2 register used as DMA Base

Table 514. TIM_DMABase values (continued)

TIM_DMABase	Description
TIM_DMABase_CCR3	CCR3 register used as DMA Base
TIM_DMABase_CCR4	CCR4 register used as DMA Base
TIM_DMABase_BDTR	BDTR register used as DMA Base
TIM_DMABase_DCR	DCR register used as DMA Base

TIM_DMABurstLength

TIM_DMABurstLength configures the TIM DMA burst length as shown in *Table 515*.

Table 515. TIM_DMABurstLength values

TIM_DMABurstLength	Description
TIM_DMABurstLength_1Byte	DMA Burst length 1 byte
TIM_DMABurstLength_2Bytes	DMA Burst length 2 bytes
TIM_DMABurstLength_3Bytes	DMA Burst length 3 bytes
TIM_DMABurstLength_4Bytes	DMA Burst length 4 bytes
TIM_DMABurstLength_5Bytes	DMA Burst length 5 bytes
TIM_DMABurstLength_6Bytes	DMA Burst length 6 bytes
TIM_DMABurstLength_7Bytes	DMA Burst length 7 bytes
TIM_DMABurstLength_8Bytes	DMA Burst length 8 bytes
TIM_DMABurstLength_9Bytes	DMA Burst length 9 bytes
TIM_DMABurstLength_10Bytes	DMA Burst length 10 bytes
TIM_DMABurstLength_11Bytes	DMA Burst length 11 bytes
TIM_DMABurstLength_12Bytes	DMA Burst length 12 bytes
TIM_DMABurstLength_13Bytes	DMA Burst length 13 bytes
TIM_DMABurstLength_14Bytes	DMA Burst length 14 bytes
TIM_DMABurstLength_15Bytes	DMA Burst length 15 bytes
TIM_DMABurstLength_16Bytes	DMA Burst length 16 bytes
TIM_DMABurstLength_17Bytes	DMA Burst length 17 bytes
TIM_DMABurstLength_18Bytes	DMA Burst length 18 bytes

Example:

```
/* Configures the TIM1 DMA Interface to transfer 1 byte and to use
the CCR1 as base address */
TIM_DMAConfig(TIM1, TIM_DMABase_CCR1, TIM_DMABurstLength_1Byte)
```

19.2.19 TIM_DMACmd function

Table 516 describes the TIM_DMACmd function.

Table 516. TIM_DMACmd function

Function name	TIM_DMACmd
Function prototype	`void TIM_DMACmd(TIM_TypeDef* TIMx, u16 TIM_DMASource, FunctionalState Newstate)`
Behavior description	Enables or disables the TIM DMA Requests.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_DMASource: DMA Request sources. Refer to *TIM_DMASource* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_DMASource

TIM_DMASource selects the TIM DMA request source. One or a combination of the following values can be used:

Table 517. TIM_DMASource values

TIM_DMASource	Description
TIM_DMA_Update	TIM Update DMA 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 COM DMA source
TIM_DMA_Trigger	TIM Trigger DMA source

Table 518 shows the DMA requests for each timer.

Table 518. TIM DMA requests

Requests	TIM1	TIM2	TIM3	TIM4	TIM5	TIM6	TIM7	TIM8
TIM_DMA_Update	x	x	x	x	x	x	x	x
TIM_DMA_CC1	x	x	x	x	x			x
TIM_DMA_CC2	x	x		x	x			x
TIM_DMA_CC2	x	x	x	x	x			x
TIM_DMA_CC3	x	x	x		x			x
TIM_DMA_CC4	x							x
TIM_DMA_Trigger	x		x	x	x			x

Example:

```
/* TIM5 Capture Compare 1 DMA Request Configuration */
TIM_DMACmd(TIM5, TIM_DMA_CC1, ENABLE);
```

19.2.20 TIM_InternalClockConfig function

Table 519 describes the TIM_InternalClockConfig function.

Table 519. TIM_InternalClockConfig function

Function name	TIM_InternalClockConfig
Function prototype	`void TIM_InternalClockConfig(TIM_TypeDef* TIMx)`
Behavior description	Configures the TIM internal clock
Input parameter	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the internal clock for TIM2 */
TIM_InternalClockConfig(TIM2);
```

19.2.21 TIM_ITRxExternalClockConfig function

Table 520 describes the TIM_ITRxExternalClockConfig function.

Table 520. TIM_ITRxExternalClockConfig function

Function name	TIM_ITRxExternalClockConfig
Function prototype	`void TIM_ITRxExternalClockConfig(TIM_TypeDef* TIMx, u16 TIM_InputTriggerSource)`
Behavior description	Configures the TIM's internal trigger as the external clock.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_InputTriggerSource: input trigger source. Refer to *TIM_InputTriggerSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_InputTriggerSource

TIM_InputTriggerSource selects the TIM Input trigger (see *Table 521*).

Table 521. TIM_InputTriggerSource values

TIM_InputTriggerSource	Description
TIM_TS_ITR0	TIM Internal Trigger 0
TIM_TS_ITR1	TIM Internal Trigger 1
TIM_TS_ITR2	TIM Internal Trigger 2
TIM_TS_ITR3	TIM Internal Trigger 3

Example:

```
/* TIM1 internal trigger 3 used as clock source */
TIM_ITRxExternalClockConfig(TIM1, TIM_TS_ITR3);
```

19.2.22 TIM_TIxExternalClockConfig function

Table 522 describes the TIM_TIxExternalClockConfig function.

Table 522. TIM_TIxExternalClockConfig function

Function name	TIM_TIxExternalClockConfig
Function prototype	`void TIM_TIxExternalClockConfig(TIM_TypeDef* TIMx, u16 TIM_TIxExternalCLKSource, u16 TIM_ICPolarity, u16 ICFilter)`
Behavior description	Configures the TIM trigger as the external clock.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_TIxExternalCLKSource: Trigger source. Refer to *TIM_TIxExternalCLKSource* for more details on the allowed values for this parameter.
Input parameter3	TIM_ICPolarity: TI polarity. Refer to *TIM_ICPolarity* for more details on the allowed values for this parameter.
Input parameter4	ICFilter: Specifies the input capture filter. This member can be a value between 0x0 and 0xF.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_TIxExternalCLKSource

TIM_TIxExternalCLKSource selects the TIx external clock source of the TIM. One of the following values can be used:

Table 523. TIM_TIxExternalCLKSource values

TIM_TIxExternalCLKSource	Description
TIM_TS_TI1FP1	IC1 is mapped on TI1.
TIM_TS_TI2FP2	IC2 is mapped on TI2.
TIM_TS_TI1F_ED	IC1 is mapped on TI1: edge detector is used

Example:

```
/* Selects the TI1 as clock for TIM1: the external clock is
connected to TI1 input pin, the rising edge is the active edge and
no filter sampling is done (ICFilter = 0) */
TIM_TIxExternalClockConfig(TIM1, TIM_TS_TI1FP1,
TIM_ICPolarity_Rising, 0);
```

19.2.23 TIM_ETRClockMode1Config function

Table 524 describes the TIM_ETRClockMode1Config function.

Table 524. TIM_ETRClockMode1Config function

Function name	TIM_ETRClockMode1Config
Function prototype	`void TIM_ETRClockMode1Config(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u16 ExtTRGFilter)`
Behavior description	Configures the TIM's External clock Mode1.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ExtTRGPrescaler: external trigger prescaler. Refer to *TIM_ExtTRGPrescaler* for more details on the allowed values for this parameter.
Input parameter3	TIM_ExtTRGPolarity: external clock polarity. Refer to *TIM_ExtTRGPolarity* for more details on the allowed values for this parameter.
Input parameter4	ExtTRGFilter: Specifies the external trigger filter. This member can assume a value between 0x0 and 0xF.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_ExtTRGPrescaler

TIM_ExtTRGPrescaler selects the external trigger prescaler. This member can be set to one of the following values:

Table 525. TIM_ExtTRGPrescaler values

TIM_ExtTRGPrescaler	Description
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

TIM_ExtTRGPolarity configures the external trigger polarity. This member can be set to one of the following values:

Table 526. TIM_ExtTRGPolarity values

TIM_ExtTRGPolarity	Description
TIM_ExtTRGPolarity_Inverted	External trigger polarity inverted: active low or falling edge active.
TIM_ExtTRGPolarity_NonInverted	External trigger polarity noninverted: active high or rising edge active.

Example:

```
/* Selects the external clock Mode 1 for TIM1: the external clock is
connected to ETR input pin, the rising edge is the active edge, no
filter sampling is done (ExtTRGFilter = 0) and the prescaler is
fixed to TIM_ExtTRGPSC_DIV2 */
TIM_ExternalCLK1Config(TIM1, TIM_ExtTRGPSC_DIV2,
TIM_ExtTRGPolarity_NonInverted, 0x0);
```

19.2.24 TIM_ETRClockMode2Config function

Table 527 describes the TIM_ETRClockMode2Config function.

Table 527. TIM_ETRClockMode2Config function

Function name	TIM_ETRClockMode2Config
Function prototype	`void TIM_ETRClockMode2Config(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u16 ExtTRGFilter)`
Behavior description	Configures the TIM's external clock mode2.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ExtTRGPrescaler: specifies the external trigger prescaler. Refer to *TIM_ExtTRGPrescaler* for more details on the allowed values for this parameter.
Input parameter3	TIM_ExtTRGPolarity: specifies the external clock polarity. Refer to *TIM_ExtTRGPolarity* for more details on the allowed values for this parameter.
Input parameter4	ExtTRGFilter: specifies the external trigger Filter. This member can assume a value between 0x0 and 0xF.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the external clock Mode 2 for TIM1: the external clock is
connected to ETR input pin, the rising edge is the active edge, no
filter sampling is done (ExtTRGFilter = 0) and the prescaler is
fixed to TIM_ExtTRGPSC_DIV2 */
TIM_ExternalCLK2Config(TIM1, TIM_ExtTRGPSC_DIV2,
TIM_ExtTRGPolarity_NonInverted, 0x0);
```

19.2.25 TIM_ETRConfig

Table 527 describes the TIM_ETRConfig function.

Table 528. TIM_ETRConfig function

Function name	TIM_ETRConfig
Function prototype	`void TIM_ETRConfig(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u8 ExtTRGFilter)`
Behavior description	Configures the TIM's external trigger (ETR).
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ExtTRGPrescaler: specifies the external trigger prescaler. Refer to *TIM_ExtTRGPrescaler* for more details on the allowed values for this parameter.
Input parameter3	TIM_ExtTRGPolarity: specifies the external clock polarity. Refer to *TIM_ExtTRGPolarity* for more details on the allowed values for this parameter.
Input parameter4	ExtTRGFilter: specifies the external trigger Filter. This member can assume a value between 0x0 and 0xF.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Configure the External Trigger (ETR) for TIM1: the rising edge is
the active edge, no filter sampling is done (ExtTRGFilter = 0) and
the prescaler is fixed to TIM_ExtTRGPSC_DIV2 */
TIM_ExternalCLK2Config(TIM1, TIM_ExtTRGPSC_DIV2,
TIM_ExtTRGPolarity_NonInverted, 0x0);
```

19.2.26 TIM_PrescalerConfig function

Table 529 describes the TIM_PrescalerConfig function.

Table 529. TIM_PrescalerConfig function

Function name	TIM_PrescalerConfig
Function prototype	`void TIM_PrescalerConfig(TIM_TypeDef* TIMx, u16 Prescaler, u16 TIM_PSCReloadMode)`
Behavior description	Configures the TIM prescaler.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Prescaler: new TIM prescaler value.
Input parameter3	TIM_PSCReloadMode: TIM prescaler reload mode. Refer to *TIM_PSCReloadMode* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_PSCReloadMode

To select the TIM Prescaler Reload mode use one of the following values:

Table 530. TIM_PSCReloadMode values

TIM_PSCReloadMode	Description
TIM_PSCReloadMode_Update	The Prescaler is loaded at the update event.
TIM_PSCReloadMode_Immediate	The Prescaler is loaded immediately.

Example:

```
/* Sets the TIM1 new Prescaler value */
u16 TIMPrescaler = 0xFF00;
TIM_SetPrescaler(TIM1, TIMPrescaler, TIM_PSCReloadMode_Update);
```

19.2.27 TIM_CounterModeConfig function

Table 531 describes the TIM_CounterModeConfig function.

Table 531. TIM_CounterModeConfig function

Function name	TIM_CounterModeConfig
Function prototype	`void TIM_CounterModeConfig(TIM_TypeDef* TIMx, u16 TIM_CounterMode)`
Behavior description	Specifies the TIM counter mode to be used.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_CounterMode: counter mode to be used. Refer to *TIM_CounterMode* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Center Aligned counter Mode 1 for the TIM1 */
TIM_CounterModeConfig(TIM1, TIM_Counter_CenterAligned1);
```

19.2.28 TIM_SelectInputTrigger function

Table 532 describes the TIM_SelectInputTrigger function.

Table 532. TIM_SelectInputTrigger function

Function name	TIM_SelectInputTrigger
Function prototype	`void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, u16 TIM_InputTriggerSource)`
Behavior description	Selects the TIM's input trigger source.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_InputTriggerSource: input trigger source. Refer to *TIM_InputTriggerSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_InputTriggerSource

TIM_InputTriggerSource selects the TIM's input trigger source. This member can be set to one of the following values:

Table 533. TIM_InputTriggerSource values

TIM_InputTriggerSource	Description
TIM_TS_ITR0	TIM Internal Trigger 0.
TIM_TS_ITR1	TIM Internal Trigger 1.
TIM_TS_ITR2	TIM Internal Trigger 2.
TIM_TS_ITR3	TIM Internal Trigger 3.
TIM_TS_TI1F_ED	TIM TI1 Edge Detector.
TIM_TS_TI1FP1	TIM Filtered Timer Input 1.
TIM_TS_TI2FP2	TIM Filtered Timer Input 2.
TIM_TS_ETRF	TIM External Trigger input.

Example:

```
/* Selects the Internal Trigger 3 as input trigger fot TIM1 */
void TIM_SelectInputTrigger(TIM1, TIM_TS_ITR3);
```

19.2.29 TIM_EncoderInterfaceConfig function

Table 534 describes the TIM_EncoderInterfaceConfig function.

Table 534. TIM_EncoderInterfaceConfig function

Function name	TIM_EncoderInterfaceConfig
Function prototype	`void TIM_EncoderInterfaceConfig(TIM_TypeDef* TIMx, u16 TIM_EncoderMode, u16 TIM_IC1Polarity, u16 TIM_IC2Polarity)`
Behavior description	Configures the TIM encoder interface.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_EncoderMode: TIM encoder mode. Refer to *TIM_EncoderMode* for more details on the allowed values for this parameter.
Input parameter3	TIM_IC1Polarity: TI1 Polarity. Refer to *TIM_ICPolarity* for more details on the allowed values for this parameter.
Input parameter4	TIM_IC2Polarity: TI2 Polarity. Refer to *TIM_ICPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_EncoderMode

TIM_EncoderMode selects the TIMx encoder mode (see *Table 535*).

Table 535. TIM_EncoderMode definition

TIM_EncoderMode	Description
TIM_EncoderMode_TI1	TIM encoder mode 1 is used.
TIM_EncoderMode_TI2	TIM encoder mode 2 is used.
TIM_EncoderMode_TI12	TIM encoder mode 3 is used.

Example:

```
/* uses of the TIM1 Encoder interface */
TIM_EncoderInterfaceConfig(TIM1, TIM_EncoderMode_1,
TIM_ICPolarity_Rising,
TIM_ICPolarity_Rising);
```

19.2.30 TIM_ForcedOC1Config function

Table 536 describes the TIM_ForcedOC1Config function.

Table 536. TIM_ForcedOC1Config function

Function name	TIM_ForcedOC1Config
Function prototype	`void TIM_ForcedOC1Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)`
Behavior description	Forces the TIM Channel 1 output waveform to active or inactive level.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ForcedAction: specified action to be forced on the output waveform. Refer to *TIM_ForcedAction* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_ForcedAction

The forced actions that can be used are listed in *Table 537*.

Table 537. TIM_ForcedAction values

TIM_ForcedAction	Description
TIM_ForcedAction_Active	Forces active level on OCxREF.
TIM_ForcedAction_InActive	Forces inactive level on OCxREF.

Example:

```
/* Forces the TIM1 Channel1 Output to the active level */
TIM_ForcedOC1Config(TIM1, TIM_ForcedAction_Active);
```

19.2.31 TIM_ForcedOC2Config function

Table 538 describes the TIM_ForcedOC2Config function.

Table 538. TIM_ForcedOC2Config function

Function name	TIM_ForcedOC2Config
Function prototype	`void TIM_ForcedOC2Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)`
Behavior description	Forces the TIM Channel2 output waveform to active or inactive level.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ForcedAction: specifies the action to be forced on the output waveform. Refer to *TIM_ForcedAction* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Forces the TIM1 Channel2 Output to the active level */
TIM_ForcedOC2Config(TIM1, TIM_ForcedAction_Active);
```

19.2.32 TIM_ForcedOC3Config function

Table 539 describes the TIM_ForcedOC3Config function.

Table 539. TIM_ForcedOC3Config function

Function name	TIM_ForcedOC3Config
Function prototype	`void TIM_ForcedOC3Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)`
Behavior description	Forces the TIM Channel3 output waveform to active or inactive level.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ForcedAction: specifies the action to be forced on the output waveform. Refer to section *TIM_ForcedAction on page 374* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Forces the TIM1 Channel3 Output to the active level */
TIM_ForcedOC3Config(TIM1, TIM_ForcedAction_Active);
```

19.2.33 TIM_ForcedOC4Config function

Table 540 describes the TIM_ForcedOC4Config function.

Table 540. TIM_ForcedOC4Config function

Function name	TIM_ForcedOC4Config
Function prototype	`void TIM_ForcedOC4Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)`
Behavior description	Forces the TIM Channel4 output waveform to active or inactive level.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_ForcedAction: specifies the action to be forced on the output waveform. Refer to *TIM_ForcedAction* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Forces the TIM1 Channel4 Output to the active level */
TIM_ForcedOC4Config(TIM1, TIM_ForcedAction_Active);
```

19.2.34 TIM_ARRPreloadConfig function

Table 541 describes the TIM_ARRPreloadConfig function.

Table 541. TIM_ARRPreloadConfig function

Function name	TIM_ARRPreloadConfig
Function prototype	`void TIM_ARRPreloadConfig(TIM_TypeDef* TIMx, FunctionalState Newstate)`
Behavior description	Enables or disables the TIM peripheral Preload register on ARR.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter3	NewState: new state of the ARPE bit in the TIM_CR1 register. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM1 Preload on ARR Register */
TIM_ARRPreloadConfig(TIM1, ENABLE);
```

19.2.35 TIM_SelectCOM function

Table 542 describes the TIM_SelectCOM function.

Table 542. TIM_SelectCOM function

Function name	TIM_SelectCOM
Function prototype	`void TIM_SelectCOM(TIM_TypeDef* TIMx, FunctionalState Newstate)`
Behavior description	Selects the TIM peripheral commutation event.
Input parameter1	TIMx: where x can be 1 or 8 to select the TIM peripheral.
Input parameter2	Newstate: new state of the commutation event. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the TIM1 Commutation event */
TIM_SelectCOM(TIM1, ENABLE);
```

19.2.36 TIM_SelectCCDMA function

Table 543 describes the TIM_SelectCCDMA function.

Table 543. TIM_SelectCCDMA function

Function name	TIM_SelectCCDMA
Function prototype	`void TIM_SelectCCDMA(TIM_TypeDef* TIMx, FunctionalState Newstate)`
Behavior description	Selects the TIM peripheral Capture Compare DMA source.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter3	NewState: new state of the Capture Compare DMA source. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the TIM1 Capture Compare DMA source */
TIM_SelectCCDMA(TIM1, ENABLE);
```

19.2.37 TIM_CCPreloadControl function

Table 544 describes the TIM_CCPreloadControl function.

Table 544. TIM_CCPreloadControl function

Function name	TIM_CCPreloadControl
Function prototype	`void TIM_CCPreloadControl(TIM_TypeDef* TIMx, FunctionalState Newstate)`
Behavior description	Sets or resets the TIM peripheral Capture Compare Preload Control bit.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Newstate: new state of the Capture Compare Preload Control bit. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the TIM1 Capture Compare Preload Control */
TIM_CCPreloadControl(TIM1, ENABLE);
```

19.2.38 TIM_OC1PreloadConfig function

Table 545 describes the TIM_OC1PreloadConfig function.

Table 545. TIM_OC1PreloadConfig function

Function name	TIM_OC1PreloadConfig
Function prototype	`void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)`
Behavior description	Enables or disables the TIM Preload register on CCR1.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPreload: Output Compare Preload state. Refer to *TIM_OCPreload* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_OCPreload

The Output Compare Preload states are listed in *Table 546*.

Table 546. TIM_OCPreload states

TIM_OCPreload	Description
TIM_OCPreload_Enable	TIM Preload register on CCR1 enable.
TIM_OCPreload_Disable	TIM Preload register on CCR1 disable.

Example:

```
/* Enables the TIM1 Preload on CC1 Register */
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

19.2.39 TIM_OC2PreloadConfig function

Table 547 describes the TIM_OC2PreloadConfig function.

Table 547. TIM_OC2PreloadConfig function

Function name	TIM_OC2PreloadConfig
Function prototype	`void TIM_OC2PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)`
Behavior description	Enables or disables the TIM Preload register on CCR2.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPreload: Output Compare Preload state. Refer to *TIM_OCPreload* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM1 Preload on CC2 Register */
TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

19.2.40 TIM_OC3PreloadConfig function

Table 548 describes the TIM_OC3PreloadConfig function.

Table 548. TIM_OC3PreloadConfig function

Function name	TIM_OC3PreloadConfig
Function prototype	`void TIM_OC3PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)`
Behavior description	Enables or disables the TIM Preload register on CCR3.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPreload: specifies the Output Compare Preload state. Refer to *TIM_OCPreload* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM1 Preload on CC3 Register */
TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

19.2.41 TIM_OC4PreloadConfig function

Table 549 describes the TIM_OC4PreloadConfig function.

Table 549. TIM_OC4PreloadConfig function

Function name	TIM_OC4PreloadConfig
Function prototype	`void TIM_OC4PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)`
Behavior description	Enables or disables the TIM Preload register on CCR4.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPreload: Output Compare Preload state. Refer to *TIM_OCPreload* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM1 Preload on CC4 Register */
TIM_OC4PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

19.2.42 TIM_OC1FastConfig function

Table 550 describes the TIM_OC1FastConfig function.

Table 550. TIM_OC1FastConfig function

Function name	TIM_OC1FastConfig
Function prototype	`void TIM_OC1FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)`
Behavior description	Configures the TIM Output Compare 1 Fast feature.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCFast: Output Compare fast feature state. Refer to *TIM_OCFast* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_OCFast

The Output Compare Preload states are listed in *Table 551*.

Table 551. TIM_OCFast states

TIM_OCFast	Description
TIM_OCFast_Enable	TIM Output Compare Fast capability enable.
TIM_OCFast_Disable	TIM Output Compare Fast capability disable.

Example:

```
/* Use the TIM1 OC1 in fast Mode */
TIM_OC1FastConfig(TIM1, TIM_OCFast_Enable);
```

19.2.43 TIM_OC2FastConfig function

Table 552 describes the TIM_OC2FastConfig function.

Table 552. TIM_OC2FastConfig function

Function name	TIM_OC2FastConfig
Function prototype	`void TIM_OC2FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)`
Behavior description	Configures the TIM Output Compare 2 Fast feature.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCFast: Output Compare fast feature state. Refer to *TIM_OCFast* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Use the TIM1 OC2 in fast Mode */
TIM_OC2FastConfig(TIM1, TIM_OCFast_Enable);
```

19.2.44 TIM_OC3FastConfig function

Table 553 describes the TIM_OC3FastConfig function.

Table 553. TIM_OC3FastConfig function

Function name	TIM_OC3FastConfig
Function prototype	`void TIM_OC3FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)`
Behavior description	Configures the TIM Output Compare 3 Fast feature.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCFast: Output Compare fast feature state. Refer to *TIM_OCFast* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Use the TIM1 OC3 in fast Mode */
TIM_OC3FastConfig(TIM1, TIM_OCFast_Enable);
```

19.2.45 TIM_OC4FastConfig function

Table 554 describes the TIM_OC4FastConfig function.

Table 554. TIM_OC4FastConfig function

Function name	TIM_OC4FastConfig
Function prototype	`void TIM_OC4FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)`
Behavior description	Configures the TIM Output Compare 4 Fast feature.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCFast: specifies the Output Compare fast feature state. Refer to *TIM_OCFast* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Use the TIM1 OC4 in fast Mode */
TIM_OC4FastConfig(TIM1, TIM_OCFast_Enable);
```

19.2.46 TIM_ClearOC1Ref

Table 555 describes the TIM_ClearOC1Ref function.

Table 555. TIM_ClearOC1Ref function

Function name	TIM_ClearOC1Ref
Function prototype	`void TIM_ClearOC1Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)`
Behavior description	Clears or safeguards the OCREF1 signal on an external event.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCClear: new state of the Output Compare Clear Enable Bit. Refer to *TIM_OCClear* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_OCClear

The values of the Output Compare Reference Clear bit that can be used are listed in *Table 556*:

Table 556. TIM_OCClear

TIM_OCClear	Description
TIM_OCClear_Enable	TIMx Output Compare Clear enable.
TIM_OCClear_Disable	TIMx Output Compare Clear disable.

Example:

```
/* Enable the TIM1 Channel1 Ouput Compare Refence clear bit */
TIM_ClearOC1Ref(TIM1, TIM_OCClear_Enable);
```

19.2.47 TIM_ClearOC2Ref

Table 557 describes the TIM_ClearOC2Ref function.

Table 557. TIM_ClearOC2Ref function

Function name	TIM_ClearOC2Ref
Function prototype	`void TIM_ClearOC2Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)`
Behavior description	Clears or safeguards the OCREF2 signal on an external event.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCClear: new state of the Output Compare Clear Enable Bit. Refer to *TIM_OCClear* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TIM1 Channel2 Ouput Compare Refence clear bit */
TIM_ClearOC2Ref(TIM1, TIM_OCClear_Enable);
```

19.2.48 TIM_ClearOC3Ref

Table 558 describes the TIM_ClearOC3Ref function.

Table 558. TIM_ClearOC3Ref function

Function name	TIM_ClearOC3Ref
Function prototype	`void TIM_ClearOC3Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)`
Behavior description	Clears or safeguards the OCREF3 signal on an external event.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCClear: new state of the Output Compare Clear Enable Bit. Refer to *TIM_OCClear* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TIM1 Channel3 Ouput Compare Refence clear bit */
TIM_ClearOC3Ref(TIM1, TIM_OCClear_Enable);
```

19.2.49 TIM_ClearOC4Ref

Table 559 describes the TIM_ClearOC4Ref function.

Table 559. TIM_ClearOC4Ref function

Function name	TIM_ClearOC4Ref
Function prototype	`void TIM_ClearOC4Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)`
Behavior description	Clears or safeguards the OCREF4 signal on an external event.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCClear: new state of the Output Compare Clear Enable Bit. Refer to *TIM_OCClear* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TIM1 Channel4 Ouput Compare Refence clear bit */
TIM_ClearOC4Ref(TIM1, TIM_OCClear_Enable);
```

19.2.50 TIM_OC1PolarityConfig function

Table 560 describes the TIM_OC1PolarityConfig function.

Table 560. TIM_OC1PolarityConfig function

Function name	TIM_OC1PolarityConfig
Function prototype	`void TIM_OC1PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)`
Behavior description	Configures the TIM Channel 1 polarity.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPolarity: Output compare polarity. Refer to *TIM_OCPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_OCPolarity

TIM_OCPolarity selects the TIM polarity (see *Table 561*).

Table 561. TIM_OCPolarity values

TIM_OCPolarity	Description
TIM_OCPolarity_High	Sets the TIM Output Polarity to high.
TIM_OCPolarity_Low	Sets the TIM Output Polarity to low.

Example:

```
/* Selects the Polarity high for TIM1 channel 1 output compare */
TIM_OC1PolarityConfig(TIM1, TIM_OCPolarity_High);
```

19.2.51 TIM_OC1NPolarityConfig function

Table 562 describes the TIM_OC1NPolarityConfig function.

Table 562. TIM_OC1NPolarityConfig function

Function name	TIM_OC1NPolarityConfig
Function prototype	`void TIM_OC1NPolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCNPolarity)`
Behavior description	Configures the TIM Channel 1N polarity.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCNPolarity: Output compare N polarity. Refer to *TIM_OCPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Polarity high for TIM1 channel 1N output compare */
TIM_OC1NPolarityConfig(TIM1, TIM_OCNPolarity_High);
```

19.2.52 TIM_OC2PolarityConfig function

Table 563 describes the TIM_OC2PolarityConfig function.

Table 563. TIM_OC2PolarityConfig function

Function name	TIM_OC2PolarityConfig
Function prototype	`void TIM_OC2PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)`
Behavior description	Configures the TIM Channel 2 polarity.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPolarity: Output compare polarity. Refer to *TIM_OCPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Polarity high for TIM1 channel 2 output compare */
TIM_OC2PolarityConfig(TIM1, TIM_OCPolarity_High);
```

19.2.53 TIM_OC2NPolarityConfig function

Table 564 describes the TIM_OC2NPolarityConfig function.

Table 564. TIM_OC2NPolarityConfig function

Function name	TIM_OC2NPolarityConfig
Function prototype	`void TIM_OC2NPolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCNPolarity)`
Behavior description	Configures the TIM Channel 2N polarity.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCNPolarity: Output compare N polarity. Refer to *TIM_OCPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Polarity high for TIM1 channel 2N output compare */
TIM_OC2NPolarityConfig(TIM1, TIM_OCNPolarity_High);
```

19.2.54 TIM_OC3PolarityConfig function

Table 565 describes the TIM_OC3PolarityConfig function.

Table 565. TIM_OC3PolarityConfig function

Function name	TIM_OC3PolarityConfig
Function prototype	`void TIM_OC3PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)`
Behavior description	Configures the TIM Channel 3 polarity.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPolarity: Output compare polarity. Refer to *TIM_OCPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Polarity high for TIM1 channel 3 output compare */
TIM_OC3PolarityConfig(TIM1, TIM_OCPolarity_High);
```

19.2.55 TIM_OC3NPolarityConfig function

Table 566 describes the TIM_OC3NPolarityConfig function.

Table 566. TIM_OC3NPolarityConfig function

Function name	TIM_OC3NPolarityConfig
Function prototype	`void TIM_OC3NPolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCNPolarity)`
Behavior description	Configures the TIM Channel 3 N polarity.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCNPolarity: Output compare N polarity. Refer to *TIM_OCPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Polarity high for TIM1 channel 3N output compare */
TIM_OC3NPolarityConfig(TIM1, TIM_OCNPolarity_High);
```

19.2.56 TIM_OC4PolarityConfig function

Table 567 describes the TIM_OC4PolarityConfig function.

Table 567. TIM_OC4PolarityConfig function

Function name	TIM_OC4PolarityConfig
Function prototype	`void TIM_OC4PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)`
Behavior description	Configures the TIM Channel 4 polarity.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_OCPolarity: Output compare polarity. Refer to *TIM_OCPolarity* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Polarity high for TIM1 channel 4 output compare */
TIM_OC4PolarityConfig(TIM1, TIM_OCPolarity_High);
```

19.2.57 TIM_CCxCmd function

Table 568 describes the TIM_CCxCmd function.

Table 568. TIM_CCxCmd function

Function name	TIM_CCxCmd
Function prototype	`void TIM_CCxCmd(TIM_TypeDef* TIMx, u16 TIM_Channel, FunctionalState Newstate)`
Behavior description	Enables or disables the TIM Capture Compare Channel x.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_Channel: TIM Channel. Refer to *TIM_Channel* for more details on the allowed values for this parameter.
Input parameter3	Newstate: specifies the TIM Channel CCxE bit new state. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM1 channel 4 */
TIM_CCxCmd(TIM1, TIM_Channel_4, ENABLE);
```

19.2.58 TIM_CCxNCmd function

Table 569 describes the TIM_CCxNCmd function.

Table 569. TIM_CCxNCmd function

Function name	TIM_CCxCmd
Function prototype	`void TIM_CCxNCmd(TIM_TypeDef* TIMx, u16 TIM_Channel, FunctionalState Newstate)`
Behavior description	Enables or disables the TIM Capture Compare Channel xN.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_Channel: TIM Channel. Refer to *TIM_Channel* for more details on the allowed values for this parameter.
Input parameter3	Newstate: specifies the TIM Channel CCxNE bit new state. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM1 channel 3N */
TIM_CCxNCmd(TIM1, TIM_Channel_3, ENABLE);
```

19.2.59 TIM_SelectOCxM function

Table 570 describes the TIM_SelectOCxM function.

Table 570. TIM_SelectOCxM function

Function name	TIM_SelectOCxM
Function prototype	`void TIM_SelectOCxM(TIM_TypeDef* TIMx, u16 TIM_Channel, u16 TIM_OCMode)`
Behavior description	Selects the TIM Output Compare mode.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_Channel: TIM Channel. Refer to *TIM_Channel* for more details on the allowed values for this parameter.
Input parameter3	TIM_OCMode: TIM Output Compare mode. Refer to *TIM_OCMode* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	This function disables the selected channel before changing the Output Compare mode. The user has to enable this channel using the TIM_CCxCmd and TIM_CCxNCmd functions.
Called functions	None

TIM_OCMode

TIM_OCMode selects the TIM Output Compare mode (see *Table 571*).

Table 571. TIM_OCMode definition

TIM_OCMode	Description
TIM_OCMode_Timing	TIM Output Compare Timing Mode.
TIM_OCMode_Active	TIM Output Compare Active Mode.
TIM_OCMode_Inactive	TIM Output Compare Inactive Mode.
TIM_OCMode_Toggle	TIM Output Compare Toggle Mode.
TIM_OCMode_PWM1	TIM Pulse Width Modulation Mode1.
TIM_OCMode_PWM2	TIM Pulse Width Modulation Mode2.
TIM_ForcedAction_Active	Force active level on OCxREF.
TIM_ForcedAction_InActive	Force inactive level on OCxREF.

Example:

```
/* Selects the TIM1 Channel 1 PWM2 Mode */
TIM_SelectOCxM(TIM1, TIM_Channel_1, TIM_OCMode_PWM2);
```

19.2.60 TIM_UpdateDisableConfig function

Table 572 describes the TIM_UpdateDisableConfig function.

Table 572. TIM_UpdateDisableConfig function

Function name	TIM_UpdateDisableConfig
Function prototype	`void TIM_UpdateDisableConfig(TIM_TypeDef* TIMx, FunctionalState Newstate)`
Behavior description	Enables or disables the TIM update event.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	NewState: new state of the UDIS bit in TIM_CR1 register. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the Update event for TIM1 */
TIM_UpdateDisableConfig(TIM1, DISABLE);
```

19.2.61 TIM_UpdateRequestConfig function

Table 573 describes the TIM_UpdateRequestConfig function.

Table 573. TIM_UpdateRequestConfig function

Function name	TIM_UpdateRequestConfig
Function prototype	`void TIM_UpdateRequestConfig(TIM_TypeDef* TIMx, u8 TIM_UpdateSource)`
Behavior description	Selects the TIM update request source.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_UpdateSource: Update Request sources. Refer to *TIM_UpdateSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_UpdateSource

TIM_UpdateSource selects the TIM Update source (see *Table 574*).

Table 574. TIM_UpdateSource values

TIM_UpdateSource	Description
TIM_UpdateSource_Global	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_Regular	Source of update is counter overflow/underflow.

Example:

```
/* Selects the regular update source for TIM1 */
TIM_UpdateRequestConfig(TIM1, TIM_UpdateSource_Regular);
```

19.2.62 TIM_SelectHallSensor function

Table 575 describes the TIM_SelectHallSensor function.

Table 575. TIM_SelectHallSensor function

Function name	TIM_SelectHallSensor
Function prototype	`void TIM_SelectHallSensor(TIM_TypeDef* TIMx, FunctionalState Newstate)`
Behavior description	Enables or disables the TIM Hall sensor interface.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	NewState: new state of the TI1S bit in the TIM_CR2 register. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Hall Sensor Interface for TIM1 */
TIM_SelectHallSensor(TIM1, ENABLE);
```

19.2.63 TIM_SelectOnePulseMode function

Table 576 describes the TIM_SelectOnePulseMode function.

Table 576. TIM_SelectOnePulseMode function

Function name	TIM_SelectOnePulseMode
Function prototype	`void TIM_SelectOnePulseMode(TIM_TypeDef* TIMx, u16 TIM_OPMode)`
Behavior description	Selects the TIM One-pulse mode.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_OPMode: specifies the One-pulse mode to be used. Refer to *TIM_OPMode* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_OPMode

TIM_OPMode selects the TIM Update source (see *Table 577*).

Table 577. TIM_OPMode definition

TIM_OPMode	Description
TIM_OPMode_Single	TIM single One-pulse mode.
TIM_OPMode_Repetitive	TIM repetitive One-pulse mode.

Example:

```
/* Selects the single One-pulse mode for TIM1 */
TIM_SelectOnePulseMode(TIM1, TIM_OPMode_Single);
```

19.2.64 TIM_SelectOutputTrigger function

Table 578 describes the TIM_SelectOutputTrigger function.

Table 578. TIM_SelectOutputTrigger function

Function name	TIM_SelectOutputTrigger
Function prototype	`void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, u16 TIM_TRGOSource)`
Behavior description	Selects the TIM Trigger Output mode.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_TRGOSource: TRGO sources. Refer to *TIM_TRGOSource* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_TRGOSource

TIM_TRGOSource selects the TIM TRGO source (see *Table 579*).

Table 579. TIM8TRGOSource values

TIM_TRGOSource	Description
TIM_TRGOSource_Reset	The UG bit in the TIM_EGR register is used as the trigger output (TRGO).
TIM_TRGOSource_Enable	The Counter Enable CEN is used as the trigger output (TRGO).
TIM_TRGOSource_Update	The update event is selected as the trigger output (TRGO).
TIM_TRGOSource_OC1	The trigger output sends a positive pulse when the CC1IF flag is to be set, as soon as a capture or compare match occurs (TRGO).
TIM_TRGOSource_OC1Ref	OC1REF signal is used as the trigger output (TRGO).
TIM_TRGOSource_OC2Ref	OC2REF signal is used as the trigger output (TRGO).
TIM_TRGOSource_OC3Ref	OC3REF signal is used as the trigger output (TRGO).
TIM_TRGOSource_OC4Ref	OC4REF signal is used as the trigger output (TRGO).

Note: TIM6 and TIM7 can only generate TIM_TRGOSource_Reset, TIM_TRGOSource_Enable or TIM_TRGOSource_Update as trigger outputs.

Example:

```
/* Selects the update event as TRGO for TIM1 */
TIM_SelectOutputTrigger(TIM1, TIM_TRGOSource_Update);
```

19.2.65 TIM_SelectSlaveMode function

Table 580 describes the TIM_SelectSlaveMode function.

Table 580. TIM_SelectSlaveMode function

Function name	TIM_SelectSlaveMode
Function prototype	`void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, u16 TIM_SlaveMode)`
Behavior description	Selects the TIM slave mode.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_SlaveMode: TIM slave mode. Refer to *TIM_SlaveMode* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_SlaveMode

TIM_SlaveMode selects the TIMx slave mode (see *Table 581*).

Table 581. TIM_SlaveMode definition

TIM_SlaveMode	Description
TIM_SlaveMode_Reset	Rising edge of the selected trigger signal (TRGI) re-initializes the counter and triggers an update of the registers.
TIM_SlaveMode_Gated	The counter clock is enabled when the trigger signal (TRGI) is high.
TIM_SlaveMode_Trigger	The counter starts at a rising edge of the trigger TRGI.
TIM_SlaveMode_External1	Rising edges of the selected trigger (TRGI) clock the counter.

Example:

```
/* Selects the Gated Mode as Slave Mode for TIM1 */
TIM_SelectSlaveMode(TIM1, TIM_SlaveMode_Gated);
```

19.2.66 TIM_SelectMasterSlaveMode function

Table 582 describes the TIM_SelectMasterSlaveMode function.

Table 582. TIM_SelectMasterSlaveMode function

Function name	TIM_SelectMasterSlaveMode
Function prototype	`void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, u16 TIM_MasterSlaveMode)`
Behavior description	Sets or resets the TIM master/slave mode.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_MasterSlaveMode: Timer master slave mode. Refer to *TIM_MasterSlaveMode* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_MasterSlaveMode

TIM_MasterSlaveMode select the TIMx master slave mode (see *Table 583*).

Table 583. TIM_MasterSlaveMode definition

TIM_MasterSlaveMode	Description
TIM_MasterSlaveMode_Enable	Enables the master slave mode.
TIM_MasterSlaveMode_Disable	Disables the master slave mode.

Example:

```
/* Enables the Master Slave Mode for TIM2 */
TIM_SelectMasterSlaveMode(TIM2, TIM_MasterSlaveMode_Enable);
```

19.2.67 TIM_SetCounter function

Table 584 describes the TIM_SetCounter function.

Table 584. TIM_SetCounter function

Function name	TIM_SetCounter
Function prototype	`void TIM_SetCounter(TIM_TypeDef* TIMx, u16 Counter)`
Behavior description	Sets the TIMx Counter Register value.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Counter: specifies the new counter register value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM1 new Counter value */
u16 TIMCounter = 0xFFFF;
TIM_SetCounter(TIM1, TIMCounter);
```

19.2.68 TIM_SetAutoreload function

Table 585 describes the TIM_SetAutoreload function.

Table 585. TIM_SetAutoreload function

Function name	TIM_SetAutoreload
Function prototype	`void TIM_SetAutoreload(TIM_TypeDef* TIMx, u16 Autoreload)`
Behavior description	Sets the TIM Autoreload Register value.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Autoreload: new TIM period value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM1 new Autoreload value */
u16 TIMAutoreload = 0xFFFF;
TIM_SetAutoreload(TIM1, TIMAutoreload);
```

19.2.69 TIM_SetCompare1 function

Table 586 describes the TIM_SetCompare1 function.

Table 586. TIM_SetCompare1 function

Function name	TIM_SetCompare1
Function prototype	`void TIM_SetCompare1(TIM_TypeDef* TIMx, u16 Compare1)`
Behavior description	Sets the TIM Capture Compare 1 value.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Compare1: new TIM Capture Compare 1 Register value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the new TIM1 Output Compare 1 value */
u16 TIMCompare1 = 0x7FFF;
TIM_SetCompare1(TIM1, TIMCompare1);
```

19.2.70 TIM_SetCompare2 function

Table 587 describes the TIM_SetCompare2 function.

Table 587. TIM_SetCompare2 function

Function name	TIM_SetCompare2
Function prototype	`void TIM_SetCompare2(TIM_TypeDef* TIMx, u16 Compare2)`
Behavior description	Sets the TIM Capture Compare 2 Register value.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Compare2: new TIM Capture Compare 2 Register value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the new TIM1 Output Compare 2 value */
u16 TIMCompare2 = 0x7FFF;
TIM_SetCompare2(TIM1, TIMCompare2);
```

19.2.71 TIM_SetCompare3 function

Table 588 describes the TIM_SetCompare3 function.

Table 588. TIM_SetCompare3 function

Function name	TIM_SetCompare3
Function prototype	`void TIM_SetCompare3(TIM_TypeDef* TIMx, u16 Compare3)`
Behavior description	Sets the TIM Capture Compare 3 value.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Compare3: new TIM Capture Compare 3 Register value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the new TIM1 Output Compare 3 value */
u16 TIMCompare3 = 0x7FFF;
TIM_SetCompare3(TIM1, TIMCompare3);
```

19.2.72 TIM_SetCompare4 function

Table 589 describes the TIM_SetCompare4 function.

Table 589. TIM_SetCompare4 function

Function name	TIM_SetCompare4
Function prototype	`void TIM_SetCompare4(TIM_TypeDef* TIMx, u16 Compare4)`
Behavior description	Sets the TIM Capture Compare 4 value.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	Compare4: new TIM Capture Compare 4 Register value.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the new TIM1 Output Compare 4 value */
u16 TIMCompare4 = 0x7FFF;
TIM_SetCompare4(TIM1, TIMCompare4);
```

19.2.73 TIM_SetIC1Prescaler function

Table 590 describes the TIM_SetIC1Prescaler function.

Table 590. TIM_SetIC1Prescaler function

Function name	TIM_SetIC1Prescaler
Function prototype	`void TIM_SetIC1Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC1Prescaler)`
Behavior description	Sets the TIM Input Capture 1 Prescaler.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_IC1Prescaler: Input Capture 1 Prescaler. Refer to *TIM_ICPrescaler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_ICPrescaler

TIM_ICPrescaler selects the TIM Input Capture prescaler (see *Table 591*).

Table 591. TIM_ICPrescaler values

TIM_ICPrescaler	Description
TIM_ICPSC_DIV1	Capture is done each time an edge is detected on the capture input.
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.

Example:

```
/* Sets the TIM1 Input Capture 1 Prescaler */
TIM_SetIC1Prescaler(TIM1, TIM_ICPSC_Div2);
```

19.2.74 TIM_SetIC2Prescaler function

Table 592 describes the TIM_SetIC2Prescaler function.

Table 592. TIM_SetIC2Prescaler function

Function name	TIM_SetIC2Prescaler
Function prototype	`void TIM_SetIC2Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC2Prescaler)`
Behavior description	Sets the TIM Input Capture 2 Prescaler.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_IC2Prescaler: Input Capture 2 prescaler. Refer to *TIM_ICPrescaler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM1 Input Capture 2 Prescaler */
TIM_SetIC2Prescaler(TIM1, TIM_ICPSC_Div2);
```

19.2.75 TIM_SetIC3Prescaler function

Table 593 describes the TIM_SetIC3Prescaler function.

Table 593. TIM_SetIC3Prescaler function

Function name	TIM_SetIC3Prescaler
Function prototype	`void TIM_SetIC3Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC3Prescaler)`
Behavior description	Sets the TIM Input Capture 3 prescaler.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_IC3Prescaler: Input Capture 3 Prescaler. Refer to *TIM_ICPrescaler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM1 Input Capture 3 Prescaler */
TIM_SetIC3Prescaler(TIM1, TIM_ICPSC_Div2);
```

19.2.76 TIM_SetIC4Prescaler function

Table 594 describes the TIM_SetIC4Prescaler function.

Table 594. TIM_SetIC4Prescaler function

Function name	TIM_SetIC4Prescaler
Function prototype	`void TIM_SetIC4Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC4Prescaler)`
Behavior description	Sets the TIM Input Capture 4 prescaler.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_IC4Prescaler: Input Capture 4 Prescaler. Refer to *TIM_ICPrescaler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM Input Capture 4 Prescaler */
TIM_SetIC4Prescaler(TIM_ICPSC_Div2);
```

19.2.77 TIM_SetClockDivision function

Table 595 describes the TIM_SetClockDivision function.

Table 595. TIM_SetClockDivision function

Function name	TIM_SetClockDivision
Function prototype	`void TIM_SetClockDivision(TIM_TypeDef* TIMx, u16 TIM_CKD)`
Behavior description	Sets the TIM clock division value.
Input parameter1	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Input parameter2	TIM_CKD: clock division value. Refer to *TIM_ClockDivision* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

TIM_CKD

TIM_CKD selects the TIM clock division (see *Table 596*).

Table 596. TIM_CKD values

TIM_CKD	Description
TIM_CKD_DIV1	$T_{DTS} = T_{ck\_tim}$
TIM_CKD_DIV2	$T_{DTS} = 2 \times T_{ck\_tim}$
TIM_CKD_DIV4	$T_{DTS} = 4 \times T_{ck\_tim}$

Example:

```
/* Sets the TIM1 CKD value */
TIM_SetClockDivision(TIM1, TIM_CKD_DIV4);
```

19.2.78 TIM_GetCapture1 function

Table 597 describes the TIM_GetCapture1 function.

Table 597. TIM_GetCapture1 function

Function name	TIM_GetCapture1
Function prototype	`u16 TIM_GetCapture1(TIM_TypeDef* TIMx)`
Behavior description	Gets the TIM input capture 1 value.
Input parameter	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Gets the Input Capture 1 value of TIM1 */
u16 IC1value = TIM_GetCapture1(TIM1);
```

19.2.79 TIM_GetCapture2 function

Table 598 describes the TIM_GetCapture2 function.

Table 598. TIM_GetCapture2 function

Function name	TIM_GetCapture2
Function prototype	`u16 TIM_GetCapture2(TIM_TypeDef* TIMx)`
Behavior description	Gets the TIM Input Capture 2 value.
Input parameter	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Gets the Input Capture 2 value of TIM1 */
u16 IC2value = TIM_GetCapture2(TIM1);
```

19.2.80 TIM_GetCapture3 function

Table 599 describes the TIM_GetCapture3 function.

Table 599. TIM_GetCapture3 function

Function name	TIM_GetCapture3
Function prototype	`u16 TIM_GetCapture3(TIM_TypeDef* TIMx)`
Behavior description	Gets the TIM Input Capture 3 value.
Input parameter	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Gets the Input Capture 3 value of TIM1 */
u16 IC3value = TIM_GetCapture3(TIM1);
```

19.2.81 TIM_GetCapture4 function

Table 600 describes the TIM_GetCapture4 function.

Table 600. TIM_GetCapture4 function

Function name	TIM_GetCapture4
Function prototype	`u16 TIM_GetCapture4(TIM_TypeDef* TIMx)`
Behavior description	Gets the TIM Input Capture 4 value.
Input parameter	TIMx: where x can be 1, 2, 3, 4, 5 or 8 to select the TIM peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Gets the Input Capture 4 value of TIM1 */
u16 IC4value = TIM_GetCapture4(TIM1);
```

19.2.82 TIM_GetCounter function

Table 601 describes the TIM_GetCounter function.

Table 601. TIM_GetCounter function

Function name	TIM_GetCounter
Function prototype	`void TIM_GetCounter(TIM_TypeDef* TIMx)`
Behavior description	Gets the TIM counter value.
Input parameter	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Gets TIM1 counter value */
u16 TIMCounter = TIM_GetCounter(TIM1);
```

19.2.83 TIM_GetPrescaler function

Table 602 describes the TIM_GetPrescaler function.

Table 602. TIM_GetPrescaler function

Function name	TIM_GetPrescaler
Function prototype	`void TIM_GetPrescaler(TIM_TypeDef* TIMx)`
Behavior description	Gets the TIM prescaler value.
Input parameter	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Gets TIM1 prescaler value */
u16 TIMPrescaler = TIM_GetPrescaler(TIM1);
```

19.2.84 TIM_GetFlagStatus function

Table 603 describes the TIM_GetFlagStatus function.

Table 603. TIM_GetFlagStatus function

Function name	TIM_GetFlagStatus
Function prototype	`FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, u16 TIM_FLAG)`
Behavior description	Checks whether the specified TIM flag is set or not.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_FLAG: specifies the flag to check. Refer to *TIM_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of TIM_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

TIM_FLAG

The TIM flags that can be checked are listed in *Table 604*.

Table 604. TIM_FLAG definition

TIM_FLAG	Description
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_COM	TIM COM flag
TIM_FLAG_Trigger	TIM Trigger flag
TIM_FLAG_Break	TIM Break flag
TIM_FLAG_CC1OF	TIM Capture/Compare 1 Overflow flag
TIM_FLAG_CC2OF	TIM Capture/Compare 2 Overflow flag
TIM_FLAG_CC3OF	TIM Capture/Compare 3 Overflow flag
TIM_FLAG_CC4OF	TIM Capture/Compare 4 Overflow flag

Note: TIM6 and TIM7 can have only one update flag. TIM_FLAG_COM and TIM_FLAG_Break are used only with TIM1 and TIM8.

Example:

```
/* Check if the TIM1 Capture Compare 1 flag is set or reset */
if(TIM_GetFlagStatus(TIM1, TIM_FLAG_CC1) == SET)
{
}
```

19.2.85 TIM_ClearFlag function

Table 605 describes the TIM_ClearFlag function.

Table 605. TIM_ClearFlag function

Function name	TIM_ClearFlag
Function prototype	`void TIM_ClearFlag(TIM_TypeDef* TIMx, u16 TIM_Flag)`
Behavior description	Clears the pending TIM flags.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_FLAG: flag to clear. Refer to *TIM_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Note: TIM6 and TIM7 can have only one update flag. TIM_FLAG_COM and TIM_FLAG_Break are used only with TIM1 and TIM8.

Example:

```
/* Clear the TIM1 Capture Compare 1 flag */
TIM_ClearFlag(TIM1, TIM_FLAG_CC1);
```

19.2.86 TIM_GetITStatus function

Table 606 describes the TIM_GetITStatus function.

Table 606. TIM_GetITStatus function

Function name	TIM_GetITStatus
Function prototype	`ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, u16 TIM_IT)`
Behavior description	Checks whether the specified TIM interrupt has occurred or not.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_IT: specifies the TIM interrupt source to check. Refer to *TIM_IT*" for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of TIM_IT (SET or RESET).
Required preconditions	None
Called functions	None

Note: TIM6 and TIM7 can generate only one update interrupt. TIM_IT_COM and TIM_IT_Break are used only with TIM1 and TIM8.

Example:

```
/*Check if the TIM1 Capture Compare 1 interrupt has occured or not*/
```

```
if(TIM_GetITStatus(TIM1, TIM_IT_CC1) == SET)
{
}
```

19.2.87 TIM_ClearITPendingBit function

Table 607 describes the TIM_ClearITPendingBit function.

Table 607. TIM_ClearITPendingBit function

Function name	TIM_ClearITPending Bit
Function prototype	`void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, u16 TIM_IT)`
Behavior description	Clears the TIM interrupt pending bits.
Input parameter1	TIMx: where x can be 1 to 8 to select the TIM peripheral.
Input parameter2	TIM_IT: specifies the interrupt pending bit to clear. Refer to *TIM_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Note: TIM6 and TIM7 can generate only one update interrupt. TIM_IT_COM and TIM_IT_Break are used only with TIM1 and TIM8.

Example:

```
/* Clear the TIM1 Capture Compare 1 interrupt pending bit */
TIM_ClearITPendingBit(TIM1, TIM_IT_CC1);
```

20 Universal synchronous asynchronous receiver transmitter (USART)

The Universal synchronous/asynchronous receiver transmitter (USART) performs flexible full-duplex data exchange with external equipment requiring industry-standard NRZ asynchronous serial data format. The SCI offers a very wide range of baud rates based on fractional baud rate generator systems. The USART interface also supports the Smart Card Protocol compliant with IrDA SIR ENDEC specifications. It can perform single-wire half-duplex communications, synchronous transmissions and modem operations (CTS/RTS).

Section 20.1: USART register structure describes the data structures used in the USART Firmware Library. *Section 20.2: Firmware library functions* presents the Firmware Library functions.

20.1 USART register structure

The USART register structure, USART_*TypeDef*, is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu16 SR;
  u16 RESERVED1;
  vu16 DR;
  u16 RESERVED2;
  vu16 BRR;
  u16 RESERVED3;
  vu16 CR1;
  u16 RESERVED4;
  vu16 CR2;
  u16 RESERVED5;
  vu16 CR3;
  u16 RESERVED6;
  vu16 GTPR;
  u16 RESERVED7;
} USART_TypeDef;
```

Table 608 gives the list of USART registers.

Table 608. USART registers

Register	Description
SR	USART Status Register
DR	USART Data Register
BRR	USART BaudRate Register
CR1	USART Control Register 1
CR2	USART Control Register 2
CR3	USART Control Register 3
GTPR	USART Guard-Time and Prescaler Register

The three USART peripherals are declared in *stm32f10x_map.h*:

```
...
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)

#define USART1_BASE           (APB2PERIPH_BASE + 0x3800)
#define USART2_BASE           (APB1PERIPH_BASE + 0x4400)
#define USART3_BASE           (APB1PERIPH_BASE + 0x4800)
#define UART4_BASE            (APB1PERIPH_BASE + 0x4C00)
#define UART5_BASE            (APB1PERIPH_BASE + 0x5000)

#ifndef DEBUG
...
#ifdef _USART1
  #define USART1              ((USART_TypeDef *) USART1_BASE)
#endif /*_USART1 */

#ifdef _USART2
  #define USART2              ((USART_TypeDef *) USART2_BASE)
#endif /*_USART2 */

#ifdef _USART3
  #define USART3              ((USART_TypeDef *) USART3_BASE)
#endif /*_USART3 */

#ifdef _UART4
  #define UART4 ((USART_TypeDef *) UART4_BASE)
#endif /*_UART4 */

#ifdef _UART5
  #define UART5 ((USART_TypeDef *) UART5_BASE)
#endif /*_UART5 */
...
#else   /* DEBUG */
...
#ifdef _USART1
  EXT USART_TypeDef           *USART1;
#endif /*_USART1 */

#ifdef _USART2
  EXT USART_TypeDef           *USART2;
#endif /*_USART2 */

#ifdef _USART3
  EXT USART_TypeDef           *USART3;
#endif /*_USART3 */

#ifdef _UART4
  EXT USART_TypeDef           *UART4;
#endif /*_UART4 */
```

```
#ifdef _UART5
  EXT USART_TypeDef           *UART5;
#endif /*_UART5 */
...
#endif
```

When using the Debug mode, the _USART1, _USART2, _USART3, _UART4 and _UART5 pointers are initialized in the *stm32f10x_lib.c* file:

```
...
#ifdef _USART1
  USART1 = (USART_TypeDef *) USART1_BASE;
#endif /*_USART1 */

#ifdef _USART2
  USART2 = (USART_TypeDef *) USART2_BASE;
#endif /*_USART2 */

#ifdef _USART3
  USART3 = (USART_TypeDef *) USART3_BASE;
#endif /*_USART3 */

#ifdef _UART4
  UART4 = (USART_TypeDef *) UART4_BASE;
#endif /*_USART4 */

#ifdef _UART5
  UART5 = (USART_TypeDef *) UART5_BASE;
#endif /*_UART5 */
...
```

To access the USART registers, _USART, _USART1, _USART2, _USART3, _UART4 and _UART5 must be defined in *stm32f10x_conf.h* as follows:

```
...
#define _USART
#define _USART1
#define _USART2
#define _USART3
#define _UART4
#define _UART5
```

20.2 Firmware library functions

Table 609 lists the various functions of the USART library.

Table 609. USART firmware library functions

Function name	Description
USART_DeInit	Resets the USARTx peripheral registers to their default reset values.
USART_Init	Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct.
USART_StructInit	Fills each USART_InitStruct member with its default value.
USART_ClockInit	Initializes the USARTx peripheral clock according to the specified parameters in the USART_ClockInitStruct.
USART_ClockStructInit	Fills each USART_ClockInitStruct member with its default value.
USART_Cmd	Enables or disables the specified USART peripheral.
USART_ITConfig	Enables or disables the specified USART interrupts.
USART_DMACmd	Enables or disables the USART DMA interface.
USART_SetAddress	Sets the address of the USART node.
USART_WakeUpConfig	Selects the USART WakeUp method.
USART_ReceiverWakeUpCmd	Determines if the USART is in mute mode or not.
USART_LINBreakDetectionConfig	Sets the USART LIN Break detection length.
USART_LINCmd	Enables or disables the USARTx LIN mode.
USART_SendData	Transmits single data through the USARTx peripheral.
USART_ReceiveData	Returns the most recent received data by the USARTx peripheral.
USART_SendBreak	Transmits break characters.
USART_SetGuardTime	Sets the specified USART guard time.
USART_SetPrescaler	Sets the USART clock prescaler.
USART_SmartCardCmd	Enables or disables the USART Smart Card mode.
USART_SmartCardNackCmd	Enables or disables NACK transmission.
USART_HalfDuplexCmd	Enables or disables the USART Half Duplex mode.
USART_IrDAConfig	Configures the USART IrDA mode.
USART_IrDACmd	Enables or disables the USART IrDA mode.
USART_GetFlagStatus	Checks whether the specified USART flag is set or not.
USART_ClearFlag	Clears the USARTx pending flags.
USART_GetITStatus	Checks whether the specified USART interrupt has occurred or not.
USART_ClearITPendingBit	Clears the USARTx interrupt pending bits.

20.2.1 USART_DeInit function

Table 610 describes the USART_DeInit function.

Table 610. USART_DeInit function

Function name	USART_DeInit
Function prototype	`void USART_DeInit(USART_TypeDef* USARTx)`
Behavior description	Resets the USARTx peripheral registers to their default reset values.
Input parameter	USARTx: selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB2PeriphResetCmd() RCC_APB1PeriphResetCmd()

Example:

```
/* Resets the USART1 registers to their default reset value */
USART_DeInit(USART1);
```

20.2.2 USART_Init function

Table 611 describes the USART_Init function. This function uses the USART_InitTypeDef structure, which is used in asynchronous mode.

Table 611. USART_Init function

Function name	USART_Init
Function prototype	`void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)`
Behavior description	Initializes the USARTx peripheral according to the parameters specified in the USART_InitStruct.
Input parameter1	USARTx: selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_InitStruct: pointer to a USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral. Refer to *USART_InitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

USART_InitTypeDef structure

The USART_InitTypeDef structure is defined in the *stm32f10x_usart.h* file:

```
typedef struct
{
  u32 USART_BaudRate;
  u16 USART_WordLength;
  u16 USART_StopBits;
  u16 USART_Parity;
  u16 USART_HardwareFlowControl;
  u16 USART_Mode;
} USART_InitTypeDef;
```

USART_BaudRate

This member configures the USART communication baud rate. The baud rate is computed using the following formula:

IntegerDivider = ((APBClock) / (16 * (USART_InitStruct->USART_BaudRate)))

FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 16) + 0.5

USART_WordLength

USART_WordLength indicates the number of data bits transmitted or received in a frame. See *Table 612* for the values of this member.

Table 612. USART_WordLength definition

USART_WordLength	Description
USART_WordLength_8b	8 bits Data
USART_WordLength_9b	9 bits Data

USART_StopBits

USART_StopBits defines the number of stop bits transmitted. See *Table 613* for the values of this member.

Table 613. USART_StopBits definition

USART_StopBits	Description
USART_StopBits_1	1 stop bit is transmitted at the end of frame
USART_StopBits_0_5	0.5 stop bit is transmitted at the end of frame
USART_StopBits_2	2 stop bits are transmitted at the end of frame
USART_StopBits_1_5	1.5 stop bit is transmitted at the end of frame

USART_Parity

USART_Parity defines the parity mode. See *Table 614* for the values of this member.

Table 614. USART_Parity definition

USART_Parity	Description
USART_Parity_No	Parity Disable
USART_Parity_Even	Even Parity
USART_Parity_Odd	Odd Parity

Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9^{th} bit when the word length is set to 9 data bits; 8^{th} bit when the word length is set to 8 data bits).

USART_HardwareFlowControl

USART_HardwareFlowControl specifies wether the hardware flow control mode is enabled or disabled. See *Table 615* for the values of this member.

Table 615. USART_HardwareFlowControl definition

USART_HardwareFlowControl	Description
USART_HardwareFlowControl_None	HFC Disabled
USART_HardwareFlowControl_RTS	RTS enabled
USART_HardwareFlowControl_CTS	CTS enabled
USART_HardwareFlowControl_RTS_CTS	RTS and CTS enabled

USART_Mode

USART_Mode specifies wether the Receive or Transmit mode is enabled or disabled. See *Table 616* for the values of this member.

Table 616. USART_Mode definition

USART_Mode	Description
USART_Mode_Tx	Transmit enabled
USART_Mode_Rx	Receive enabled

Example:

```
/* The following example illustrates how to configure the USART1 */
USART_InitTypeDef USART_InitStructure;

USART_InitStructure.USART_BaudRate = 9600;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_Odd;
USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_RTS_CTS;
USART_InitStructure.USART_Mode = USART_Mode_Tx I  USART_Mode_Rx;
USART_Init(USART1, &USART_InitStructure);
```

20.2.3 USART_StructInit function

Table 617 describes the USART_StructInit function.

Table 617. USART_StructInit function

Function name	USART_StructInit
Function prototype	`void USART_StructInit(USART_InitTypeDef* USART_InitStruct)`
Behavior description	Fills each USART_InitStruct member with its default value.
Input parameter	USART_InitStruct: pointer to the USART_InitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The USART_InitStruct members have the following default values:

Table 618. USART_InitStruct default values

Member	Default value
USART_BaudRate	9600
USART_WordLength	USART_WordLength_8b
USART_StopBits	USART_StopBits_1
USART_Parity	USART_Parity_No
USART_HardwareFlowControl	USART_HardwareFlowControl_None
USART_Mode	USART_Mode_Rx | USART_Mode_Tx

Example:

```
/* The following example illustrates how to initialize a
USART_InitTypeDef structure */
USART_InitTypeDef USART_InitStructure;
USART_StructInit(&USART_InitStructure);
```

20.2.4 USART_ClockInit function

Table 619 describes the USART_ClockInit function. This function uses the USART_ClockInitTypeDef structure, which is used in synchronous mode.

Table 619. USART_ClockInit function

Function name	USART_ClockInit
Function prototype	`void USART_ClockInit(USART_TypeDef* USARTx, USART_ClockInitTypeDef* USART_ClockInitStruct)`
Behavior description	Initializes the USARTx peripheral Clock according to the specified parameters in the USART_ClockInitStruct.
Input parameter1	USARTx: where x can be 1, 2, 3 to select the USART peripheral. **Note:** The Smart Card mode is not available for UART4 and UART5.

Table 619. USART_ClockInit function (continued)

Input parameter2	USART_ClockInitStruct: pointer to a USART_ClockInitTypeDef structure that contains the configuration information for the specified USART peripheral clock. Refer to *USART_ClockInitTypeDef structure* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

USART_ClockInitTypeDef structure

The USART_InitTypeDef structure is defined in the *stm32f10x_usart.h* file:

```
typedef struct
{
  u16 USART_Clock;
  u16 USART_CPOL;
  u16 USART_CPHA;
  u16 USART_LastBit;
} USART_ClockInitTypeDef;
```

USART_Clock

USART_Clock indicates whether the USART clock specified in the USART_Clock member is enabled or disabled. See *Table 620* for the values of this member.

Table 620. USART_Clock definition

USART_Clock	Description
USART_Clock_Enable	USART Clock enabled
USART_Clock_Disable	USART Clock disabled

USART_CPOL

USART_CPOL specifies the steady state value of the serial clock. See *Table 621* for the values of this member.

Table 621. USART_CPOL definition

USART_CPOL	Description
USART_CPOL_High	Clock is active high
USART_CPOL_Low	Clock is active low

USART_CPHA

USART_CPHA defines the clock transition on which the bit capture is made. See *Table 622* for the values of this member.

Table 622. USART_CPHA definition

USART_CPHA	Description
USART_CPHA_1Edge	Data are captured on the first clock edge
USART_CPHA_2Edge	Data are captured on the second clock edge

USART_LastBit

USART_LastBit defines whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. See *Table 623* for the values of this member.

Table 623. USART_LastBit definition

USART_LastBit	Description
USART_LastBit_Disable	The clock pulse of the last data bit is not output on the SCLK pin.
USART_LastBit_Enable	The clock pulse of the last data bit is output on the SCLK pin.

Example:

```
/* The following example illustrates how to configure the USART1
Clock */
USART_ClockInitTypeDef USART_ClockInitStructure;

USART_ClockInitStructure.USART_Clock = USART_Clock_Disable;
USART_ClockInitStructure.USART_CPOL = USART_CPOL_High;
USART_ClockInitStructure.USART_CPHA = USART_CPHA_1Edge;
USART_ClockInitStructure.USART_LastBit = USART_LastBit_Enable;
USART_Init(USART1, &USART_ClockInitStructure);
```

20.2.5 USART_ClockStructInit function

Table 617 describes the USART_ClockStructInit function.

Table 624. USART_ClockStructInit function

Function name	USART_ClockStructInit
Function prototype	`void USART_ClockStructInit(USART_InitTypeDef* USART_ClockInitStruct)`
Behavior description	Fills each USART_ClockInitStruct member with its default value.
Input parameter	USART_ClockInitStruct: pointer to the USART_ClockInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The USART_ClockInitStruct members have the following default values:

Table 625. USART_ClockInitStruct default values

Member	Default value
USART_Clock	USART_Clock_Disable
USART_CPOL	USART_CPOL_Low
USART_CPHA	USART_CPHA_1Edge
USART_LastBit	USART_LastBit_Disable

Example:

```
/* The following example illustrates how to initialize a
USART_ClockInitTypeDef structure */
USART_ClockInitTypeDef USART_ClockInitStructure;
USART_ClockStructInit(&USART_ClockInitStructure);
```

20.2.6 USART_Cmd function

Table 626 describes the USART_Cmd function.

Table 626. USART_Cmd function

Function name	USART_Cmd
Function prototype	`void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState)`
Behavior description	Enables or disables the specified USART peripheral.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	NewState: new state of the USARTx peripheral. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the USART1 */
USART_Cmd(USART1, ENABLE);
```

20.2.7 USART_ITConfig function

Table 627 describes the USART_ITConfig function.

Table 627. USART_ITConfig function

Function name	USART_ITConfig
Function prototype	`void USART_ITConfig(USART_TypeDef* USARTx, u16 USART_IT, FunctionalState NewState)`
Behavior description	Enables or disables the specified USART interrupts.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_IT: specifies the USART interrupt sources to be enabled or disabled. Refer to *USART_IT* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the specified USARTx interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

USART_IT

USART_IT is used to enable or disable USART interrupts. Refer to *Table 628* for the values taken by this parameter.

Table 628. USART_IT values

USART_IT	Description
USART_IT_PE	Parity Error interrupt
USART_IT_TXE	Transmit Data Register empty interrupt
USART_IT_TC	Transmission complete interrupt
USART_IT_RXNE	Receive Data register not empty interrupt
USART_IT_IDLE	Idle line detection interrupt
USART_IT_LBD	LIN break detection interrupt
USART_IT_CTS	CTS change interrupt (not available for UART4 and UART5)
USART_IT_ERR	Error interrupt (Frame error, noise error, overrun error)

Example:

```
/* Enables the USART1 transmit interrupt */
USART_ITConfig(USART1, USART_IT_TXE ENABLE);
```

20.2.8 USART_DMACmd function

Table 629 describes the USART_DMACmd function.

Table 629. USART_DMACmd function

Function name	USART_DMACmd
Function prototype	`void USART_DMACmd(USART_TypeDef* USARTx, u16 USART_DMAReq, FunctionalState Newstate)`
Behavior description	Enables or disables the USART DMA interface.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3 or UART4. **Note:** The DMA mode is not available for UART5.
Input parameter2	USART_DMAReq: specifies the DMA request. Refer to *USART_DMAReq* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

USART_DMAReq

USART_DMAReq selects the DMA request to be enabled or disabled. Refer to *Table 630* for the values taken by this parameter.

Table 630. USART_DMAReq values

USART_DMAReq	Description
USART_DMAReq_Tx	Transmit DMA request
USART_DMAReq_Rx	Receive DMA request

Example:

```
/* Enable the DMA transfer on Rx and Tx action for USART2 */
USART_DMACmd(USART2, USART_DMAReq_Rx | USART_DMAReq_Tx, ENABLE);
```

20.2.9 USART_SetAddress function

Table 631 describes the USART_SetAddress function.

Table 631. USART_SetAddress function

Function name	USART_SetAddress
Function prototype	`void USART_SetAddress(USART_TypeDef* USARTx, u8 USART_Address)`
Behavior description	Sets the address of the USART node.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_Address indicates the address of the USART node.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the USART2 address node to 0x5 */
USART_SetAddress(USART2, 0x5);
```

20.2.10 USART_WakeUpConfig function

Table 632 describes the USART_WakeUpConfig function.

Table 632. USART_WakeUpConfig function

Function name	USART_WakeUpConfig
Function prototype	`void USART_WakeUpConfig(USART_TypeDef* USARTx, u16 USART_WakeUp)`
Behavior description	Selects the USART WakeUp method.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_WakeUp: specifies the USART wake-up method. Refer to *USART_WakeUp* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

USART_WakeUp

USART_WakeUp selects the wake-up method. Refer to *Table 633* for the values taken by this parameter.

Table 633. USART_WakeUp values

USART_WakeUp	Description
USART_WakeUp_IdleLine	Wakeup by an idle line detection
USART_WakeUp_AddressMark	Wakeup by an address mark

Example:

```
/* Selects the IDLE Line as USART1 WakeUp */
USART_WakeUpConfig(USART1, USART_WakeUp_IdleLine);
```

20.2.11 USART_ReceiverWakeUpCmd function

Table 634 describes the USART_ReceiverWakeUpCmd function.

Table 634. USART_ReceiverWakeUpCmd function

Function name	USART_ReceiverWakeUpCmd
Function prototype	`void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState Newstate)`
Behavior description	Determines if the USART is in mute mode or not.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	NewState: new state of the USART mute mode. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* USART3 in normal mode */
USART_ReceiverWakeUpCmd(USART3, DISABLE);
```

20.2.12 USART_LINBreakDetectLengthConfig function

Table 635 describes the USART_LINBreakDetectLengthConfig function.

Table 635. USART_LINBreakDetectLengthConfig function

Function name	USART_LINBreakDetectLengthConfig
Function prototype	`void USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, u16 USART_LINBreakDetectLength)`
Behavior description	Sets the USART LIN Break detection length.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_LINBreakDetectLength specifies the LIN break detection length. Refer to *USART_LINBreakDetectLength* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

USART_LINBreakDetectLength

USART_LINBreakDetectLength selects the LIN break detection length. Refer to *Table 636* for the values taken by this parameter.

Table 636. USART_LINBreakDetectionLength values

USART_LINBreakDetectionLength	Description
USART_LINBreakDetectLength_10b	10 bit break detection
USART_LINBreakDetectLength_11b	11 bit break detection

Example:

```
/* Selects 10 bit break detection for USART1 */
USART_LINBreakDetectLengthConfig(USART1,
USART_LINDetectLength_10b);
```

20.2.13 USART_LINCmd function

Table 637 describes the USART_LINCmd function.

Table 637. USART_LINCmd function

Function name	USART_LINCmd
Function prototype	`void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState Newstate)`
Behavior description	Enables or disables the USART LIN mode.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	NewState: new state of the USART LIN mode. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the USART2 LIN mode */
USART_LINCmd(USART2, ENABLE);
```

20.2.14 USART_SendData function

Table 638 describes the USART_SendData function.

Table 638. USART_SendData function

Function name	USART_SendData
Function prototype	`void USART_SendData(USART_TypeDef* USARTx, u16 Data)`
Behavior description	Transmits single data through the USARTx peripheral.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	Data: the data to transmit.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send one HalfWord on USART3 */
USART_SendData(USART3, 0x26);
```

20.2.15 USART_ReceiveData function

Table 639 describes the USART_ReceiveData function.

Table 639. USART_ReceiveData function

Function name	USART_ReceiveData
Function prototype	`u16 USART_ReceiveData(USART_TypeDef* USARTx)`
Behavior description	Returns the most recent data received through the USARTx peripheral.
Input parameter	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Output parameter	None
Return parameter	The received data.
Required preconditions	None
Called functions	None

Example:

```
/* Receive one halfword on USART2 */
u16 RxData;
RxData = USART_ReceiveData(USART2);
```

20.2.16 USART_SendBreak function

Table 640 describes the USART_SendBreak function.

Table 640. USART_SendBreak function

Function name	USART_SendBreak
Function prototype	`void USART_SendBreak(USART_TypeDef* USARTx)`
Behavior description	Transmits a break character
Input parameter	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send break character on USART1 */
USART_SendBreak(USART1);
```

20.2.17 USART_SetGuardTime function

Table 641 describes the USART_SetGuardTime function.

Table 641. USART_SetGuardTime function

Function name	USART_SetGuardTime
Function prototype	`void USART_SetGuardTime(USART_TypeDef* USARTx, u8 USART_GuardTime)`
Behavior description	Sets the specified USART guard time.
Input parameter1	USARTx: where x can be 1, 2 or 3 to select the USART peripheral. **Note:** The guard time bits are not available for UART4 and UART5.
Input parameter2	USART_GuardTime: specifies the guard time.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the guard time to 0x78 */
USART_SetGuardTime(0x78);
```

20.2.18 USART_SetPrescaler function

Table 642 describes the USART_SetPrescaler function.

Table 642. USART_SetPrescaler function

Function name	USART_SetPrescaler
Function prototype	`void USART_SetPrescaler(USART_TypeDef* USARTx, u8 USART_Prescaler)`
Behavior description	Sets the USART clock prescaler.
Input parameter1	USARTx: Selects the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5. **Note:** This function is used for IrDA mode with UART4 and UART5.
Input parameter2	USART_Prescaler: specifies the prescaler.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the system clock prescaler to 0x56 */
USART_SetPrescaler(0x56);
```

20.2.19 USART_SmartCardCmd function

Table 643 describes the USART_SmartCardCmd function.

Table 643. USART_SmartCardCmd function

Function name	USART_SmartCardCmd
Function prototype	`void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState Newstate)`
Behavior description	Enables or disables the USART Smartcard mode.
Input parameter1	USARTx: where x can be 1, 2 or 3 to select the USART peripheral. **Note:** The Smartcard mode is not available for UART4 and UART5.
Input parameter2	NewState: new state of the Smart Card mode. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the USART1 Smart Card mode */
USART_SmartCardCmd(USART1, ENABLE);
```

20.2.20 USART_SmartCardNACKCmd function

Table 644 describes the USART_SmartCardNACKCmd function.

Table 644. USART_SmartCardNACKCmd function

Function name	USART_SmartCardNACKCmd
Function prototype	`void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState Newstate)`
Behavior description	Enables or disables NACK transmission.
Input parameter1	USARTx: where x can be 1, 2 or 3 to select the USART peripheral. **Note:** The Smartcard mode is not available for UART4 and UART5.
Input parameter3	NewState: new state of the NACK transmission. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the USART1 NACK transmission during parity error */
USART_SmartCardNACKCmd(USART1, ENABLE);
```

20.2.21 USART_HalfDuplexCmd function

Table 645 describes the USART_HalfDuplexCmd function.

Table 645. USART_HalfDuplexCmd function

Function name	USART_HalfDuplexCmd
Function prototype	`void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState Newstate)`
Behavior description	Enables or disables the USART's Half Duplex mode.
Input parameter1	USARTx: Select the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	NewState: new state of the Half Duplex mode. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enabe HalfDuplex mode for USART2 */
USART_HalfDuplexCmd(USART2, ENABLE);
```

20.2.22 USART_IrDAConfig function

Table 646 describes the USART_IrDAConfig function.

Table 646. USART_IrDAConfig function

Function name	USART_IrDAConfig
Function prototype	`void USART_IrDAConfig(USART_TypeDef* USARTx, u16 USART_IrDAMode)`
Behavior description	Configures the USART IrDA mode.
Input parameter1	USARTx: Select the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_IrDAMode: specifies the IrDA mode. Refer to *USART_IrDAMode* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

USART_IrDAMode

USART_IrDAMode select the IrDA mode. Refer to *Table 647* for the values taken by this parameter.

Table 647. USART_IrDAMode values

USART_IrDAMode	Description
USART_IrDAMode_LowPower	IrDA low Power mode
USART_IrDAMode_Normal	IrDA normal mode

Example:

```
/* USART2 IrDA Low Power Selection */
USART_IrDAConfig(USART2,USART_IrDAMode_LowPower);
```

20.2.23 USART_IrDACmd function

Table 648 describes the USART_IrDACmd function.

Table 648. USART_IrDACmd function

Function name	USART_IrDACmd
Function prototype	`void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState Newstate)`
Behavior description	Enables or disables the USART IrDA mode.
Input parameter1	USARTx: Select the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	NewState: new state of the IrDA mode. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the USART1 IrDA Mode */
USART_IrDACmd(USART1, ENABLE);
```

20.2.24 USART_GetFlagStatus function

Table 649 describes the USART_GetFlagStatus function.

Table 649. USART_GetFlagStatus function

Function name	USART_GetFlagStatus
Function prototype	`FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, u16 USART_FLAG)`
Behavior description	Checks whether the specified USART flag is set or not.
Input parameter1	USARTx: Select the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_FLAG: specifies the flag to check. Refer to *USART_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of USART_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

USART_FLAG

The USART flags that can be checked are listed in the following table:

Table 650. USART_FLAG definition

USART_FLAG	Description
USART_FLAG_CTS	CTS change flag (not available for UART4 and UART5)
USART_FLAG_LBD	LIN Break detection flag
USART_FLAG_TXE	Transmit data register empty flag
USART_FLAG_TC	Transmission complete flag
USART_FLAG_RXNE	Read data register Not empty flag
USART_FLAG_IDLE	Idle line detected
USART_FLAG_ORE	Overrun Error
USART_FLAG_NE	Noise Error
USART_FLAG_FE	Framing Error
USART_FLAG_PE	Parity Error

Example:

```
/* Check if the transmit data register is full or not */
FlagStatus Status;
Status = USART_GetFlagStatus(USART1, USART_FLAG_TXE);
```

20.2.25 USART_ClearFlag function

Table 651 describes the USART_ClearFlag function.

Table 651. USART_ClearFlag function

Function name	USART_ClearFlag
Function prototype	`void USART_ClearFlag(USART_TypeDef* USARTx, u16 USART_FLAG)`
Behavior description	Clears the USARTx pending flags.
Input parameter1	USARTx: Select the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_FLAG: specifies the flag to clear. This parameter can be any combination of the following values: – USART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5). – USART_FLAG_LBD: LIN Break detection flag. – USART_FLAG_TC: Transmission Complete flag. – USART_FLAG_RXNE: Receive data register not empty flag Refer to *Table 652: USART_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 652. USART_FLAG

USART_FLAG	Description
USART_FLAG_CTS	CTS change flag (not available for UART4 and UART5)
USART_FLAG_LBD	LIN Break detection flag
USART_FLAG_TC	Transmission complete flag
USART_FLAG_RXNE	Read data register not empty flag

Note: 1 The PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by a software sequence: a read operation to the USART_SR register (`USART_GetFlagStatus()`) followed by a read operation to the USART_DR register (`USART_ReceiveData()`).

2 The RXNE flag can also be cleared by a read operation to the USART_DR register (`USART_ReceiveData()`).

3 The TC flag can also be cleared by a software sequence: a read operation to the USART_SR register (`USART_GetFlagStatus()`) followed by a write operation to the USART_DR register (`USART_SendData()`).

4 The TXE flag is cleared only by a write operation to the USART_DR register (`USART_SendData()`).

Example:

```
/* Clear LBD flag */
USART_ClearFlag(USART1,USART_FLAG_LBD);
```

20.2.26 USART_GetITStatus function

Table 653 describes the USART_GetITStatus function.

Table 653. USART_GetITStatus function

Function name	USART_GetITStatus
Function prototype	`ITStatus USART_GetITStatus(USART_TypeDef* USARTx, u16 USART_IT)`
Behavior description	Checks whether the specified USART interrupt has occurred or not.
Input parameter1	USARTx: Select the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_IT: specifies the USART interrupt source to check. Refer to *USART_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of USART_IT (SET or RESET).
Required preconditions	None
Called functions	None

USART_IT

USART_IT is used to read the status of USART interrupt pending bits. Refer to *Table 654* for the values taken by this parameter.

Table 654. USART_IT definition

USART_IT	Description
USART_IT_PE	Parity Error interrupt
USART_IT_TXE	Transmit interrupt
USART_IT_TC	Transmission Complete interrupt
USART_IT_RXNE	Receive interrupt
USART_IT_IDLE	IDLE line interrupt
USART_IT_LBD	LIN break detection interrupt
USART_IT_CTS	CTS change interrupt (not available for UART4 and UART5)
USART_IT_ORE	Overrun Error interrupt
USART_IT_NE	Noise Error interrupt
USART_IT_FE	Frame Error interrupt

Example:

```
/* Get the USART1 Overrun Error interrupt status */
ITStatus ErrorITStatus;
ErrorITStatus = USART_GetITStatus(USART1, USART_IT_ORE);
```

20.2.27 USART_ClearITPendingBit function

Table 655 describes the USART_ClearITPendingBit function.

Table 655. USART_ClearITPendingBit function

Function name	USART_ClearITPending Bit
Function prototype	`void USART_ClearITPendingBit(USART_TypeDef* USARTx, u16 USART_IT)`
Behavior description	Clears the USARTx interrupt pending bits.
Input parameter1	USARTx: Select the USART or UART peripheral. This parameter can assume one of the following values: USART1, USART2, USART3, UART4 or UART5.
Input parameter2	USART_IT: specifies the interrupt pending bit to clear. This parameter can be one of the following values: – USART_IT_CTS: CTS change interrupt (not available for UART4 and UART5) – USART_IT_LBD: LIN Break detection interrupt – USART_IT_TC: Transmission complete interrupt. – USART_IT_RXNE: Receive Data register not empty interrupt Refer to *Table 656: USART_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Note: 1 The PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) pending bits are cleared by a software sequence: a read operation to the USART_SR register (`USART_GetITStatus()`) followed by a read operation to the USART_DR register (`USART_ReceiveData()`).

2 The RXNE pending bit can also be cleared by a read operation to the USART_DR register (`USART_ReceiveData()`).

3 The TC pending bit can also be cleared by a software sequence: a read operation to the USART_SR register (`USART_GetITStatus()`) followed by a write operation to the USART_DR register (`USART_SendData()`).

4 The TXE pending bit is cleared only by a write operation to the USART_DR register (`USART_SendData()`).

Table 656. USART_IT

USART_IT	Description
USART_IT_CTS	CTS change interrupt (not available for UART4 and UART5)
USART_IT_LBD	LIN break detection interrupt
USART_IT_TC	Transmission Complete interrupt
USART_IT_RXNE	Receive interrupt

Example:

```
/* Clear the CTS interrupt pending bit */
USART_ClearITPendingBit(USART1, USART_IT_CTS);
```

21 Window watchdog (WWDG)

The window watchdog (WWDG) is used to detect if a software fault has occurred. A software fault is usually generated by external interference or by unforeseen logical conditions which cause the application program to abandon its normal sequence.

Section 21.1: WWDG registers describes the data structures used in the WWDG Firmware Library. *Section 21.2: Firmware library functions* presents the Firmware Library functions.

21.1 WWDG registers

The WWDG register structure, *WWDG_TypeDef, is* defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 CR;
  vu32 CFR;
  vu32 SR;
} WWDG_TypeDef;
```

Table 657 gives the list of WWDG registers.

Table 657. WWDG registers

Register	Description
CR	Window Watchdog Control register
CFR	Window Watchdog Configuration Register
SR	Window Watchdog Status Register

The WWDG peripheral is declared in *stm32f10x_map.h*, as following:

```
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)

#define WWDG_BASE             (APB1PERIPH_BASE + 0x2C00)

#ifndef DEBUG
...
#ifdef _WWDG
  #define WWDG                ((WWDG_TypeDef *) WWDG_BASE)
#endif /*_WWDG */
...
#else   /* DEBUG */
...
#ifdef _WWDG
  EXT WWDG_TypeDef            *WWDG;
#endif /*_WWDG */
...
```

```
#endif
```

When using the Debug mode, WWDG pointer is initialized in *stm32f10x_lib.c*:

```
#ifdef _WWDG
  WWDG = (WWDG_TypeDef *)  WWDG_BASE;
#endif /*_WWDG */
```

To access the window watchdog registers, _WWDG must be defined in *stm32f10x_conf.h*, as follows:

```
#define _WWDG
```

21.2 Firmware library functions

Table 658 gives the list of the various functions in the WWDG library.

Table 658. WWGD firmware library functions

Function name	Description
WWDG_DeInit	Resets the WWDG peripheral registers to their default reset values.
WWDG_SetPrescaler	Sets the WWDG Prescaler.
WWDG_SetWindowValue	Sets the WWDG window value.
WWDG_EnableIT	Enables the WWDG Early Wake-up interrupt (EWI).
WWDG_SetCounter	Sets the WWDG counter value.
WWDG_Enable	Enables WWDG and load the counter value.
WWDG_GetFlagStatus	Checks whether the Early Wake-up interrupt flag is set or not.
WWDG_ClearFlag	Clears Early Wake-up interrupt flag.

21.2.1 WWDG_DeInit function

Table 659 describes the WWDG_DeInit function.

Table 659. WWDG_DeInit function

Function name	WWDG_DeInit
Function prototype	`void WWDG_DeInit(void)`
Behavior description	Resets the WWDG peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	RCC_APB1PeriphResetCmd

Example:

```
/* Deinitialize the WWDG registers */
WWDG_DeInit();
```

21.2.2 WWDG_SetPrescaler function

Table 660 describes the WWDG_SetPrescaler function.

Table 660. WWDG_SetPrescaler function

Function name	WWDG_SetPrescaler
Function prototype	`void WWDG_SetPrescaler(u32 WWDG_Prescaler)`
Behavior description	Sets the WWDG Prescaler.
Input parameter	WWDG_Prescaler: specifies the WWDG Prescaler. Refer to *WWDG_Prescaler* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

WWDG_Prescaler

WWDG_Prescaler selects the WWDG Prescaler. Refer to *Table 661* for the values taken by this parameter.

Table 661. WWGD_Prescaler values

WWDG_Prescaler	Description
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

Example:

```
/* Set WWDG prescaler to 8 */
WWDG_SetPrescaler(WWDG_Prescaler_8);
```

21.2.3 WWDG_SetWindowValue function

Table 662 describes WWDG_SetWindowValue function.

Table 662. WWDG_SetWindowValue function

Function name	WWDG_SetWindowValue
Function prototype	`void WWDG_SetWindowValue(u8 WindowValue)`
Behavior description	Sets the WWDG window value.
Input parameter	WindowValue: specifies the window value to be compared to the downcounter. This parameter value must be lower than 0x80.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set WWDG window value to 0x50 */
WWDG_SetWindowValue(0x50);
```

21.2.4 WWDG_EnableIT function

Table 663 describes WWDG_EnableIT function.

Table 663. WWDG_EnableIT function

Function name	WWDG_EnableIT
Function prototype	`void WWDG_EnableIT(void)`
Behavior description	Enables the WWDG Early Wake-up interrupt(EWI).
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable WWDG Early wakeup interrupt */
WWDG_EnableIT();
```

21.2.5 WWDG_SetCounter function

Table 664 describes WWDG_SetCounter function.

Table 664. WWDG_SetCounter function

Function name	WWDG_SetCounter
Function prototype	`void WWDG_SetCounter(u8 Counter)`
Behavior description	Sets the WWDG counter value.
Input parameter	Counter: specifies the watchdog counter value. This parameter must be a number between 0x40 and 0x7F.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set WWDG counter value to 0x70 */
WWDG_SetCounter(0x70);
```

21.2.6 WWDG_Enable function

Table 665 describes WWDG_Enable function.

Table 665. WWDG_Enable function

Function name	WWDG_Enable
Function prototype	`void WWDG_Enable(u8 Counter)`
Behavior description	Enables WWDG and load the counter value(1)
Input parameter	Counter: specifies the watchdog counter value. This parameter must be a number between 0x40 and 0x7F.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

1. Once enabled the WWDG can not be disabled any more.

Example:

```
/* Enable WWDG and set counter value to 0x7F */
WWDG_Enable(0x7F);
```

21.2.7 WWDG_GetFlagStatus function

Table 666 describes WWDG_GetFlagStatus function.

Table 666. WWDG_GetFlagStatus function

Function name	WWDG_GetFlagStatus
Function prototype	`FlagStatus WWDG_GetFlagStatus(void)`
Behavior description	Checks whether the Early Wake-up interrupt flag is set or not.
Input parameter	None
Output parameter	None
Return parameter	The new state of the Early Wake-up interrupt flag (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Test if the counter has reached the value 0x40 */
FlagStatus Status;
Status = WWDG_GetFlagStatus();
if(Status == RESET)
{
...
}
else
{
...
}
```

21.2.8 WWDG_ClearFlag function

Table 667 describes WWDG_ClearFlag function.

Table 667. WWDG_ClearFlag function

Function name	WWDG_ClearFlag
Function prototype	`void WWDG_ClearFlag(void)`
Behavior description	Clears Early Wake-up interrupt flag.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear EWI flag */
WWDG_ClearFlag();
```

22 Digital/analog converter (DAC)

The digital/analog converter (DAC) module is a 12-bit, voltage output digital/analog converter.

Section 22.1 describes the data structures used in the DAC firmware library. *Section 22.2* one presents the firmware library functions.

22.1 DAC register structure

The *DAC_TypeDef* DAC register structure is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 CR;
  vu32 SWTRIGR;
  vu32 DHR12R1;
  vu32 DHR12L1;
  vu32 DHR8R1;
  vu32 DHR12R2;
  vu32 DHR12L2;
  vu32 DHR8R2;
  vu32 DHR12RD;
  vu32 DHR12LD;
  vu32 DHR8RD;
  vu32 DOR1;
  vu32 DOR2;
} DAC_TypeDef;
```

Table 668 shows the DAC registers.

Table 668. DAC registers

Register	Description
CR	DAC Control Register
SWTRIGR	DAC Software Trigger Register
DHR12R1	DAC channel1 12-bit Right Aligned Data Holding Register
DHR12L1	DAC channel1 12-bit Left Aligned Data Holding Register
DHR8R1	DAC channel1 8-bit Right Aligned Data Holding Register
DHR12R2	DAC channel2 12-bit Right Aligned Data Holding Register
DHR12L2	DAC channel2 12-bit Left Aligned Data Holding Register
DHR8R2	DAC channel2 8-bit Right Aligned Data Holding Register
DHR12RD	Dual DAC 12-bit Right Aligned Data Holding Register
DHR12LD	Dual DAC 12-bit Left Aligned Data Holding Register
DHR8RD	Dual DAC 8-bit Right Aligned Data Holding Register
DOR1	DAC channel1 Data Output Register
DOR2	DAC channel2 Data Output Register

The DAC peripheral is declared in the same file:

```
...
#define PERIPH_BASE   0x40000000
#define APB1PERIPH_BASE       (PERIPH_BASE)
....
#define DAC_BASE    (APB1PERIPH_BASE + 0x7400)
....
#ifndef DEBUG
...
#define DAC ((DAC_TypeDef *) DAC_BASE)
...
#else
...
#ifdef _DAC
  EXT DAC_TypeDef    *DAC;
#endif /*_DAC */
...
#endif
```

When the debug mode is used, _DAC pointer is initialized in the *stm32f10x_lib.c* file:

```
...
#ifdef _DAC
  DAC = (DAC_TypeDef *) DAC_BASE;
#endif /*_DAC */
...
```

_DAC must be defined in the *stm32f10x_conf.h* file, to access the peripheral registers as follows:

```
...
#define _DAC
...
```

22.2 Firmware library functions

Table 669 gives the list of the DAC library functions.

Table 669. DAC firmware library functions

Function name	Description
DAC_DeInit	De-initializes the DAC peripheral registers to their default reset values.
DAC_Init	Initializes the DAC peripheral according to the specified parameters in DAC_InitStruct.
DAC_StructInit	Fills each DAC_InitStruct member with its default value.
DAC_Cmd	Enables or disables the specified DAC channel.
DAC_DMACmd	Enables or disables the specified DAC channel DMA request.
DAC_SoftwareTriggerCmd	Enables or disables the selected DAC channel software trigger.
DAC_DualSoftwareTriggerCmd	Simultaneously enables or disables the two DAC channel software triggers.
DAC_WaveGenerationCmd	Enables or disables the selected DAC channel wave generation.
DAC_SetChannel1Data	Sets the specified data holding register value for DAC channel1.
DAC_SetChannel2Data	Sets the specified data holding register value for DAC channel2.
DAC_SetDualChannelData	Sets the specified data holding register value for dual channel DAC.
DAC_GetDataOutputValue	Returns the last data output value for the selected DAC channel.

22.2.1 DAC_DeInit

Table 670 describes the DAC_DeInit function.

Table 670. DAC_DeInit function

Function name	DAC_DeInit
Function prototype	`void DAC_DeInit(void)`
Behavior description	De-initializes the DAC peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called function	RCC_APB1PeriphClockCmd().

Example:

```
/* Deinitialize the DAC */
DAC_DeInit();
```

22.2.2 DAC_Init

Table 671 describes the DAC_Init function.

Table 671. DAC_Init function

Function name	DAC_Init
Function prototype	`void DAC_Init(u32 DAC_Channel, DAC_InitTypeDef* DAC_InitStruct)`
Behavior description	Initializes the DAC peripheral according to the specified parameters in DAC_InitStruct.
Input parameter1	DAC_Channel: specifies the selected DAC channel. Refer to *DAC_Channel* for more details on the allowed values for this parameter.
Input parameter2	DAC_InitStruct: pointer to a DAC_InitTypeDef structure that contains the configuration information for the DAC peripheral. Refer to *DAC_InitTypeDef* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

DAC_Channel

Specifies the DAC channel to configure. Refer to *Table 672* for the values taken by this parameter.

Table 672. DAC_Channel definition

DAC_Channel	Description
DAC_Channel_1	DAC Channel1 selected
DAC_Channel_2	DAC Channel2 selected

DAC_InitTypeDef

The DAC_InitTypeDef structure is defined in the *stm32f10x_dac.h* file:

```
typedef struct
{
u32 DAC_Trigger;
  u32 DAC_WaveGeneration;
u32 DAC_LFSRUnmask_TriangleAmplitude;
u32 DAC_OutputBuffer;
} DAC_InitTypeDef;
```

DAC_Trigger

Specifies the external trigger for the selected DAC channel (see *Table 673*).

Table 673. DAC_Trigger definition

DAC_Trigger	Description
DAC_Trigger_None	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
DAC_Trigger_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_Trigger_T8_TRGO	TIM8 TRGO selected as external conversion trigger for DAC channel
DAC_Trigger_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_Trigger_T5_TRGO	TIM5 TRGO selected as external conversion trigger for DAC channel
DAC_Trigger_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_Trigger_T4_TRGO	TIM4 TRGO selected as external conversion trigger for DAC channel
DAC_Trigger_Ext_IT9	External interrupt 9 event selected as external conversion trigger for DAC channel
DAC_Trigger_Software	Conversion started by external software trigger for DAC channel

DAC_WaveGeneration

Specifies whether DAC channel noise waves or triangle waves are generated, or whether no wave is generated. *Table 674* shows the values that can be assumed by this member.

Table 674. DAC_WaveGeneration definition

DAC_WaveGeneration	Description
DAC_WaveGeneration_None	No wave is generated on DAC channel
DAC_WaveGeneration_Noise	Noise wave is generated on DAC channel
DAC_WaveGeneration_Triangle	Triangle wave is generated on DAC channel

DAC_LFSRUnmask_TriangleAmplitude

Specifies the LFSR mask for noise wave generation or the maximum amplitude triangle generation for the DAC channel. *Table 675* shows the values that can be assumed by this member.

Table 675. DAC_LFSRUnmask_TriangleAmplitude definition

DAC_LFSRUnmask_TriangleAmplitude	Description
DAC_LFSRUnmask_Bit0 / DAC_TriangleAmplitude_1	Unmask DAC channel LFSR bit0 for noise wave generation / Select max triangle amplitude of 1
DAC_LFSRUnmask_Bits1_0 / DAC_TriangleAmplitude_3	Unmask DAC channel LFSR bit[1:0] for noise wave generation / Select max triangle amplitude of 3
DAC_LFSRUnmask_Bits2_0 / DAC_TriangleAmplitude_7	Unmask DAC channel LFSR bit[2:0] for noise wave generation / Select max triangle amplitude of 7
DAC_LFSRUnmask_Bits3_0 / DAC_TriangleAmplitude_15	Unmask DAC channel LFSR bit[3:0] for noise wave generation / Select max triangle amplitude of 15
DAC_LFSRUnmask_Bits4_0 / DAC_TriangleAmplitude_31	Unmask DAC channel LFSR bit[4:0] for noise wave generation / Select max triangle amplitude of 31

Table 675. DAC_LFSRUnmask_TriangleAmplitude definition (continued)

DAC_LFSRUnmask_TriangleAmplitude	Description
DAC_LFSRUnmask_Bits5_0 / DAC_TriangleAmplitude_63	Unmask DAC channel LFSR bit[5:0] for noise wave generation / Select max triangle amplitude of 63
DAC_LFSRUnmask_Bits6_0 / DAC_TriangleAmplitude_127	Unmask DAC channel LFSR bit[6:0] for noise wave generation / Select max triangle amplitude of 127
DAC_LFSRUnmask_Bits7_0 / DAC_TriangleAmplitude_255	Unmask DAC channel LFSR bit[7:0] for noise wave generation / Select max triangle amplitude of 255
DAC_LFSRUnmask_Bits8_0 / DAC_TriangleAmplitude_511	Unmask DAC channel LFSR bit[8:0] for noise wave generation / Select max triangle amplitude of 511
DAC_LFSRUnmask_Bits9_0 / DAC_TriangleAmplitude_1023	Unmask DAC channel LFSR bit[9:0] for noise wave generation / Select max triangle amplitude of 1023
DAC_LFSRUnmask_Bits10_0 / DAC_TriangleAmplitude_2047	Unmask DAC channel LFSR bit[10:0] for noise wave generation / Select max triangle amplitude of 2047
DAC_LFSRUnmask_Bits11_0 / DAC_TriangleAmplitude_4095	Unmask DAC channel LFSR bit[11:0] for noise wave generation / Select max triangle amplitude of 4095

DAC_OutputBuffer

Specifies whether the DAC channel output buffer is enabled or disabled. *Table 676* shows the values that can be assumed by this member.

Table 676. DAC_OutputBuffer definition

DAC_OutputBuffer	Description
DAC_OutputBuffer_Enable	Output buffer is enabled for DAC channel
DAC_OutputBuffer_Disable	Output buffer is disabled for DAC channel

Example:

```
/* Initialize the DAC channel1 according to the DAC_InitStructure
members */
  DAC_InitTypeDef DAC_InitStructure;

  DAC_InitStructure.DAC_Trigger = DAC_Trigger_T6_TRGO;
  DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_Noise;
  DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude =
DAC_LFSRUnmask_Bits11_0;
 DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
  DAC_Init(DAC_Channel_1, &DAC_InitStructure);
```

22.2.3 DAC_StructInit

Table 677 describes the DAC_StructInit function.

Table 677. DAC_StructInit function

Function name	DAC_StructInit
Function prototype	`void DAC_StructInit(DAC_InitTypeDef* DAC_InitStruct)`
Behavior description	Fills each DAC_InitStruct member with its default value.
Input parameter	DAC_InitStruct: pointer to an DAC_InitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

DAC_InitStruct

Table 678 gives the values that can be assumed by the DAC_InitStruct members.

Table 678. DAC_InitStruct definition

DAC_InitStruct	Default values
DAC_Trigger	DAC_Trigger_None
DAC_WaveGeneration	DAC_WaveGeneration_None
DAC_LFSRUnmask_TriangleAmplitude	DAC_LFSRUnmask_Bit0
DAC_OutputBuffer	DAC_OutputBuffer_Enable

Example:

```
/* Initialize a DAC_InitTypeDef structure. */
DAC_InitTypeDef  DAC_InitStructure;
DAC_StructInit(&DAC_InitStructure);
```

22.2.4 DAC_Cmd

Table 679 describes the DAC_Cmd function.

Table 679. DAC_Cmd function

Function name	DAC_Cmd
Function prototype	`void DAC_Cmd(u32 DAC_Channel, FunctionalState NewState)`
Behavior description	Enables or disables the specified DAC channel.
Input parameter1	DAC_Channel: specifies the selected DAC channel. Refer to *DAC_Channel* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the selected DAC channel. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable DAC channel1 */
DAC_Cmd(DAC_Channel_1, ENABLE);
```

22.2.5 DAC_DMACmd

Table 680 describes the DAC_DMACmd function.

Table 680. DAC_DMACmd function

Function name	DAC_DMACmd
Function prototype	`DAC_DMACmd(u32 DAC_Channel, FunctionalState NewState)`
Behavior description	Enables or disables the specified DAC channel DMA request.
Input parameter1	DAC_Channel: specifies the selected DAC channel Refer to *DAC_Channel* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the selected DAC channel DMA request. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Enable DAC channel2 DMA request */
DAC_DMACmd(DAC_Channel_2, ENABLE);
```

22.2.6 DAC_SoftwareTriggerCmd

Table 681 describes the DAC_SoftwareTriggerCmd function.

Table 681. DAC_SoftwareTriggerCmd function

Function name	DAC_SoftwareTriggerCmd
Function prototype	`void DAC_SoftwareTriggerCmd(u32 DAC_Channel, FunctionalState NewState)`
Behavior description	Enables or disables the selected DAC channel software trigger.
Input parameter1	DAC_Channel: specifies the selected DAC channel. Refer to *DAC_Channel* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the selected DAC channel software trigger. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable DAC channel1 software trigger */
DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE);
```

22.2.7 DAC_DualSoftwareTriggerCmd

Table 677 describes the DAC_DualSoftwareTriggerCmd function.

Table 682. DAC_DualSoftwareTriggerCmd function

Function name	DAC_DualSoftwareTriggerCmd
Function prototype	`void DAC_DualSoftwareTriggerCmd(FunctionalState NewState)`
Behavior description	Enables or disables the selected DAC channel software trigger.
Input parameter1	NewState: new state of the dual DAC channel software trigger. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable both DAC channels software trigger */
DAC_DualSoftwareTriggerCmd(ENABLE);
```

22.2.8 DAC_WaveGenerationCmd

Table 677 describes the DAC_WaveGenerationCmd function.

Table 683. DAC_WaveGenerationCmd function

Function name	DAC_WaveGenerationCmd
Function prototype	`void DAC_WaveGenerationCmd(u32 DAC_Channel, u32 DAC_Wave, FunctionalState NewState)`
Behavior description	Enables or disables the selected DAC channel wave generation.
Input parameter1	DAC_Channel: specifies the selected DAC channel Refer to *DAC_Channel* for more details on the allowed values for this parameter.
Input parameter2	DAC_Wave: specifies the wave type to enable or disable. Refer to *DAC_Wave* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the selected DAC channel wave generation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

DAC_Wave

Table 684 gives the values to be selected to have the desired DAC wave type.

Table 684. Dac_Wave definition

DAC_Wave	Description
DAC_Wave_Noise	Selects noise wave generation
DAC_Wave_Triangle	Selects triangle wave generation

Example:

```
/* Enable DAC channel1 noise wave generation */
DAC_Wave GenerationCmd(DAC_Channel_1, DAC_Wave_Noise, ENABLE);
```

22.2.9 DAC_SetChannel1Data

Table 677 describes the DAC_SetChannel1Data function.

Table 685. DAC_SetChannel1Data function

Function name	DAC_SetChannel1Data
Function prototype	`void DAC_SetChannel1Data(u32 DAC_Align, u16 Data)`
Behavior description	Set the specified data holding register value for DAC channel1.
Input parameter1	DAC_Align: Specifies the data alignment for DAC channel1. Refer to *DAC_Align* for more details on the allowed values for this parameter.
Input parameter2	Data: the data to be loaded in the selected data holding register. The value must be: - 12bit right data alignment: Data<= 0x0FFF - 12bit left data alignment: Data<= 0xFFF0 - 8bit right data alignment: Data<= 0x00FF
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

DAC_Align

Table 686 gives the values used to select the desired data holding register.

Table 686. DAC_Align definition

DAC_Align	Description
DAC_Align_12b_R	12-bit right data alignment for the selected DAC channel
DAC_Align_12b_L	12-bit left data alignment for the selected DAC channel
DAC_Align_8b_R	8-bit right data alignment for the selected DAC channel

Example:

```
/* Set 0x500 value in the DAC channel1 12bit right alignement data
holding register */
DAC_SetChannel1Data(DAC_Align_12b_R, 0x500);
```

22.2.10 DAC_SetChannel2Data

Table 677 describes the DAC_SetChannel2Data function.

Table 687. DAC_SetChannel2Data function

Function name	DAC_SetChannel2Data
Function prototype	`void DAC_SetChannel2Data(u32 DAC_Align, u16 Data)`
Behavior description	Sets the specified data holding register value for DAC channel2.
Input parameter1	DAC_Align: specifies the data alignment for DAC channel2. Refer to *DAC_Align* for more details on the allowed values for this parameter.
Input parameter2	Data: specifies the data to be loaded in the selected data holding register. The value must be: – 12-bit right data alignment: Data<= 0x0FFF – 12-bit left data alignment: Data<= 0xFFF0 – 8-bit right data alignment: Data<= 0x00FF
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set 0x8880 value in the DAC channel2 12bit left alignement data
holding register */
DAC_SetChannel2Data(DAC_Align_12b_L, 0x8880);
```

22.2.11 DAC_SetDualChannelData

Table 677 describes the DAC_SetDualChannelData function.

Table 688. DAC_SetDualChannelData function

Function name	DAC_SetDualChannelData
Function prototype	`void DAC_SetDualChannelData(u32 DAC_Align, u16 Data2, u16 Data1)`
Behavior description	Sets the specified data holding register value for dual channel DAC.
Input parameter1	DAC_Align: specifies the data alignment for dual channel DAC. Refer to *DAC_Align* for more details on the allowed values for this parameter.
Input parameter2	Data2: specifies the data for DAC channel2 to be loaded in the selected data holding register. The value must be: – 12-bit right data alignment: Data<= 0x0FFF – 12-bit left data alignment: Data<= 0xFFF0 – 8-bit right data alignment: Data<= 0x00FF
Input parameter3	Data1: specifies the data for DAC channel1 to be loaded in the selected data holding register. The value must be: – 12-bit right data alignment: Data<= 0x0FFF – 12-bit left data alignment: Data<= 0xFFF0 – 8-bit right data alignment: Data<= 0x00FF
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set 0xF1 value for DAC channel1 and 0x40 for DAC channel2, in the
dual channel DAC 8bit right alignement data holding register */
DAC_SetDualChannelData(DAC_Align_8b_R, 0x40, 0xF1);
```

22.2.12 DAC_GetDataOutputValue

Table 677 describes the DAC_GetDataOutputValue function.

Table 689. DAC_GetDataOutputValue function

Function name	DAC_GetDataOutputValue
Function prototype	`u16 DAC_GetDataOutputValue(u32 DAC_Channel)`
Behavior description	Returns the last data output value of the selected DAC channel.
Input parameter	DAC_Channel: specifies the selected DAC channel Refer to *DAC_Channel* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The selected DAC channel data output value.
Required preconditions	None
Called functions	None

Example:

```
/* Returns the DAC channel1 data output value */
u16 DataValue;
DataValue = DAC_GetDataOutputValue(DAC_Channel_1);
```

23 Flexible static memory controller (FSMC)

The FSMC block is able to interface with

Section 23.1 describes the data structures used in the FSMC firmware library. *Section 23.2* presents the firmware library functions.

23.1 FSMC register structure

The *FSMC_TypeDef* FSMC register structure is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 BTCR[8];
} FSMC_Bank1_TypeDef;

typedef struct
{
  vu32 BWTR[7];
} FSMC_Bank1E_TypeDef;

typedef struct
{
  vu32 PCR2;
  vu32 SR2;
  vu32 PMEM2;
  vu32 PATT2;
  u32  RESERVED0;
  vu32 ECCR2;
} FSMC_Bank2_TypeDef;

typedef struct
{
  vu32 PCR3;
  vu32 SR3;
  vu32 PMEM3;
  vu32 PATT3;
  u32  RESERVED0;
  vu32 ECCR3;
} FSMC_Bank3_TypeDef;

typedef struct
{
  vu32 PCR4;
  vu32 SR4;
  vu32 PMEM4;
  vu32 PATT4;
  vu32 PIO4;
} FSMC_Bank4_TypeDef;
```

Table 690 gives the list of FSMC registers.

Table 690. FSMC registers

Register	Description
FSMC_BCR1	SRAM/NOR-Flash chip-select control register1
FSMC_BTR1	SRAM/NOR-Flash chip-select timing register1
FSMC_BWTR1	SRAM/NOR-Flash chip-select timing register1
FSMC_BCR2	SRAM/NOR-Flash chip-select control register2
FSMC_BTR2	SRAM/NOR-Flash chip-select timing register2
FSMC_BWTR2	SRAM/NOR-Flash chip-select timing register2
FSMC_BCR3	SRAM/NOR-Flash chip-select control register3
FSMC_BTR3	SRAM/NOR-Flash chip-select timing register4
FSMC_BWTR3	SRAM/NOR-Flash chip-select timing register3
FSMC_BCR4	SRAM/NOR-Flash chip-select control register4
FSMC_BTR4	SRAM/NOR-Flash chip-select timing register3
FSMC_BWTR4	SRAM/NOR-Flash chip-select timing register4
FSMC_PCR2	PC Card/NAND Flash control register2
FSMC_SR2	FIFO status and interrupt register2
FSMC_PMEM2	Common memory space timing register2
FSMC_PATT2	Attribute memory space timing register2
FSMC_ECCR2	ECC result registers2
FSMC_PCR3	PC Card/NAND Flash control register3
FSMC_SR3	FIFO status and interrupt register3
FSMC_PMEM3	Common memory space timing register3
FSMC_PATT3	Attribute memory space timing register3
FSMC_ECCR3	ECC result registers3
FSMC_PCR4	PC Card/NAND Flash control register4
FSMC_PMEM4	Common memory space timing register4
FSMC_PATT4	Attribute memory space timing register3
FSMC_PIO4	I/O space timing register4

When the debug mode is used, the _FSMC pointer is initialized in the *stm32f10x_lib.c* file. _FSMC must be defined in the *stm32f10x_conf.h* file to access the peripheral registers. The FSMC peripheral is declared in the same file:

```
/* FSMC registers base address */
#define FSMC_R_BASE           ((u32)0xA0000000)
...
/* FSMC Bankx registers base address */
#define FSMC_Bank1_R_BASE     (FSMC_R_BASE + 0x0000)
#define FSMC_Bank1E_R_BASE    (FSMC_R_BASE + 0x0104)
#define FSMC_Bank2_R_BASE     (FSMC_R_BASE + 0x0060)
```

```
#define FSMC_Bank3_R_BASE     (FSMC_R_BASE + 0x0080)
#define FSMC_Bank4_R_BASE     (FSMC_R_BASE + 0x00A0)
...
#ifndef DEBUG
...
#ifdef _FSMC
  #define FSMC_Bank1          ((FSMC_Bank1_TypeDef *)
FSMC_Bank1_R_BASE)
  #define FSMC_Bank1E         ((FSMC_Bank1E_TypeDef *)
FSMC_Bank1E_R_BASE)
  #define FSMC_Bank2          ((FSMC_Bank2_TypeDef *)
FSMC_Bank2_R_BASE)
  #define FSMC_Bank3          ((FSMC_Bank3_TypeDef *)
FSMC_Bank3_R_BASE)
  #define FSMC_Bank4          ((FSMC_Bank4_TypeDef *)
FSMC_Bank4_R_BASE)
#endif /*_FSMC */
...
#else   /* DEBUG */
...
#ifdef _FSMC
  EXT FSMC_Bank1_TypeDef      *FSMC_Bank1;
  EXT FSMC_Bank1E_TypeDef     *FSMC_Bank1E;
  EXT FSMC_Bank2_TypeDef      *FSMC_Bank2;
  EXT FSMC_Bank3_TypeDef      *FSMC_Bank3;
  EXT FSMC_Bank4_TypeDef      *FSMC_Bank4;
#endif /*_FSMC */
...
#endif  /* DEBUG*/
When debug mode is used, _FSMC pointer is initialized in
stm32f10x_lib.c file :
...
#ifdef _FSMC
  FSMC_Bank1 = (FSMC_Bank1_TypeDef *)    FSMC_Bank1_R_BASE;
  FSMC_Bank1E = (FSMC_Bank1E_TypeDef *)  FSMC_Bank1E_R_BASE;
  FSMC_Bank2 = (FSMC_Bank2_TypeDef *)    FSMC_Bank2_R_BASE;
  FSMC_Bank3 = (FSMC_Bank3_TypeDef *)    FSMC_Bank3_R_BASE;
  FSMC_Bank4 = (FSMC_Bank4_TypeDef *)    FSMC_Bank4_R_BASE;
#endif /*_FSMC */
...
_FSMC must be defined, in stm32f10x_conf.h file, to access the
peripheral registers as follows:
...
#define _FSMC
...
```

23.2 Firmware library functions

Table 691 gives the list of the FSMC library functions.

Table 691. FSMC firmware library functions

Function name	Description
FSMC_NORSRAMDeInit	Re-initializes the FSMC NOR bank registers to their default reset values.
FSMC_NANDDeInit	Re-initializes the FSMC NAND bank registers to their default reset values.
FSMC_PCCARDDeInit	Re-initializes the FSMC PCCARD bank registers to their default reset values.
FSMC_NORSRAMInit	Initializes the FSMC NOR memory bank according to the parameters specified in FSMC_NORInitStruct.
FSMC_NANDInit	Initializes the FSMC NOR memory bank according to the parameters specified in FSMC_NANDInitStruct.
FSMC_PCCARDInit	Initializes the FSMC PCCARD memory bank according to the parameters specified in FSMC_NORInitStruct.
FSMC_NORSRAMStructInit	Fills each FSMC_NORInitStruct member with its default value.
FSMC_NANDStructInit	Fills each FSMC_NORInitStruct member with its default value.
FSMC_PCCARDStructInit	Fills each FSMC_NORInitStruct member with its default value.
FSMC_NORSRAMCmd	Enables or disables the NOR/SRAM memory bank1.
FSMC_NANDCmd	Enables or disables the specified NAND memory bank(1 or 2).
FSMC_PCCARDCmd	Enables or disables the PCCARD memory bank3.
FSMC_NANDECCCmd	Enables or disables the NAND ECC feature.
FSMC_GetECC	Returns the error correction code register value.
FSMC_ITConfig	Enables or disables the specified interrupts.
FSMC_GetFlagStatus	Checks whether the specified FSMC flag is set or not.
FSMC_ClearFlag	Clears the FSMC's pending flags.
FSMC_GetITStatus	Checks whether the specified FSMC interrupt has occurred or not.
FSMC_ClearITPendingBit	Clears the FSMC's interrupt pending bits.

23.2.1 FSMC_NORSRAMDeInit

Table 692 describes the FSMC_NORSRAMDeInit function.

Table 692. FSMC_NORSRAMDeInit function

Function name	FSMC_NORSRAMDeInit
Function prototype	`void FSMC_NORSRAMDeInit(u32 FSMC_Bank)`
Behavior description	Re-initializes the FSMC NOR/SRAM bank registers to their default reset values.
Input parameter	FSMC_Bank: specifies the FSMC NOR/SRAM bank to be used. Refer to *FSMC_Bank* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FSMC_Bank

Specifies the NOR/SRAM Banks that can be used. *Table 693* shows the values assumed by this parameter.

Table 693. FSMC_Bank definition

FSMC_Bank	Description
FSMC_Bank1_NORSRAM1	FSMC Bank1 NOR/SRAM1
FSMC_Bank1_NORSRAM2	FSMC Bank1 NOR/SRAM2
FSMC_Bank1_NORSRAM3	FSMC Bank1 NOR/SRAM3
FSMC_Bank1_NORSRAM4	FSMC Bank1 NOR/SRAM4

Example:

```
/* Deinitialize the FSMC NOR/SRAM Memory Bank1 */
FSMC_NORSRAMDeInit(FSMC_Bank1_NORSRAM1);
```

23.2.2 FSMC_NANDDeInit

Table 694 describes the FSMC_NANDDeInit function.

Table 694. FSMC_NANDDeInit function

Function name	FSMC_NANDDeInit
Function prototype	`void FSMC_NANDDeInit(u32 FSMC_Bank)`
Behavior description	Re-initializes the FSMC NAND bank registers to their default reset values.
Input parameter	FSMC_Bank: specifies the NAND FSMC bank to be used. Refer to *FSMC_Bank* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FSMC_Bank

Specifies the NAND banks that can be used. *Table 693* shows the values assumed by this parameter.

Table 695. FSMC_Bank definition

FSMC_Bank	Description
FSMC_Bank2_NAND	FSMC Bank2 NAND
FSMC_Bank3_NAND	FSMC Bank3 NAND

Example:

```
/* Deinitialize the FSMC NAND Memory Bank3 */
FSMC_NANDDeInit(FSMC_Bank3_NAND);
```

23.2.3 FSMC_PCCARDDeInit

Table 696 describes the FSMC_PCCARDDeInit function.

Table 696. FSMC_PCCARDDeInit function

Function name	FSMC_PCCARDDeInit
Function prototype	`void FSMC_PCCARDDeInit(void)`
Behavior description	Re-initializes the FSMC PCCARD bank registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Deinitialize the FSMC PCCARD Memory Bank */
FSMC_PCCARDDeInit();
```

23.2.4 FSMC_NORSRAMInit

Table 697 describes the FSMC_NORSRAMInit function.

Table 697. FSMC_NORSRAMInit function

Function name	FSMC_NORSRAMInit
Function prototype	`void FSMC_NORSRAMInit(FSMC_NORSRAMInitTypeDef* FSMC_NORSRAMInitStruct)`
Behavior description	Initializes the FSMC NOR/SRAM banks according to the specified parameters in FSMC_NORSRAMInitStruct.
Input parameter1	FSMC_NORSRAMInitStruct: pointer to an FSMC_NORSRAMInitTypeDef structure that contains the configuration information for the specified FSMC NOR/SRAM banks. Refer to *FSMC_NORSRAMInitTypeDef* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The FSMC_NORSRAMTimingInitTypeDef and FSMC_NORSRAMInitTypeDef structures are defined in the *stm32f10x_fsmc.h* file:

FSMC_NORSRAMTimingInitTypeDef

```
typedef struct
{
  u32 FSMC_AddressSetupTime;
  u32 FSMC_AddressHoldTime;
  u32 FSMC_DataSetupTime;
  u32 FSMC_BusTurnaroundDuration;
  u32 FSMC_CLKDivision;
  u32 FSMC_DataLatency;
  u32 FSMC_AccessMode;
}FSMC_NORSRAMTimingInitTypeDef;
```

FSMC_AddressSetupTime

Defines the number of HCLK cycles to configure the duration of the address setup time. This parameter can be a value between 0 and 0xF. It is not used with synchronous NOR Flash memories.

FSMC_AddressHoldTime

Defines the number of HCLK cycles to configure the duration of the address hold time. This parameter can be a value between 0 and 0xF. It is not used with synchronous NOR Flash memories.

FSMC_DataSetupTime

Defines the number of HCLK cycles to configure the duration of the data setup time. This parameter can be a value between 0 and 0xFF. It is used for SRAMs, ROMs and asynchronous multiplexed NOR Flash memories.

FSMC_BusTurnaroundDuration

Defines the number of HCLK cycles to configure the duration of the bus turnaround. This parameter can be a value between 0 and 0xF. It is only used for multiplexed NOR Flash memories.

FSMC_CLKDivision

Defines the number of HCLK cycles to configure the duration of the data setup time. This parameter can be a value between 0 and 0xF. This parameter is not used for asynchronous NOR Flash, SRAM or ROM accesses.

FSMC_DataLatency

Defines the number of memory clock cycles to issue to the memory before getting the first data. The value of this parameter depends on the memory type as shown below:

- It must be set to 0 in case of a CRAM
- It is don't care in asynchronous NOR, SRAM or ROM accesses
- It may assume a value between 0 and 0xF in NOR Flash memories with synchronous burst mode enable

FSMC_AccessMode

Specifies the asynchronous access mode. *Table 698* gives the values assumed by this parameter.

Table 698. FSMC_AccessMode definition

FSMC_AccessMode	Description
FSMC_AccessMode_A	Access mode A.
FSMC_AccessMode_B	Access mode B.
FSMC_AccessMode_C	Access mode C.
FSMC_AccessMode_D	Access mode D.

FSMC_NORSRAMInitTypeDef

```
typedef struct
{
  u32 FSMC_Bank;
  u32 FSMC_DataAddressMux;
  u32 FSMC_MemoryType;
  u32 FSMC_MemoryDataWidth;
  u32 FSMC_BurstAccessMode;
  u32 FSMC_WaitSignalPolarity;
  u32 FSMC_WrapMode;
  u32 FSMC_WaitSignalActive;
  u32 FSMC_WriteOperation;
  u32 FSMC_WaitSignal;
  u32 FSMC_ExtendedMode;
  u32 FSMC_WriteBurst;
  /* Timing Parameters for write and read access if the
ExtendedMode is not used*/
  FSMC_NORSRAMTimingInitTypeDef* FSMC_ReadWriteTimingStruct;
  /* Timing Parameters for write access if the  ExtendedMode is
used*/
  FSMC_NORSRAMTimingInitTypeDef* FSMC_WriteTimingStruct;
}FSMC_NORSRAMInitTypeDef;
```

FSMC_Bank

Specifies the memory bank that will be used. *Table 699* gives the values assumed by this parameter.

Table 699. FSMC_Bank definition

FSMC_Bank	Description
FSMC_Bank1_NORSRAM1	BANK1 NOR SRAM1
FSMC_Bank1_NORSRAM2	BANK1 NOR SRAM2
FSMC_Bank1_NORSRAM3	BANK1 NOR SRAM3
FSMC_Bank1_NORSRAM4	BANK1 NOR SRAM4

FSMC_DataAddressMux

Specifies whether the address and data values are multiplexed on the databus or not. *Table 700* gives the values assumed by this member.

Table 700. FSMC_DataAddressMux definition

FSMC_DataAddressMux	Description
FSMC_DataAddressMux_Disable	Address/Data non multiplexed
FSMC_DataAddressMux_Enable	Address/Data multiplexed on databus

FSMC_MemoryType

Specifies the type of external memory attached to the corresponding memory bank. *Table 701* gives the values assumed by this member.

Table 701. FSMC_MemoryType definition

FSMC_MemoryType	Description
FSMC_MemoryType_SRAM	SRAM and ROM memory
FSMC_MemoryType_PSRAM	PSRAM memory
FSMC_MemoryType_NOR	NOR memory

FSMC_MemoryDataWidth

Specifies the external memory device width. *Table 701* gives the values assumed by this member.

Table 702. FSMC_MemoryDataWidth definition

FSMC_MemoryDataWidth	Description
FSMC_MemoryDataWidth_8b	8-bit external memory device data width.
FSMC_MemoryDataWidth_16b	16-bit external memory device data width.

FSMC_BurstAccessMode

Enables or disables the burst access mode for Flash memory, valid only with synchronous burst Flash memories. *Table 701* gives the values assumed by this member.

Table 703. FSMC_BurstAccessMode definition

FSMC_BurstAccessMode	Description
FSMC_BurstAccessMode_Disable	Disables the burst access mode
FSMC_BurstAccessMode_Enable	Enables the burst access mode

FSMC_WaitSignalPolarity

Specifies the wait signal polarity, valid only when accessing the Flash memory in burst mode. *Table 704* gives the values assumed by this member.

Table 704. FSMC_WaitSignalPolarity definition

FSMC_WaitSignalPolarity	Description
FSMC_WaitSignalPolarity_Low	Wait signal active low.
FSMC_WaitSignalPolarity_High	Wait signal active high.

FSMC_WrapMode

Enables or disables the Wrapped burst access mode for Flash memory, valid only when accessing Flash memories in burst mode. *Table 705* gives the values assumed by this member.

Table 705. FSMC_WrapMode definition

FSMC_WrapMode	Description
FSMC_WrapMode_Disable	Direct wrapped burst is disabled
FSMC_WrapMode_Enable	Direct wrapped burst is enabled

FSMC_WaitTiming

Specifies the wait signal polarity, valid only when accessing Flash memories in burst mode. *Table 701* gives the values assumed by this member.

Table 706. FSMC_WaitTiming definition

FSMC_WaitTiming	Description
FSMC_WaitTiming_BeforeWaitState	WAITn signal is active one data cycle before the wait state.
FSMC_WaitTiming_DuringWaitState	WAITn signal is active during the wait state.

FSMC_WriteOperation

Enables or disables the write operation to be accepted by the FSMC. *Table 701* gives the values assumed by this member.

Table 707. FSMC_WriteOperation definition

FSMC_WriteOperation	Description
FSMC_WriteOperation_Disable	Write operations are disabled in the bank by the FSMC.
FSMC_WriteOperation_Enable	Write operations are enabled in the bank by the FSMC

FSMC_WaitSignal

Enables or disables the wait-state insertion via WAITn signal, valid for Flash memory access in burst mode. *Table 708* gives the values assumed by this member.

Table 708. FSMC_WaitSignal defintiion

FSMC_WaitSignal	Description
FSMC_WaitSignal_Disable	NWAIT signal is disabled.
FSMC_WaitSignal_Enable	NWAIT signal is enabled.

FSMC_ExtendedMode

Enables or disables the extended mode. *Table 701* gives the values assumed by this member.

Table 709. FSMC_ExtendedMode defintiion

FSMC_ExtendedMode	Description
FSMC_ExtendedMode_Disable	Extended mode is disabled
FSMC_ExtendedMode_Enable	Extended mode is enabled

FSMC_WriteBurst

Enables or disables the write burst operation. *Table 701* gives the values assumed by this member.

Table 710. FSMC_WriteBurst definition

FSMC_WriteBurst	Description
FSMC_WriteBurst_Disable	Write operations are always performed in asynchronous mode
FSMC_WriteBurst_Enable	Write operations are always performed in synchronous mode

Example:

```
/* Initialize the FSMC NOR memory according to the
FSMC_NORSRAMInitStructure members */
  FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
FSMC_NORSRAMTimingInitTypeDef* FSMC_NORSRAMTimingStructure;

FSMC_NORSRAMTimingStructure.FSMC_AddressSetupTime = 0x2;
FSMC_NORSRAMTimingStructure.FSMC_AddressHoldTime = 0x2;
FSMC_NORSRAMTimingStructure.FSMC_DataSetupTime = 0x2;
FSMC_NORSRAMTimingStructure.FSMC_BusTurnaroundDuration = 0x0;
FSMC_NORSRAMTimingStructure.FSMC_CLKDivision = 0x0;
FSMC_NORSRAMTimingStructure.FSMC_DataLatency = 0x0;
FSMC_NORSRAMTimingStructure.FSMC_AccessMode = FSMC_AccessMode_A;

FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM1;
FSMC_NORSRAMInitStructure.FSMC_DataAddressMux =
FSMC_DataAddressMux_Disable;
FSMC_NORSRAMInitStructure.FSMC_MemoryType = FSMC_MemoryType_NOR;
FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth =
FSMC_MemoryDataWidth_16b;
```

```
FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode =
FSMC_BurstAccessMode_Disable;
FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity =
FSMC_WaitSignalPolarity_Low;
FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
FSMC_NORSRAMInitStructure.FSMC_WaitTiming =
FSMC_WaitSignalActive_BeforeWaitState;
FSMC_NORSRAMInitStructure.FSMC_WriteOperation =
FSMC_WriteOperation_Enable;
FSMC_NORSRAMInitStructure.FSMC_WaitSignal =
FSMC_WaitSignal_Disable;
FSMC_NORSRAMInitStructure.FSMC_ExtendedMode =
FSMC_ExtendedMode_Disable ;
FSMC_NORSRAMInitStructure.FSMC_WriteBurst =
FSMC_WriteBurst_Disable;
FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStructure =
&FSMC_NORSRAMTimingStructure;

FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);
```

23.2.5 FSMC_NANDInit

Table 711 describes the FSMC_NANDInit function.

Table 711. FSMC_NANDInit function

Function name	FSMC_NANDInit
Function prototype	`void FSMC_NANDInit(FSMC_NAND_PCCARDInitTypeDef* FSMC_NANDInitStruct)`
Behavior description	Initializes the FSMC NAND banks according to the parameters specified in FSMC_NANDInitStruct.
Input parameter	FSMC_NANDInitStruct: pointer to an FSMC_NANDInitTypeDef structure that contains the configuration information for the specified FSMC NAND banks. Refer to *FSMC_NANDInitTypeDef* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The FSMC_NAND_PCCARDTimingInitTypeDef and FSMC_NANDInitTypeDef are defined in the *stm32f10x_fsmc.h* file:

FSMC_NAND_PCCARDTimingInitTypeDef

```
typedef struct
{
  u32 FSMC_SetupTime;
  u32 FSMC_WaitSetupTime;
  u32 FSMC_HoldSetupTime;
  u32 FSMC_HiZSetupTime;
}FSMC_NAND_PCCARDTimingInitTypeDef;
```

FSMC_SetupTime

Defines the number of HCLK cycles to setup address before the command assertion for NAND-Flash read or write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_WaitSetupTime

Defines the minimum number of HCLK cycles to assert the command for NAND-Flash read or write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_HoldSetupTime

Defines the number of HCLK clock cycles to hold address (and data for write access) after the command deassertion for NAND-Flash read or write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_HiZSetupTime

Defines the number of HCLK clock cycles during which the databus is kept in HiZ after the start of a NAND-Flash write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_NANDInitTypeDef

The FSMC_NANDInitTypeDef structure is defined in the *stm32f10x_fsmc.h* file:

```
/* FSMC NAND Init structure definition */
typedef struct
{
  u32 FSMC_Bank;
  u32 FSMC_Waitfeature;
  u32 FSMC_MemoryDataWidth;
  u32 FSMC_ECC;
  u32 FSMC_ECCPageSize;
  u32 FSMC_AddressLowMapping;
  u32 FSMC_TCLRSetupTime;
  u32 FSMC_TARSetupTime;
  /* FSMC Common Space Timing */
  FSMC_NAND_PCCARDTimingInitTypeDef*  FSMC_CommonSpaceTimingStruct;
  /* FSMC Attribute Space Timing */
  FSMC_NAND_PCCARDTimingInitTypeDef*
FSMC_AttributeSpaceTimingStruct;
}FSMC_NANDInitTypeDef;
```

FSMC_Bank

Specifies the memory bank that will be used. *Table 712* gives the values assumed by this member.

Table 712. FSMC_Bank definition

FSMC_Bank	Description
FSMC_Bank2_NAND	BANK2 NAND
FSMC_Bank3_NAND	BANK3 NAND

FSMC_Waitfeature

Enables or disables the Wait feature for the NAND Memory Bank. *Table 712* gives the values assumed by this member.

Table 713. FSMC_Waitfeature definition

FSMC_Waitfeature	Description
FSMC_Waitfeature_Disable	Disables the Wait feature for the NAND memory bank.
FSMC_Waitfeature_Enable	Enables the Wait feature for the NAND memory bank.

FSMC_MemoryDataWidth

Specifies the external memory device width. *Table 712* gives the values assumed by this member.

Table 714. FSMC_MemoryDataWidth definition

FSMC_MemoryDataWidth	Description
FSMC_MemoryDataWidth_8b	8-bit external memory device data width.
FSMC_MemoryDataWidth_16b	16-bit external memory device data width.

FSMC_ECC

Enables or disables the ECC computation. *Table 712* gives the values assumed by this member.

Table 715. FSMC_ECC definition

FSMC_ECC	Description
FSMC_ECC_Disable	Disables the ECC logic.
FSMC_ECC_Enable	Enables the ECC logic.

FSMC_ECCPageSize

Defines the page size for the extended ECC. *Table 712* gives the values assumed by this member.

Table 716. FSMC_ECCPageSize definition

FSMC_ECCPageSize	Description
FSMC_ECCPageSize_256Bytes	256 byte ECC page size
FSMC_ECCPageSize_512Bytes	512 byte ECC page size
FSMC_ECCPageSize_1024Bytes	1024 byte ECC page size
FSMC_ECCPageSize_2048Bytes	2048 byte ECC page size
FSMC_ECCPageSize_4096Bytes	4096 byte ECC page size
FSMC_ECCPageSize_8192Bytes	8192 byte ECC page size

FSMC_AddressLowMapping

Defined which NAND-Flash controller address bits are delivered on A[24:16] signals. *Table 712* gives the values assumed by this member.

Table 717. FSMC_AddressLowMapping definition

FSMC_AddressLowMapping	Description
FSMC_AddressLowMapping_Direct	Direct mapping: A[24:16] delivers the [24:16] AHB address lines
FSMC_AddressLowMapping_InDirect	Low-address bit mapping: A[24:16] delivers the [8:0] AHB address lines

FSMC_TCLRSetupTime

Defines the number of HCLK cycles to configure the delay between CLE low and RE low. This parameter can assume a value between 0 and 0xFF.

FSMC_TARSetupTime

Defines the number of HCLK cycles to configure the delay between ALE low and RE low. This parameter can assume a value between 0 and 0xFF.

Example:

```
/* Initialize the FSMC NAND memory Bank2 according to the
FSMC_NANDInitStructure members */
  FSMC_NANDSRAMInitTypeDef  FSMC_NANDSRAMInitStructure;
FSMC_NAND_PCCARDTimingInitTypeDef FSMC_CommonSpaceTimingStructure;
FSMC_NAND_PCCARDTimingInitTypeDef
FSMC_AttributeSpaceTimingStructure;
FSMC_CommonSpaceTimingStructure.FSMC_SetupTime = 0x4;
FSMC_CommonSpaceTimingStructure.FSMC_WaitSetupTime = 0x0;
FSMC_CommonSpaceTimingStructure.FSMC_HoldSetupTime = 0x7;
FSMC_CommonSpaceTimingStructure.FSMC_HiZSetupTime = 0x0;
FSMC_AttributeSpaceTimingStructure.FSMC_SetupTime = 0x4;
FSMC_AttributeSpaceTimingStructure.FSMC_WaitSetupTime = 0x0;
FSMC_AttributeSpaceTimingStructure.FSMC_HoldSetupTime = 0x7;
FSMC_AttributeSpaceTimingStructure.FSMC_HiZSetupTime = 0x0;

FSMC_NANDSRAMInitStructure.FSMC_Bank = FSMC_Bank2_NAND;
FSMC_NANDSRAMInitStructure.FSMC_Waitfeature =
FSMC_Waitfeature_Enable;
FSMC_NANDSRAMInitStructure.FSMC_MemoryDataWidth =
FSMC_MemoryDataWidth_8b;
FSMC_NANDSRAMInitStructure.FSMC_ECC = FSMC_ECC_Enable;
FSMC_NANDSRAMInitStructure.FSMC_ECCPageSize =
FSMC_ECCPageSize_512Bytes;
FSMC_NANDSRAMInitStructure.FSMC_AddressLowMapping =
FSMC_AddressLowMapping_Direct;
FSMC_NANDSRAMInitStructure.FSMC_TCLRSetupTime = 0x1;
FSMC_NANDSRAMInitStructure.FSMC_TARSetupTime = 0x1;

FSMC_NANDSRAMInitStructure.FSMC_CommonSpaceTimingStructure =
&FSMC_CommonSpaceTimingStructure;

FSMC_NANDSRAMInitStructure.FSMC_AttributeSpaceTimingStructure =
&FSMC_AttributeSpaceTimingStructure;

FSMC_NANDInit(&FSMC_NANDSRAMInitStructure);
```

23.2.6 FSMC_PCCARDInit

Table 692 describes the FSMC_PCCARDInit function.

Table 718. FSMC_PCCARDInit function

Function name	FSMC_PCCARDInit
Function prototype	`void FSMC_PCCARDInit(FSMC_NAND_PCCARDInitTypeDef* FSMC_PCCARDInitStruct)`
Behavior description	Initializes the FSMC PC-CARD bank according to the parameters specified in FSMC_PCCARDInitStruct.
Input parameter	FSMC_PCCARDInitStruct: pointer to an FSMC_PCCARDInitTypeDef structure that contains the configuration information for the FSMC PC-CARD bank. Refer to *FSMC_PCCARDInitTypeDef* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FSMC_NAND_PCCARDTimingInitTypeDef and FSMC_PCCARDInitTypeDef are defined in the *stm32f10x_fsmc.h* file:

FSMC_NAND_PCCARDTimingInitTypeDef

```
typedef struct
{
  u32 FSMC_SetupTime;
  u32 FSMC_WaitSetupTime;
  u32 FSMC_HoldSetupTime;
  u32 FSMC_HiZSetupTime;
}FSMC_NAND_PCCARDTimingInitTypeDef;
```

FSMC_SetupTime

Defines the number of HCLK cycles to setup the address before the command assertion for PCCARD read or write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_WaitSetupTime

Defines the minimum number of HCLK cycles to assert the command for PCCARD read or write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_HoldSetupTime

Defines the number of HCLK clock cycles to hold the address (and data for write access) after the command deassertion for PCCARD read or write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_HiZSetupTime

Defines the number of HCLK clock cycles during which the databus is kept in HiZ after the start of a PCCARD write access to common/Attribute or I/O memory space (depending on the memory space timing to be configured). This parameter can assume a value between 0 and 0xFF.

FSMC_PCCARDInitTypeDef

```
/* FSMC PCCARD Init structure definition */
typedef struct
{
  u32 FSMC_Waitfeature;
  u32 FSMC_AddressLowMapping;
  u32 FSMC_TCLRSetupTime;
  u32 FSMC_TARSetupTime;
  /* FSMC Common Space Timing */
  FSMC_NAND_PCCARDTimingInitTypeDef*  FSMC_CommonSpaceTimingStruct;
  /* FSMC Attribute Space Timing */
  FSMC_NAND_PCCARDTimingInitTypeDef*
FSMC_AttributeSpaceTimingStruct;
  /* FSMC IO Space Timing */
  FSMC_NAND_PCCARDTimingInitTypeDef*  FSMC_IOSpaceTimingStruct;
}FSMC_PCCARDInitTypeDef;
```

FSMC_Waitfeature

Enables or disables the Wait feature for the NAND memory bank. *Table 719* gives the list of values assumed by this member.

Table 719. FSMC_Waitfeature definition

FSMC_Waitfeature	Description
FSMC_Waitfeature_Disable	Disables the Wait feature for the NAND memory bank.
FSMC_Waitfeature_Enable	Enables the Wait feature for the NAND memory bank.

FSMC_AddressLowMapping

Defined which NAND-Flash controller address bits are delivered on the A[24:16] signals. *Table 719* gives the list of values assumed by this member.

Table 720. FSMC_AddressLowMapping definition

FSMC_AddressLowMapping	Description
FSMC_AddressLowMapping_Direct	Direct mapping: A[24:16] delivers the [24:16] AHB address lines
FSMC_AddressLowMapping_InDirect	Low address bit mapping: A[24:16] delivers the [8:0] AHB address lines

FSMC_TCLRSetupTime

Defines the number of HCLK cycles to configure the delay between CLE low and RE low. This parameter can assume a value between 0 and 0xFF.

FSMC_TARSetupTime

Defines the number of HCLK cycles to configure the delay between ALE low and RE low. This parameter can assume a value between 0 and 0xFF.

Example:

```
/* Initialize the FSMC PC-CARD memory Bank4 according to the
FSMC_PCCARDInitStructure members */
  FSMC_PCCARDInitTypeDef  FSMC_PCCARDInitStructure;
FSMC_NAND_PCCARDTimingInitTypeDef FSMC_CommonSpaceTimingStructure;
FSMC_NAND_PCCARDTimingInitTypeDef
FSMC_AttributeSpaceTimingStructure;
FSMC_NAND_PCCARDTimingInitTypeDef FSMC_IOSpaceTimingStructure;

FSMC_CommonSpaceTimingStructure.FSMC_SetupTime = 0x4;
FSMC_CommonSpaceTimingStructure.FSMC_WaitSetupTime = 0x0;
FSMC_CommonSpaceTimingStructure.FSMC_HoldSetupTime = 0x7;
FSMC_CommonSpaceTimingStructure.FSMC_HiZSetupTime = 0x0;

FSMC_AttributeSpaceTimingStructure.FSMC_SetupTime = 0x4;
FSMC_AttributeSpaceTimingStructure.FSMC_WaitSetupTime = 0x0;
FSMC_AttributeSpaceTimingStructure.FSMC_HoldSetupTime = 0x7;
FSMC_AttributeSpaceTimingStructure.FSMC_HiZSetupTime = 0x0;

FSMC_IOSpaceTimingStructure.FSMC_SetupTime = 0x4;
FSMC_IOSpaceTimingStructure.FSMC_WaitSetupTime = 0x0;
FSMC_IOSpaceTimingStructure.FSMC_HoldSetupTime = 0x7;
FSMC_IOSpaceTimingStructure.FSMC_HiZSetupTime = 0x0;

FSMC_PCCARDInitStructure.FSMC_Waitfeature =
FSMC_Waitfeature_Enable;
FSMC_PCCARDInitStructure.FSMC_AddressLowMapping =
FSMC_AddressLowMapping_Direct;
FSMC_PCCARDInitStructure.FSMC_TCLRSetupTime = 0x1;
FSMC_PCCARDInitStructure.FSMC_TARSetupTime = 0x1;
FSMC_PCCARDInit(&FSMC_PCCARDInitStructure);
```

23.2.7 FSMC_NORSRAMStructInit

Table 692 describes the FSMC_NORSRAMStructInit function.

Table 721. FSMC_NORSRAMStructInit function

Function name	FSMC_NORSRAMStructInit
Function prototype	`void FSMC_NORSRAMStructInit(FSMC_NORSRAMInitTypeDef* FSMC_NORSRAMInitStruct)`
Behavior description	Fills each FSMC_NORSRAMInitStruct member with its default value.
Input parameter	FSMC_NORSRAMInitStruct: pointer to an FSMC_NORSRAMInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 722 gives the default values of the FSMC_NORSRAMInitStruct members.

Table 722. FSMC_NORSRAMInitStruct member definition

Member	Default value
FSMC_Bank	FSMC_Bank1_NORSRAM1
FSMC_DataAddressMux	FSMC_DataAddressMux_Enable
FSMC_MemoryType	FSMC_MemoryType_SRAM
FSMC_MemoryDataWidth	FSMC_MemoryDataWidth_8b
FSMC_BurstAccessMode	FSMC_BurstAccessMode_Disable
FSMC_WaitSignalPolarity	FSMC_WaitSignalPolarity_Low
FSMC_WrapMode	FSMC_WrapMode_Disable
FSMC_WaitSignalActive	FSMC_WaitSignalActive_BeforeWaitState
FSMC_WriteOperation	FSMC_WriteOperation_Enable
FSMC_WaitSignal	FSMC_WaitSignal_Enable
FSMC_ExtendedMode	FSMC_ExtendedMode_Disable
FSMC_WriteBurst	FSMC_WriteBurst_Disable
FSMC_ReadWriteTimingStruct->FSMC_AddressSetupTime	0xF
FSMC_ReadWriteTimingStruct->FSMC_AddressHoldTime	0xF
FSMC_ReadWriteTimingStruct->FSMC_DataSetupTime	0xFF
FSMC_ReadWriteTimingStruct->FSMC_BusTurnaroundDuration	0xF
FSMC_ReadWriteTimingStruct->FSMC_CLKDivision	0xF
FSMC_ReadWriteTimingStruct->FSMC_DataLatency	0xFF
FSMC_ReadWriteTimingStruct->FSMC_AccessMode	FSMC_AccessMode_A
FSMC_WriteTimingStruct->FSMC_AddressSetupTime	0xF

Table 722. FSMC_NORSRAMInitStruct member definition (continued)

Member	Default value
FSMC_WriteTimingStruct->FSMC_AddressHoldTime	0xF
FSMC_WriteTimingStruct->FSMC_DataSetupTime	0xFF
FSMC_WriteTimingStruct->FSMC_BusTurnaroundDuration	0xF
FSMC_WriteTimingStruct->FSMC_CLKDivision	0xF
FSMC_WriteTimingStruct->FSMC_DataLatency	0xFF
FSMC_WriteTimingStruct->FSMC_AccessMode	FSMC_AccessMode_A

Example:

```
/* Initialize a FSMC_NORSRAMInitTypeDef structure. */
FSMC_NORSRAMInitTypeDef FSMC_NORSRAMInitStructure;
FSMC_NORSRAMStructInit(&FSMC_NORSRAMInitStructure);
```

23.2.8 FSMC_NANDStructInit

Table 692 describes the FSMC_NANDStructInit function.

Table 723. FSMC_NANDStructInit function

Function name	FSMC_NANDStructInit
Function prototype	`void FSMC_NANDStructInit(FSMC_NANDInitTypeDef* FSMC_NANDInitStruct)`
Behavior description	Fills each FSMC_NANDInitStruct member with its default value.
Input parameter	FSMC_NANDInitStruct: pointer to an FSMC_NANDInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 722 gives the default values of the FSMC_NANDInitStruct members.

Table 724. FSMC_NANDInitStruct member definitions

Member	Default value
FSMC_Bank	FSMC_Bank2_NAND
FSMC_Waitfeature	FSMC_Waitfeature_Disable
FSMC_MemoryDataWidth	FSMC_MemoryDataWidth_8b
FSMC_ECC	FSMC_ECC_Disable
FSMC_ECCPageSize	FSMC_ECCPageSize_256Bytes
FSMC_AddressLowMapping	FSMC_AddressLowMapping_Direct
FSMC_TCLRSetupTime	0x0
FSMC_TARSetupTime	0x0
FSMC_CommonSpaceTimingStruct->FSMC_CommonSetupTime	0xFC

Table 724. FSMC_NANDInitStruct member definitions (continued)

Member	Default value
FSMC_CommonSpaceTimingStruct->FSMC_CommonWaitSetupTime	0xFC
FSMC_CommonSpaceTimingStruct->FSMC_CommonHoldSetupTime	0xFC
FSMC_CommonSpaceTimingStruct->FSMC_CommonHiZSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeWaitSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeHoldSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeHiZSetupTime	0xFC

Example:

```
/* Initialize a FSMC_NANDInitTypeDef structure. */
FSMC_NANDInitTypeDef FSMC_NANDInitStructure;
FSMC_NAND_PCCARDTimingInitTypeDef FSMC_CommonSpaceTimingStructure;
FSMC_NAND_PCCARDTimingInitTypeDef
FSMC_AttributeSpaceTimingStructure;
FSMC_NANDStructInit(&FSMC_NANDInitStructure);
```

23.2.9 FSMC_PCCARDStructInit

Table 692 describes the FSMC_PCCARDStructInit function.

Table 725. FSMC_PCCARDStructInit function

Function name	FSMC_PCCARDStructInit
Function prototype	`void FSMC_PCCARDStructInit(FSMC_NAND_PCCARDInitTypeDef* FSMC_PCCARDInitStruct)`
Behavior description	Fills each FSMC_PCCARDInitStruct member with its default value.
Input parameter	FSMC_PCCARDInitStruct: pointer to an FSMC_NAND_PCCARDInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 722 gives the default values of the FSMC_PCCARDInitStruct members.

Table 726. FSMC_PCCARDInitStruct member definition

Member	Default value
FSMC_Waitfeature	FSMC_Waitfeature_Disable
FSMC_AddressLowMapping	FSMC_AddressLowMapping_Direct
FSMC_TCLRSetupTime	0x0
FSMC_TARSetupTime	0x0
FSMC_CommonSpaceTimingStruct->FSMC_CommonSetupTime	0xFC
FSMC_CommonSpaceTimingStruct->FSMC_CommonWaitSetupTime	0xFC
FSMC_CommonSpaceTimingStruct->FSMC_CommonHoldSetupTime	0xFC
FSMC_CommonSpaceTimingStruct->FSMC_CommonHiZSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeWaitSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeHoldSetupTime	0xFC
FSMC_AttributeSpaceTimingStruct->FSMC_AttributeHiZSetupTime	0xFC
FSMC_IOSpaceTimingStruct->FSMC_AttributeSetupTime	0xFC
FSMC_IOSpaceTimingStruct->FSMC_AttributeWaitSetupTime	0xFC
FSMC_IOSpaceTimingStruct->FSMC_AttributeHoldSetupTime	0xFC
FSMC_IOSpaceTimingStruct->FSMC_AttributeHiZSetupTime	0xFC

Example:

```
/* Initialize a FSMC_PCCARDInitTypeDef structure. */
FSMC_PCCARDInitTypeDef FSMC_PCCARDInitStructure;
FSMC_NAND_PCCARDTimingInitTypeDef FSMC_CommonSpaceTimingStructure;
FSMC_NAND_PCCARDTimingInitTypeDef FSMC_AttributeSpaceTimingStructure;
FSMC_NAND_PCCARDTimingInitTypeDef FSMC_IOSpaceTimingStructure;
FSMC_PCCARDStructInit(&FSMC_PCCARDInitStructure);
```

23.2.10 FSMC_NORSRAMCmd

Table 692 describes the FSMC_NORSRAMCmd function.

Table 727. FSMC_NORSRAMCmd function

Function name	FSMC_NORSRAMCmd
Function prototype	`void FSMC_NORSRAMCmd(u32 FSMC_Bank, FunctionalState NewState)`
Behavior description	Enables or disables the NOR/SRAM memory bankx.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. Refer to *FSMC_Bank* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the FSMC_Bank. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the FSMC Bank 2 for NOR/SRAM Memory use */
FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM2, ENABLE);
```

23.2.11 FSMC_NANDCmd

Table 692 describes the FSMC_NANDCmd function.

Table 728. FSMC_NANDCmd function

Function name	FSMC_NANDCmd
Function prototype	`void FSMC_NANDCmd(u32 FSMC_Bank, FunctionalState NewState)`
Behavior description	Enables or disables the NAND memory bank.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. Refer to *FSMC_Bank* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the FSMC_Bank. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the FSMC Bank 2 or NAND Memory */
FSMC_NANDCmd(FSMC_Bank2_NAND, ENABLE);
```

23.2.12 FSMC_PCCARDCmd

Table 692 describes the FSMC_PCCARDCmd function.

Table 729. FSMC_PCCARDCmd function

Function name	FSMC_PCCARDCmd
Function prototype	`void FSMC_PCCARDCmd(FunctionalState NewState)`
Behavior description	Enables or disables the PC-CARD memory bank.
Input parameter	NewState: new state of the PCCARD memory bank. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the FSMC Bank 3 for PC-CARD Memory */
FSMC_PCCARDCmd(ENABLE);
```

23.2.13 FSMC_PCCARDCmd

Table 692 describes the FSMC_PCCARDCmd function.

Table 730. FSMC_PCCARDCmd function

Function name	FSMC_PCCARDCmd
Function prototype	`void FSMC_PCCARDCmd(FunctionalState NewState)`
Behavior description	Enables or disables the PC-CARD memory bank.
Input parameter	NewState: new state of the PCCARD memory bank. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the FSMC Bank 3 for PC-CARD Memory */
FSMC_PCCARDCmd(ENABLE);
```

23.2.14 FSMC_NANDECCCmd

Table 692 describes the FSMC_NANDECCCmd function.

Table 731. FSMC_NANDECCCmd function

Function name	FSMC_NANDECCCmd
Function prototype	`void FSMC_NANDECCCmd(u32 FSMC_Bank, FunctionalState NewState)`
Behavior description	Enables or disables the FSMC NAND ECC feature.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. Refer to *FSMC_Bank* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the FSMC NAND ECC feature. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables FSMC NAND Bank2 ECC functionnality */
FSMC_NANDECCCmd(FSMC_Bank2_NAND, ENABLE);
```

23.2.15 FSMC_ITConfig

Table 692 describes the FSMC_ITConfig function.

Table 732. FSMC_ITConfig function

Function name	FSMC_ITConfig
Function prototype	`void FSMC_ITConfig(u32 FSMC_Bank, u32 FSMC_IT, FunctionalState NewState)`
Behavior description	Enables or disables the specified FSMC interrupts.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. This parameter can assume one of the following values: – FSMC_Bank2_NAND: FSMC Bank2 NAND – FSMC_Bank3_NAND: FSMC Bank3 NAND – FSMC_Bank4_PCCARD: FSMC Bank4 PC memory card
Input parameter2	FSMC_IT: specifies the FSMC interrupt sources to be enabled or disabled. Refer to *FSMC_IT* for more details on the allowed values for this parameter.
Input parameter3	NewState: new state of the FSMC interrupt source. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

FSMC_IT

Table 733 shows the values that can be combined to enable or disable FSMC interrupts.

Table 733. FSMC_IT definition

FSMC_IT	Description
FSMC_IT_RisingEdge	Interrupt rising edge detection
FSMC_IT_Level	Interrupt level detection
FSMC_IT_FallingEdge	Interrupt falling edge detection

Example:

```
/* Enables the FSMC_Bank2 Rising edge detection Interrupt source */
FSMC_ITConfig(FSMC_Bank2, FSMC_IT_RisingEdge, ENABLE);
```

23.2.16 FSMC_GetFlagStatus

Table 692 describes the FSMC_GetFlagStatus function.

Table 734. FSMC_GetFlagStatus function

Function name	FSMC_GetFlagStatus
Function prototype	`FlagStatus FSMC_GetFlagStatus(u32 FSMC_Bank, u32 FSMC_FLAG)`
Behavior description	Checks whether the specified FSMC flag is set or not.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. This parameter can assume one of the following values: – FSMC_Bank2_NAND: FSMC Bank2 NAND – FSMC_Bank3_NAND: FSMC Bank3 NAND – FSMC_Bank4_PCCARD: FSMC Bank4 PC memory card
Input parameter2	FSMC_FLAG: specifies the flag to check. Refer to *ADC_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of FSMC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

FSMC_FLAG

Table 735 gives the list of the flags that can be checked.

Table 735. FSMC_FLAG definition

FSMC_FLAG	Description
FSMC_FLAG_RisingEdge	Rising Edge detection flag
FSMC_FLAG_Level	Level detection flag
FSMC_FLAG_FallingEdge	Falling Edge detection flag
FSMC_FLAG_FEMPT	FIFO empty flag

Example:

```
/* Check if the FSMC_Bank2 FIFO is empty or not */
if(FSMC_GetFlagStatus(FSMC_Bank2_NAND, FSMC_FLAG_FEMPT) == SET)
{
}
```

23.2.17 FSMC_ClearFlag

Table 692 describes the FSMC_ClearFlag function.

Table 736. FSMC_ClearFlag function

Function name	FSMC_ClearFlag
Function prototype	`void FSMC_ClearFlag(u32 FSMC_Bank, u32 FSMC_FLAG)`
Behavior description	Clears the FSMC's pending flags.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. This parameter can assume one of the following values: – FSMC_Bank2_NAND: FSMC Bank2 NAND – FSMC_Bank3_NAND: FSMC Bank3 NAND – FSMC_Bank4_PCCARD: FSMC Bank4 PC memory card
Input parameter2	FSMC_FLAG: specifies the flag to check. Refer to *ADC_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of FSMC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Clear the FSMC_Bank2 FIFO flag */
FSMC_ClearFlag(FSMC_Bank2_NAND, FSMC_FLAG_FEMPT);
```

23.2.18 FSMC_GetITStatus

Table 692 describes the FSMC_GetITStatus function.

Table 737. FSMC_GetITStatus function

Function name	FSMC_GetITStatus
Function prototype	`ITStatus FSMC_GetITStatus(u32 FSMC_Bank, u32 FSMC_IT)`
Behavior description	Checks whether the specified FSMC interrupt has occurred or not.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. This parameter can assume one of the following values: – FSMC_Bank2_NAND: FSMC Bank2 NAND – FSMC_Bank3_NAND: FSMC Bank3 NAND – FSMC_Bank4_PCCARD: FSMC Bank4 PC memory card
Input parameter2	FSMC_IT: specifies the FSMC interrupt source to check.Refer to *FSMC_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of FSMC_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the FSMC_Bank2 interrupt Rising edge detection */
FSMC_GetITStatus(FSMC_Bank2_NAND, FSMC_IT_RisingEdge);
```

23.2.19 FSMC_ClearITPendingBit

Table 692 describes the FSMC_ClearITPendingBit function.

Table 738. FSMC_ClearITPendingBit function

Function name	FSMC_ClearITPendingBit
Function prototype	`void FSMC_ClearITPendingBit(u32 FSMC_Bank, u32 FSMC_IT)`
Behavior description	Clears the FSMC's interrupt pending bits.
Input parameter1	FSMC_Bank: specifies the FSMC bank to be used. This parameter can assume one of the following values: – FSMC_Bank2_NAND: FSMC Bank2 NAND – FSMC_Bank3_NAND: FSMC Bank3 NAND – FSMC_Bank4_PCCARD: FSMC Bank4 PC memory card
Input parameter2	FSMC_IT: specifies the FSMC interrupt source to check. Refer to *FSMC_IT* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of FSMC_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Clear the FSMC_Bank2 interrupt Rising edge detection */
FSMC_ClearITPendingBit(FSMC_Bank2_NAND, FSMC_IT_RisingEdge);
```

24 SDIO interface (SDIO)

The SD/SDIO MMC card host interface (SDIO) provides an interface between the AHB peripheral bus and MultiMediaCards (MMCs), SD memory cards, SDIO cards and CE-ATA devices.

*Section 24.1*describes the data structures used in the SDIO firmware library. *Section 24.2* presents the firmware library functions.

24.1 SDIO register structure

The *SDIO_TypeDef* SDIO register structure is defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
vu32 POWER;
vu32 CLKCR;
vu32 ARG;
vu32 CMD;
vuc32 RESPCMD;
vuc32 RESP1;
vuc32 RESP2;
vuc32 RESP3;
vuc32 RESP4;
vu32 DTIMER;
vu32 DLEN;
vu32 DCTRL;
vuc32 DCOUNT;
vuc32 STA;
vu32 ICR;
vu32 MASK;
u32 RESERVED0[2];
vuc32 FIFOCNT;
u32 RESERVED1[13];
vu32 FIFO;
} SDIO_TypeDef;
```

Table 739 gives the list of SDIO registers.

Table 739. SDIO registers

Register	Description
POWER	SDIO Power Control Register
CLKCR	SDIO Clock Control Register
ARG	SDIO Argument Register
CMD	SDIO Command Register
RESPCMD	SDIO Command Response Register
RESP1	SDIO response 1 register

Table 739. SDIO registers (continued)

Register	Description
RESP2	SDIO response 2 register
RESP3	SDIO response 3 register
RESP4	SDIO response 4 register
DTIMER	SDIO Data Timer Register
DLEN	SDIO Data Length Register
DCTRL	SDIO Data Control Register
DCOUNT	SDIO Data Counter Register
STA	SDIO Status Register
ICR	SDIO Interrupt Clear Register
MASK	SDIO Mask Register
FIFOCNT	SDIO FIFO Counter Register
FIFO	SDIO Data FIFO Register

The SDIO peripheral is declared in the same file:

```
...
#define PERIPH_BASE   0x40000000
#define APB3PERIPH_BASE       (PERIPH_BASE + 18000)
....
#define SDIO_BASE (APB3PERIPH_BASE)
....
#ifndef DEBUG
...
#define SDIO ((SDIO_TypeDef *) SDIO_BASE)
...
#else
...
#ifdef _SDIO
  EXT SDIO_TypeDef    *SDIO;
#endif /*_SDIO */
...
#endif
```

When debug mode is used, _SDIO pointer is initialized in *stm32f10x_lib.c* file :

```
...
#ifdef _SDIO
  SDIO = (SDIO_TypeDef *) SDIO_BASE;
#endif /*_SDIO */
...
```

_SDIO must be defined, in *stm32f10x_conf.h* file, to access the peripheral registers as follows:

```
...
#define _SDIO
...
```

24.2 Firmware library functions

Table 740 gives the lists of the SDIO library functions.

Table 740. SDIO firmware library functions

Function name	Description
SDIO_DeInit	Resets the SDIO peripheral registers to their default reset values.
SDIO_Init	Initializes the SDIO peripheral according to the specified parameters in the SDIO_InitStruct.
SDIO_StructInit	Fills each SDIO_InitStruct member with its default value.
SDIO_ClockCmd	Enables or disables the SDIO Clock.
SDIO_SetPowerState	Sets the power status of the controller.
SDIO_GetPowerState	Gets the power status of the controller.
SDIO_ITConfig	Enables or disables SDIO interrupts.
SDIO_DMACmd	Enables or disables SDIO DMA request.
SDIO_SendCommand	Initializes the SDIO command according to the parameters specified in SDIO_CmdInitStruct ,and sends the command.
SDIO_CmdStructInit	Fills each SDIO_CmdInitStruct member with its default value.
SDIO_GetCommandResponse	Returns command index of last command for which a response was received.
SDIO_GetResponse	Returns the response received from the card for the last command.
SDIO_DataConfig	Initializes the SDIO data path according to the parameters specified in the SDIO_DataInitStruct.
SDIO_DataStructInit	Fills each SDIO_DataInitStruct member with its default value.
SDIO_GetDataCounter	Returns the number of remaining data bytes to be transferred.
SDIO_ReadData	Reads one data word from RX FIFO.
SDIO_WriteData	Writes one data word to TX FIFO.
SDIO_GetFIFOCount	Returns the number of words left to be written to or read from FIFO.
SDIO_StartSDIOReadWait	Starts the SD I/O Read Wait operation.
SDIO_StopSDIOReadWait	Stops the SD I/O Read Wait operation.
SDIO_SetSDIOReadWaitMode	Sets one of the two options of inserting read wait interval.
SDIO_SetSDIOOperation	Enables or disables the SD I/O mode operation.
SDIO_SendSDIOSuspendCmd	Enables or disables the SD I/O mode suspend command.
SDIO_CommandCompletionCmd	Enables or disables the command completion signal.
SDIO_CEATAITCmd	Enables or disables the CE-ATA interrupt.
SDIO_SendCEATACmd	Sends CE-ATA command (CMD61).
SDIO_GetFlagStatus	Checks whether the specified SDIO flag is set or not.
SDIO_ClearFlag	Clears the SDIO's pending flags.

Table 740. SDIO firmware library functions (continued)

Function name	Description
SDIO_GetITStatus	Checks whether the specified SDIO interrupt has occurred or not.
SDIO_ClearITPendingBit	Clears the SDIO's interrupt pending bits.

24.2.1 SDIO_DeInit

Table 741 describes the SDIO_DeInit function.

Table 741. SDIO_DeInit function

Function name	SDIO_DeInit
Function prototype	`void SDIO_DeInit(void)`
Behavior description	Resets the SDIO peripheral registers to their default reset values.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Deinitialize the SDIO */
SDIO_DeInit();
```

24.2.2 SDIO_Init

Table 742 describes the SDIO_Init function.

Table 742. SDIO_Init function

Function name	SDIO_Init
Function prototype	`void SDIO_Init(SDIO_InitTypeDef* SDIO_InitStruct)`
Behavior description	Initializes the SDIO peripheral according to the parameters specified in the SDIO_InitStruct.
Input parameter	SDIO_InitStruct: pointer to an SDIO_InitTypeDef structure that contains the configuration information for the SDIO peripheral. Refer to *DAC_InitTypeDef* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SDIO_InitTypeDef

The SDIO_InitTypeDef structure is defined in the *stm32f10x_sdio.h* file:

```
typedef struct
{
  u8 SDIO_ClockDiv;
  u32 SDIO_ClockEdge;
  u32 SDIO_MCLKBypass;
  u32 SDIO_ClockPowerSave;
  u32 SDIO_BusWide;
  u32 SDIO_HardwareFlowControl;
} SDIO_InitTypeDef;
```

SDIO_ClockDiv

Specifies the clock frequency of the SDIO controller. Its value ranges be from 0x00 to 0xFF.

SDIO_ClockEdge

Specifies the clock transition on which the bit capture is made. *Table 743* shows the values this member can assume.

Table 743. SDIO_ClockEdge definition

SDIO_ClockEdge	Description
SDIO_ClockEdge_Rising	SDIO clock generated on the rising edge of master clock MCLK
SDIO_ClockEdge_Falling	SDIO clock generated on the falling edge of master clock MCLK

SDIO_MCLKBypass

Specifies whether the SDIO Clock divider bypass is enabled or disabled. *Table 744* shows the values this member can assume.

Table 744. SDIO_MCLKBypass definition

SDIO_MCLKBypass	Description
SDIO_MCLKBypass_Disable	SDIO Clock divider bypass is disabled
SDIO_MCLKBypass_Enable	SDIO Clock divider bypass is enabled

SDIO_ClockPowerSave

Specifies whether SDIO Clock output is enabled or disabled when the bus is idle. *Table 745* shows the values this member can assume.

Table 745. SDIO_ClockPowerSave definition

SDIO_ClockPowerSave	Description
SDIO_ClockPowerSave_Disable	SDIO Clock output is disabled when the bus is idle
SDIO_ClockPowerSave_Enable	SDIO Clock output is enabled when the bus is idle

SDIO_BusWide

Specifies the SDIO bus width. *Table 746* shows the values this member can assume.

Table 746. SDIO_BusWide definition

SDIO_BusWide	Description
SDIO_BusWide_1b	1-bit wide bus mode
SDIO_BusWide_4b	4-bit wide bus mode
SDIO_BusWide_8b	8-bit wide bus mode

SDIO_HardwareFlowControl

Specifies whether the SDIO hardware flow control is enabled or disabled. *Table 747* shows the values this member can assume.

Table 747. SDIO_HardwareFlowControl definition

SDIO_HardwareFlowControl	Description
SDIO_HardwareFlowControl_Disable	SDIO hardware flow control is disabled
SDIO_HardwareFlowControl_Enable	SDIO hardware flow control is enabled

Example:

```
/* Configure the SDIO peripheral */
SDIO_InitTypeDef SDIO_InitStructure;

SDIO_InitStructure.SDIO_ClockDiv = 0xB2;
SDIO_InitStructure.SDIO_ClockEdge = SDIO_ClockEdge_Rising;
SDIO_InitStructure.SDIO_MCLKBypass = SDIO_MCLKBypass_Disable;
SDIO_InitStructure.SDIO_ClockPowerSave =
SDIO_ClockPowerSave_Enable;
SDIO_InitStructure.SDIO_BusWide = SDIO_BusWide_4b;
SDIO_InitStructure.SDIO_HardwareFlowControl =
SDIO_HardwareFlowControl_Enable;
SDIO_Init(&SDIO_InitStructure);
```

24.2.3 SDIO_StructInit

Table 741 describes the SDIO_StructInit function.

Table 748. SDIO_StructInit function

Function name	SDIO_StructInit
Function prototype	`void SDIO_StructInit(SDIO_InitTypeDef* SDIO_InitStruct)`
Behavior description	Fills each SDIO_InitStruct member with its default value.
Input parameter	SDIO_InitStruct: pointer to an SDIO_InitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 749 shows the values assumed by the SDIO_InitStruct members.

Table 749. SDIO_InitStruct member definition

Member	Default value
SDIO_ClockDiv	0x00
SDIO_ClockEdge	SDIO_ClockEdge_Rising
SDIO_MCLKBypass	SDIO_MCLKBypass_Disable
SDIO_ClockPowerSave	SDIO_ClockPowerSave_Disable
SDIO_BusWide	SDIO_BusWide_1b
SDIO_HardwareFlowControl	SDIO_HardwareFlowControl_Disable

Example:

```
/* Initialize a SDIO_InitTypeDef structure. */
SDIO_InitTypeDef  SDIO_InitStructure;
SDIO_StructInit(&SDIO_InitStructure);
```

24.2.4 SDIO_ClockCmd

Table 750 describes the SDIO_ClockCmd function.

Table 750. SDIO_ClockCmd function

Function name	SDIO_ClockCmd
Function prototype	`void SDIO_ClockCmd(FunctionalState NewState)`
Behavior description	Enables or disables the SDIO Clock.
Input parameter	NewState: new state of the SDIO Clock. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable SDIO Clock*/
SDIO_ClockCmd(ENABLE);
```

24.2.5 SDIO_SetPowerState

Table 751 describes the SDIO_SetPowerState function.

Table 751. SDIO_SetPowerState function

Function name	SDIO_SetPowerState
Function prototype	`void SDIO_SetPowerState(u32 SDIO_PowerState)`
Behavior description	Sets the power status of the controller.
Input parameter	SDIO_PowerState: new power state. Refer to *SDIO_PowerState* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SDIO_PowerState

Specifies Power state to set. *Table 752* shows the values assumed by this member.

Table 752. SDIO_PowerState definition

SDIO_PowerState	Description
SDIO_PowerState_OFF	Power off: the clock to card is stopped.
SDIO_PowerState_ON	Power on: the card is clocked.

Example:

```
/* Set SDIO Power Status */
SDIO_SetPowerState(SDIO_PowerState_ON);
```

24.2.6 SDIO_GetPowerState

Table 753 describes the SDIO_GetPowerState function.

Table 753. SDIO_GetPowerState function

Function name	SDIO_GetPowerState
Function prototype	`u32 SDIO_GetPowerState(void)`
Behavior description	Gets the power status of the controller.
Input parameter	None
Output parameter	None
Return parameter	Power status of the controller.
Required preconditions	None
Called functions	None

Example:

```
/* Get SDIO Power Status */
u32 PowerState;
PowerState = SDIO_GetPowerState();
```

24.2.7 SDIO_ITConfig

Table 754 describes the SDIO_ITConfig function.

Table 754. SDIO_ITConfig function

Function name	SDIO_ITConfig
Function prototype	`void SDIO_ITConfig(u32 SDIO_IT, FunctionalState NewState)`
Behavior description	Enables or disables the SDIO interrupts.
Input parameter1	SDIO_IT: specifies the SDIO interrupt sources to be enabled or disabled. Refer to *SDIO_IT* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified SDIO interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SDIO_IT

Table 755 shows the values that can be combined to enable or disable the SDIO interrupts.

Table 755. SDIO_IT definition

SDIO_IT	Description
SDIO_IT_CCRCFAIL	Command response received (CRC check failed) interrupt mask
SDIO_IT_DCRCFAIL	Data block sent/received (CRC check failed) interrupt mask
SDIO_IT_CTIMEOUT	Command response timeout interrupt mask
SDIO_IT_DTIMEOUT	Data timeout interrupt mask
SDIO_IT_TXUNDERR	Transmit FIFO underrun error interrupt mask
SDIO_IT_RXOVERR	Received FIFO overrun error interrupt mask
SDIO_IT_CMDREND	Command response received (CRC check passed) interrupt mask
SDIO_IT_CMDSENT	Command sent (no response required) interrupt mask
SDIO_IT_DATAEND	Data end (data counter SDIDCOUNT is zero) interrupt mask
SDIO_IT_STBITERR	Start bit not detected on all data signals in wide bus mode interrupt mask
SDIO_IT_DBCKEND	Data block sent/received (CRC check passed) interrupt mask
SDIO_IT_CMDACT	Command transfer in progress interrupt mask
SDIO_IT_TXACT	Data transmit in progress interrupt mask
SDIO_IT_RXACT	Data receive in progress interrupt mask
SDIO_IT_TXFIFOBW	Transmit FIFO burst writable interrupt mask
SDIO_IT_RXFIFOBR	Receive FIFO burst readable interrupt mask.
SDIO_IT_TXFIFOF	Transmit FIFO full interrupt mask
SDIO_IT_RXFIFOF	Receive FIFO full interrupt mask.
SDIO_IT_TXFIFOE	Transmit FIFO empty interrupt mask
SDIO_IT_RXFIFOE	Receive FIFO empty interrupt mask
SDIO_IT_TXDAVL	Data available in transmit FIFO interrupt mask
SDIO_IT_RXDAVL	Data available in receive FIFO interrupt mask
SDIO_IT_SDIOIT	SDIO interrupt received interrupt mask
SDIO_IT_CEATAEND	CE-ATA command completion signal received for CMD61

Example:

```
/* Enable Receive FIFO full interrupt */
SDIO_ITConfig(SDIO_IT_RXFIFOF, ENABLE);
```

24.2.8 SDIO_DMACmd

Table 756 describes the SDIO_DMACmd function.

Table 756. SDIO_DMACmd function

Function name	SDIO_DMACmd
Function prototype	`void SDIO_DMACmd(FunctionalState NewState)`
Behavior description	Enables or disables the SDIO DMA request.
Input parameter	NewState: new state of the selected SDIO DMA request. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Enable SDIO DMA request */
SDIO_DMACmd(ENABLE);
```

24.2.9 SDIO_SendCommand

Table 757 describes the SDIO_SendCommand function.

Table 757. SDIO_SendCommand function

Function name	SDIO_SendCommand
Function prototype	`void SDIO_SendCommand(SDIO_CmdInitTypeDef *SDIO_CmdInitStruct)`
Behavior description	Initializes the SDIO command according to the parameters specified in SDIO_CmdInitStruct, and sends the command.
Input parameter	SDIO_CmdInitStruct: pointer to an SDIO_CmdInitTypeDef structure that contains the configuration information for the SDIO command. Refer to *SDIO_CmdInitTypeDef* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SDIO_CmdInitTypeDef

The SDIO_CmdInitTypeDef structure is defined in the *stm32f10x_sdio.h* file:

```
typedef struct
{
  u32 SDIO_Argument;
  u32 SDIO_CmdIndex;
  u32 SDIO_Response;
  u32 SDIO_Wait;
  u32 SDIO_CPSM;
} SDIO_CmdInitTypeDef;
```

SDIO_Argument

Specifies the SDIO command argument which is sent to a card as part of a command message. If a command contains an argument, it must be loaded into this register before writing the command to the command register.

SDIO_CmdIndex

Specifies the SDIO command index. It must be lower than 0x40.

SDIO_Response

Specifies the SDIO response type. *Table 758* gives the values assumed by this parameter.

Table 758. SDIO_Response definition

SDIO_Response	Description
SDIO_Response_No	No Response is expected
SDIO_Response_Short	Short Response is expected
SDIO_Response_Long	Long Response is expected

SDIO_Wait

Specifies whether SDIO wait-for-interrupt request is enabled or disabled.*Table 759* gives the values assumed by this parameter.

Table 759. SDIO_Wait definition

SDIO_Wait	Description
SDIO_Wait_NO	No wait is requested
SDIO_Wait_IT	SDIO wait for interrupt request is enabled
SDIO_Wait_Pend	SDIO Wait End of transfer is enabled

SDIO_CPSM

Specifies whether SDIO Command path state machine (CPSM) is enabled or disabled.*Table 760* gives the values assumed by this parameter

Table 760. SDIO_CPSM definition

SDIO_CPSM	Description
SDIO_CPSM_Enable	SDIO command path state machine (CPSM) is enabled
SDIO_CPSM_Disable	SDIO command path state machine (CPSM) is disabled

Example:

```
/* Configure the SDIO Command */
SDIO_CmdInitTypeDef SDIO_CmdInitStructure;
SDIO_CmdInitStructure.SDIO_Argument = 0x0;
SDIO_CmdInitStructure.SDIO_CmdIndex = 0x0;
SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_IT;
SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
SDIO_SendCommand(&SDIO_CmdInitStructure);
```

24.2.10 SDIO_CmdStructInit

Table 761 describes the SDIO_CmdStructInit function.

Table 761. SDIO_CmdStructInit function

Function name	SDIO_CmdStructInit
Function prototype	`void SDIO_CmdStructInit(SDIO_CmdInitTypeDef* SDIO_CmdInitStruct)`
Behavior description	Fills each SDIO_CmdInitStruct member with its default value.
Input parameter	SDIO_CmdInitStruct: pointer to an SDIO_CmdInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 762 gives the default values of the SDIO_CmdInitStruct members.

Table 762. SDIO_CmdInitStruct member definition

Member	Default value
SDIO_Argument	0x00
SDIO_CmdIndex	0x00
SDIO_RespType	SDIO_RespType_No
SDIO_Wait	SDIO_Wait_No
SDIO_CPSM	SDIO_CPSM_Disable

Example:

```
/* Initialize a SDIO_CmdInitTypeDef structure */
SDIO_CmdInitTypeDef  SDIO_CmdInitStructure;
SDIO_CmdStructInit(&SDIO_CmdInitStructure);
```

24.2.11 SDIO_GetCommandResponse

Table 763 describes the SDIO_GetCommandResponse function.

Table 763. SDIO_GetCommandResponse function

Function name	SDIO_GetCommandResponse
Function prototype	`u8 SDIO_GetCommandResponse(void)`
Behavior description	Returns command index of last command for which a response was received.
Input parameter	None
Output parameter	None
Return parameter	Returns the command index of the last command response received.
Required preconditions	None
Called Functions	None

Example:

```
/* Get the Command Response */
u8 CmdResp = 0;
CmdResp = SDIO_GetCommandResponse();
```

24.2.12 SDIO_GetResponse

Table 764 describes the SDIO_GetResponse function.

Table 764. SDIO_GetResponse function

Function name	SDIO_GetResponse
Function prototype	`u32 SDIO_GetResponse(u32 SDIO_RESP)`
Behavior description	Returns the response received from the card for the last command.
Input parameter	SDIO_RESP: specifies the SDIO response register. Refer to *SDIO_RESP* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The corresponding response register value.
Required preconditions	None
Called Functions	None

SDIO_RESP

Specifies the Response register to be read. *Table 765* gives the values assumed by this member.

Table 765. SDIO_RESP definition

SDIO_RESP	Description
SDIO_RESP1	SDIO Response register 1
SDIO_RESP2	SDIO Response register 2
SDIO_RESP3	SDIO Response register 3
SDIO_RESP4	SDIO Response register 4

Example:

```
/* Get the Data Response received */
u32 Response = 0;
Response = SDIO_GetResponse(SDIO_RESP1);
```

24.2.13 SDIO_DataConfig

Table 766 describes the SDIO_DataConfig function.

Table 766. SDIO_DataConfig function

Function name	SDIO_DataConfig
Function prototype	`void SDIO_DataConfig(SDIO_DataInitTypeDef* SDIO_DataInitStruct)`
Behavior description	Initializes the SDIO data path according to the parameters specified in SDIO_DataInitStruct.
Input parameter	SDIO_DataInitStruct: pointer to an SDIO_DataInitTypeDef structure that contains the configuration information for the SDIO command. Refer to *SDIO_DataInitTypeDef*" for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SDIO_DataInitTypeDef

The SDIO_DataInitTypeDef structure is defined in the *stm32f10x_sdio.h* file:

```
typedef struct
{
  u32 SDIO_DataTimeOut;
  u32 SDIO_DataLength;
  u32 SDIO_DataBlockSize;
  u32 SDIO_TransferDir;
  u32 SDIO_TransferMode;
  u32 SDIO_DPSM;
} SDIO_DataInitTypeDef;
```

SDIO_DataTimeOut

Specifies the data timeout period in card bus clock periods.

SDIO_DataLength

Specifies the number of data bytes to be transferred.

SDIO_DataBlockSize

Specifies the data block size for block transfer. *Table 767* gives the values assumed by this member.

Table 767. SDIO_DataBlockSize definition

SDIO_DataBlockSize	Description
SDIO_DataBlockSize_1b	Data Block length = 2^0 = 1 bytes
SDIO_DataBlockSize_2b	Data Block length = 2^1 = 2 bytes
SDIO_DataBlockSize_4b	Data Block length = 2^2 = 4 bytes
SDIO_DataBlockSize_8b	Data Block length = 2^3 = 8 bytes
SDIO_DataBlockSize_16b	Data Block length = 2^4 = 16 bytes
SDIO_DataBlockSize_32b	Data Block length = 2^5 = 32 bytes
SDIO_DataBlockSize_64b	Data Block length = 2^6 = 64 bytes
SDIO_DataBlockSize_128b	Data Block length = 2^7 = 128 bytes
SDIO_DataBlockSize_256b	Data Block length = 2^8 = 256 bytes
SDIO_DataBlockSize_512b	Data Block length = 2^9 = 512 bytes
SDIO_DataBlockSize_1024b	Data Block length = 2^{10} = 1024 bytes
SDIO_DataBlockSize_2048b	Data Block length = 2^{11} = 2048 bytes
SDIO_DataBlockSize_4096b	Data Block length = 2^{12} = 4096 bytes
SDIO_DataBlockSize_8192b	Data Block length = 2^{13} = 8192 bytes
SDIO_DataBlockSize_16384b	Data Block length = 2^{14} = 16384 bytes

SDIO_TransferDir

Specifies the data transfer direction, whether the transfer is a read or write. *Table 768* gives the values assumed by this member.

Table 768. SDIO_TransferDir definition

SDIO_TransferDir	Description
SDIO_TransferDir_ToCard	From controller to card
SDIO_TransferDir_ToSDIO	From card to controller

SDIO_TransferMode

Specifies whether data transfer is in stream or block mode. *Table 769* gives the values assumed by this member.

Table 769. SDIO_TransferMode definition

SDIO_TransferMode	Description
SDIO_TransferMode_Stream	Stream data transfer
SDIO_TransferMode_Block	Block data transfer

SDIO_DPSM

Specifies whether SDIO Data path state machine (DPSM) is enabled or disabled.*Table 769* gives the values assumed by this member.

Table 770. SDIO_DPSM definition

SDIO_DPSM	Description
SDIO_DPSM_Enable	SDIO Data path state machine (DPSM) is enabled
SDIO_DPSM_Disable	SDIO Data path state machine (DPSM) is disabled

Example:

```
/* Configure the SDIO Data Path State Machine */
SDIO_DataInitTypeDef SDIO_DataInitStructure;
SDIO_DataInitStructure.SDIO_DataTimeOut = 0xFFFFFF;
SDIO_DataInitStructure.SDIO_DataLength = 0x100;
SDIO_DataInitStructure.SDIO_DataBlockSize = SDIO_DataBlockSize_16b;
SDIO_DataInitStructure.SDIO_TransferDir = SDIO_TransferDir_ToCard;
SDIO_DataInitStructure.SDIO_TransferMode = SDIO_TansferMode_Block;
SDIO_DataInitStructure.SDIO_DPSM = SDIO_DPSM_Enable;
SDIO_DataConfig(&SDIO_DataInitStructure);
```

24.2.14 SDIO_DataStructInit

Table 771 describes the SDIO_DataStructInit function.

Table 771. SDIO_DataStructInit function

Function name	SDIO_DataStructInit
Function prototype	`void SDIO_DataStructInit(SDIO_DataInitTypeDef* SDIO_DataInitStruct)`
Behavior description	Fills each SDIO_DataInitStruct member with its default value.
Input parameter	SDIO_DataInitStruct: pointer to an SDIO_DataInitTypeDef structure which will be initialized.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 772 gives the default values of the SDIO_DataInitStruct members.

Table 772. SDIO_DataInitStruct member definition

Member	Default value
SDIO_DataTimeOut	0xFFFFFFFF
SDIO_DataLength	0x00
SDIO_DataBlockSize	SDIO_DataBlockSize_1b
SDIO_TransferDir	SDIO_TransferDir_ToCard
SDIO_TransferMode	SDIO_TransferMode_Block
SDIO_DPSM	SDIO_DPSM_Disable

Example:

```
/* Initialize a SDIO_DataInitTypeDef structure */
SDIO_DataInitTypeDef  SDIO_DataInitStructure;
SDIO_DataStructInit(&SDIO_DataInitStructure);
```

24.2.15 SDIO_GetDataCounter

Table 773 describes the SDIO_GetDataCounter function.

Table 773. SDIO_GetDataCounter function

Function name	SDIO_GetDataCounter
Function prototype	`u32 SDIO_GetDataCounter(void)`
Behavior description	Returns number of remaining data bytes to be transferred.
Input parameter	None
Output parameter	None
Return parameter	Number of remaining data bytes to be transferred
Required preconditions	None
Called functions	None

Example:

```
/* Get the Data Counter */
u32 DataCounter = 0;
DataCounter = SDIO_GetDataCounter();
```

24.2.16 SDIO_ReadData

Table 774 describes the SDIO_ReadData function.

Table 774. SDIO_ReadData function

Function name	SDIO_ReadData
Function prototype	`u32 SDIO_ReadData(void)`
Behavior description	Read one data word from Rx FIFO.
Input parameter	None
Output parameter	None
Return parameter	Data received
Required preconditions	None
Called functions	None

Example:

```
/* Read Data */
u32 Data = 0;
Data = SDIO_ReadData();
```

24.2.17 SDIO_WriteData

Table 775 describes the SDIO_WriteData function.

Table 775. SDIO_WriteData function

Function name	SDIO_WriteData
Function prototype	`void SDIO_WriteData(u32 Data)`
Behavior description	Write one data word to Tx FIFO.
Input parameter	Data: 32-bit data word to write.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Write Data */
SDIO_WriteData(0xFFF);
```

24.2.18 SDIO_GetFIFOCount

Table 776 describes the SDIO_GetFIFOCount function.

Table 776. SDIO_GetFIFOCount function

Function name	SDIO_GetFIFOCount
Function prototype	`u32 SDIO_GetFIFOCount(void)`
Behavior description	Returns the number of words left to be written to or read from FIFO.
Input parameter	None
Output parameter	None
Return parameter	Remaining number of words.
Required preconditions	None
Called functions	None

Example:

```
/* Get the FIFO Data Counter */
u32 FIFODataCounter = 0;
FIFODataCounter = SDIO_GetFIFOCount();
```

24.2.19 SDIO_StartSDIOReadWait

Table 777 describes the SDIO_StartSDIOReadWait function.

Table 777. SDIO_StartSDIOReadWait function

Function name	SDIO_StartSDIOReadWait
Function prototype	`void SDIO_StartSDIOReadWait(FunctionalState NewState)`
Behavior description	Starts the SD I/O Read Wait operation.
Input parameter	NewState: new state of the Start SDIO Read Wait operation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Start the Read Wait Operation in SDIO mode */
SDIO_StartSDIOReadWait(ENABLE);
```

24.2.20 SDIO_StopSDIOReadWait

Table 778 describes the SDIO_StopSDIOReadWait function.

Table 778. SDIO_StopSDIOReadWait function

Function name	SDIO_StopSDIOReadWait
Function prototype	`void SDIO_StopSDIOReadWait(FunctionalState NewState)`
Behavior description	Stops the SD I/O Read Wait operation.
Input parameter	NewState: new state of the Stop SDIO Read Wait operation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Stop the Read Wait Operation in SDIO mode */
SDIO_StopSDIOReadWait(ENABLE);
```

24.2.21 SDIO_SetSDIOReadWaitMode

Table 779 describes the SDIO_SetSDIOReadWaitMode function.

Table 779. SDIO_SetSDIOReadWaitMode function

Function name	SDIO_SetSDIOReadWaitMode
Function prototype	`void SDIO_SetSDIOReadWaitMode(u32 SDIO_ReadWaitMode)`
Behavior description	Sets one of the two options of inserting read wait interval.
Input parameter	SDIOReadWaitMode: SD I/O Read Wait operation mode. This parameter can be: – SDIO_ReadWaitMode_CLK: Read Wait control by stopping SDIOCLK – SDIO_ReadWaitMode_DATA2: Read Wait control using SDIO_DATA2
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the Read Wait Mode to SDIO CLK */
SDIO_SetSDIOReadWaitMode(SDIO_ReadWaitMode_CLK);
```

24.2.22 SDIO_SetSDIOOperation

Table 780 describes the SDIO_SetSDIOOperation function.

Table 780. SDIO_SetSDIOOperation function

Function name	SDIO_SetSDIOOperation
Function prototype	`void SDIO_SetSDIOOperation(FunctionalState NewState)`
Behavior description	Enables or disables the SD I/O mode operation.
Input parameter	NewState: new state of SDIO specific operation. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the SDIO Operation */
SDIO_SetSDIOOperation(ENABLE);
```

24.2.23 SDIO_SendSDIOSuspendCmd

Table 781 describes the SDIO_SendSDIOSuspendCmd function.

Table 781. SDIO_SendSDIOSuspendCmd function

Function name	SDIO_SendSDIOSuspendCmd
Function prototype	`void SDIO_SendSDIOSuspendCmd(FunctionalState NewState)`
Behavior description	Enables or disables the SD I/O Mode suspend command sending.
Input parameter	NewState: new state of the SD I/O Mode suspend command. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send the SDIO Suspend Command */
SDIO_SendSDIOSuspendCmd(ENABLE);
```

24.2.24 SDIO_CommandCompletionCmd

Table 782 describes the SDIO_CommandCompletionCmd function.

Table 782. SDIO_CommandCompletionCmd function

Function name	SDIO_CommandCompletionCmd
Function prototype	`void SDIO_CommandCompletionCmd(FunctionalState NewState)`
Behavior description	Enables or disables the command completion signal.
Input parameter	NewState: new state of command completion signal. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the Command Compeltion signal */
SDIO_CommandCompletionCmd(ENABLE);
```

24.2.25 SDIO_CEATAITCmd

Table 783 describes the SDIO_CEATAITCmd function.

Table 783. SDIO_CEATAITCmd function

Function name	SDIO_CEATAITCmd
Function prototype	`void SDIO_CEATAITCmd(FunctionalState NewState)`
Behavior description	Enables or disables the CE-ATA interrupt.
Input parameter	NewState: new state of CE-ATA interrupt. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the CEATA interrupt */
SDIO_CEATAITCmd(ENABLE);
```

24.2.26 SDIO_SendCEATACmd

Table 784 describes the SDIO_SendCEATACmd function.

Table 784. SDIO_SendCEATACmd function

Function name	SDIO_SendCEATACmd
Function prototype	`void SDIO_SendCEATACmd(FunctionalState NewState)`
Behavior description	Sends CE-ATA command (CMD61).
Input parameter	NewState: new state of CE-ATA command. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sends the CEATA command (CMD61) */
SDIO_SendCEATACmd(ENABLE);
```

24.2.27 SDIO_GetFlagStatus

Table 785 describes the SDIO_GetFlagStatus function.

Table 785. SDIO_GetFlagStatus function

Function name	SDIO_GetFlagStatus
Function prototype	`FlagStatus SDIO_GetFlagStatus(u32 SDIO_FLAG)`
Behavior description	Checks whether the specified SDIO flag is set or not.
Input parameter	SDIO_FLAG: specifies the flag to check. Refer to *ADC_FLAG* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of SDIO_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

SDIO_FLAG

Table 786 gives the list of the SDIO flags that can be checked.

Table 786. SDIO_FLAG definition

SDIO_FLAG	Description
SDIO_FLAG_CCRCFAIL	Command response received (CRC check failed) flag
SDIO_FLAG_DCRCFAIL	Data block sent/received (CRC check failed) flag
SDIO_FLAG_CTIMEOUT	Command response timeout flag
SDIO_FLAG_DTIMEOUT	Data timeout flag
SDIO_FLAG_TXUNDERR	Transmit FIFO underrun error flag
SDIO_FLAG_RXOVERR	Received FIFO overrun error flag
SDIO_FLAG_CMDREND	Command response received (CRC check passed)flag
SDIO_FLAG_CMDSENT	Command sent (no response required) flag
SDIO_FLAG_DATAEND	Data end (data counter, SDIDCOUNT, is zero) flag
SDIO_FLAG_STBITERR	Start bit not detected on all data signals in wide bus mode flag
SDIO_FLAG_DBCKEND	Data block sent/received (CRC check passed)flag
SDIO_FLAG_CMDACT	Command transfer in progress flag
SDIO_FLAG_TXACT	Data transmit in progress flag
SDIO_FLAG_RXACT	Data receive in progress flag
SDIO_FLAG_TXFIFOBW	Transmit FIFO burst writable flag
SDIO_FLAG_RXFIFOBR	Receive FIFO burst readable flag
SDIO_FLAG_TXFIFOF	Transmit FIFO full flag
SDIO_FLAG_RXFIFOF	Receive FIFO full flag
SDIO_FLAG_TXFIFOE	Transmit FIFO empty flag
SDIO_FLAG_RXFIFOE	Receive FIFO empty flag
SDIO_FLAG_TXDAVL	Data available in transmit FIFO flag
SDIO_FLAG_RXDAVL	Data available in receive FIFO flag
SDIO_FLAG_SDIOIT	SDIO interrupt received flag
SDIO_FLAG_CEATAEND	CE-ATA command completion signal received for CMD61 flag

Example:

```
/* Get the SDIO Data available in transmit FIFO flag status */
FlagStatus Status = RESET;
Status = SDIO_GetFlagStatus(SDIO_FLAG_TXDAVL);
```

24.2.28 SDIO_ClearFlag

Table 787 describes the SDIO_ClearFlag function.

Table 787. SDIO_ClearFlag function

Function name	SDIO_ClearFlag
Function prototype	`void SDIO_ClearFlag(u32 SDIO_FLAG)`
Behavior description	Clears the SDIOx's pending flags.
Input parameter	SDIO_FLAG: specifies the flag to clear. Refer to *SDIO_FLAG on page 512* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SDIO_FLAG

Table 788 gives the list of the SDIO flags that can be checked.

Table 788. SDIO_FLAG definition

SDIO_FLAG	Description
SDIO_FLAG_CCRCFAIL	Command response received (CRC check failed) flag
SDIO_FLAG_DCRCFAIL	Data block sent/received (CRC check failed) flag
SDIO_FLAG_CTIMEOUT	Command response timeout flag
SDIO_FLAG_DTIMEOUT	Data timeout flag
SDIO_FLAG_TXUNDERR	Transmit FIFO underrun error flag
SDIO_FLAG_RXOVERR	Received FIFO overrun error flag
SDIO_FLAG_CMDREND	Command response received (CRC check passed) flag
SDIO_FLAG_CMDSENT	Command sent (no response required) flag
SDIO_FLAG_DATAEND	Data end (data counter, SDIDCOUNT, is zero) flag
SDIO_FLAG_STBITERR	Start bit not detected on all data signals in wide bus mode flag
SDIO_FLAG_DBCKEND	Data block sent/received (CRC check passed) flag
SDIO_FLAG_SDIOIT	SDIO interrupt received flag
SDIO_FLAG_CEATAEND	CE-ATA command completion signal received for CMD61 flag

Example:

```
/* Clear the SDIO Received FIFO overrun error flag */
SDIO_ClearFlag(SDIO_FLAG_RXOVERR);
```

24.2.29 SDIO_GetITStatus

Table 789 describes the SDIO_GetITStatus function.

Table 789. SDIO_GetITStatus function

Function name	SDIO_GetITStatus
Function prototype	`ITStatus SDIO_GetITStatus(u32 SDIO_IT)`
Behavior description	Checks whether the specified SDIO interrupt has occurred or not.
Input parameter	SDIO_IT: specifies the SDIO interrupt source to check. Refer to *SDIO_IT on page 496* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	The new state of SDIO_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the SDIO Data available in transmit FIFO IT status */
ITStatus Status = RESET;
Status = SDIO_GetITStatus(SDIO_IT_TXDAVL);
```

24.2.30 SDIO_ClearITPendingBit

Table 790 describes the SDIO_ClearITPendingBit function.

Table 790. SDIO_ClearITPendingBit function

Function name	SDIO_ClearITPending Bit
Function prototype	`void SDIO_ClearITPendingBit(u32 SDIO_IT)`
Behavior description	Clears the SDIO's interrupt pending bits.
Input parameter	SDIO_IT: specifies the interrupt pending bit to clear. Refer to *SDIO_IT on page 514* for more details on the allowed values for this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

SDIO_IT

To enable or disable SDIO interrupts, use a combination of one or more of the values given in *Table 791*.

Table 791. SDIO_IT definitions

SDIO_IT	Description
SDIO_IT_CCRCFAIL	Command response received (CRC check failed) interrupt mask
SDIO_IT_DCRCFAIL	Data block sent/received (CRC check failed) interrupt mask
SDIO_IT_CTIMEOUT	Command response timeout interrupt mask
SDIO_IT_DTIMEOUT	Data timeout interrupt mask
SDIO_IT_TXUNDERR	Transmit FIFO underrun error interrupt mask
SDIO_IT_RXOVERR	Received FIFO overrun error interrupt mask
SDIO_IT_CMDREND	Command response received (CRC check passed) interrupt mask
SDIO_IT_CMDSENT	Command sent (no response required) interrupt mask
SDIO_IT_DATAEND	Data end (data counter SDIDCOUNT is zero) interrupt mask
SDIO_IT_STBITERR	Start bit not detected on all data signals in wide bus mode interrupt mask
SDIO_IT_DBCKEND	Data block sent/received (CRC check passed) interrupt mask
SDIO_IT_SDIOIT	SDIO interrupt received interrupt mask
SDIO_IT_CEATAEND	CE-ATA command completion signal received for CMD61 interrupt mask

Example:

```
/* Clear the SDIO Data block sent/received (CRC check
passed)interrupt pending bit*/
SDIO_ClearITPendingBit(SDIO_IT_DBCKEND);
```

25 Debug MCU

The DBGMCU can be used for a variety of purposes, including getting the device identifier, peripheral and low-power mode behavior when the MCU is in Debug mode.

Section 25.1: DBGMCU register structure describes the data structures used in the DBGMCU firmware library. *Section 25.2: Firmware library functions* presents the firmware library functions.

25.1 DBGMCU register structure

The DBGMCU register structure, *DBGMCU_TypeDef, is* defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 IDCODE;
  vu32 CR;
}DBGMCU_TypeDef;
```

Table 792 gives the list of DBGMCU registers.

Table 792. DBGMCU registers

Register	Description
IDCODE	MCU device ID code register
CR	Control register

The DBGMCU peripheral is declared in the same file:

```
#define DBGMCU_BASE           ((u32)0xE0042000)
#ifndef DEBUG
...
#ifdef _DBGMCU
  #define DBGMCU              ((DBGMCU_TypeDef *) DBGMCU_BASE)
#endif /*_DBGMCU */
...
#else   /* DEBUG */
...
#ifdef _DBGMCU
  EXT DBGMCU_TypeDef          *DBGMCU;
#endif /*_DBGMCU */
...
#endif
```

When using the Debug mode, the DBGMCU pointer is initialized in the *stm32f10x_lib.c* file:

```
#ifdef _DBGMCU
  DBGMCU = (DBGMCU_TypeDef *)  DBGMCU_BASE;
#endif /*_DBGMCU */
```

To access the Debug MCU registers, _DBGMCU must be defined in *stm32f10x_conf.h* as follows:

```
#define _DBGMCU
```

25.2 Firmware library functions

Table 793 gives the list of the various functions in the DBGMCU library.

Table 793. DBGMCU firmware library functions

Function name	Description
DBGMCU_GetREVID	Returns the device revision identifier.
DBGMCU_GetDEVID	Returns the device identifier.
DBGMCU_Config	Configures the specified peripheral and low-power mode behavior when the MCU is in Debug mode.

25.2.1 DBGMCU_GetREVID function

Table 794 describes the DBGMCU_GetREVID function.

Table 794. DBGMCU_GetREVID function

Function name	DBGMCU_GetREVID
Function prototype	u32 DBGMCU_GetREVID(void)
Behavior description	Returns the device revision identifier.
Input parameter	None
Output parameter	None
Return parameter	Device revision identifier
Required preconditions	None
Called functions	None

Example:

```
/* Get the device revision identifier */
u32 RevID = 0;
RevID = DBGMCU_GetREVID();
```

25.2.2 DBGMCU_GetDEVID function

Table 795 describes the DBGMCU_GetDEVID function.

Table 795. DBGMCU_GetDEVID function

Function name	DBGMCU_GetDEVID
Function prototype	u32 DBGMCU_GetDEVID(void)
Behavior description	Returns the device identifier.
Input parameter	None
Output parameter	None
Return parameter	Device identifier
Required preconditions	None
Called functions	None

Example:

```
/* Get the device identifier */
u32 DevID = 0;
DevID = DBGMCU_GetDEVID();
```

25.2.3 DBGMCU_Config function

Table 796 describes the DBGMCU_Config function.

Table 796. DBGMCU_Config function

Function name	DBGMCU_Config
Function prototype	void DBGMCU_Config(u32 DBGMCU_Periph, FunctionalState NewState)
Behavior description	Configures the specified peripheral and low-power mode behavior when the MCU is in Debug mode.
Input parameter1	DBGMCU_Periph: specifies the peripheral and low power mode. Refer to *DBGMCU_Periph* for more details on the allowed values for this parameter.
Input parameter2	NewState: new state of the specified peripheral in Debug mode. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

DBGMCU_Periph

This parameter selects the peripheral to configure (see *Table 797*).

Table 797. DBGMCU_Periph definition

DBGMCU_Periph	Description
DBGMCU_SLEEP	Keep debugger connection during Sleep mode
DBGMCU_STOP	Keep debugger connection during Stop mode
DBGMCU_STANDBY	Keep debugger connection during Standby mode
DBGMCU_IWDG_STOP	Debug IWDG stopped when Core is halted
DBGMCU_WWDG_STOP	Debug WWDG stopped when Core is halted
DBGMCU_TIM1_STOP	TIM1 counter stopped when Core is halted
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_CAN_STOP	Debug CAN 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_TIM5_STOP	TIM5 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_TIM8_STOP	TIM8 counter stopped when Core is halted

Example:

```
/* Set PLL clock output to 72MHz using HSE (8MHz) as entry clock */
RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
```

26 CRC calculation unit

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code from a 32-bit data word and a fixed generator polynomial.

Section 26.1: CRC register structure describes the data structures used in the CRC Firmware Library. *Section 26.2: Firmware library functions* presents the Firmware Library functions.

26.1 CRC register structure

The CRC register structure, *CRC_TypeDef, is* defined in the *stm32f10x_map.h* file as follows:

```
typedef struct
{
  vu32 DR;
  vu8  IDR;
  u8   RESERVED0;
  u16  RESERVED1;
  vu32 CR;
} CRC_TypeDef;
```

Table 798 gives the list of CRC registers.

Table 798. CRC registers

Register	Description
DR	Data register
IDR	Independent Data register
CR	Control register

The CRC peripheral is declared in the same file:

```
#define PERIPH_BASE           ((u32)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
#define CRC_BASE              (AHBPERIPH_BASE + 0x3000)

#ifndef DEBUG
...
#ifdef _CRC
  #define CRC                 ((CRC_TypeDef *) CRC_BASE)
#endif /*_CRC */
...
#else   /* DEBUG */
...
#ifdef _CRC
  EXT CRC_TypeDef             *CRC;
#endif /*_CRC */
...
```

```
#endif
```

When using the Debug mode, CRC pointer is initialized in *stm32f10x_lib.c* file:

```
#ifdef _CRC
  CRC = (CRC_TypeDef *)  CRC_BASE;
#endif /*_CRC */
```

To access the CRC calculation unit registers, _CRC must be defined in *stm32f10x_conf.h* as follows:

```
#define _CRC
```

26.2 Firmware library functions

Table 799 gives the list of the various functions of the CRC library.

Table 799. CRC firmware library functions

Function name	Description
CRC_ResetDR	Resets the CRC Data register (DR).
CRC_CalcCRC	Computes the 32-bit CRC of a given data word (32-bit).
CRC_CalcBlockCRC	Computes the 32-bit CRC of a given data word buffer (32-bit).
CRC_GetCRC	Returns the current CRC value
CRC_SetIDRegister	Stores a 8-bit data in the independent data (ID) register.
CRC_GetIDRegister	Returns the 8-bit data stored in the independent data (ID) register

26.2.1 CRC_ResetDR function

Table 800 describes the CRC_ResetDR function.

Table 800. CRC_ResetDR function

Function name	CRC_ResetDR
Function prototype	void CRC_ResetDR(void)
Behavior description	Resets the CRC Data register (DR).
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Deinitialize the CRC Data register (DR) */
CRC_ResetDR();
```

26.2.2 CRC_CalcCRC function

Table 801 describes the CRC_CalcCRC function.

Table 801. CRC_CalcCRC function

Function name	CRC_CalcCRC
Function prototype	u32 CRC_CalcCRC(u32 Data)
Behavior description	Computes the 32-bit CRC of a given data word (32-bit).
Input parameter	Data: data word (32-bit) to compute its CRC
Output parameter	None
Return parameter	32-bit CRC
Required preconditions	None
Called functions	None

Example:

```
/* Compute the CRC of 0x32F103 data */
u32 DataCRC = 0;
DataCRC = CRC_CalcCRC(0x32F103);
```

26.3 CRC_CalcBlockCRC function

Table 802 describes the CRC_CalcBlockCRC function.

Table 802. CRC_CalcBlockCRC function

Function name	CRC_CalcBlockCRC
Function prototype	u32 CRC_CalcBlockCRC(u32 pBuffer[], u32 BufferLength)
Behavior description	Computes the 32-bit CRC of a given data word buffer (32-bit).
Input parameter1	pBuffer: pointer to the buffer containing the data to be computed
Input parameter2	BufferLength: length of the buffer to be computed
Output parameter	None
Return parameter	32-bit CRC
Required preconditions	None
Called functions	None

Example:

```
u32 DATA_t[2] = {0x32F103, 0x32F101};
u32 DATACRC = 0;

/* Compute the CRC of DATA_t buffer */
DATACRC = CRC_CalcBlockCRC(DATA_t, 2);
```

26.3.1 CRC_GetCRC function

Table 803 describes the CRC_GetCRC function.

Table 803. CRC_GetCRC function

Function name	CRC_GetCRC
Function prototype	u32 CRC_GetCRC(void)
Behavior description	Returns the current CRC value.
Input parameter	None
Output parameter	None
Return parameter	32-bit CRC
Required preconditions	None
Called functions	None

Example:

```
/* Get the current CRC value */
u32 DataCRC = 0;
DataCRC = CRC_GetCRC();
```

26.3.2 CRC_SetIDRegister function

Table 804 describes the CRC_SetIDRegister function.

Table 804. CRC_SetIDRegister function

Function name	CRC_SetIDRegister
Function prototype	void CRC_SetIDRegister(u8 IDValue)
Behavior description	Stores 8-bit data into the independent data (ID) register.
Input parameter	IDValue: 8-bit value to be stored into the ID register
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Store 0xF1 value into the Independent Data(ID) register */
CRC_SetIDRegister(0xF1);
```

26.3.3 CRC_GetIDRegister function

Table 805 describes the CRC_GetIDRegister function.

Table 805. CRC_GetIDRegister function

Function name	CRC_GetIDRegister
Function prototype	u8 CRC_GetIDRegister(void)
Behavior description	Returns the 8-bit data stored in the Independent Data(ID) register
Input parameter	None
Output parameter	None
Return parameter	8-bit value of the ID register
Required preconditions	None
Called functions	None

Example:

```
/* Get the current ID register value */
u8 IDValue = 0;
IDValue = CRC_GetIDRegister();
```

27 Revision history

Table 806. Revision history

Date	Revision	Changes
28-May-2007	1	Initial release.
05-Oct-2007	2	*Section 1.3.1: Variables on page 38* updated. In *Peripheral declaration on page 40*, `#define DEBUG` replaced by `#define DEBUG 1`. *assert* replaced by *assert_param* and *#undef assert* removed from document. *Figure 1: Firmware library folder structure* updated. RIDE added in *Section 2.1.3: Project folder on page 43*. Targeted bit position modified in *Section 2.4.1: Mapping formula on page 47*. BKP_RTCOutputConfig modified in *Table 54: BKP library functions*. In *Section 5.2: Firmware library functions on page 86*, BKP_RTCCalibrationClockOutputCmd() function replaced by BKP_RTCOutputConfig(). *Table 75: CAN_SJW values* modified. Required preconditions updated in *Table 162: FLASH_ReadOutProtection function* and note added in *Section 9.2.13: FLASH_ReadOutProtection function*. RTC_GetPrescaler function removed (see *Section 16.2: Firmware library functions*). Descriptions changed in *Table 416: SPI_CPOL definition*. *Section 19.2.2: TIM_TimeBaseInit function* modified. TIM_InitTypeDef replaced by TIM_OCInitTypeDef and example updated in *Section 19.2.3: TIM_OC1Init function*. *Table 489: TIM_ICSelection definition* and *Table 525: TIM_ExtTRGPrescaler values* modified. *Section 19.2.51: TIM_OC1NPolarityConfig function*, *Section 19.2.53: TIM_OC2NPolarityConfig function* and *Section 19.2.55: TIM_OC3NPolarityConfig function* modified. Note added in *USART_Parity on page 415*. *Section 5.2.5: BKP_RTCOutputConfig function* modified. Examples modified in *Section 16.2.10: RTC_WaitForSynchro function* and *Section 19.2.53: TIM_OC2NPolarityConfig function*.

Table 806. Revision history

Date	Revision	Changes
22-May-2008	3	*User manual updated to support high-density STM32F10xxx devices.* *Section 4: Analog/digital converter (ADC) on page 52: 3 ADCs available.* *Section 5: Backup registers (BKP) on page 83: 42 registers available.* *Value range modified in ExtId on page 106 and ExtId on page 110. Table 87: IDE values on page 106 and Table 94: IDE values on page 110 modified.* *Section 7: DMA controller (DMA) on page 117: 12 channels available.* *Section 9: Flash memory (FLASH) on page 145: FLASH_Pages updated.* *Section 10: General purpose I/O (GPIO) on page 166: GPIO ports F and G added, GPIO_PortSource added, Table 203: GPIO_Remap values updated.* *Section 13: Nested vectored interrupt controller (NVIC) on page 220: Table 267: NVIC registers updated, NVIC_BASEPRICONFIG modified in Table 268: NVIC firmware library functions, Table 274: NVIC_IRQChannels updated,* Input parameter2 modified in *Table 291: NVIC_SetVectorTable function, NVIC_TypeDef* modified in *Section 13.1: NVIC register structure on page 220.* *Section 15: Reset and clock control (RCC) on page 260: Notes modified in Table 340: RCC_DeInit function on page 263, Table 373: RCC_AHBPeriph values updated, Table 375: RCC_APB2Periph values updated, Table 377: RCC_APB1Periph values updated.* *Section 17: Serial peripheral interface (SPI) on page 303: 3 SPIs available, I^2S feature added, Table 410: SPI firmware library functions updated.* *Section 17: Serial peripheral interface (SPI) on page 303 updated with I^2S functions.* *Section 19: Advanced-control timer, general-purpose timer and basic timer (TIM) on page 334 updated (Advanced-control timer section and General-purpose timer section merged).* *Added sections:* – *Digital/analog converter (DAC) on page 443* – *Flexible static memory controller (FSMC) on page 457* – *SDIO interface (SDIO) on page 487* – *Debug MCU on page 515* – *CRC calculation unit on page 519*
13-Jun-2008	4	*Figure 1: Firmware library folder structure* modified.

Table 806. Revision history

Date	Revision	Changes
18-Jul-2008	5	Input parameter2, I2C_FLAG, modified in *Table 250: I2C_ClearFlag function on page 209*. *I2C_FLAG on page 210* modified. Input parameter2, I2C_IT, modified in *Table 254: I2C_ClearITPendingBit function on page 212*. *Table 255: I2C_IT definition on page 212* modified and notes added below. SPI_I2S_FLAG_MODF and SPI_I2S_FLAG_CRCER flag names changed to SPI_FLAG_MODF and SPI_FLAG_CRCER in *Table 453: SPI_I2S_FLAG flags on page 324*. *Section 17.2.21: SPI_I2S_ClearFlag function on page 325* modified. I2S_IT_UDR added to and SPI_I2S_IT_CRCERR and SPI_I2S_IT_MODF flag names changed to SPI_IT_CRCERR and SPI_IT_MODF in *Table 456: SPI_I2S_IT flags on page 326*. *Section 17.2.23: SPI_I2S_ClearITPendingBit function on page 327* modified. *Table: USART_InitTypeDef members versus USART mode* removed. Example modified below *USART_Mode on page 416*. *Section 20.2.25: USART_ClearFlag function on page 433* modified. *Section 20.2.27: USART_ClearITPendingBit function on page 435* modified.
19-Sep-2008	6	FSMC_AddHoldTime replaced by FSMC_AddressHoldTime and FSMC_AddSetupTime replaced by FSMC_AddressSetupTime. *Section 2.1: Package description on page 42* updated. stm32f10x_conf.h description modified in *Table 2: Firmware library files on page 44*. Note added to *Section 15.2.3: RCC_WaitForHSEStartUp function on page 264*. I2C_EVENT_MASTER_BYTE_TRANSMITTING added to *Table 247: I2C_Event on page 207*. FSMC_AsyncWait removed: – *FSMC_NORSRAMInitTypeDef* updated – *FSMC_AsyncWait* section removed – *Table 722: FSMC_NORSRAMInitStruct member definition* updated *Table 701: FSMC_MemoryType definition on page 466* modified (COSMORAM and OneNAND removed, FSMC_MemoryType_CRAM replaced by FSMC_MemoryType_PSRAM). Small text changes.

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 any time, 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 AN AUTHORIZED ST REPRESENTATIVE, 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 void any 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.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com