User Guide

AT32F413 Firmware BSP&Pack

Introduction

This application note is written to give a quick guideline on how to apply AT32F413 BSP (Board Support Package) and install AT32 Pack.

Contents

List of tables

List of figures

1 Overview

In order to allow users to use Artery MCU in an efficiently and quickly manner, we provide a complete set of BSP & Pack tools for the sake of application development. They include peripheral driver library, core-related documents and application cases as well as Pack documents supporting a variety of development environments, such as Keil_v5, Keil_v4, IAR_v6, IAR_v7 and IAR_v8.

This application note gives a detailed account of how to use BSP and Pack.

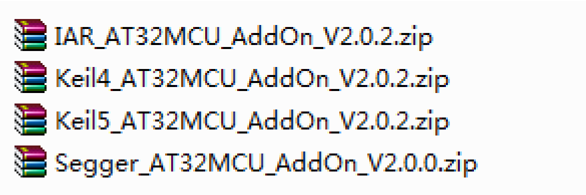
2 How to install Pack

ArteryTek provides a complete set of Pack documents that support various development environments such as Keil_v5, Keil_v4, IAR_v6, IAR_v7 and IAR_v8. Double-click on the corresponding Pack to finish one-stop installation.

Note: This section takes AT32F403A as an example, and other AT32 MCUs have similar Pack installation methods.

The pack installation documents are as follows (the specific version information is subject to the actual conditions):

Figure 1. Pack kit



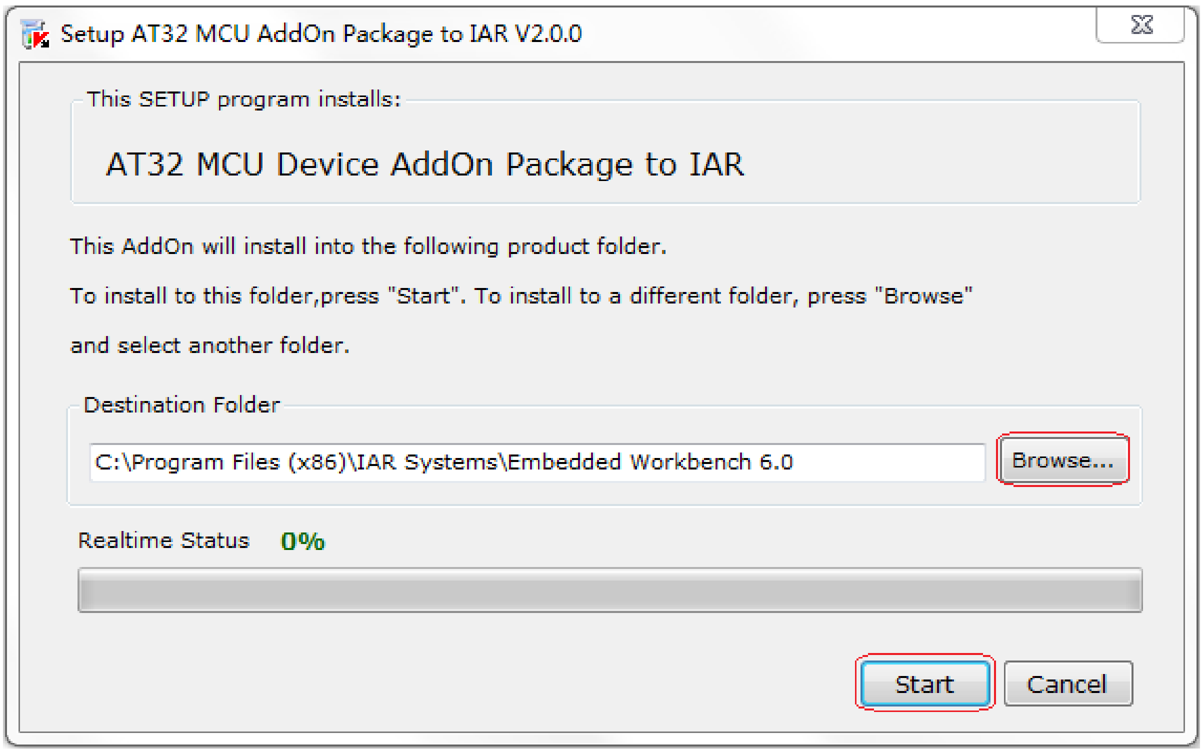
2.1 IAR Pack installation

IAR_AT32MCU_AddOn.zip: This is a zip file supporting IAR_V6, IAR_V7 and IAR_V8 Follow the steps below to install:

① Unzip *IAR_AT32MCU_AddOn.zip*

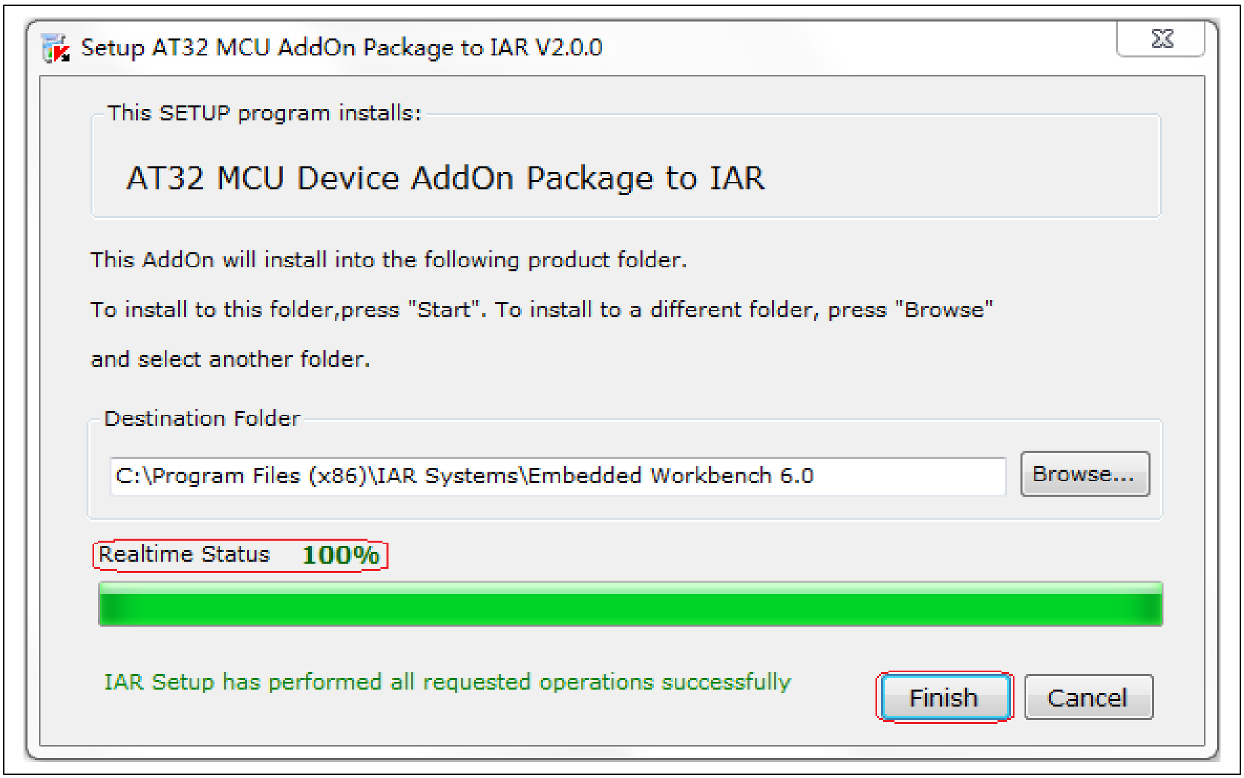
② Double click on *IAR_AT32MCU_AddOn.exe*, and a dialogue box pops up (the specific version information is subject to the actual conditions).

Figure 2. IAR Pack installation window



Note: If the installation path of IAR does not match the Destination Folder, click on "Browse" to select a correct path, then click on "Start", as shown below.

Figure 3. IAR Pack installation

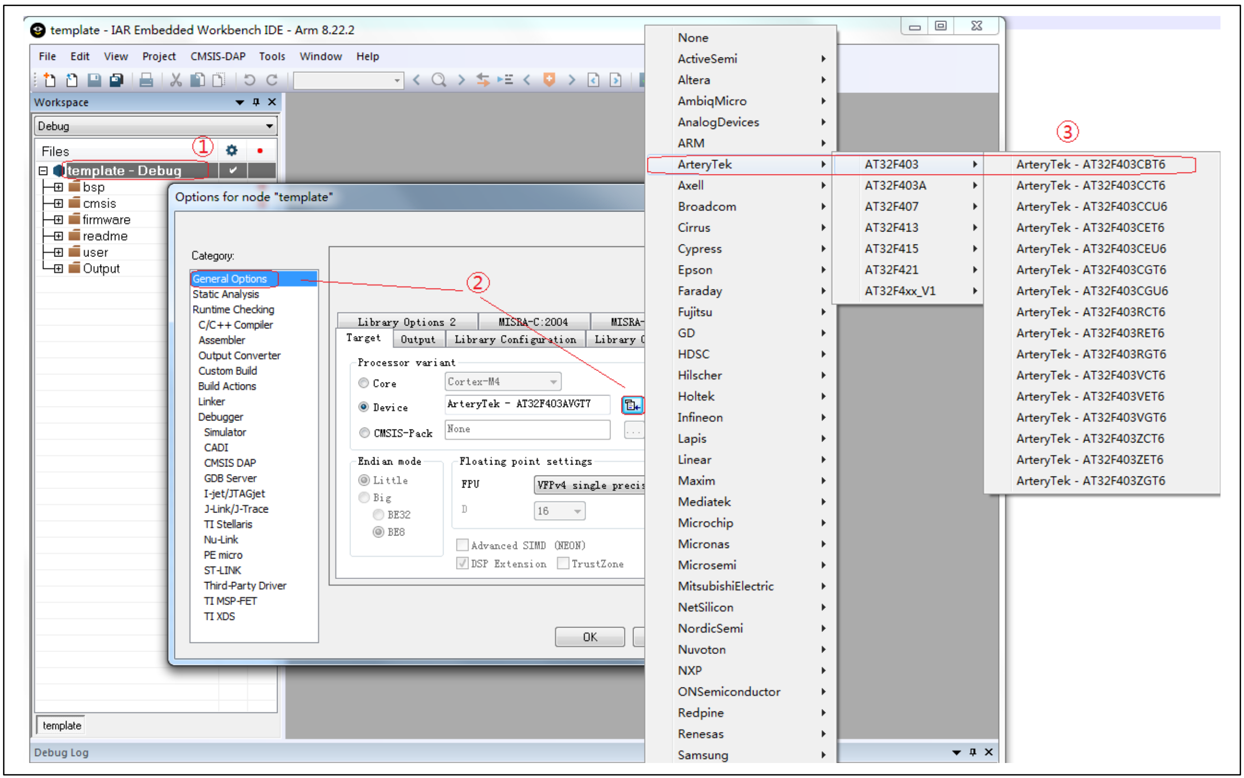


③ Click on “Finish”

④ To check whether the IAR Pack is installed successfully or not, open an IAR project and follow the steps below:

- Right click on a project name, and select “Options…”
- Select “General Options”, and click on the check box
- Click on “ArteryTek” and view AT32 MCU-related information

Figure 4. View IAR Pack installation status



2.2 Keil_v5 Pack installation

Keil5_AT32MCU_AddOn.zip: This is a zip file supporting Keil_v5. Follow the steps below to install:

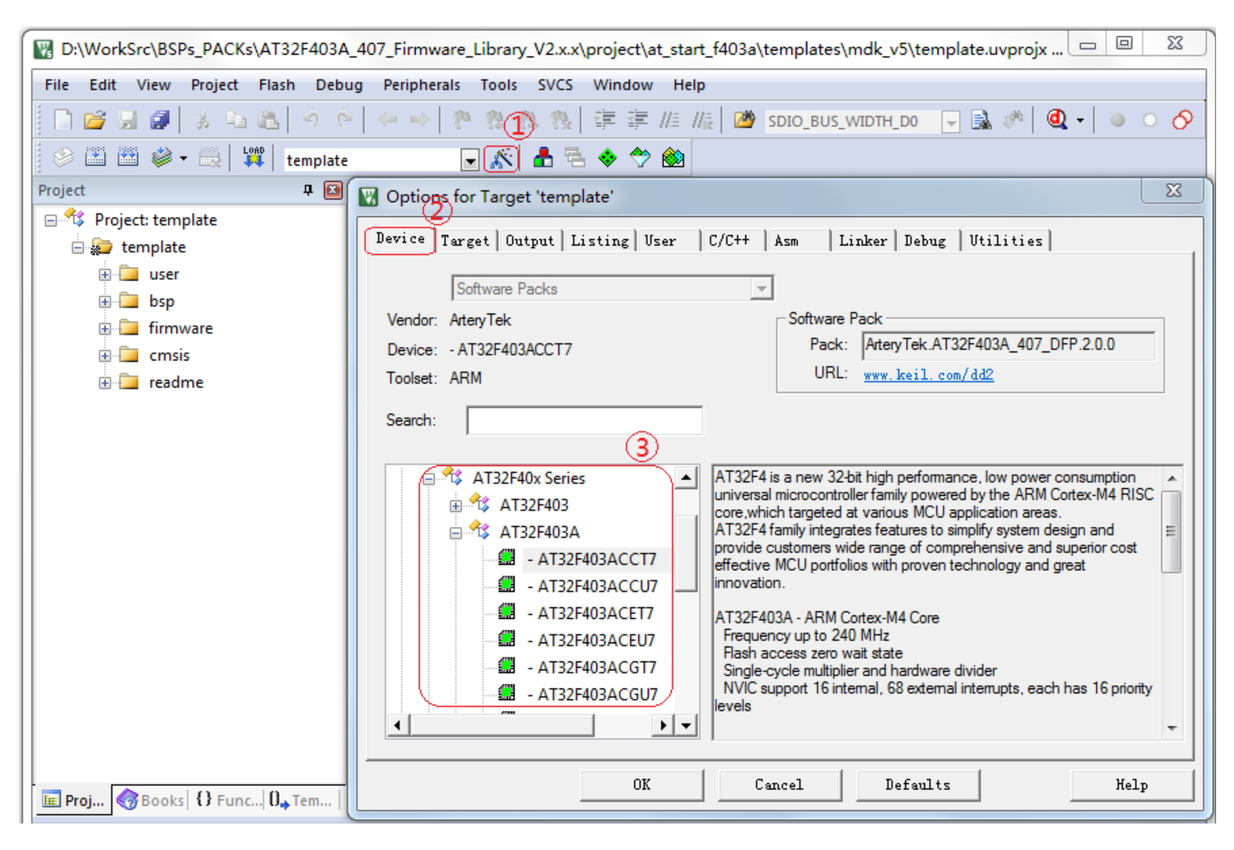
① Unzip Keil5_AT32MCU_AddOn.zip. This zip file includes all Keil5 packs supported, all of which are standard Keil_v5 DFP installation files.

② Select the desired Pack, and double click on *ArteryTek.AT32xxxx_DFP.2.x.x.pack* to get one-stop installation.

⑤ To check whether the Keil_v5 Pack is installed successfully or not, follow the steps below:

- Click on wand
- Select "Device"
- View AT32 MCU-related information

Figure 5. View Keil_v5 Pack installation status



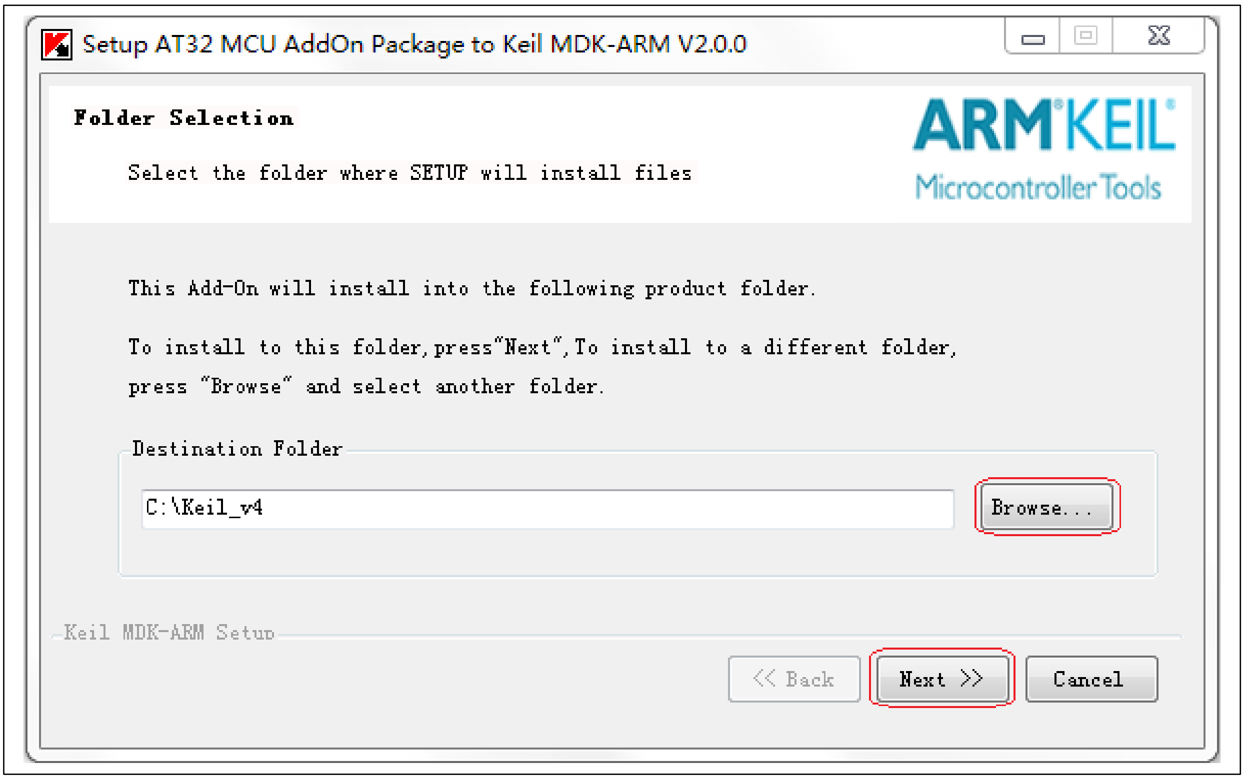
2.3 Keil_v4 Pack installation

Keil4_AT32MCU_AddOn.zip: This is a zip file supporting Keil_v4. Follow the steps below to finish installation.

① Unzip Keil4_AT32MCU_AddOn.zip

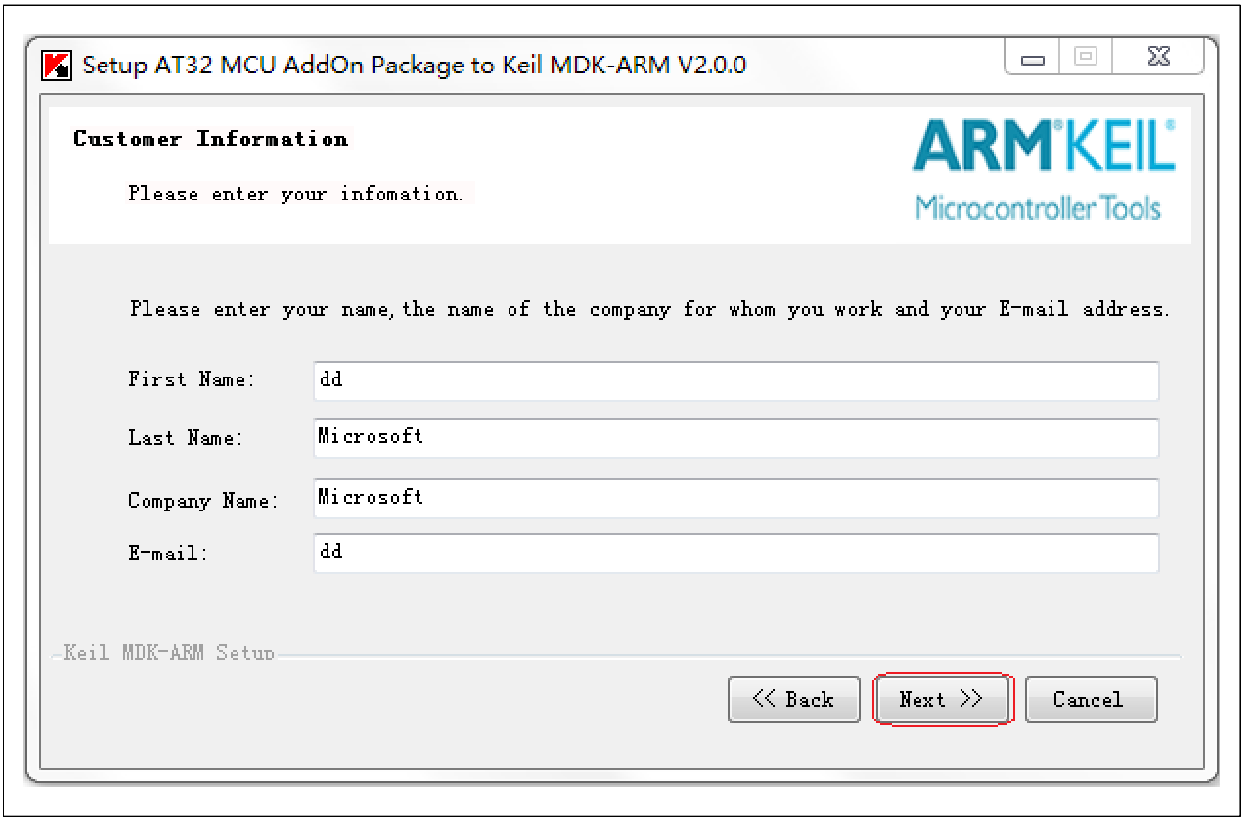
② Double click on *Keil4_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 6. Keil_v4 Pack installation window



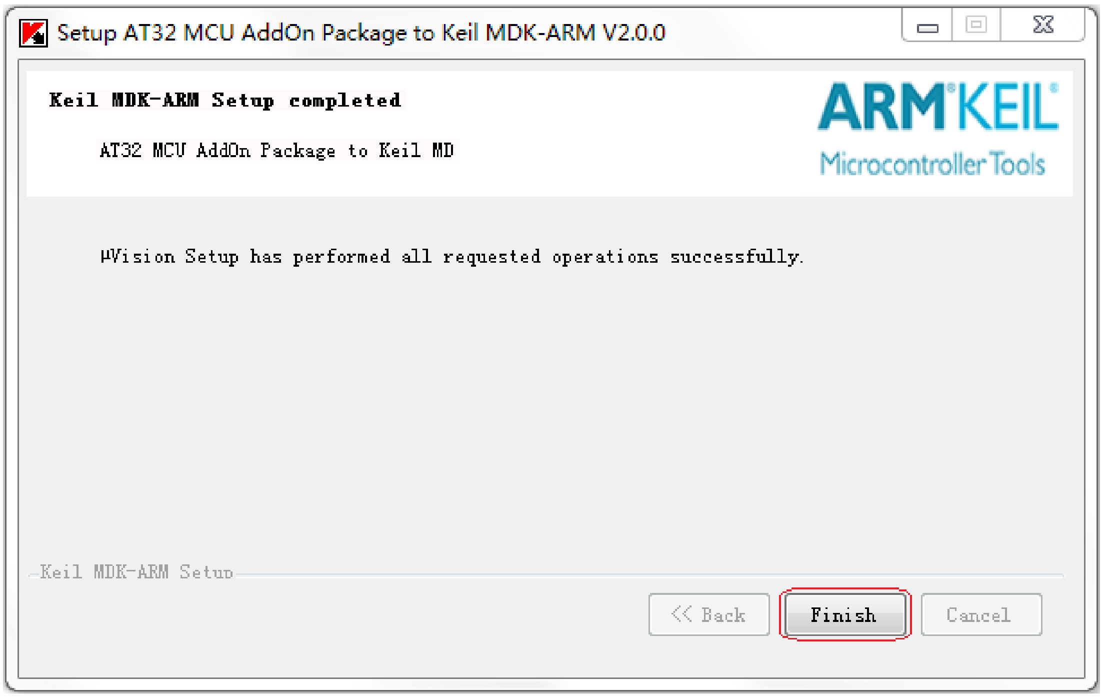
③ If the installation path of Keil_v4 does not match the "Destination Folder", click on "Browse" to select the actual correct path, then click on "Next", as shown below.

Figure 7. Keil_v4 Pack installation process



④ In the above "Customer Information" window, you can make some changes, but usually it is unnecessary. Then click on "Next" to start installation. The installation result is as follows.

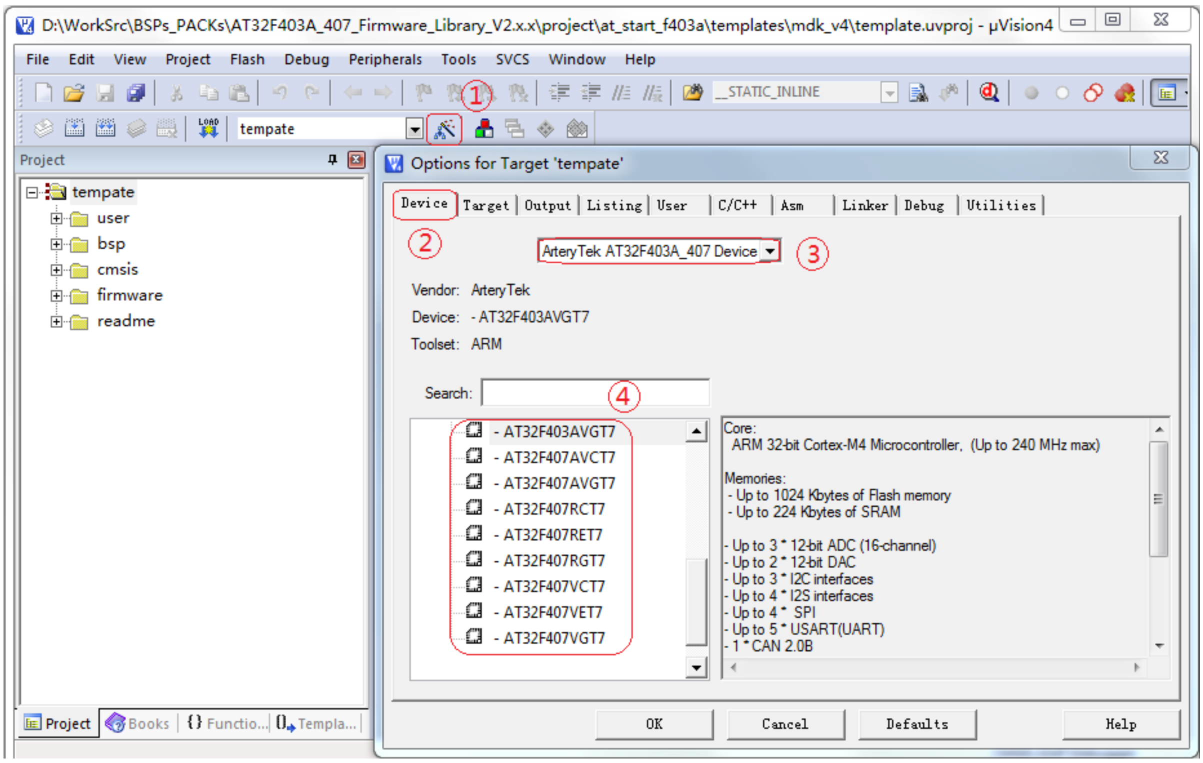
Figure 8. Keil_v4 Pack installation complete



⑤ Click on "Finish". To check whether Keil_v4 Pack is installed successfully or not, follow the below steps:

- Click on wand
- Select "Device"
- Select the desired pack file
- View ArteryTek-related information

Figure 9. View Keil_v4 Pack installation status



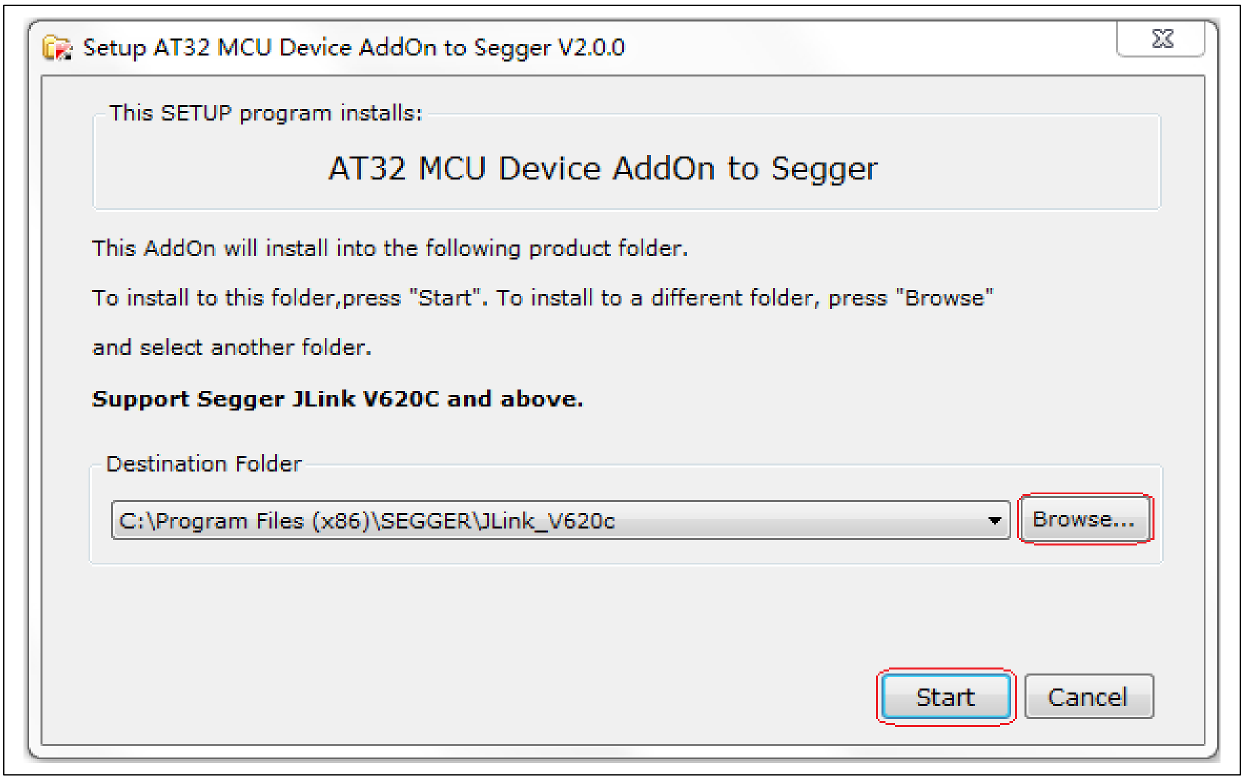
2.4 Segger Pack installation

Segger_AT32MCU_AddOn.zip: This is used to download J-Flash. Follow the steps below to install.

① Unzip Segger_AT32MCU_AddOn.zip

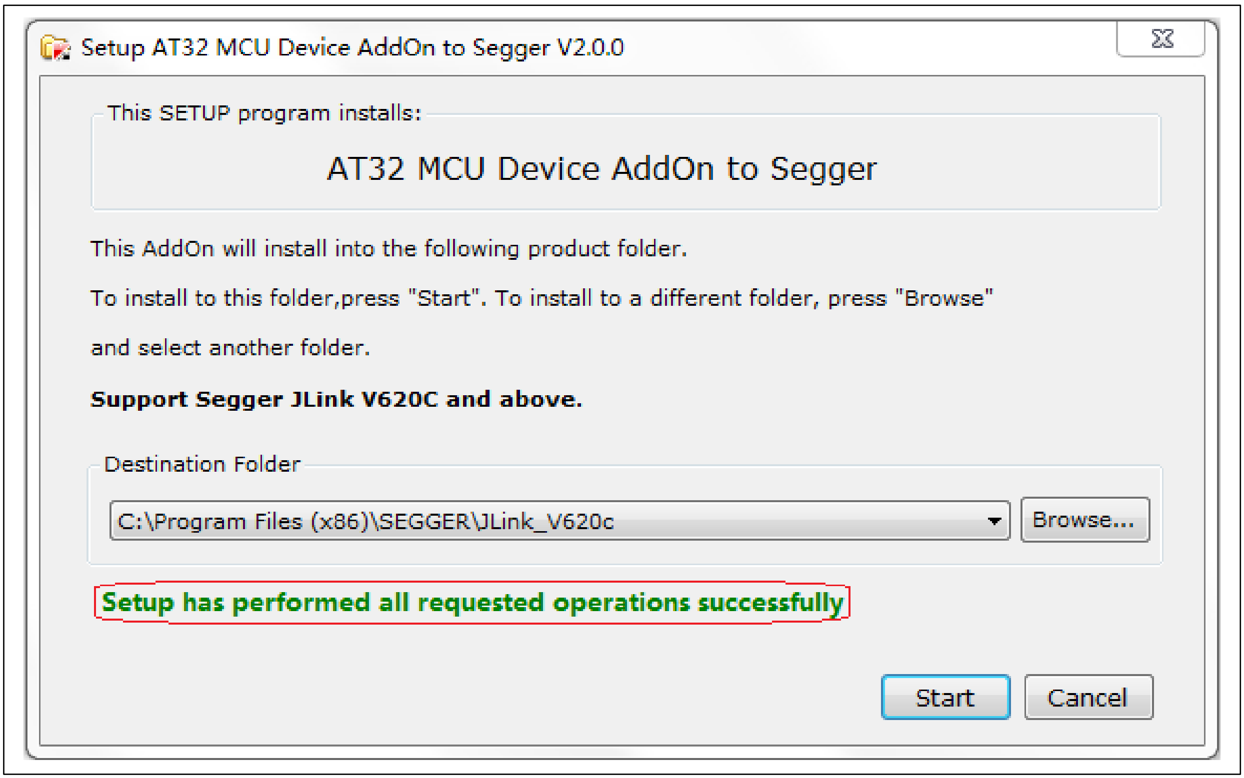
② Double click on *Segger_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions)

Figure 10. Segger pack installation window



Note: If the installation path of Segger does not match the "Destination Folder", click on "Browse" to select a correct path, then click on "Start", as shown below.

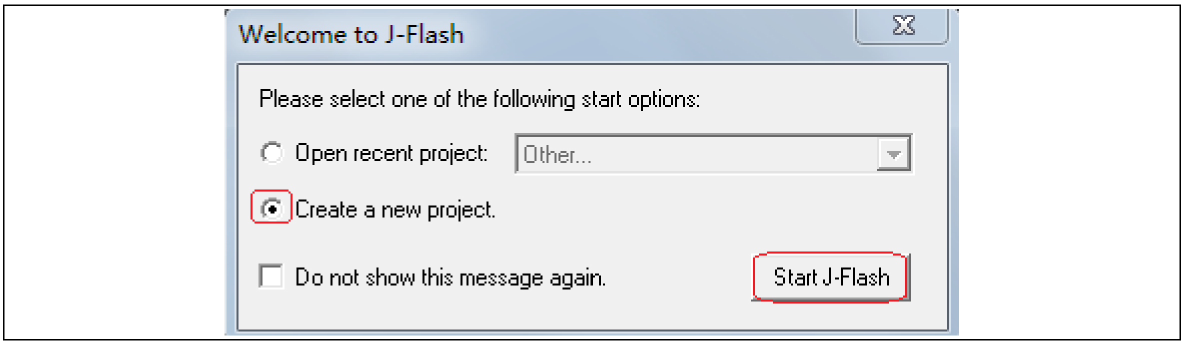
Figure 11. Segger pack installation process



③ If the "Setup has performed all requested operations successfully" appears, it indicates successful installation. To check whether the installation is successful or not, follow the steps below:

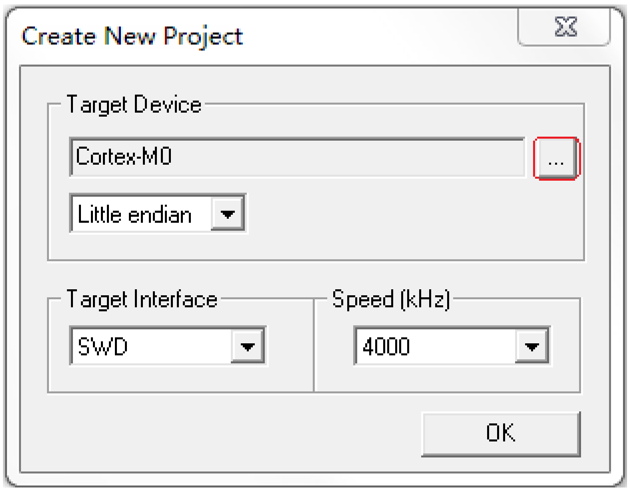
– Open J-Flash.exe, a dialog box appears, tick "Create a new project" and click on "Start J-Flash":

Figure 12. Open J-Flash



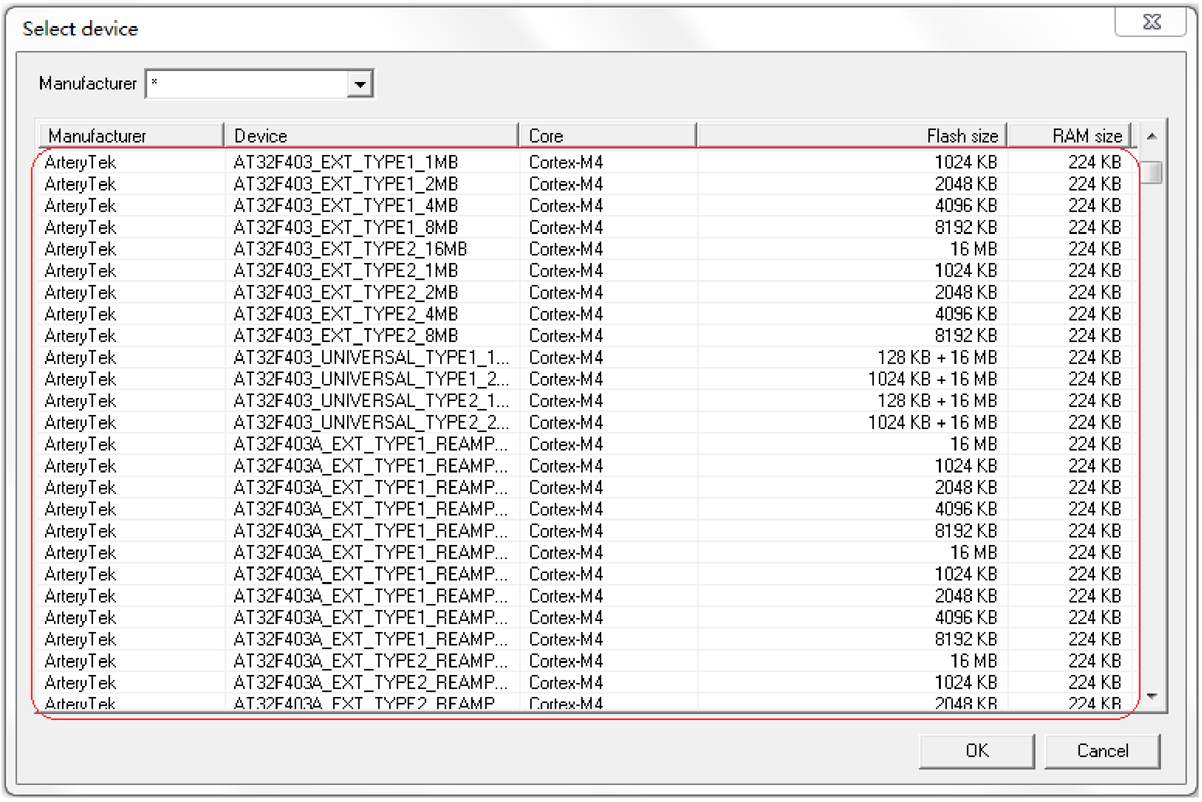
– After "Start J-Flash", click on the check box under "Target Device":

Figure 13. Create a new project using J-Flash



– Drag the scroll bar up and down in the check box. If the ArteryTek-related information and algorithm documents can be found, the installation is successful, as shown below:

Figure 14. View Device information



Manufacturer	Device	Core	Flash size	RAM size
ArteryTek	AT32F403_EXT_TYPE1_1MB	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE1_2MB	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE1_4MB	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE1_8MB	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_16MB	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_1MB	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_2MB	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_4MB	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_8MB	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE1_1...	Cortex-M4	128 KB + 16 MB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE1_2...	Cortex-M4	1024 KB + 16 MB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE2_1...	Cortex-M4	128 KB + 16 MB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE2_2...	Cortex-M4	1024 KB + 16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE2_REAMP...	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE2_REAMP...	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE2_REAMP...	Cortex-M4	2048 KB	224 KB

3 Flash algorithm file

Flash algorithm files are included in the Pack for online download through IDE tools such as KEIL/IAR. This section describes how to use Flash algorithm files.

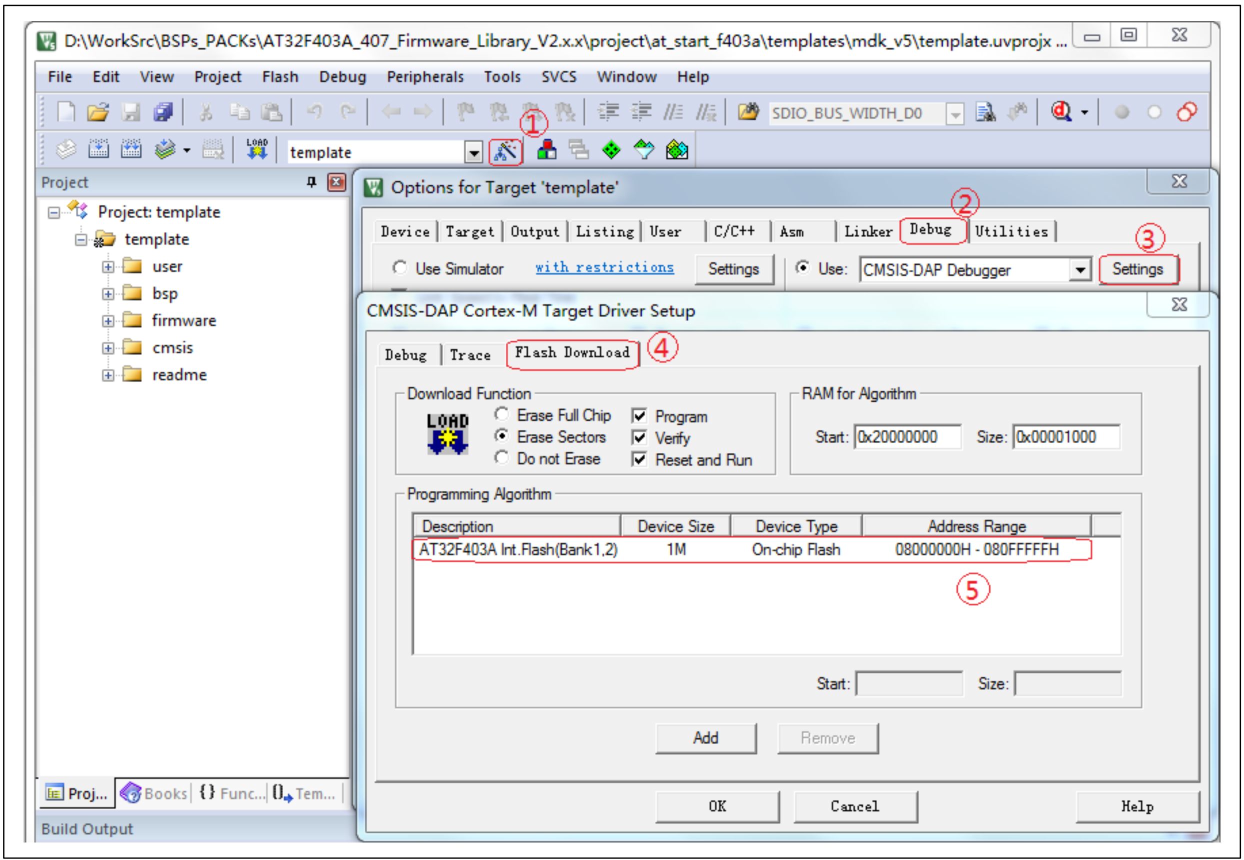
Note: AT32 MUCs have similar Flash algorithms, and this section uses AT32F403A as an example.

3.1 How to use Keil algorithm file

Common IDE tools such as Keil_v4 and Keil_v5 adopt a similar method to select and use the algorithm files. Here we take Keil_v5 as an example.

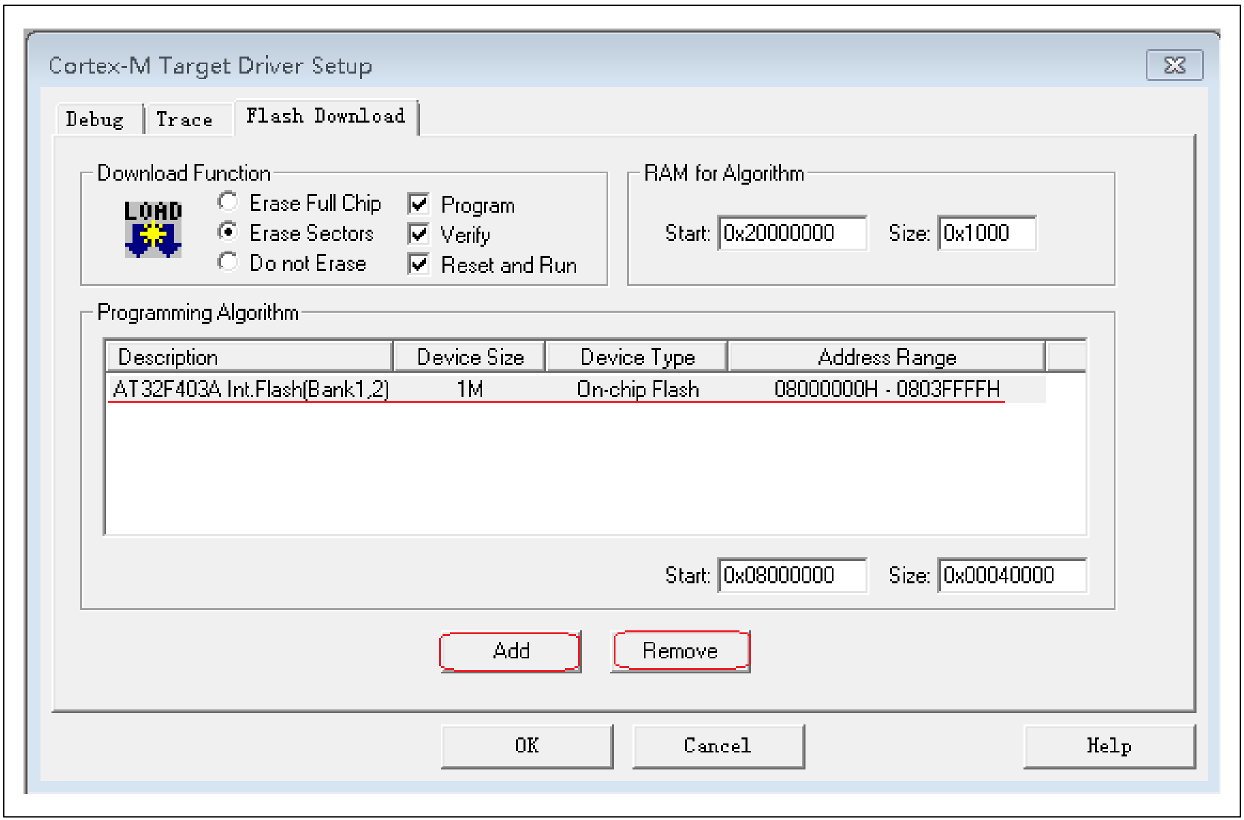
After creating a Keil IDE development tool project, the user can start Debug configuration and select the Flash algorithms. Go to *wand—>Debug—>Settings—>Flash Download*, as shown below:

Figure 15. Keil algorithm file settings



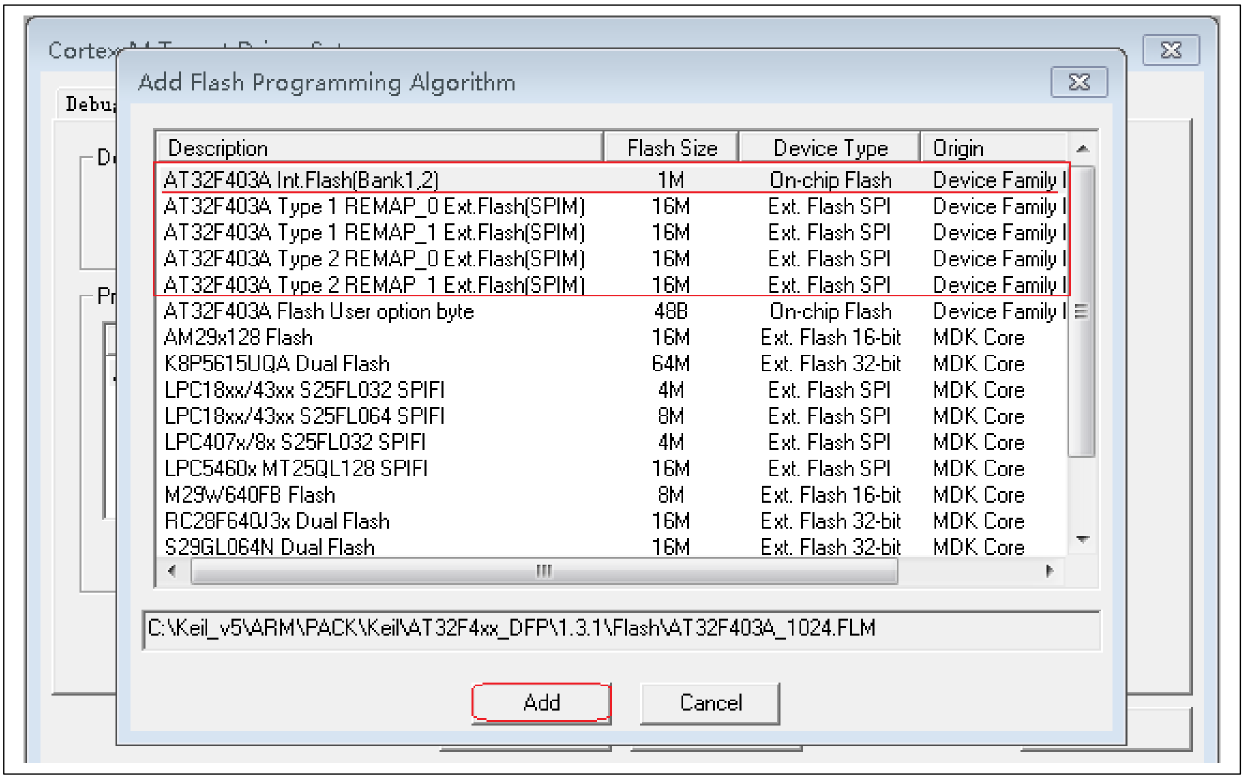
In this example, the selected Flash algorithm file is the default one. To change or remove it, click on this algorithm file, then click on *Add* or *Remove*. If the selected algorithm does not match the MCU, please follow the method below to modify.

Figure 16. Keil algorithm file configuration



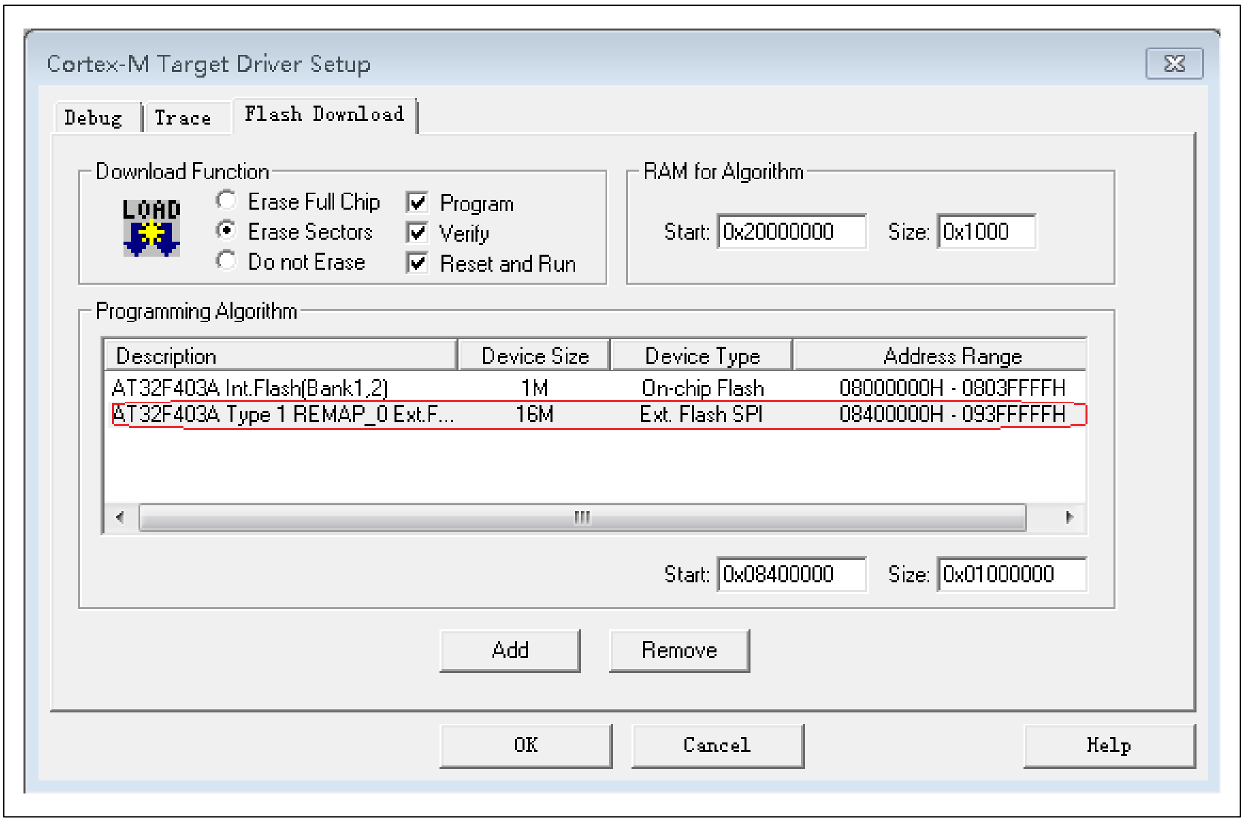
Click on *Remove* to remove the existing algorithm from the configuration, then click on *Add* to view the algorithm files associated with a MCU model and select them, as shown below:

Figure 17. Select algorithm files using Keil



After selection, click on *Add* to add the selected algorithm files into the current configuration. For example, a new SPIM algorithm is added into the project.

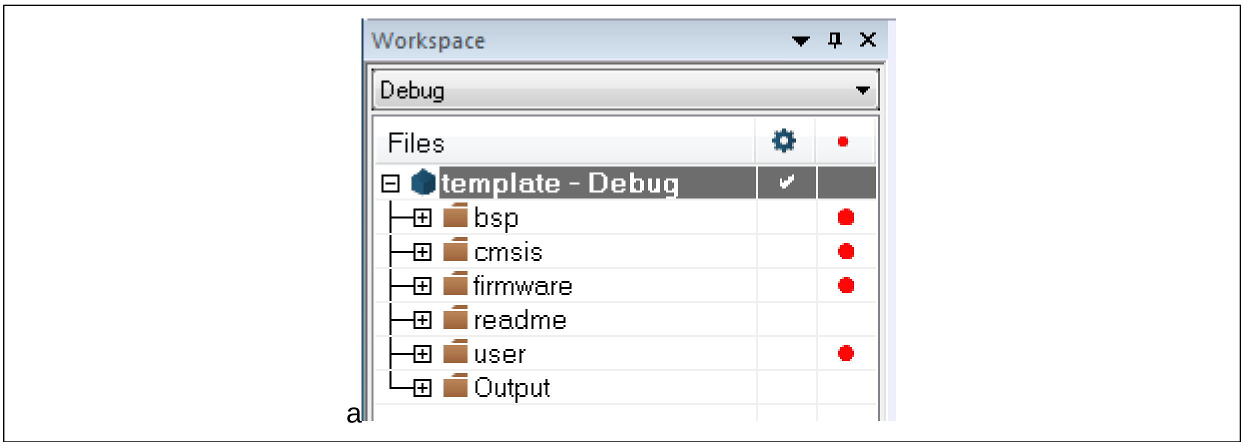
Figure 18. Add algorithm files using Keil



3.2 How to use IAR algorithm files

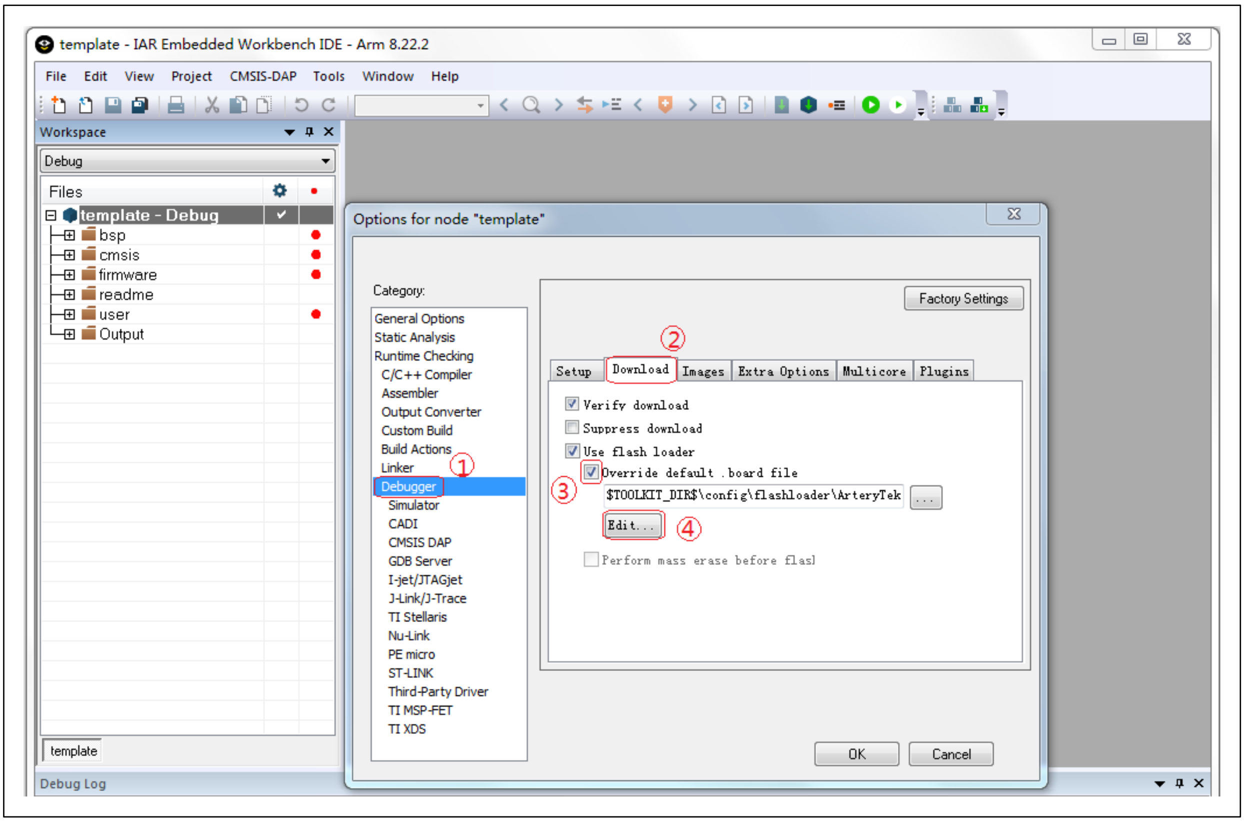
In IAR environment, the Flash algorithm files are automatically selected according to the selected MCU model during a new project configuration. To configure/modify an algorithm file manually, right-click on the file name (after an IAR project is created) in the following gray box:

Figure 19. IAR project name



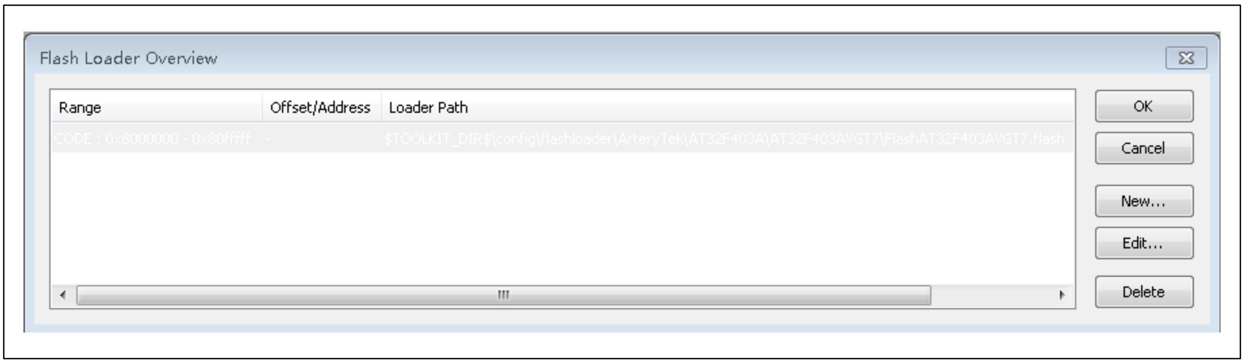
Go to *Options*—>*Debugger*—>*Download*—>Tick *Override default .board file*—>Click on *Edit*, as shown below:

Figure 20. IAR algorithm file configuration



Then the following window will be displayed.

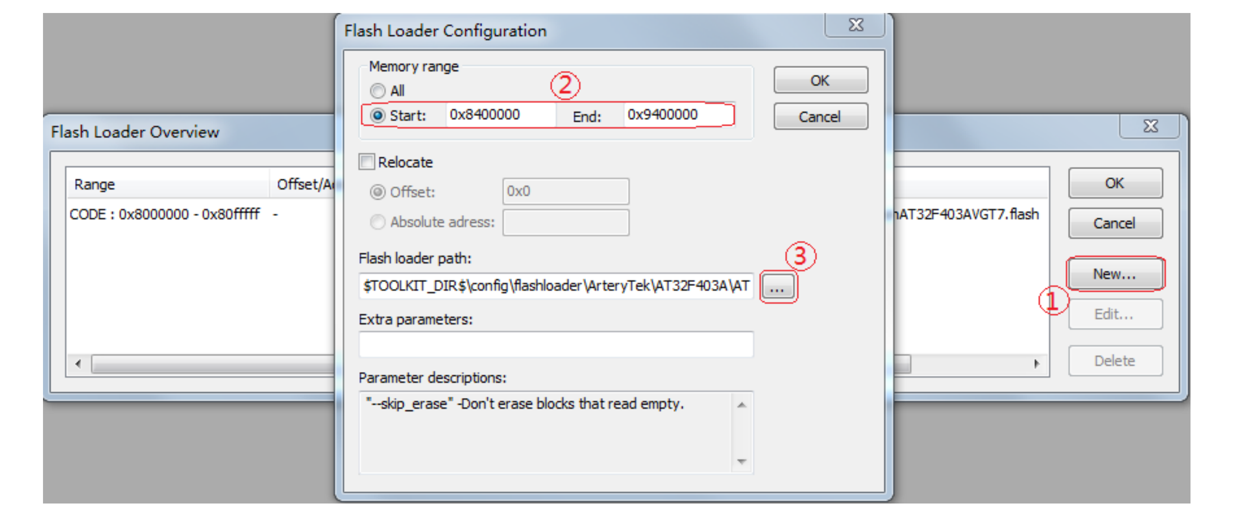
Figure 21. IAR Flash Loader Overview



Flash algorithm configuration is designated by default after selecting a MCU part number. To modify it, click on New/Edit/Delete.

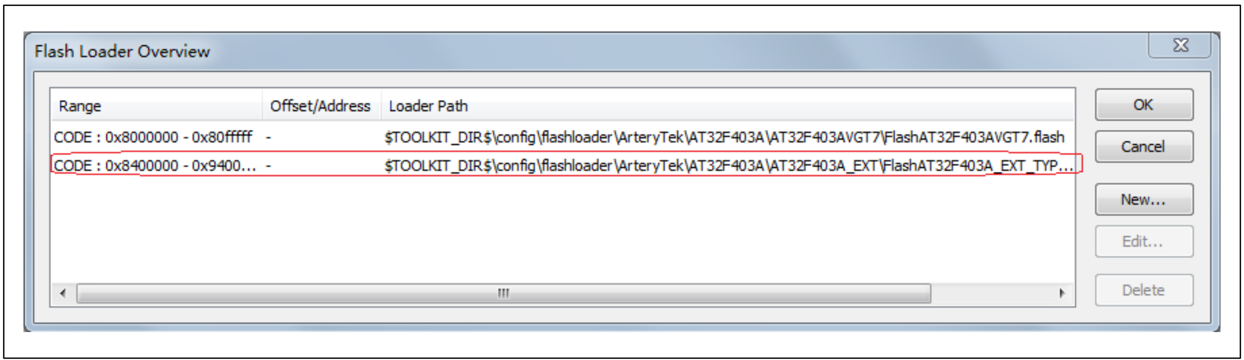
For example, click on *New*—>2. *Memory range*—>3. Select a Flash algorithm file, as shown below:

Figure 22. IAR Flash Loader configuration



This example shows how to add a SPIM Flash algorithm file. The user needs to select the corresponding MCU part number and a correct Flash algorithm file. The selected Flash algorithm configuration file is installed into IAR development environment using IAR_AT32MCU_AddOn tool. After a successful configuration, a new SPIM Flash algorithm is shown below:

Figure 23. IAR Flash Loader configuration success



1. Description of SPIM algorithms

Some Artery MCUs support Bank3 (refer to the Reference Manual or Datasheet on Artery official website for details), which can be used as an expansion of Flash memory in case of insufficient internal Flash or special application requirements. When the compiling addresses of some code or data are stored in the SPIM, these algorithm files are used for external Flash programming during online IDE tool download.

Naming rules of Artery SPIM algorithm file: AT32F4xxTypeNREMAP_P Ext.Flash.

N=1,2

P=0,1

TYPEN: External SPI Flash. Select it according to the external Flash type and part number. Refer to the FLASH_SELECT register section of the corresponding MCU Reference Manual.

REMAP_P: Select multiplex-function MCU SPIM PIN. Select it according to the connection method of pins connected to external Flash. Refer to the external SPIF remapping section in the corresponding MCU reference manual.

REMAP0: EXT_SPIF_GRMP=000

REMAP1: EXT_SPIF_GRMP=001

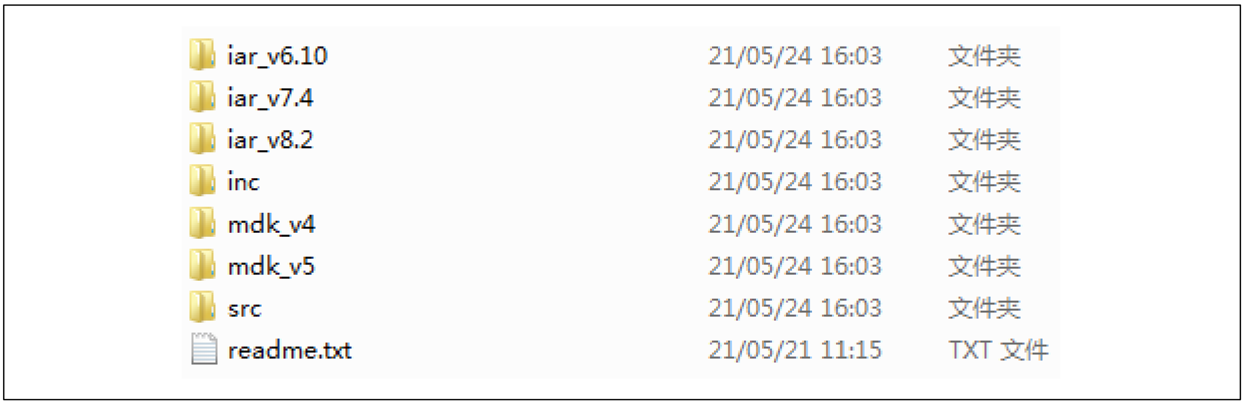
4 BSP introduction

4.1 Quick start

4.1.1 Template project

Artery firmware library BSP comes with a series of template projects built around Keil and IAR. For example, the template project of AT32F403A/407 series is located in AT32F403A_407_Firmware_Library_V2.x.x/project/at_start_xxx/templates.

Figure 24. Template content



The above template project includes various versions such as Keil_v5, Keil_v4, IAR_6.10, IAR_7.4 and IAR_8.2. Of those, “inc” and “src” folders contain header files and source code files. Open a corresponding folder and click on the corresponding file to open an IDE project. Figure 25 presents an example of Keil_v5 template project (its details and version are subject to the actual firmware library).

Figure 25. Keil_v5 template project example

The contents in a project include: (using AT32F403A/407 as an example, other products are similar)

① at32f403a_407_clock.c (clock configuration file) defines the default clock frequency and clock paths

② at32f403a_407_int.c (interrupt file) contains some interrupt handling codes

③ main.c contains the main code files

④ at32f403a_407_board.c (board configuration file) contains common hardware configurations such as buttons and LEDs on the AT-START-Evaluation Board

⑤ at32f403a_407_xx.c under firmware folder contains driver files of on-chip peripherals

⑥ system_at32f403a_407.c is the system initialization file

⑦ startup_at32f403a_407.s is a startup file

⑧ readme.txt is a readme file, containing functional description and configuration information

Note: AT32 MUCs share similar BSP usage method, and this section uses AT32F403A as an example.

4.1.2 BSP macro definitions

① To create a project, it is necessary to enable a startup code (startup_at32f403a_407.s) and open the appropriate macro definitions according to MCU part number before compiling code. Table 1 presents the correspondence between the MCUs and their macro definitions.

Table 1. Summary of macro definitions

MCU part number	Macro definitions	PINs	Flash size (KB)
AT32F403ACCT7	AT32F403ACCT7	48	256
AT32F403ACET7	AT32F403ACET7	48	512
AT32F403ACGT7	AT32F403ACGT7	48	1024
AT32F403ACCU7	AT32F403ACCU7	48	256
AT32F403ACEU7	AT32F403ACEU7	48	512
AT32F403ACGU7	AT32F403ACGU7	48	1024
AT32F403ARCT7	AT32F403ARCT7	64	256
AT32F403ARET7	AT32F403ARET7	64	512
AT32F403ARGT7	AT32F403ARGT7	64	1024
AT32F403AVCT7	AT32F403AVCT7	100	256
AT32F403AVET7	AT32F403AVET7	100	512
AT32F403AVGT7	AT32F403AVGT7	100	1024
AT32F407RCT7	AT32F407RCT7	64	256
AT32F407RET7	AT32F407RET7	64	512
AT32F407RGT7	AT32F407RGT7	64	1024
AT32F407VCT7	AT32F407VCT7	100	256
AT32F407VET7	AT32F407VET7	100	512
AT32F407VGT7	AT32F407VGT7	100	1024
AT32F407AVCT7	AT32F407AVCT7	100	256
AT32F407AVGT7	AT32F407AVGT7	100	1024

② In the header file (at32f403a_407.h), USE_STDPERIPH_DRIVER (macro definition) is used to determine whether the Keil RTE feature is used or not. Enabling this definition while Keil RTE is unused can prevent some versions of Keil-MDK from opening _RTE_ accidentally.

③ The configuration header file (at32f403a_407_conf.h) defines macro definitions that enable peripherals. The file can be used to control the use of peripherals. The peripherals can be disabled simply by masking _MODULE_ENABLED pertaining to peripherals, as shown below:

Figure 26. Peripheral enable macro definitions

```
#define CRM_MODULE_ENABLED
#define TMR_MODULE_ENABLED
#define RTC_MODULE_ENABLED
#define BPR_MODULE_ENABLED
#define GPIO_MODULE_ENABLED
#define I2C_MODULE_ENABLED
#define USART_MODULE_ENABLED
#define PWC_MODULE_ENABLED
#define CAN_MODULE_ENABLED
#define ADC_MODULE_ENABLED
#define DAC_MODULE_ENABLED
#define SPI_MODULE_ENABLED
#define DMA_MODULE_ENABLED
#define DEBUG_MODULE_ENABLED
#define FLASH_MODULE_ENABLED
#define CRC_MODULE_ENABLED
#define WWDT_MODULE_ENABLED
#define WDT_MODULE_ENABLED
#define EXINT_MODULE_ENABLED
#define SDIO_MODULE_ENABLED
#define XMC_MODULE_ENABLED
#define USB_MODULE_ENABLED
#define ACC_MODULE_ENABLED
#define MISC_MODULE_ENABLED
#define EMAC_MODULE_ENABLED
```

The at32f403a_407_conf.h also defines the HEXT_VALUE (high-speed external clock value), which should be modified accordingly when changing an external high-speed crystal oscillator.

④ The system clock configuration file (at32f403a_407_clock.c/.h) defines the default system clock frequency and clock paths. The user, if needed, can customize the frequency multiplication process and factors, or generate corresponding clock configuration files using the clock configuration host of ArteryTek.

4.2 BSP specifications

The subsequent sections give a description of BSP specifications.

4.2.1 List of abbreviations for peripherals

Table 2. List of abbreviations for peripherals

Abbreviations	Description
ADC	Analog-to-digital converter
BPR	Battery powered register
CAN	Controller area network
CRC	CRC calculation unit
CRM	Clock and reset manage
DAC	Digital-to-analog converter
DMA	Direct memory access
DEBUG	Debug
EXINT	External interrupt/event controller
GPIO	General-purpose I/Os
IOMUX	Multiplexed I/Os
I2C	Inter-integrated circuit interface
NVIC	Nested vectored interrupt controller
PWC	Power controller
RTC	Real-time clock
SPI	Serial peripheral interface
I2S	Inter-IC Sound
SysTick	System tick timer
TMR	Timer
USART	Universal synchronous asynchronous receiver transmitter
WDT	Watchdog timer
WWDT	Window watchdog timer
XMC	External memory controller

4.2.2 Naming rules

The naming rules for BSP are described as follows:
"ip" indicates an abbreviation of a peripheral, for example, ADC, TMR, GPIO, etc., regardless of upper and lower case letters, such as adc, tmr, gpio.

- **Source code file**
  The file name starts with **"**at32fxxx_ip.c**"**, for example, at32f403a_407_adc.c.
- **Header file**
  The file name starts with **"**at32fxxx_ip.h**"**, for example, at32f403a_407_adc.h.
- **Constant**
  If it is used in a single one file, the constant is then defined in this file; if it is used in multiple files, the constant is defined in corresponding header file.
  All constants are in written in English capital letters.
- **Variable**
  If it is used in a single one file, the variable is then defined in this file; if it is used in multiple files, the variable is declared with "extern" in the corresponding header file.
- **Naming rules for functions**
  The peripheral functions are named based on the rule of "**peripheral abbreviation_attribute_action**" or "**peripheral abbreviation_action**".
  The commonly used functions are as follows:

Function type	**Naming rule**	**Example**
Peripheral reset	ip_reset	adc_reset
Peripheral enable	ip_enable	adc_enable
Peripheral structure parameter initialize	ip_default_para_init	spi_default_para_init
Peripheral initialize	ip_init	spi_init
Peripheral interrupt enable	ip_interrupt_enable	adc_interrupt_enable
Peripheral flag get	ip_flag_get	adc_flag_get
Peripheral flag clear	ip_flag_clear	adc_flag_clear

4.2.3 Encoding rules

This section describes the encoding rules related to firmware function library.

Type of variables:

```
typedef int32_t   INT32;
typedef int16_t   INT16;
typedef int8_t    INT8;
typedef uint32_t UINT32;
typedef uint16_t UINT16;
typedef uint8_t   UINT8;

typedef int32_t   s32;
typedef int16_t   s16;
typedef int8_t    s8;

typedef const int32_t sc32;     /*!< read only */
typedef const int16_t sc16;     /*!< read only */
typedef const int8_t   sc8;     /*!< read only */

typedef __IO int32_t   vs32;
typedef __IO int16_t   vs16;
typedef __IO int8_t    vs8;

typedef __I int32_t vsc32;      /*!< read only */
typedef __I int16_t vsc16;      /*!< read only */
typedef __I int8_t   vsc8;      /*!< read only */

typedef uint32_t u32;
typedef uint16_t u16;
typedef uint8_t   u8;

typedef const uint32_t uc32;   /*!< read only */
typedef const uint16_t uc16;   /*!< read only */
typedef const uint8_t   uc8;   /*!< read only */

typedef __IO uint32_t vu32;
typedef __IO uint16_t vu16;
typedef __IO uint8_t   vu8;

typedef __I uint32_t vuc32;    /*!< read only */
typedef __I uint16_t vuc16;    /*!< read only */
typedef __I uint8_t   vuc8;    /*!< read only */
```

4.2.3.1 Flag type

typedef enum {RESET = 0, SET = !RESET} flag_status;

4.2.3.2 Function status type

typedef enum {FALSE = 0, TRUE = !FALSE} confirm_state;

4.2.3.3 Error status type

typedef enum {ERROR = 0, SUCCESS = !ERROR} error_status;

4.2.3.4 Peripheral type

① Peripherals

Define the base address of peripheral in the at32fxxx_ip.h, for example, in the at32f403a_407.h:

```
#define ADC1_BASE                 (APB2PERIPH_BASE + 0x2400)
#define ADC2_BASE                 (APB2PERIPH_BASE + 0x2800)
```

Define the type of a peripheral in the at32fxxx_ip.h, for example, in the at32f403a_407_adc.h:

```
#define ADC1                      ((adc_type *) ADC1_BASE)
#define ADC2                      ((adc_type *) ADC2_BASE)
```

② Peripheral registers and bits

Define the type of a peripheral in the at32fxxx_ip.h, for example, in the at32f403a_407_adc.h:

```
/**
  * @brief type define adc register all
  */
typedef struct
{

  /**
    * @brief adc sts register, offset:0x00
    */
  union
  {
    __IO uint32_t sts;
    struct
    {
      __IO uint32_t vmor               : 1; /* [0] */
      __IO uint32_t cce                : 1; /* [1] */
      __IO uint32_t pcce               : 1; /* [2] */
      __IO uint32_t pccs               : 1; /* [3] */
      __IO uint32_t occs               : 1; /* [4] */
      __IO uint32_t reserved1          : 27;/* [31:5] */
    } sts_bit;
  };
…
```

```
…
…
  /**
  * @brief adc odt register, offset:0x4C
  */
  union
  {
    __IO uint32_t odt;
    struct
    {
      __IO uint32_t odt                   : 16; /* [15:0] */
      __IO uint32_t adc2odt               : 16; /* [31:16] */
    } odt_bit;
  };

} adc_type;
```

③ Examples of peripheral register access

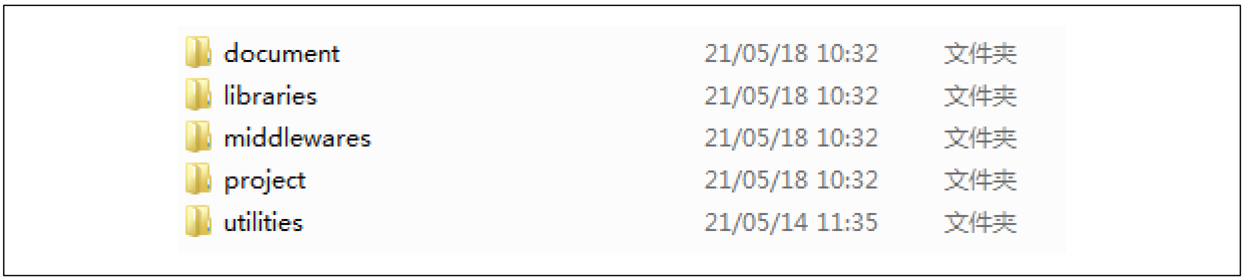
Read peripheral	i = ADC1-> ctrl1;
Write peripheral	ADC1-> ctrl1 = i;
Read bit 5 in bit-field mode	i = ADC1-> ctrl1. cceien;
Write 1 to bit 5 in bit-field mode	ADC1-> ctrl1. cceien= TRUE;
Write 1 to bit 5	ADC1-> ctrl1 |= 1<<5;
Write 0 to bit 5	ADC1-> ctrl1&= ~(1<<5) ;

4.3 BSP structure

4.3.1 BSP folder structure

BSP (Board Support Package) structure is shown in Figure 27.

Figure 27. BSP folder structure



document

- AT32Fxxx firmware library BSP&Pack user guide.pdf: refer to BSP/Pack user manual
- ReleaseNotes_AT32F403A_407_Firmware_Library.pdf: document revision history

libraries

- **drivers**: driver library for peripherals
  src folder: low-level driver source file for peripherals, such as at32fxxx_ip.c
  inc folder: low-level driver header file for peripherals, such as at32fxxx_ip.h

- **cmsis**: core-related files
  cm4 folder: core-related files, including cortex-m4 library, system initialization file, startup file, etc.
  dsp folder: dsp-related files

middlewares

Third-party software or public protocols, including USB protocol layer driver, network protocol driver, operating system source code, etc.

project

examples: demo
templates: template project, including Keil4, keil5, IAR6, IAR7, IAR8 and eclipse_gcc

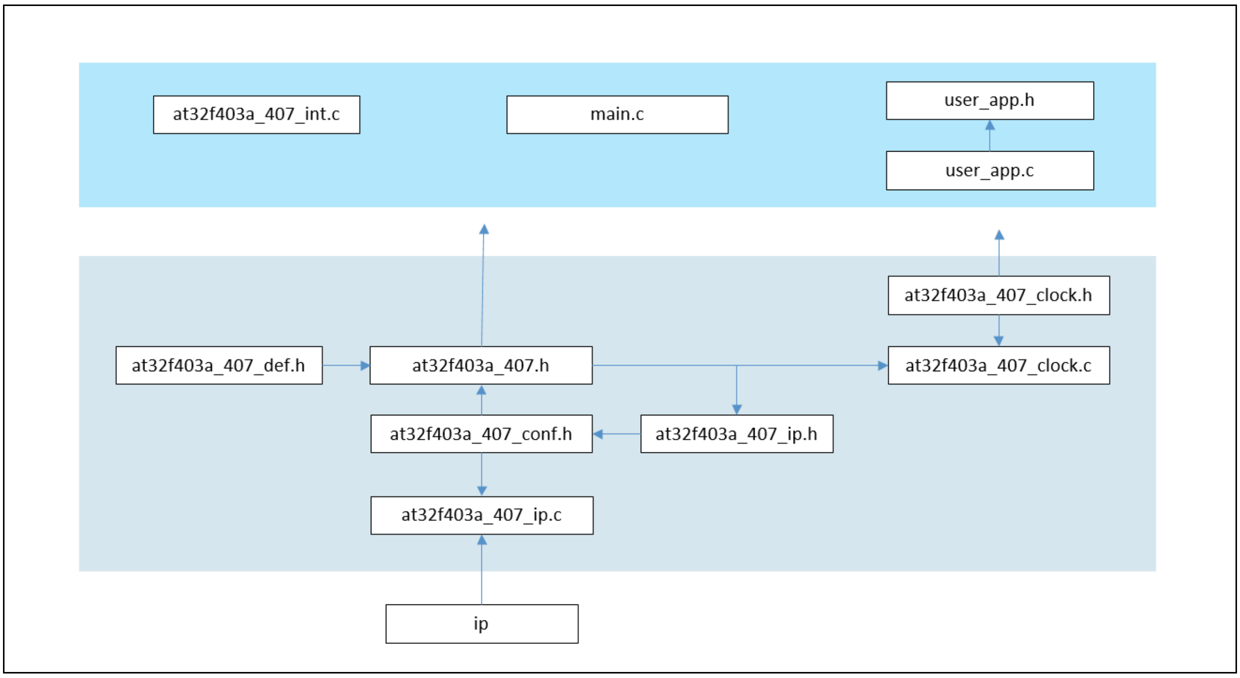
utilities

Store application cases

4.3.2 BSP function library structure

Figure 28 shows the architecture of BSP function library.

Figure 28. BSP function library structure



BSP function library files are described in Table 3.

Table 3. Summary of BSP function library files

File name	Description
at32f403a_407_conf.h	Macro definition for peripheral enable, and external high-speed clock HEXT_VALUE
main.c	Main function
at32f403a_407_ip.c	Driver source file for a peripheral, for example,at32f403a_407_adc.c
at32f403a_407_ip.h	Driver header file for a peripheral, for example,at32f403a_407_adc.h
at32f403a_407.h	In the header file (at32f403a_407.h), the definition USE_STDPERIPH_DRIVER is used to determine whether the Keil RTE is used or not. Enabling the definition while Keil RTE is unused can prevent Keil-MDK from enabling _RTE_ accidentally.
at32f403a_407_clock.c	This is a clock configuration file used to configure default clock frequency and clock path.
at32f403a_407_clock.h	This is a clock configure header file.
at32f403a_407_int.c	This is a source file for interrupt functions that programs interrupt handling code.
at32f403a_407_int.h	This is a header file for interrupt functions.
at32f403a_407_misc.c	This is a source file for other configurations, such as, nvic configuration function, systick clock source selection.
at32f403a_407_misc.h	This is a header file for other configurations.
startup_at32f403a_407.s	This is a startup file.

4.3.3 Initialization and configuration for peripherals

This section describes how to initialize and configure peripherals using GPIO as an example.

GPIO initialization

Step 1: Define the gpio_init_type, for example, gpio_init_type gpio_init_struct;

Step 2: Enable GPIO clock using the function crm_periph_clock_enable;

Step 3: De-initialize the structure gpio_init_struct o allow the values of other members (mostly default values) to be correctly written, for example, gpio_default_para_init(&gpio_init_struct);

Step 4: Configure member of the structure, and write structure parameters into GPIO registers through the gpio_init, for example,

gpio_init_struct.gpio_pins = GPIO_PINS_2 | GPIO_PINS_3;
gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOA, &gpio_init_struct);

For more information on peripheral initialization procedure, refer to the section of peripherals of the Reference Manual, and the section of peripherals of the AT32Fxxx_Firmware_Library_V2.x.x.zip\project\at_start_fxxx\examples.

4.3.4 Peripheral functions format description

Table 4. Function format description for peripherals

Name	Description
Function name	The name of a peripheral function.
Function prototype	Prototype declaration
Function description	Brief description of how the function is executed
Input parameter n	Description of the input parameters
Output parameter n	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

5 AT32F413 peripheral library functions

5.1 HICK automatic clock calibration (ACC)

The ACC register structure acc_type is defined in the “at32f413_acc.h”.

```
/**
  * @brief type define acc register all
  */
typedef struct
{
      ……
} acc_type;
```

The table below gives a list of the ACC registers.

Table 5. Summary of ACC registers

Register	Description
acc_sts	ACC status register
acc_ctrl1	ACC control register 1
acc_ctrl2	ACC control register 2
acc_c1	ACC compare value 1
acc_c2	ACC compare value 2
acc_c3	ACC compare value 3

The table below gives a list of the ACC library functions.

Table 6. Summary of ACC library functions

Function name	Description
acc_calibration_mode_enable	ACC calibration mode enable
acc_step_set	Configure ACC calibration step length
acc_interrupt_enable	ACC interrupt enable
acc_hicktrim_get	Get ACC trimming calibration value
acc_hickcal_get	Get ACC coarse calibration value
acc_write_c1	Write ACC C1 register value
acc_write_c2	Write ACC C2 register value
acc_write_c3	Write ACC C3 register value
acc_read_c1	Read ACC C1 register value
acc_read_c2	Read ACC C2 register value
acc_read_c3	Read ACC C3 register value
acc_flag_get	Get ACC interrupt flag
acc_flag_clear	Clear ACC interrupt flag

5.1.1 acc_calibration_mode_enable function

The table below describes the function acc_calibration_mode_enable.

Table 7. acc_calibration_mode_enable function

Name	Description
Function name	acc_calibration_mode_enable
Function prototype	void acc_calibration_mode_enable(uint16_t acc_trim, confirm_state new_state);
Function description	ACC calibration mode enable
Input parameter 1	acc_trim: calibration mode selection ACC_CAL_HICKCAL or ACC_CAL_HICKTRIM
Input parameter 2	new_state: Enable or disable ACC
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

acc_trim

Calibration mode selection

ACC_CAL_HICKCAL: Coarse calibration mode

ACC_CAL_HICKTRIM: Fine calibration mode

new_state

Enable or disable ACC

FALSE: Disabled

TRUE: Enabled

Example:

```
/* open acc calibration */
acc_calibration_mode_enable(ACC_CAL_HICKTRIM, TRUE);
```

5.1.2 acc_step_set function

The table below describes the function acc_step_set.

Table 8. acc_step_set function

Name	Description
Function name	acc_step_set
Function prototype	void acc_step_set(uint8_t step_value);
Function description	Configure ACC calibration step length
Input parameter 1	step_value: step value for calibration
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

step_value

This 4-bit field defines the value to be changed for each calibration.

Note: To obtain better calibration accuracy, it is recommended to set the step value to 1.

When ENTRIM=0, only HICKCAL is calibrated. If the step value is incremented or decremented by one, the corresponding HICKCAL follows the change rule (increased or decreased by one), and the HICK frequency will increase or decrease by 40 KHz (design value), meaning a positive correlation between them.

When ENTRIM=1, only the HICKTRIM is calibrated. If the step value is incremented or decremented by one, the corresponding HICKTRIM follows the change rule (increased or decreased by one), and the HICK will increase or decrease by 20 KHz(design value), meaning a positive correlation between them.

Example:

```
/* set acc step value */
acc_step_set(0x1);
```

5.1.3 acc_interrupt_enable function

The table below describes the function acc_interrupt_enable.

Table 9. acc_interrupt_enable function

Name	Description
Function name	dma_ interrupt_enable
Function prototype	void acc_interrupt_enable(uint16_t acc_int, confirm_state new_state);
Function description	Enable acc interrupts
Input parameter 1	acc_int: interrupt source selection
Input parameter 2	new_state: enable or disable interrupts
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

acc_int

Interrupt source selection

ACC_CALRDYIEN_INT: Calibration complete interrupt

ACC_EIEN_INT: Reference signal lost interrupt

new_state

Enable or disable interrupts

FALSE: Interrupt disabled

TRUE: Interrupt enabled

Example:

```
/* enable the acc reference signal lost interrupt */
acc_interrupt_enable(ACC_EIEN_INT, TRUE);
```

5.1.4 acc_hicktrim_get function

The table below describes the function acc_hicktrim_get.

Table 10. acc_hicktrim_get function

Name	Description
Function name	acc_hicktrim_get
Function prototype	uint8_t acc_hicktrim_get(void);
Function description	Get acc trimming calibration value
Input parameter	NA
Output parameter	NA
Return value	Return acc trimming calibration value
Required preconditions	NA
Called functions	NA

Example:

```
/* get trim value*/
uint8_t trim_value;
trim_value = acc_hicktrim_get();
```

5.1.5 acc_hickcal_get function

The table below describes the function acc_hicktrim_get.

Table 11. acc_hickcal_get function

Name	Description
Function name	acc_hickcal_get
Function prototype	uint8_t acc_hickcal_get(void);
Function description	Get acc coarse calibration value
Input parameter	NA
Output parameter	NA
Return value	Return acc coarse calibration value
Required preconditions	NA
Called functions	NA

Example:

```
/* get cal value*/
uint8_t cal_value;
cal_value = acc_hickcal_get ();
```

5.1.6 acc_write_c1 function

The table below describes the function acc_write_c1.

Table 12. acc_write_c1 function

Name	Description
Function name	acc_write_c1
Function prototype	void acc_write_c1(uint16_t acc_c1_value);
Function description	Write ACC C1 register value
Input parameter	acc_c1_value: the value to be written in ACC C1 register
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* update the c1 value */
acc_c2_value = 8000;
acc_write_c1(acc_c2_value - 10);
```

5.1.7 acc_write_c2 function

The table below describes the function acc_write_c2.

Table 13. acc_write_c2 function

Name	Description
Function name	acc_write_c2
Function prototype	void acc_write_c2(uint16_t acc_c2_value);
Function description	Write ACC C2 register value
Input parameter	acc_c2_value: the value to be written in ACC C2 register
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* update the c2 value */
acc_c2_value = 8000;
acc_write_c2(acc_c2_value - 10);
```

5.1.8 acc_write_c3 function

The table below describes the function acc_write_c3.

Table 14. acc_write_c3 function

Name	Description
Function name	acc_write_c3
Function prototype	void acc_write_c3(uint16_t acc_c3_value);
Function description	Write ACC C3 register value
Input parameter	acc_c3_value: the value to be written in ACC C3 register
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* update the c3 value */
acc_c2_value = 8000;
acc_write_c3(acc_c2_value - 10);
```

5.1.9 acc_read_c1 function

The table below describes the function acc_read_c1.

Table 15. acc_read_c1 function

Name	Description
Function name	acc_read_c1
Function prototype	uint16_t acc_read_c1(void);
Function description	Read ACC C1 register value
Input parameter	NA
Output parameter	NA
Return value	ACC C1 register value
Required preconditions	NA
Called functions	NA

Example:

```
/* get the c1 value */
uint16_t    acc_c1_value;
acc_c1_value = acc_read_c1();
```

5.1.10 acc_read_c2 function

The table below describes the function acc_read_c2.

Table 16. acc_read_c2 function

Name	Description
Function name	acc_read_c2
Function prototype	uint16_t acc_read_c2(void);
Function description	Read ACC C2 register value
Input parameter	NA
Output parameter	NA
Return value	ACC C2 register value
Required preconditions	NA
Called functions	NA

Example:

```
/* get the c2 value */
uint16_t    acc_c2_value;
acc_c2_value = acc_read_c2();
```

5.1.11 acc_read_c3 function

The table below describes the function acc_read_c3.

Table 17. acc_read_c3 function

Name	Description
Function name	acc_read_c3
Function prototype	uint16_t acc_read_c3(void);
Function description	Read ACC C3 register value
Input parameter	NA
Output parameter	NA
Return value	ACC C3 register value
Required preconditions	NA
Called functions	NA

Example:

```
/* get the c3 value */
uint16_t    acc_c3_value;
acc_c3_value = acc_read_c3();
```

5.1.12 acc_flag_get function

The table below describes the function acc_flag_get.

Table 18. acc_flag_get function

Name	Description
Function name	acc_flag_get
Function prototype	flag_status acc_flag_get(uint16_t acc_flag);
Function description	Get acc flag status
Input parameter 1	acc_flag: ACC flag selection
Output parameter	NA
Return value	flag_status: indicates whether or not the flag has been set
Required preconditions	NA
Called functions	NA

acc_flag

The acc_flag is used for flag selection, including:

ACC_RSLOST_FLAG: Reference signal lost interrupt

ACC_CALRDY_FLAG: Calibration complete interrupt

flag_status

RESET: Corresponding flag bit is not set

SET: Corresponding flag bit is set

Example:

```
if(acc_flag_get(ACC_CALRDY_FLAG) != RESET)
{
   at32_led_toggle(LED2);
   /* clear acc calibration ready flag */
   acc_flag_clear(ACC_CALRDY_FLAG);
}
```

5.1.13 acc_flag_clear function

The table below describes the function acc_flag_clear.

Table 19. acc_flag_clear function

Name	Description
Function name	acc_flag_clear
Function prototype	void acc_flag_clear(uint16_t acc_flag);
Function description	Clear acc flag
Input parameter 1	acc_flag: ACC flag selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

acc_flag

The acc_flag is used for flag selection, including:

ACC_RSLOST_FLAG: Reference signal lost interrupt

ACC_CALRDY_FLAG: Calibration complete interrupt

Example:

```
if(acc_flag_get(ACC_CALRDY_FLAG) != RESET)
{
    at32_led_toggle(LED2);
    /* clear acc calibration ready flag */
    acc_flag_clear(ACC_CALRDY_FLAG);
}
```

5.2 Analog-to-digital converter (ADC)

The ADC register structure adc_type is defined in the "at32f413_adc.h".

```
/**
  * @brief type define adc register all
  */
typedef struct
{
      ......
} adc_type;
```

The table below gives a list of the ADC registers.

Table 20. Summary of ADC registers

Register	Description
sts	ADC status register
ctrl1	ADC control register 1
ctrl2	ADC control register 2
spt1	ADC sample time register 1
spt2	ADC sample time register 2
pcdto1	ADC preempted channel data offset register 1
pcdto2	ADC preempted channel data offset register 2
pcdto3	ADC preempted channel data offset register 3
pcdto4	ADC preempted channel data offset register 4
vmhb	ADC voltage monitor high boundary register
vmlb	ADC voltage monitor low boundary register
osq1	ADC ordinary sequence register 1
osq2	ADC ordinary sequence register 2
osq3	ADC ordinary sequence register 3
psq	ADC preempted sequence register
pdt1	ADC preempted data register 1
pdt2	ADC preempted data register 2
pdt3	ADC preempted data register 3
pdt4	ADC preempted data register 4
odt	ADC ordinary data register

The table below gives a list of the ADC library functions.

Table 21. Summary of ADC library functions

Function name	Description
adc_reset	Reset all ADC registers to their reset values
adc_enable	Enable A/D converter
adc_combine_mode_select	Select master/slave mode
adc_base_default_para_init	Define an initial valur for adc_base_struct
adc_base_config	Configure ADC registers with the initialized parameters of the adc_base_struct
adc_dma_mode_enable	Enable DMA transfer for ordinary group
adc_interrupt_enable	Enable the selected ADC event interrupt
adc_calibration_init	Initialization calibration
adc_calibration_init_status_get	Get initialization calibration status
adc_calibration_start	Start calibration
adc_calibration_status_get	Get calibration status
adc_voltage_monitor_enable	Enable voltage monitoring for ordinary/preempted channels and a single channel
adc_voltage_monitor_threshold_value_set	Set the threshold of voltage monitoring
adc_voltage_monitor_single_channel_select	Select a single channel for voltage monitoring
adc_ordinary_channel_set	Configure ordinary channels, including channel selection, conversion sequence number and sampling time
adc_preempt_channel_length_set	Configure the length of preempted group conversion sequence
adc_preempt_channel_set	Configure preempted channels, including channel selection, conversion sequence number and sampling time
adc_ordinary_conversion_trigger_set	Enable trigger mode and trigger event selection for ordinary conversion
adc_preempt_conversion_trigger_set	Enable trigger mode and trigger event selection for preempted conversion
adc_preempt_offset_value_set	Set data offset for preempted conversion
adc_ordinary_part_count_set	Set the number of ordinary channels for each triggered conversion in partition mode
adc_ordinary_part_mode_enable	Enable partition mode for ordinary channels
adc_preempt_part_mode_enable	Enable partition mode for preempted channels
adc_preempt_auto_mode_enable	Enable auto conversion of preempted group at the end of ordinary conversion
adc_tempersensor_vintrv_enable	Enable internal temperature sensor and V_{INTRV}
adc_ordinary_software_trigger_enable	Software trigger ordinary group conversion
adc_ordinary_software_trigger_status_get	Get the status of ordinary group conversion triggered by software
adc_preempt_software_trigger_enable	Software trigger preempted group conversion
adc_preempt_software_trigger_status_get	Get the status of preempted group conversion triggered by software
adc_ordinary_conversion_data_get	Get data of ordinary group conversion in non-master-slave mode
adc_combine_ordinary_conversion_data_get	Get data of ordinary group conversion in master-slave mode
adc_preempt_conversion_data_get	Get the converted data of preempted group
adc_flag_get	Get the status of flag bits
adc_flag_clear	Clear Clear flag bits

5.2.1 adc_reset function

The table below describes the function adc_reset.

Table 22. adc_reset function

Name	Description
Function name	adc_reset
Function prototype	void adc_reset(adc_type *adc_x)
Function description	Reset all ADC registers to their reset values
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset()

Example:

```
/* deinitialize adc1 */
adc_reset(ADC1);
```

5.2.2 adc_enable function

The table below describes the function adc_enable.

Table 23. adc_enable function

Name	Description
Function name	adc_enable
Function prototype	void adc_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable/disable A/D converter
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	new_state: indicates the pre-configured status of A/D converter This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable adc1 */
adc_enable(ADC1, TRUE);
```

Note: When the ADC is enabled, calling the function adc_enable triggers ordinary channel to start conversion.

5.2.3 adc_combine_mode_select function

The table below describes the function adc_combine_mode_select.

Table 24. adc_combine_mode_select function

Name	Description
Function name	adc_combine_mode_select
Function prototype	void adc_combine_mode_select(adc_combine_mode_type combine_mode)
Function description	Select ADC1 master/slave mode
Input parameter	combine_mode: indicates the master/slave mode supported by ADC1 This parameter can be any enumated value in the adc_combine_mode_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

combine_mode

The combine_mode is used to select master/slave combined mode, including:

ADC_INDEPENDENT_MODE: Non-combined (independent) mode

ADC_ORDINARY_SMLT_PREEMPT_SMLT_MODE:

Ordinary simultaneous + preempted simultaneous

ADC_ORDINARY_SMLT_PREEMPT_INTERLTRIG_MODE:

Ordinary simultaneous + interleaved preempted

ADC_ORDINARY_SHORTSHIFT_PREEMPT_SMLT_MODE:

Preempted simultaneous + ordinary group short shift

ADC_ORDINARY_LONGSHIFT_PREEMPT_SMLT_MODE:

Preempted simultaneous + ordinary group long shift

ADC_PREEMPT_SMLT_ONLY_MODE: Preempted simultaneous

ADC_ORDINARY_SMLT_ONLY_MODE: Ordinary simultaneous

ADC_ORDINARY_SHORTSHIFT_ONLY_MODE: Ordinary group short shift

ADC_ORDINARY_LONGSHIFT_ONLY_MODE: Ordinary group long shift

ADC_PREEMPT_INTERLTRIG_ONLY_MODE: interleaved preempted group trigger

Example:

```
/* select combine mode as independent mode */
adc_combine_mode_select(ADC_INDEPENDENT_MODE);
```

Note: The adc_combine_mode_select function is used for ADC1 noly and is invalid for ADC2.

5.2.4 adc_base_default_para_init function

The table below describes the function adc_base_default_para_init.

Table 25. adc_base_default_para_init function

Name	Description
Function name	adc_ base_default_para_init
Function prototype	void adc_base_default_para_init(adc_base_config_type *adc_base_struct)
Function description	Set the initial value for the adc_base_struct
Input parameter	adc_base_struct: adc_base_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The default values of members in the adc_base_struct:

sequence_mode: FALSE
repeat_mode: FALSE
data_align: ADC_RIGHT_ALIGNMENT
ordinary_channel_length: 1

Example:

```
/* initialize a adc_base_config_type structure */
adc_base_config_type adc_base_struct;
adc_base_default_para_init(&adc_base_struct);
```

5.2.5 adc_base_config function

The table below describes the function adc_base_config.

Table 26. adc_base_config function

Name	Description
Function name	adc_base_config
Function prototype	void adc_base_config(adc_type *adc_x, adc_base_config_type *adc_base_struct);
Function description	Initialize ADC registers with the specified parameters in the adc_base_struct
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_base_struct: adc_base_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_base_config_type structure

The adc_base_config_type is defined in the at32f413_adc.h.

```
typedef struct
{
    confirm_state                sequence_mode;
    confirm_state                repeat_mode;
```

adc_data_align_type data_align;
uint8_t ordinary_channel_length;
} adc_base_config_type; the member parameters are described as follows:

sequence_mode

Set ADC sequence mode

FALSE: Select a single channel for conversion

TRUE: Select multiple channels for conversion

repeat_mode

Set ADC repeat mode

FALSE: When SQEN=0 , trigger a single channel conversion each time; when SQEN=, trigger the conversion of a group of channels each time.

TRUE: When SQEN =0 repeatedly convert a single channel at each trigger; when SQEN=1, repeatedly convert a group of channels at each trigger until the ADCEN bit is cleared

data_align

Set data alignment of ADC

ADC_RIGHT_ALIGNMENT: right-aligned

ADC_LEFT_ALIGNMENT: left-aligned

ordinary_channel_length

Set the length of ordinary group ADC conversion

Example:

```
adc_base_config_type adc_base_struct;
adc_base_struct.sequence_mode = TRUE;
adc_base_struct.repeat_mode = FALSE;
adc_base_struct.data_align = ADC_RIGHT_ALIGNMENT;
adc_base_struct.ordinary_channel_length = 3;
adc_base_config(ADC1, &adc_base_struct);
```

5.2.6 adc_dma_mode_enable function

The table below describes the function adc_dma_mode_enable.

Table 27. adc_dma_mode_enable function

Name	Description
Function name	adc_dma_mode_enable
Function prototype	void adc_dma_mode_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable DMA transfer for ordinary group conversion
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	new_state: DMA pre-configured status of ordinary group in DMA transfer mode This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable dma transfer adc ordinary conversion data */
```

```
adc_dma_mode_enable(ADC1, TRUE);
```

Note: The function adc_dma_mode_enable is used for ADC1 only and is invalid for ADC2.

5.2.7 adc_interrupt_enable function

The table below describes the function adc_interrupt_enable.

Table 28. adc_interrupt_enable function

Name	Description
Function name	adc_interrupt_enable
Function prototype	void adc_interrupt_enable(adc_type *adc_x, uint32_t adc_int, confirm_state new_state)
Function description	Enable the selected ADC event interrupt
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_int: indicates the selected ADC This parameter is used to select any event interrupt supported by ADC.
Input parameter 3	new_state: indicates the pre-configured status of ADC event interrupts This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_int

The adc_int is used to select and set event interrupts, with the following parameters:

ADC_CCE_INT: Interrupt enabled at the end of group conversion

ADC_VMOR_INT: Interrupt enabled when voltage monitor is outside the threshold

ADC_PCCE_INT: Interrpt enabled at the end of preempted group conversion

Example:

```
/* enable voltage monitoring out of range interrupt */
adc_interrupt_enable(ADC1, ADC_VMOR_INT, TRUE);
```

5.2.8 adc_calibration_init function

The table below describes the function adc_calibration_init.

Table 29. adc_calibration_init function

Name	Description
Function name	adc_calibration_init
Function prototype	void adc_calibration_init(adc_type *adc_x)
Function description	Initialization calibration
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* initialize A/D calibration */
adc_calibration_init(ADC1);
```

5.2.9 adc_calibration_init_status_get function

The table below describes the function adc_calibration_init_status_get.

Table 30. adc_calibration_init_status_get function

Name	Description
Function name	adc_calibration_init_status_get
Function prototype	flag_status adc_calibration_init_status_get(adc_type *adc_x)
Function description	Get the status of initialization calibration
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	flag_status: indicates the status of calibration initialization Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* wait initialize A/D calibration success */
while(adc_calibration_init_status_get(ADC1));
```

5.2.10 adc_calibration_start function

The table below describes the function adc_calibration_start.

Table 31. adc_calibration_start function

Name	Description
Function name	adc_calibration_start
Function prototype	void adc_calibration_start(adc_type *adc_x)
Function description	Start calibration
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* start calibration process */
adc_calibration_start(ADC1);
```

5.2.11 adc_calibration_status_get function

The table below describes the function adc_calibration_status_get.

Table 32. adc_calibration_status_get function

Name	Description
Function name	adc_calibration_status_get
Function prototype	flag_status adc_calibration_status_get(adc_type *adc_x)
Function description	Get the status of calibration
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	flag_status: indicates the status of calibration Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* wait calibration success */
while(adc_calibration_status_get(ADC1));
```

5.2.12 adc_voltage_monitor_enable function

The table below describes the function adc_voltage_monitor_enable.

Table 33. adc_voltage_monitor_enable function

Name	Description
Function name	adc_voltage_monitor_enable
Function prototype	void adc_voltage_monitor_enable(adc_type *adc_x, adc_voltage_monitoring_type adc_voltage_monitoring)
Function description	Enable voltage monitor for ordinary/preempted group and a single channel
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_voltage_monitoring: select ordinary group, preempted group or a single channel for voltage monitoring This parameter can be any enumerated value in the adc_voltage_monitoring_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_voltage_monitoring

The adc_voltage_monitoring is used to select one or more channels of ordinary group/preempted group for voltage monitoring, including:

ADC_VMONITOR_SINGLE_ORDINARY:

Select a single ordinary channel for voltage monitoring

ADC_VMONITOR_SINGLE_PREEMPT:

Select a single preempted channel for voltage monitoring

ADC_VMONITOR_SINGLE_ORDINARY_PREEMPT:
Select a single channel from ordinary or preempted group for voltage monitoring
ADC_VMONITOR_ALL_ORDINARY:
Select all ordinary channels for voltage monitoring
ADC_VMONITOR_ALL_PREEMPT:
Select all preempted channels for voltage monitoring
ADC_VMONITOR_ALL_ORDINARY_PREEMPT:
Select all ordinary and preempted channels for voltage monitoring
ADC_VMONITOR_NONE:
No channels need voltage monitoring

Example:

```
/* enable the voltage monitoring on all ordinary and preempt channels */
adc_voltage_monitor_enable(ADC1, ADC_VMONITOR_ALL_ORDINARY_PREEMPT);
```

5.2.13 adc_voltage_monitor_threshold_value_set function

The table below describes the function adc_voltage_monitor_threshold_value_set.

Table 34. adc_voltage_monitor_threshold_value_set function

Name	Description
Function name	adc_voltage_monitor_threshold_value_set
Function prototype	void adc_voltage_monitor_threshold_value_set(adc_type *adc_x, uint16_t adc_high_threshold, uint16_t adc_low_threshold)
Function description	Configure the threshold of voltage monitoring
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_high_threshold: indicates the upper limit for voltage monitoring This parameter can be any value between 0x000~0xFFF.
Input parameter 3	adc_low_threshold: indicates the lower limit for voltage monitoring This parameter can be any value lower than that of adc_high_threshold in the range of 0x000~0xFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set voltage monitoring's high and low thresholds value */
adc_voltage_monitor_threshold_value_set(ADC1, 0xBBB, 0xAAA);
```

5.2.14 adc_voltage_monitor_single_channel_select function

The table below describes the function adc_voltage_monitor_single_channel_select.

Table 35. adc_voltage_monitor_single_channel_select function

Name	Description
Function name	adc_voltage_monitor_single_channel_select
Function prototype	void adc_voltage_monitor_single_channel_select(adc_type *adc_x, adc_channel_select_type adc_channel)
Function description	Select a single channel for voltage monitoring
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_channel: select a single channel for voltage monitoring Refer to *adc_channel* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_channel

The adc_channel is used to select a single channel for voltage monitoring, including

ADC_CHANNEL_0: ADC channel 0

ADC_CHANNEL_1: ADC channel 1

……

ADC_CHANNEL_16: ADC channel 16

ADC_CHANNEL_17: ADC channel 17

Example:

```
/* select the voltage monitoring's channel */
adc_voltage_monitor_single_channel_select(ADC1, ADC_CHANNEL_5);
```

5.2.15 adc_ordinary_channel_set function

The table below describes the function adc_ordinary_channel_set.

Table 36. adc_ordinary_channel_set function

Name	Description
Function name	adc_ordinary_channel_set
Function prototype	void adc_ordinary_channel_set(adc_type *adc_x, adc_channel_select_type adc_channel, uint8_t adc_sequence, adc_sampletime_select_type adc_sampletime)
Function description	Configure ordinary channels, including parameters such as channel selection, conversion sequence number and sampling time
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_channel: indicates the selected channel Refer to *adc_channel* for details.
Input parameter 3	adc_sequence: defines the sequence of channel conversion This parameter can be any value from 1~16.

Name	Description
Input parameter 4	adc_sampletime: defines the sampling time for channel Refer to *adc_sampletime* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_sampletime

The adc_sampletime is used to configure the sampling time of channels, including:

ADC_SAMPLETIME_1_5: sampling time is 1.5 ADCCLK cycles
ADC_SAMPLETIME_7_5: sampling time is 7.5 ADCCLK cycles
ADC_SAMPLETIME_13_5: sampling time is 13.5 ADCCLK cycles
ADC_SAMPLETIME_28_5: sampling time is 28.5 ADCCLK cycles
ADC_SAMPLETIME_41_5: sampling time is 41.5 ADCCLK cycles
ADC_SAMPLETIME_55_5: sampling time is 55.5 ADCCLK cycles
ADC_SAMPLETIME_71_5: sampling time is 71.5 ADCCLK cycles
ADC_SAMPLETIME_239_5: sampling time is 239.5 ADCCLK cycles

Example:

```
/* set ordinary channel's corresponding rank in the sequencer and sample time */
adc_ordinary_channel_set(ADC1, ADC_CHANNEL_4, 1, ADC_SAMPLETIME_239_5);
adc_ordinary_channel_set(ADC1, ADC_CHANNEL_5, 2, ADC_SAMPLETIME_239_5);
```

5.2.16 adc_preempt_channel_length_set function

The table below describes the function adc_preempt_channel_length_set.

Table 37. adc_preempt_channel_length_set function

Name	Description
Function name	adc_preempt_channel_length_set
Function prototype	void adc_preempt_channel_length_set(adc_type *adc_x, uint8_t adc_channel_lenght)
Function description	Set the length of preempted channel conversion
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_channel_lenght: set the length of preempted channel conversion This parameter can be any value from 0x1~0x4.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set preempt channel lenghth */
adc_preempt_channel_length_set(ADC1, 3);
```

5.2.17 adc_preempt_channel_set function

The table below describes the function adc_preempt_channel_set.

Table 38. adc_preempt_channel_set function

Name	Description
Function name	adc_preempt_channel_set
Function prototype	void adc_preempt_channel_set(adc_type *adc_x, adc_channel_select_type adc_channel, uint8_t adc_sequence, adc_sampletime_select_type adc_sampletime)
Function description	Configure preempted group, including parameters such as channel selection, conversion sequence number and sampling time
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_channel: indicates the selected channel Refer to *adc_channel* for details.
Input parameter 3	adc_sequence: set the sequence number for channel conversion This parameter can be any value from 1~4.
Input parameter 4	adc_sampletime: set the sampling time for channels Refer to *adc_sampletime* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set ordinary channel's corresponding rank in the sequencer and sample time */
adc_preempt_channel_set(ADC1, ADC_CHANNEL_7, 1, ADC_SAMPLETIME_239_5);
adc_preempt_channel_set(ADC1, ADC_CHANNEL_8, 2, ADC_SAMPLETIME_239_5);
```

5.2.18 adc_ordinary_conversion_trigger_set function

The table below describes the function adc_ordinary_conversion_trigger_set.

Table 39. adc_ordinary_conversion_trigger_set function

Name	Description
Function name	adc_ordinary_conversion_trigger_set
Function prototype	void adc_ordinary_conversion_trigger_set(adc_type *adc_x, adc_ordinary_trig_select_type adc_ordinary_trig, confirm_state new_state)
Function description	Enable trigger mode and select trigger events for ordinary group conversion
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_ordinary_trig: indicates the selected trigger event for ordinary group This parameter can be any enumerated value in the adc_ordinary_trig_select_type.
Input parameter 3	new_state: indicates the pre-configured status of trigger mode This parameter can be TRUE or FALSE.
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

adc_ordinary_trig

The adc_ordinary_trig is used to select a trigger event for ordinary group conversion, including:

ADC1 &ADC2 trigger events

ADC12_ORDINARY_TRIG_TMR1CH1: TMR1 CH1 event
ADC12_ORDINARY_TRIG_TMR1CH2: TMR1 CH2 event
ADC12_ORDINARY_TRIG_TMR1CH3: TMR1 CH3 event
ADC12_ORDINARY_TRIG_TMR2CH2: TMR2 CH2 event
ADC12_ORDINARY_TRIG_TMR3TRGOUT: TMR3 TRGOUT event
ADC12_ORDINARY_TRIG_TMR4CH4: TMR4 CH4 event
ADC12_ORDINARY_TRIG_EXINT11_TMR8TRGOUT: EXINT 11/TMR8 TRGOUT event
ADC12_ORDINARY_TRIG_SOFTWARE: Software trigger event
ADC12_ORDINARY_TRIG_TMR1TRGOUT: TMR1 TRGOUT event
ADC12_ORDINARY_TRIG_TMR8CH1: TMR8 CH1 event
ADC12_ORDINARY_TRIG_TMR8CH2: TMR8 CH2 event

Example:

```
/* set ordinary external trigger event */
adc_ordinary_conversion_trigger_set(ADC1, ADC12_ORDINARY_TRIG_TMR1CH1, TRUE);
```

5.2.19 adc_preempt_conversion_trigger_set function

The table below describes the function adc_preempt_conversion_trigger_set.

Table 40. adc_preempt_conversion_trigger_set function

Name	Description
Function name	adc_preempt_conversion_trigger_set
Function prototype	void adc_preempt_conversion_trigger_set(adc_type *adc_x, adc_preempt_trig_select_type adc_preempt_trig, confirm_state new_state)
Function description	Enable trigger mode and trigger events for preempted group conversion
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_ preempt _trig: selected trigger event for preempted group This parameter can be any enumerated value in the adc_preempt_trig_select_type.
Input parameter 3	new_state: indicates the pre-configured status of trigger mode This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_preempt _trig

The adc_preempt _trig is used to select a trigger event for preempted group conversion, including ADC1 &ADC2 trigger events

ADC12_PREEMPT_TRIG_TMR1TRGOUT:	TMR1 TRGOUT event
ADC12_PREEMPT_TRIG_TMR1CH4:	TMR1 CH4 event
ADC12_PREEMPT_TRIG_TMR2TRGOUT:	TMR2 TRGOUT event
ADC12_PREEMPT_TRIG_TMR2CH1:	TMR2 CH1 event
ADC12_PREEMPT_TRIG_TMR3CH4:	TMR3 CH4 event
ADC12_PREEMPT_TRIG_TMR4TRGOUT:	TMR4 TRGOUT event
ADC12_PREEMPT_TRIG_EXINT15_TMR8CH4:	EXINT 15/TMR8 CH4 event
ADC12_PREEMPT_TRIG_SOFTWARE:	Software trigger event
ADC12_PREEMPT_TRIG_TMR1CH1:	TMR1 CH1 event
ADC12_PREEMPT_TRIG_TMR8CH1:	TMR8 CH1 event
ADC12_PREEMPT_TRIG_TMR8TRGOUT:	TMR8 TRGOUT event

Example:

```
/* set preempt external trigger event */
adc_preempt_conversion_trigger_set(ADC1, ADC12_PREEMPT_TRIG_SOFTWARE, TRUE);
```

5.2.20 adc_preempt_offset_value_set function

The table below describes the function adc_preempt_offset_value_set.

Table 41. adc_preempt_offset_value_set function

Name	Description
Function name	adc_preempt_offset_value_set
Function prototype	void adc_preempt_offset_value_set(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel, uint16_t adc_offset_value)
Function description	Set the offset value of preempted group conversion
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_preempt_channel: indicates the selected channel Refer to *adc_preempt_channel* for details.
Input parameter 3	adc_offset_value: set the offset value for the selected channel This parameter can be any value from 0x000~0xFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_preempt_channel

The adc_preempt_channel is used to set an offset value for the selected channel, including

ADC_PREEMPT_CHANNEL_1:	Preempted channel 1
ADC_PREEMPT_CHANNEL_2:	Preempted channel 2
ADC_PREEMPT_CHANNEL_3:	Preempted channel 3
ADC_PREEMPT_CHANNEL_4:	Preempted channel 4

Example:

```
/* set preempt channel's conversion value offset */
adc_preempt_offset_value_set(ADC1, ADC_PREEMPT_CHANNEL_1, 0x111);
adc_preempt_offset_value_set(ADC1, ADC_PREEMPT_CHANNEL_2, 0x222);
adc_preempt_offset_value_set(ADC1, ADC_PREEMPT_CHANNEL_3, 0x333);
```

5.2.21 adc_ordinary_part_count_set function

The table below describes the function adc_ordinary_part_count_set.

Table 42. adc_ordinary_part_count_set function

Name	Description
Function name	adc_ordinary_part_count_set
Function prototype	void adc_ordinary_part_count_set(adc_type *adc_x, uint8_t adc_channel_count)
Function description	Set the number of ordinary channels at each triggered conversion in partition mode
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_channel_count: indicates the number of ordinary group in partition mode This parameter can be any value from 0x1~0x8.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set partitioned mode channel count */
adc_ordinary_part_count_set(ADC1, 2);
```

Note: In partition mode, only the number of ordinary group is configurable, and that of preempted group is fixed 1.

5.2.22 adc_ordinary_part_mode_enable function

The table below describes the function adc_ordinary_part_mode_enable.

Table 43. adc_ordinary_part_mode_enable function

Name	Description
Function name	adc_ordinary_part_mode_enable
Function prototype	void adc_ordinary_part_mode_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable partition mode for ordinary channels
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	new_state: indicates the pre-configured status for partition mode of ordinary channels This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the partitioned mode on ordinary channel */
adc_ordinary_part_mode_enable(ADC1, TRUE);
```

5.2.23 adc_preempt_part_mode_enable function

The table below describes the function adc_preempt_part_mode_enable.

Table 44. adc_preempt_part_mode_enable function

Name	Description
Function name	adc_preempt_part_mode_enable
Function prototype	void adc_preempt_part_mode_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable partition mode for preempted channels
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	new_state: indicates the pre-configured status for partition mode of preempted channels This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the partitioned mode on preempt channel */
adc_preempt_part_mode_enable(ADC1, TRUE);
```

5.2.24 adc_preempt_auto_mode_enable function

The table below describes the function adc_preempt_auto_mode_enable.

Table 45. adc_preempt_auto_mode_enable function

Name	Description
Function name	adc_preempt_auto_mode_enable
Function prototype	void adc_preempt_auto_mode_enable(adc_type *adc_x, confirm_state,new_state)
Function description	Enable auto preempted group conversion at the end of ordinary group conversion
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	new_state: indicates the pre-configured status for auto preempted group conversion This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable automatic preempt group conversion */
adc_preempt_auto_mode_enable(ADC1, TRUE);
```

5.2.25 adc_tempersensor_vintrv_enable function

The table below describes the function adc_tempersensor_vintrv_enable.

Table 46. adc_tempersensor_vintrv_enable function

Name	Description
Function name	adc_tempersensor_vintrv_enable
Function prototype	void adc_tempersensor_vintrv_enable(confirm_state new_state)
Function description	Enable internal temperature sensor and V_{INTRV}
Input parameter	new_state: indicates the pre-configured status or internal temperature sensor and V_{INTRV} This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the temperature sensor and vintrv channel */
adc_tempersensor_vintrv_enable(TRUE);
```

5.2.26 adc_ordinary_software_trigger_enable function

The table below describes the function adc_ordinary_software_trigger_enable.

Table 47. adc_ordinary_software_trigger_enable function

Name	Description
Function name	adc_ordinary_software_trigger_enable
Function prototype	void adc_ordinary_software_trigger_enable(adc_type *adc_x, confirm_state new_state)
Function description	Trigger ordinary group conversion by software
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	new_state: indicates the pre-configured status for software-triggered ordinary group conversion This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable ordinary software start conversion */
adc_ordinary_software_trigger_enable(ADC1, TRUE);
```

5.2.27 adc_ordinary_software_trigger_status_get function

The table below describes the function adc_ordinary_software_trigger_status_get.

Table 48. adc_ordinary_software_trigger_status_get function

Name	Description
Function name	adc_ordinary_software_trigger_status_get
Function prototype	flag_status adc_ordinary_software_trigger_status_get(adc_type *adc_x)
Function description	Get the status of software-triggered ordinary group conversion
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	flag_status: indicates the status of software-triggered ordinary group conversion Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* wait ordinary software start conversion */
while(adc_ordinary_software_trigger_status_get(ADC1));
```

5.2.28 adc_preempt_software_trigger_enable function

The table below describes the function adc_preempt_software_trigger_enable.

Table 49. adc_preempt_software_trigger_enable function

Name	Description
Function name	adc_preempt_software_trigger_enable
Function prototype	void adc_preempt_software_trigger_enable(adc_type *adc_x, confirm_state new_state)
Function description	Preempted group conversion triggered by software
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	new_state: indicates the pre-configured status of software-triggered preempted group conversion This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable preempt software start conversion */
adc_preempt_software_trigger_enable(ADC1, TRUE);
```

5.2.29 adc_preempt_software_trigger_status_get function

The table below describes the function adc_preempt_software_trigger_status_get.

Table 50. adc_preempt_software_trigger_status_get function

Name	Description
Function name	adc_preempt_software_trigger_status_get
Function prototype	flag_status adc_preempt_software_trigger_status_get(adc_type *adc_x)
Function description	Get the status of software-triggered preempted group conversion
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	flag_status: indicates the status of software-triggered preempted group conversion Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* wait preempt software start conversion */
while(adc_preempt_software_trigger_status_get(ADC1));
```

5.2.30 adc_ordinary_conversion_data_get function

The table below describes the function adc_ordinary_conversion_data_get.

Table 51. adc_ordinary_conversion_data_get function

Name	Description
Function name	adc_ordinary_conversion_data_get
Function prototype	uint16_t adc_ordinary_conversion_data_get(adc_type *adc_x)
Function description	Get the converted data of ordinary group in non-master/slave mode
Input parameter	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Output parameter	NA
Return value	16-bit converted data by ordinary group
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t adc1_ordinary_index = 0;
adc1_ordinary_index = adc_ordinary_conversion_data_get(ADC1);
```

Note: This function can beused only when the ADC is in independent mode and each ADC is configured with a single channel.

5.2.31 adc_combine_ordinary_conversion_data_get function

The table below describes the function adc_combine_ordinary_conversion_data_get.

Table 52. adc_combine_ordinary_conversion_data_get function

Name	Description
Function name	adc_combine_ordinary_conversion_data_get
Function prototype	uint32_t adc_combine_ordinary_conversion_data_get(void)
Function description	Get the converted data of ordinary group in master/slave mode
Input parameter	NA
Output parameter	NA
Return value	32-bit converted data by ordinary group (high 16 bits for ADC2 and low 16 bits for ADC1).
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t common_ordinary_index = 0;
common_ordinary_index = adc_combine_ordinary_conversion_data_get();
```

Note: This function is used only when ADC is in master/slave mode and each ADC is configured with a single channel.

5.2.32 adc_preempt_conversion_data_get function

The table below describes the function adc_preempt_conversion_data_get.

Table 53. adc_preempt_conversion_data_get function

Name	Description
Function name	adc_preempt_conversion_data_get
Function prototype	uint16_t adc_preempt_conversion_data_get(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel)
Function description	Get the converted data of preempted group
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_preempt_channel: indicates the selected preempted channel Refer to *adc_preempt_channel* for details.
Output parameter	NA
Return value	16-bit converted data by preempted group
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t adc1_preempt_valuetab[3] = {0};
adc1_preempt_valuetab[0] = adc_preempt_conversion_data_get(ADC1, ADC_PREEMPT_CHANNEL_1);
adc1_preempt_valuetab[1] = adc_preempt_conversion_data_get(ADC1, ADC_PREEMPT_CHANNEL_2);
adc1_preempt_valuetab[2] = adc_preempt_conversion_data_get(ADC1, ADC_PREEMPT_CHANNEL_3);
```

5.2.33 adc_flag_get function

The table below describes the function adc_flag_get.

Table 54. adc_flag_get function

Name	Description
Function name	adc_flag_get
Function prototype	flag_status adc_flag_get(adc_type *adc_x, uint8_t adc_flag)
Function description	Get the status of the flag bit
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_flag: indicates the selected flag. Refer to *adc_flag* for details.
Output parameter	NA
Return value	flag_status: the status for the selected flag bit Return SET or RESET.
Required preconditions	NA
Called functions	NA

adc_flag

The adc_flag is used to select a flag to get its status, including

ADC_VMOR_FLAG: Voltage monitor outside threshold
ADC_CCE_FLAG: End of channel conversion
ADC_PCCE_FLAG: End of preempted channel conversion
ADC_PCCS_FLAG: Start of preempted channel conversion
ADC_OCCS_FLAG: Start of ordinary channel conversion

Example:

```
/* check if wakeup preempted channelsconversion end flag is set */
if(adc_flag_get(ADC1, ADC_PCCE_FLAG) != RESET)
```

5.2.34 adc_interrupt_flag_get function

The table below describes the function adc_interrupt_flag_get.

Table 55. adc_flag_clear function

Name	Description
Function name	adc_interrupt_flag_get
Function prototype	flag_status adc_interrupt_flag_get(adc_type *adc_x, uint8_t adc_flag)
Function description	Get the selected interrupt flag status
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_flag: indicates the selected flag. Refer to *adc_flag* for details.
Output parameter	NA
Return value	flag_status: the status for the selected flag bit Return SET or RESET.
Required preconditions	NA
Called functions	NA

adc_flag

The adc_flag is used to select a flag to get its status, including

ADC_VMOR_FLAG: Voltage monitor outside threshold

ADC_CCE_FLAG: End of channel conversion

ADC_PCCE_FLAG: End of preempted channel conversion

Example

```
/* check if wakeup preempted channelsconversion end flag is set */
if(adc_interrupt_flag_get(ADC1, ADC_PCCE_FLAG) != RESET)
```

5.2.35 adc_flag_clear function

The table below describes the function adc_flag_clear.

Table 56. adc_flag_clear function

Name	Description
Function name	adc_flag_clear
Function prototype	void adc_flag_clear(adc_type *adc_x, uint32_t adc_flag)
Function description	Clear the flag bits that have been set
Input parameter 1	adc_x: selected ADC peripheral This parameter can be ADC1 or ADC2.
Input parameter 2	adc_flag: select a flag to be cleard Refer to *adc_flag* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* preempted channelsconversion end flag clear */
adc_flag_clear(ADC1, ADC_PCCE_FLAG);
```

5.3 Battery powered registers (BPR)

The BPR register structure bpr_type is defined in the "at32f413_bpr.h".

```
/**
  * @brief type define bpr register all
  */
typedef struct
{


} bpr_type;
```

The table below gives a list of the BPR registers.

Table 57. Summary of BPR registers

Register	Description
dt1	Battery powered data register 1
dt2	Battery powered data register 2
dt3	Battery powered data register 3
dt4	Battery powered data register 4
dt5	Battery powered data register 5
dt6	Battery powered data register 6
dt7	Battery powered data register 7
dt8	Battery powered data register 8
dt9	Battery powered data register 9
dt10	Battery powered data register 10
rtccal	RTC calibration register
ctrl	BPR control register
ctrlsts	BPR control/status register
dt11	Battery powered data register 11
dt12	Battery powered data register 12
dt13	Battery powered data register 13
dt14	Battery powered data register 14
dt15	Battery powered data register 15
dt16	Battery powered data register 16
dt17	Battery powered data register 17
dt18	Battery powered data register 18
dt19	Battery powered data register 19
dt20	Battery powered data register 20
dt21	Battery powered data register 21
dt22	Battery powered data register 22
dt23	Battery powered data register 23
dt24	Battery powered data register 24
dt25	Battery powered data register 25
dt26	Battery powered data register 26
dt27	Battery powered data register 27

Register	Description
dt28	Battery powered data register 28
dt29	Battery powered data register 29
dt30	Battery powered data register 30
dt31	Battery powered data register 31
dt32	Battery powered data register 32
dt33	Battery powered data register 33
dt34	Battery powered data register 34
dt35	Battery powered data register 35
dt36	Battery powered data register 36
dt37	Battery powered data register 37
dt38	Battery powered data register 38
dt39	Battery powered data register 39
dt40	Battery powered data register 40
dt41	Battery powered data register 41
dt42	Battery powered data register 42

The table below gives a list of the BPR library functions.

Table 58. Summary of BPR library functions

Function name	Description
bpr_reset	Reset all battery powered data registers to their reset values
bpr_flag_get	Get the flag
bpr_flag_clear	Clear the flag
bpr_interrupt_enable	Enable tamper detection interrupt
bpr_data_read	Read data from battery powered data registers
bpr_data_write	Write data into battery powered data registers
bpr_rtc_output_select	Configure event output
bpr_rtc_clock_calibration_value_set	Configure clock calibration
bpr_tamper_pin_enable	Enable tamper detection
bpr_tamper_pin_active_level_set	Configure tamper detection active level

5.3.1 bpr_reset function

The table below describes the function bpr_reset.

Table 59. bpr_reset function

Name	Description
Function name	bpr_reset
Function prototype	void bpr_reset(void);
Function description	Reset all battery powered data registers to their reset values
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_battery_powered_domain_reset(confirm_state new_state);

Example:

```
bpr_reset();
```

5.3.2 bpr_flag_get function

The table below describes the function bpr_flag_get.

Table 60. bpr_flag_get function

Name	Description
Function name	bpr_flag_get
Function prototype	flag_status bpr_flag_get(uint32_t flag);
Function description	Get the status of flag bit
Input parameter 1	flag: flag selection Refer to the “flag” description below for details.
Output parameter	NA
Return value	flag_status: status of flag Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This is used to select a flag and get its status, including

BPR_TAMPER_INTERRUPT_FLAG: Tamper detection interrupt

BPR_TAMPER_EVENT_FLAG: Tamper detection event

Example:

```
bpr_flag_get(BPR_TAMPER_INTERRUPT_FLAG);
```

5.3.3 bpr_interrupt_flag_get function

The table below describes the function bpr_interrupt_flag_get.

Table 61. bpr_interrupt_flag_get function

Name	Description
Function name	bpr_interrupt_flag_get
Function prototype	flag_status bpr_interrupt_flag_get(uint32_t flag);
Function description	Get the status of flag bit and judge the corresponding interrupt enable bit
Input parameter 1	flag: flag selection Refer to the “flag” description below for details.
Output parameter	NA
Return value	flag_status: status of flag Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This is used to select a flag and get its status, including

BPR_TAMPER_INTERRUPT_FLAG: Tamper detection interrupt

BPR_TAMPER_EVENT_FLAG: Tamper detection event

Example

```
bpr_interrupt_flag_get(BPR_TAMPER_INTERRUPT_FLAG);
```

5.3.4 bpr_flag_clear function

The table below describes the function bpr_flag_clear.

Table 62. bpr_flag_clear function

Name	Description
Function name	bpr_flag_clear
Function prototype	void bpr_flag_clear(uint32_t flag);
Function description	Clear the flag
Input parameter 1	flag: the flag to be cleared Refer to the “flag” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

This is used to clear the selected flag, including:

BPR_TAMPER_INTERRUPT_FLAG: Tamper detection interrupt

BPR_TAMPER_EVENT_FLAG: Tamper detection event

Example:

```
bpr_flag_clear(BPR_TAMPER_INTERRUPT_FLAG);
```

5.3.5 bpr_interrupt_enable function

The table below describes the function bpr_interrupt_enable.

Table 63. bpr_interrupt_enable function

Name	Description
Function name	bpr_interrupt_enable
Function prototype	void bpr_interrupt_enable(confirm_state new_state);
Function description	Enable tamper detection interrupt
Input parameter 1	new_state: enable or disable tamper detection interrupt This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
bpr_interrupt_enable(TRUE);
```

5.3.6 bpr_data_read function

The table below describes the function bpr_data_read.

Table 64. bpr_data_read function

Name	Description
Function name	bpr_data_read
Function prototype	uint16_t bpr_data_read(bpr_data_type bpr_data);
Function description	Read data from battery powered data registers
Input parameter 1	bpr_data: selected battery powered data registers Refer to the “bpr_data” description below for details.
Output parameter	NA
Return value	Return the data read from battery powered data register.
Required preconditions	NA
Called functions	NA

bpr_data

This is used to select a data register.

BPR_DATA1: Battery powered data register 1

BPR_DATA2: Battery powered data register 2

BPR_DATA41: Battery powered data register 41

BPR_DATA42: Battery powered data register 42

Example:

```
bpr_data_read(BPR_DATA1);
```

5.3.7 bpr_data_write function

The table below describes the function bpr_data_write.

Table 65. bpr_data_write function

Name	Description
Function name	bpr_data_write
Function prototype	void bpr_data_write(bpr_data_type bpr_data, uint16_t data_value);
Function description	Write data into battery powered data registers
Input parameter 1	bpr_data: selected battery powered data registers Refer to the “bpr_data” description below for details.
Input parameter 2	data_value: 16-bit data
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

bpr_data

This is used to select a data register.

BPR_DATA1: Battery powered data register 1

BPR_DATA2: Battery powered data register 2

BPR_DATA41: Battery powered data register 41

BPR_DATA42: Battery powered data register 42

Example:

```
bpr_data_write(BPR_DATA1, 0x5A5A);
```

5.3.8 bpr_rtc_output_select function

The table below describes the function bpr_rtc_output_select.

Table 66. bpr_rtc_output_select function

Name	Description
Function name	bpr_rtc_output_select
Function prototype	void bpr_rtc_output_select(bpr_rtc_output_type output_source);
Function description	Configure event output
Input parameter 1	output_source: output event Refer to the “output_source” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

output_source

This is used to select an output event.

BPR_RTC_OUTPUT_NONE: No output

BPR_RTC_OUTPUT_CLOCK_CAL_BEFORE:

Output the RTC clock with a frequency divided by 64 before calibration

BPR_RTC_OUTPUT_ALARM: Pulse output alarm event

BPR_RTC_OUTPUT_SECOND: Pulse output second event
BPR_RTC_OUTPUT_CLOCK_CAL_AFTER:
Output the RTC clock with a frequency divided by 64 after calibraion
BPR_RTC_OUTPUT_ALARM_TOGGLE: Toggle output alarm event
BPR_RTC_OUTPUT_SECOND_TOGGLE: Toggle output second event

Example:

```
bpr_rtc_output_select(BPR_RTC_OUTPUT_ALARM);
```

5.3.9 bpr_rtc_clock_calibration_value_set function

The table below describes the function bpr_rtc_clock_calibration_value_set.

Table 67. bpr_rtc_clock_calibration_value_set function

Name	Description
Function name	bpr_rtc_clock_calibration_value_set
Function prototype	void bpr_rtc_clock_calibration_value_set(uint8_t calibration_value);
Function description	Configure clock calibration
Input parameter 1	value: calibration value, range: 0~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
bpr_rtc_clock_calibration_value_set(0x7F);
```

5.3.10 bpr_tamper_pin_enable function

The table below describes the function bpr_tamper_pin_enable.

Table 68. bpr_tamper_pin_enable function

Name	Description
Function name	bpr_tamper_pin_enable
Function prototype	void bpr_tamper_pin_enable(confirm_state new_state);
Function description	Enable tamper detection
Input parameter 1	new_state: enable or disable interrupt This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
bpr_tamper_pin_enable(TRUE);
```

5.3.11 bpr_tamper_pin_active_level_set function

The table below describes the function bpr_tamper_pin_active_level_set.

Table 69. bpr_tamper_pin_active_level_set function

Name	Description
Function name	bpr_tamper_pin_active_level_set
Function prototype	void bpr_tamper_pin_active_level_set(bpr_tamper_pin_active_level_type active_level);
Function description	Configure tamper detection active level
Input parameter 1	active_level: tamper detection active level Refer to the “active_level” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

active_level

It is used to select the tamper detection active level.

BPR_TAMPER_PIN_ACTIVE_HIGH: Active high

BPR_TAMPER_PIN_ACTIVE_LOW: Active low

Example:

```
bpr_tamper_pin_active_level_set(BPR_TAMPER_PIN_ACTIVE_HIGH);
```

5.4 Controller area network (CAN)

The CAN register structure can_type is defined in the “at32f413_can.h”.

```
/**
  * @brief type define can register all
  */
typedef struct
{
    …
} can_type;
```

The table below gives a list of the CAN registers.

Table 70. Summary of CAN registers

Register	Description
mctrl	CAN master control register
msts	CAN master status register
tsts	CAN transmit status register
rf0	CAN receive FIFO 0 register
fr1	CAN receive FIFO 1 register
inten	CAN interrupt enable register
ests	CAN error status register
btmg	CAN bit timing register
tmi0	Transmit mailbox 0 identifier register
tmc0	Transmit mailbox 0 data length and time stamp register
tmdtl0	Transmit mailbox 0 data byte low register
tmdth0	Transmit mailbox 0 data byte high register
tmi1	Transmit mailbox 1 identifier register
tmc1	Transmit mailbox 1 data length and time stamp register
tmdtl1	Transmit mailbox 1 data byte low register
tmdth1	Transmit mailbox 1 data byte high register
tmi2	Transmit mailbox 2 identifier register
tmc2	Transmit mailbox 2 data length and time stamp register
tmdtl2	Transmit mailbox 2 data byte low register
tmdth2	Transmit mailbox 2 data byte high register
rfi0	Receive FIFO0 mailbox identifier register
rfc0	Receive FIFO0 mailbox data length and time stamp register
rfdtl0	Receive FIFO0 mailbox data byte low register
rfdth0	Receive FIFO0 mailbox data byte high register
rfi1	Receive FIFO1 mailbox identifier register
rfc1	Receive FIFO1 mailbox data length and time stamp register
rfdtl1	Receive FIFO1 mailbox data byte low register
rfdth1	Receive FIFO1 mailbox data byte high register
fctrl	CAN filter control register
fmcfg	CAN filter mode configuration register
fscfg	CAN filter size configuration register

Register	Description
frf	CAN filter FIFO assosication register
facfg	CAN filter activate control register
fb0f1	CAN filter bank 0 filter register 1
fb0f2	CAN filter bank 0 filter register 2
fb1f1	CAN filter bank 1 filter register 1
fb1f2	CAN filter bank 1 filter register 2
…	…
fb13f1	CAN filter bank 13 filter register 1
fb13f2	CAN filter bank 13 filter register 2

The table below gives a list of the CAN library functions.

Table 71. Summary of CAN library functions

Function name	Description
can_reset	Reset all CAN registers to their reset values
can_baudrate_default_para_init	Configure the CAN baud rate initial structure with the initial value
can_baudrate_set	Configure CAN baud rate
can_default_para_init	Configure the CAN initial structure with the initial value
can_base_init	Initialize CAN registers with the specified parameters in the can_base_struct
can_filter_default_para_init	Configure the CAN filter initial structure with the initial value
can_filter_init	Initialize CAN registers with the specified parameters in the can_filter_init_struct
can_debug_transmission_prohibit	Select to disalbe/enable message reception and transmission when debug
can_ttc_mode_enable	Enable time-triggered mode
can_message_transmit	Transmit a frame of message
can_transmit_status_get	Get the status of transmission
can_transmit_cancel	Cancel transmission
can_message_receive	Receive a frame of message
can_receive_fifo_release	Release receive FIFO
can_receive_message_pending_get	Get the count of pending messages in FIFO
can_operating_mode_set	Configure CAN operating mode
can_doze_mode_enter	Enter sleep mode
can_doze_mode_exit	Exit sleep mode
can_error_type_record_get	Read CAN error type
can_receive_error_counter_get	Read CAN receive error counter
can_transmit_error_counter_get	Read CAN transmit error counter
can_interrupt_enable	Enable the selected CAN interrupt
can_flag_get	Read the selected CAN flag
can_flag_clear	Clear the selected CAN flag

5.4.1 can_reset function

The table below describes the function can_reset.

Table 72. can_reset function

Name	Description
Function name	can_reset
Function prototype	void can_reset(can_type* can_x);
Function description	Reset CAN registers to their default values
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
can_reset(CAN1);
```

5.4.2 can_baudrate_default_para_init function

The table below describes the function can_baudrate_default_para_init.

Table 73. can_baudrate_default_para_init function

Name	Description
Function name	can_baudrate_default_para_init
Function prototype	void can_baudrate_default_para_init(can_baudrate_type* can_baudrate_struct);
Function description	Configure the CAN baud rate initial structure with the initial value
Input parameter 1	can_baudrate_struct: *can_baudrate_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of can_baudrate_type before starting.
Called functions	NA

Example:

```
can_baudrate_type can_baudrate_struct;
can_baudrate_default_para_init(&can_baudrate_struct);
```

5.4.3 can_baudrate_set function

The table below describes the function can_baudrate_set.

Table 74. can_baudrate_set function

Name	Description
Function name	can_baudrate_set
Function prototype	error_status can_baudrate_set(can_type* can_x, can_baudrate_type* can_baudrate_struct);
Function description	Set CAN baud rate
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	can_baudrate_struct: *can_baudrate_type* pointer
Output parameter	NA
Return value	status_index: check if baud rate is configured successfully
Required preconditions	It is necessary to define a variable of can_baudrate_type before starting.
Called functions	NA

The can_baudrate_type is defined in the at32f413_can.h.

```
typedef struct
{
    uint16_t          baudrate_div;
    can_rsaw_type     rsaw_size;
    can_bts1_type     bts1_size;
    can_bts2_type     bts2_size;
} can_baudrate_type;
```

baudrate_div

CAN clock division factor

Value range: 0x001~0x400

rsaw_size

Defines the maximum of time unit that the CAN is allowed to lengthen or shorten in a bit

CAN_RSAW_1TQ: Resynchronization width is 1 time unit

CAN_RSAW_2TQ: Resynchronization width is 2 time units

CAN_RSAW_3TQ: Resynchronization width is 3 time units

CAN_RSAW_4TQ: Resynchronization width is 4 time units

bts1_size

segment1 time duration

bts1_size description

CAN_BTS1_1TQ: the bit time segment 1 has 1 time unit

……

CAN_BTS1_16TQ: the bit time segment 1 has 16 time units

bts2_size

segment2 time duration

CAN_BTS2_1TQ: the bit time segment 2 has 1 time unit

……

CAN_BTS2_8TQ: the bit time segment 2 has 8 time units

Example:

```
/* can baudrate, set baudrate = pclk/(baudrate_div *(1 + bts1_size + bts2_size)) */
can_baudrate_struct.baudrate_div = 10;
can_baudrate_struct.rsaw_size = CAN_RSAW_3TQ;
can_baudrate_struct.bts1_size = CAN_BTS1_8TQ;
can_baudrate_struct.bts2_size = CAN_BTS2_3TQ;
can_baudrate_set(CAN1, &can_baudrate_struct);
```

5.4.4 can_default_para_init function

The table below describes the function can_default_para_init.

Table 75. can_default_para_init function

Name	Description
Function name	can_default_para_init
Function prototype	void can_default_para_init(can_base_type* can_base_struct);
Function description	Set an initial value for CAN initial structure
Input parameter 1	can_base_struct: *can_base_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of can_base_type before starting.
Called functions	NA

Example:

```
can_base_type can_base_struct;
can_default_para_init (&can_base_struct);
```

5.4.5 can_base_init function

The table below describes the function can_base_init.

Table 76. can_base_init function

Name	Description
Function name	can_base_init
Function prototype	error_status can_base_init(can_type* can_x, can_base_type* can_base_struct);
Function description	Initialize CAN registers with the specified parameters in the can_base_struct
Input parameter 1	can_base_struct: *can_base_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of can_base_type before starting.
Called functions	NA

The can_base_type is defined in the at32f413_can.h.

```
typedef struct
{
  can_mode_type                 mode_selection;
  confirm_state                 ttc_enable;
  confirm_state                 aebo_enable;
```

```
    confirm_state                   aed_enable;
    confirm_state                   prsf_enable;
    can_msg_discarding_rule_type    mdrsel_selection;
    can_msg_sending_rule_type       mmssr_selection;
} can_base_type;
```

mode_selection

Test mode selection

CAN_MODE_COMMUNICATE: Communication mode

CAN_MODE_LOOPBACK: Loopback mode

CAN_MODE_LISTENONLY: Listen only mode

CAN_MODE_LISTENONLY_LOOPBACK: Loopback + listen only mode

ttc_enable

Enable/disable time-triggered communication mode

FALSE: Disable time-triggered communication mode

TRUE: Enable time-triggered communication mode (while receiving/sending messages, capture time stamp and store it into the CAN_RFCx and CAN_TMCx registers)

aebo_enable

Enable auto exit of bus-off mode

FALSE: Automatic exit of bus-off mode is disabled

TRUE: Automatic exit of bus-off mode is enabled

aed_enable

Enable auto exit of sleep mode

FALSE: Auto exit of sleep mode is disabled

TRUE: Auto exit of sleep mode is enabled

prsf_enable

Disable retransmission when transmit failed

FALSE: Retransmission is enabled

TRUE: Retransmission is disabled

mdrsel_selection

Define message discard rule when reception overflows

CAN_DISCARDING_FIRST_RECEIVED: The previous message is discarded

CAN_DISCARDING_LAST_RECEIVED: The new incoming message is discarded

mmssr_selection

Define multiple message transmit sequence rule

CAN_SENDING_BY_ID: The message with the smallest identifier number is first transmitted

CAN_SENDING_BY_REQUEST: The message with the first request order is first transmitted

Example:

```
/* can base init */
can_base_struct.mode_selection = CAN_MODE_COMMUNICATE;
can_base_struct.ttc_enable = FALSE;
can_base_struct.aebo_enable = TRUE;
can_base_struct.aed_enable = TRUE;
can_base_struct.prsf_enable = FALSE;
can_base_struct.mdrsel_selection = CAN_DISCARDING_FIRST_RECEIVED;
can_base_struct.mmssr_selection = CAN_SENDING_BY_ID;
can_base_init(CAN1, &can_base_struct);
```

5.4.6 can_filter_default_para_init function

The table below describes the function can_filter_default_para_init.

Table 77. can_filter_default_para_init function

Name	Description
Function name	can_filter_default_para_init
Function prototype	void can_filter_default_para_init(can_filter_init_type* can_filter_init_struct);
Function description	Configure CAN filter initialization structure with the initial value
Input parameter 1	can_filter_init_struct: *can_filter_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of can_filter_init_type before starting.
Called functions	NA

Example:

```
can_filter_init_type can_filter_init_struct;
can_filter_default_para_init(&can_filter_init_struct);
```

5.4.7 can_filter_init function

The table below describes the function can_filter_init.

Table 78. can_filter_init function

Name	Description
Function name	can_filter_init
Function prototype	void can_filter_init(can_type* can_x, can_filter_init_type* can_filter_init_struct);
Function description	Initialize all CAN registers with the specified parameters in the can_base_struct
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	can_filter_init_struct: *can_filter_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of can_filter_init_type before starting.
Called functions	NA

The can_filter_init_type is defined in the at32f413_can.h.

```
typedef struct
{
  confirm_state               filter_activate_enable;
  can_filter_mode_type        filter_mode;
  can_filter_fifo_type        filter_fifo;
  uint8_t                     filter_number;
  can_filter_bit_width_type   filter_bit;
  uint16_t                    filter_id_high;
  uint16_t                    filter_id_low;
  uint16_t                    filter_mask_high;
```

uint16_t filter_mask_low;

} can_filter_init_type;

filter_activate_enable

Enable/disable filter bank

FALSE: Disable filter bank

TRUE: Enable filter bank

filter_mode

Select filter mode

CAN_FILTER_MODE_ID_MASK: Identifier mask mode

CAN_FILTER_MODE_ID_LIST: Identifier list mode

filter_fifo

Select filter associated FIFO

CAN_FILTER_FIFO0: Associated with FIFO0

CAN_FILTER_FIFO1: Associated with FIFO1

filter_number

Select filter bank

Value range: 0~13

filter_bit

Select filter bit width

CAN_FILTER_16BIT: 16-bit

CAN_FILTER_32BIT: 32-bit

filter_id_high

The filter_id_high is used to configure the upper 16 bits of the filter identifier 1 (32-bit width, Mask/List mode), the filter identifier 2 (16-bit width, List mode) or the filter mask identifier 1 (16-bit width, Mask mode).

Value range: 0x0000~0xFFFF

filter_id_low

The filter_id_low is used to configure the lower 16 bits of the filter identifier 1 (32-bit width, Mask/List mode), or the filter identifier 1 (16-bit width, Mask/List mode)

Value range: 0x0000~0xFFFF

filter_mask_high

The filter_mask_high is used to configure the upper 16 bits of the filter identifier 1 (32-bit width, Mask mode), the filter mask identifier 2 (16-bit width, Mask mode), the upper 16 bits of the filter mask identifier 2 (32-bit width, List mode), or the filter mask identifier 4 (16-bit width, List mode).

Value range: 0x0000~0xFFFF

filter_mask_low

The filter_mask_low is used to configure the lower 16 bits of the filter identifier 1 (32-bit width, Mask mode), the filter mask identifier 2 (16-bit width, Mask mode), the lower 16 bits of the filter identifier 2 (32-bit width, List mode), or the filter mask identifier (16-bit width, List mode).

Value range: 0x0000~0xFFFF

Example:

```
/* can filter init */
can_filter_init_struct.filter_activate_enable = TRUE;
can_filter_init_struct.filter_mode = CAN_FILTER_MODE_ID_MASK;
can_filter_init_struct.filter_fifo = CAN_FILTER_FIFO0;
can_filter_init_struct.filter_number = 0;
```

```
can_filter_init_struct.filter_bit = CAN_FILTER_32BIT;
can_filter_init_struct.filter_id_high = 0;
can_filter_init_struct.filter_id_low = 0;
can_filter_init_struct.filter_mask_high = 0;
can_filter_init_struct.filter_mask_low = 0;
can_filter_init(CAN1, &can_filter_init_struct);
```

5.4.8 can_debug_transmission_prohibit function

The table below describes the function can_debug_transmission_prohibit.

Table 79. can_debug_transmission_prohibit function

Name	Description
Function name	can_debug_transmission_prohibit
Function prototype	void can_debug_transmission_prohibit(can_type* can_x, confirm_state new_state);
Function description	Disable/enable message transceiver when debugging
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* prohibit can trans when debug*/
can_debug_transmission_prohibit(CAN1, TRUE);
```

5.4.9 can_ttc_mode_enable function

The table below describes the function can_ttc_mode_enable.

Table 80. can_ttc_mode_enable function

Name	Description
Function name	can_ttc_mode_enable
Function prototype	void can_ttc_mode_enable(can_type* can_x, confirm_state new_state);
Function description	Enable time-triggered mode
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* can time trigger operation communication mode enable*/
can_ttc_mode_enable (CAN1, TRUE);
```

Note: When the ttc_enable is enabled in the can_base_init, it indicates that only the time stamp is enabled (during message receive and transmit, the time stamp is captured and stored in the CAN_RFCx and CAN_TMCx registers). But when the can_ttc_mode_enable is enabled, not only the time stamp is enable, but the time stamp transmission feature is enabled (during message transmission, the time stamp is sent on the 7th and 8th data byte).

5.4.10 can_message_transmit function

The table below describes the function can_message_transmit.

Table 81. can_message_transmit function

Name	Description
Function name	can_message_transmit
Function prototype	uint8_t can_message_transmit(can_type* can_x, can_tx_message_type* tx_message_struct);
Function description	Transmit a frame of message.
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 orCAN2.
Input parameter 2	tx_message_struct: message pending for transmission, refer to *can_tx_message_type*
Output parameter	NA
Return value	transmit_mailbox: indicates the mailbox number required to send message
Required preconditions	Write the to-be-sent message in the tx_message_struct
Called functions	NA

The can_tx_message_type is defined in the at32f413_can.h.

```
typedef struct
{
    uint32_t                standard_id;
    uint32_t                extended_id;
    can_identifier_type     id_type;
    can_trans_frame_type    frame_type;
    uint8_t                 dlc;
    uint8_t                 data[8];
} can_tx_message_type;
```

standard_id

Standard identifier (11 bits active)

Value range: 0x000~0x7FF

extended_id

Extended identifier (29 bits active)

Value range: 0x000~0x1FFFFFFF

id_type

Identifier type

CAN_ID_STANDARD: Standard identifier

CAN_ID_EXTENDED: Extended identifier

frame_type

Frame type

CAN_TFT_DATA: Data frame

CAN_TFT_REMOTE: Remote frame

dlc

Data length (in byte)

Value range: 0~8

data[8]

Data pending for transmission

Value range: 0x00~0xFF

Example:

```
/* can transmit data */
static void can_transmit_data(void)
{
  uint8_t transmit_mailbox;
  can_tx_message_type tx_message_struct;
  tx_message_struct.standard_id = 0x400;
  tx_message_struct.extended_id = 0;
  tx_message_struct.id_type = CAN_ID_STANDARD;
  tx_message_struct.frame_type = CAN_TFT_DATA;
  tx_message_struct.dlc = 8;
  tx_message_struct.data[0] = 0x11;
  tx_message_struct.data[1] = 0x22;
  tx_message_struct.data[2] = 0x33;
  tx_message_struct.data[3] = 0x44;
  tx_message_struct.data[4] = 0x55;
  tx_message_struct.data[5] = 0x66;
  tx_message_struct.data[6] = 0x77;
  tx_message_struct.data[7] = 0x88;
  transmit_mailbox = can_message_transmit(CAN1, &tx_message_struct);
  while(can_transmit_status_get(CAN1, (can_tx_mailbox_num_type)transmit_mailbox) !=
CAN_TX_STATUS_SUCCESSFUL);
}
```

5.4.11 can_transmit_status_get function

The table below describes the function can_transmit_status_get.

Table 82. can_transmit_status_get function

Name	Description
Function name	can_transmit_status_get
Function prototype	can_transmit_status_type can_transmit_status_get(can_type* can_x, can_tx_mailbox_num_type transmit_mailbox);
Function description	Get the status of transmission
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	transmit_mailbox: indicates the mailbox number required to send messge
Output parameter	NA
Return value	state_index: transmission status
Required preconditions	First send a frame of message and get a transmit mailbox number
Called functions	NA

Example:

```
/* can transmit data */
static void can_transmit_data(void)
{
  uint8_t transmit_mailbox;
  can_tx_message_type tx_message_struct;
  tx_message_struct.standard_id = 0x400;
  tx_message_struct.extended_id = 0;
  tx_message_struct.id_type = CAN_ID_STANDARD;
  tx_message_struct.frame_type = CAN_TFT_DATA;
  tx_message_struct.dlc = 8;
  tx_message_struct.data[0] = 0x11;
  tx_message_struct.data[1] = 0x22;
  tx_message_struct.data[2] = 0x33;
  tx_message_struct.data[3] = 0x44;
  tx_message_struct.data[4] = 0x55;
  tx_message_struct.data[5] = 0x66;
  tx_message_struct.data[6] = 0x77;
  tx_message_struct.data[7] = 0x88;
  transmit_mailbox = can_message_transmit(CAN1, &tx_message_struct);
  while(can_transmit_status_get(CAN1, (can_tx_mailbox_num_type)transmit_mailbox) !=
CAN_TX_STATUS_SUCCESSFUL);
}
```

5.4.12 can_transmit_cancel function

The table below describes the function can_transmit_cancel.

Table 83. can_transmit_cancel function

Name	Description
Function name	can_transmit_cancel
Function prototype	void can_transmit_cancel(can_type* can_x, can_tx_mailbox_num_type transmit_mailbox);
Function description	Cancel transmission
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	transmit_mailbox: indicates the mailbox number required to send message
Output parameter	NA
Return value	NA
Required preconditions	First send a frame of message and get a transmit mailbox number
Called functions	NA

Example:

```
/* cancel a transmit request */
uint8_t transmit_mailbox;
transmit_mailbox = can_message_transmit(CAN1, &tx_message_struct);
can_transmit_cancel(CAN1, (can_tx_mailbox_num_type)transmit_mailbox);
```

5.4.13 can_message_receive function

The table below describes the function can_message_receive.

Table 84. can_message_receive function

Name	Description
Function name	can_message_receive
Function prototype	void can_message_receive(can_type* can_x, can_rx_fifo_num_type fifo_number, can_rx_message_type* rx_message_struct);
Function description	Receive a frame of message
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	fifo_number: receive FIFO This parameter can be CAN_RX_FIFO0 or CAN_RX_FIFO1.
Output parameter	rx_message_struct: indicates the received message, refer to *can_rx_message_type*
Return value	NA
Required preconditions	Receive FIFO not empty (FIFO message count is not zero)
Called functions	void can_receive_fifo_release(can_type* can_x, can_rx_fifo_num_type fifo_number);

The can_rx_message_type is defined in the at32f413_can.h.

```
typedef struct
{
    uint32_t            standard_id;
    uint32_t            extended_id;
```

```
    can_identifier_type       id_type;
    can_trans_frame_type      frame_type;
    uint8_t                   dlc;
    uint8_t                   data[8];
    uint8_t                   filter_index;
} can_rx_message_type;
```

standard_id

Standard identifier (11 bits active)

Value range: 0x000~0x7FF

extended_id

Extended identifier (29 bits active)

Value range: 0x000~0x1FFFFFFF

id_type

Identifier type

CAN_ID_STANDARD: Standard identifier

CAN_ID_EXTENDED: Extended identifier

frame_type

Frame type

CAN_TFT_DATA: Data frame

CAN_TFT_REMOTE: Remote frame

dlc

Data length (in byte)

Value range: 0~8

data[8]

Data pending for transmission

Value range: 0x00~0xFF

filter_index

Filter match index (indicating the filter number that a message has passed through)

Value range: 0x00~0xFF

Example:

```
/* can receive message */
can_rx_message_type rx_message_struct;
can_message_receive(CAN1, CAN_RX_FIFO0, &rx_message_struct);
```

5.4.14 can_receive_fifo_release function

The table below describes the function can_receive_fifo_release.

Table 85. can_receive_fifo_release function

Name	Description
Function name	can_receive_fifo_release
Function prototype	void can_receive_fifo_release(can_type* can_x, can_rx_fifo_num_type fifo_number);
Function description	Release receive FIFO
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	fifo_number: receive FIFO This parameter can be CAN_RX_FIFO0 or CAN_RX_FIFO1.
Output parameter	NA
Return value	NA
Required preconditions	Message in FIFO has already been read
Called functions	NA

Example:

```
/* can receive message */
void can_message_receive(can_type* can_x, can_rx_fifo_num_type fifo_number, can_rx_message_type*
rx_message_struct)
{
  /* get the id type */
  rx_message_struct->id_type = (can_identifier_type)can_x->fifo_mailbox[fifo_number].rfi_bit.rfidi;
  …

  /* get the data field */
  rx_message_struct->data[0] = can_x->fifo_mailbox[fifo_number].rfdtl_bit.rfdt0;
  …
  rx_message_struct->data[7] = can_x->fifo_mailbox[fifo_number].rfdth_bit.rfdt7;

  /* FIFO must be read before releasing FIFO */
  /* release the fifo */
  can_receive_fifo_release(can_x, fifo_number);
}
```

5.4.15 can_receive_message_pending_get function

The table below describes the function can_receive_message_pending_get.

Table 86. can_receive_message_pending_get function

Name	Description
Function name	can_receive_message_pending_get
Function prototype	uint8_t can_receive_message_pending_get(can_type* can_x, can_rx_fifo_num_type fifo_number);
Function description	Get the number of message pending for read in FIFO
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	fifo_number: receive FIFO This parameter can be CAN_RX_FIFO0 or CAN_RX_FIFO1.
Output parameter	NA
Return value	message_pending: the count of message pending for read in FIFO
Required preconditions	NA
Called functions	NA

Example:

```
/* return the number of pending messages of */
can_receive_message_pending_get (CAN1, CAN_RX_FIFO0);
```

5.4.16 can_operating_mode_set function

The table below describes the function can_operating_mode_set.

Table 87. can_operating_mode_set function

Name	Description
Function name	can_operating_mode_set
Function prototype	error_status can_operating_mode_set(can_type* can_x, can_operating_mode_type can_operating_mode);
Function description	Configure CAN operating modes
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	*can_operating_mode*: CAN operating mode selection
Output parameter	NA
Return value	status: indicates whether configuration is successful or not
Required preconditions	NA
Called functions	NA

can_operating_mode

CAN_OPERATINGMODE_FREEZE: Freeze mode—for CAN controller initialization

CAN_OPERATINGMODE_DOZE: Sleep mode—CAN clock stopped to save power consumption

CAN_OPERATINGMODE_COMMUNICATE: Communication mode—for communication

Example:

```
/* set the operation mode –enter freeze mode*/
```

```
can_operating_mode_set (CAN1, CAN_OPERATINGMODE_FREEZE);

/* Initialize CAN controller */
…

/* set the operation mode –enter communicate mode*/
can_operating_mode_set (CAN1, CAN_OPERATINGMODE_COMMUNICATE);

/* Start communication: send and receive message */
…
```

5.4.17 can_doze_mode_enter function

The table below describes the function can_doze_mode_enter.

Table 88. can_doze_mode_enter function

Name	Description
Function name	can_doze_mode_enter
Function prototype	can_enter_doze_status_type can_doze_mode_enter(can_type* can_x);
Function description	Enter sleep mode
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	*can_enter_doze_status*: indicates wheter the Sleep mode is entered
Required preconditions	NA
Called functions	NA

can_enter_doze_status

Indicates whether the Sleep mode is entered or not

CAN_ENTER_DOZE_FAILED: Sleep mode entry failure

CAN_ENTER_DOZE_SUCCESSFUL: Sleep mode entry success

Example:

```
/* can enter the low power mode */
can_enter_doze_status_type can_enter_doze_status;
can_enter_doze_status = can_doze_mode_enter(CAN1);
```

5.4.18 can_doze_mode_exit function

The table below describes the function can_doze_mode_exit.

Table 89. can_doze_mode_exit function

Name	Description
Function name	can_doze_mode_exit
Function prototype	can_quit_doze_status_type can_doze_mode_exit(can_type* can_x);
Function description	Exit Sleep mode
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	*can_quit_doze_status*: indicates whethe the Sleep mode has been left
Required preconditions	NA
Called functions	NA

can_quit_doze_status

Indicates whethe the Sleep mode has been left successfully

CAN_QUIT_DOZE_FAILED: Sleep mode exit failure

CAN_QUIT_DOZE_SUCCESSFUL: Sleep mode exit success

Example:

```
/* can exit the low power mode */
can_quit_doze_status_type can_quit_doze_status;
can_quit_doze_status = can_doze_mode_exit (CAN1);
```

5.4.19 can_error_type_record_get function

The table below describes the function can_error_type_record_get.

Table 90. can_error_type_record_get function

Name	Description
Function name	can_error_type_record_get
Function prototype	can_error_record_type can_error_type_record_get(can_type* can_x);
Function description	Read CAN error type
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	*can_error_record*: error type
Required preconditions	NA
Called functions	NA

can_error_record

CAN error record

CAN_ERRORRECORD_NOERR: No error

CAN_ERRORRECORD_STUFFERR: Bit stuffing error

CAN_ERRORRECORD_FORMERR: Format error

CAN_ERRORRECORD_ACKERR: Acknowledge error

CAN_ERRORRECORD_BITRECESSIVEERR: Recessive bit error
CAN_ERRORRECORD_BITDOMINANTERR: Dominant bit error
CAN_ERRORRECORD_CRCERR: CRC error
CAN_ERRORRECORD_SOFTWARESETERR: Set by software

Example:

```
/* get the error type record (etr) */
can_error_record_type can_error_record;
can_error_record = can_error_type_record_get (CAN1);
```

5.4.20 can_receive_error_counter_get function

The table below describes the function can_receive_error_counter_get.

Table 91. can_receive_error_counter_get function

Name	Description
Function name	can_receive_error_counter_get
Function prototype	uint8_t can_receive_error_counter_get(can_type* can_x);
Function description	Read CAN receive error counter
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	receive_error_counter: Receive error counter Valur range: 0x00~0xFF
Required preconditions	NA
Called functions	NA

Example:

```
/* get the receive error counter (rec) */
uint8_t receive_error_counter;
receive_error_counter = can_receive_error_counter_get (CAN1);
```

5.4.21 can_transmit_error_counter_get function

The table below describes the function can_transmit_error_counter_get.

Table 92. can_transmit_error_counter_get function

Name	Description
Function name	can_transmit_error_counter_get
Function prototype	uint8_t can_transmit_error_counter_get(can_type* can_x);
Function description	Read CAN transmit error counter
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	transmit_error_counter: Transmit error counter Value range: 0x00~0xFF
Required preconditions	NA
Called functions	NA

Example:

```
/* get the transmit error counter (tec) */
uint8_t transmit_error_counter;
transmit_error_counter = can_transmit_error_counter_get (CAN1);
```

5.4.22 can_interrupt_enable function

The table below describes the function can_interrupt_enable.

Table 93. can_interrupt_enable function

Name	Description
Function name	can_interrupt_enable
Function prototype	void can_interrupt_enable(can_type* can_x, uint32_t can_int, confirm_state new_state);
Function description	Enable the selected CAN interrupt
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	*can_int*: Select CAN interrupts
Input parameter 3	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

can_int

CAN interrupt selection

CAN_TCIEN_INT: Transmit mailbox empty interrupt enable
CAN_RF0MIEN_INT: FIFO 0 receive message interrupt enable
CAN_RF0FIEN_INT: Receive FIFO0 full interrupt enable
CAN_RF0OIEN_INT: Receive FIFO0 overflow interrupt enable
CAN_RF1MIEN_INT: FIFO1 receive message interrupt enable
CAN_RF1FIEN_INT: Receive FIFO1 full interrupt enable
CAN_RF1OIEN_INT: Receive FIFO1 overflow interrupt enable
CAN_EAIEN_INT: Error active interrupt enable
CAN_EPIEN_INT: Error passive interrupt enable
CAN_BOIEN_INT: Bus-off interrupt enable
CAN_ETRIEN_INT: Error type record interrupt enable
CAN_EOIEN_INT: Error occur interrupt enable
CAN_QDZIEN_INT: Quit Sleep mode interrupt enable
CAN_EDZIEN_INT: Enter Sleep mode interrupt enable

Example:

```
/* can interrupt config */
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);/*CAN1 error/status change interrupt */
```

```
nvic_irq_enable(USBFS_L_CAN1_RX0_IRQn, 0x00, 0x00);/*CAN1 FIFO0 receive interrrupt */

/* FIFO 0 receive message interrupt enable */
can_interrupt_enable(CAN1, CAN_RF0MIEN_INT, TRUE);
/* error type record interrupt enable */
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);

/*This parameter is an error interrupt controller and it is enabled before error-related interrupts */
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
```

5.4.23 can_flag_get function

The table below describes the function can_flag_get.

Table 94. can_flag_get function

Name	Description
Function name	can_flag_get
Function prototype	flag_status can_flag_get(can_type* can_x, uint32_t can_flag);
Function description	Get the status of the selected CAN flag
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	*can_flag:* indicates the selected flag Refer to the "can_flag" description below for details.
Output parameter	NA
Return value	flag_status: the status of the selected flag Return SET or RESET.
Required preconditions	NA
Called functions	NA

can_flag

This is used to select a flag and get its status, including

CAN_EAF_FLAG: Error active flag
CAN_EPF_FLAG: Error passive flag
CAN_BOF_FLAG: Bus-off flag
CAN_ETR_FLAG: Error type record (non-zero error type flag)
CAN_EOIF_FLAG: Error occur interrupt flag
CAN_TM0TCF_FLAG: Mailbox 0 transmission complete flag
CAN_TM1TCF_FLAG: Mailbox 1 transmission complete flag
CAN_TM2TCF_FLAG: Mailbox 2 transmission complete flag
CAN_RF0MN_FLAG: FIFO0 non-empty flag
CAN_RF0FF_FLAG: FIFO0 full flag
CAN_RF0OF_FLAG: FIFO0 overflow flag
CAN_RF1MN_FLAG: FIFO1 non-empty flag
CAN_RF1FF_FLAG: FIFO1 full flag
CAN_RF1OF_FLAG: FIFO1 overflow flag
CAN_QDZIF_FLAG: Quit Sleep mode flag
CAN_EDZC_FLAG: Enter Sleep mode flag

CAN_TMEF_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

Example:

```
/* get receive fifo 0 message num flag */
flag_status bit_status = RESET;
bit_status = can_flag_get (CAN1, CAN_RF0MN_FLAG);
```

5.4.24 can_interrupt_flag_get function

The table below describes the function can_interrupt_flag_get.

Table 95. can_interrupt_flag_get function

Name	Description
Function name	can_interrupt_flag_get
Function prototype	flag_status can_interrupt_flag_get(can_type* can_x, uint32_t can_flag);
Function description	Get the selected CAN interrupt flag
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	can_flag: indicates the selected flag Refer to the "can_flag" description below for details.
Output parameter	NA
Return value	flag_status: the status of the selected flag Return SET or RESET.
Required preconditions	NA
Called functions	NA

can_flag

This is used to select a flag and get its status, including

CAN_EAF_FLAG: Error active flag
CAN_EPF_FLAG: Error passive flag
CAN_BOF_FLAG: Bus-off flag
CAN_ETR_FLAG: Error type record (non-zero error type flag)
CAN_EOIF_FLAG: Error occur interrupt flag
CAN_TM0TCF_FLAG: Mailbox 0 transmission complete flag
CAN_TM1TCF_FLAG: Mailbox 1 transmission complete flag
CAN_TM2TCF_FLAG: Mailbox 2 transmission complete flag
CAN_RF0MN_FLAG: FIFO0 non-empty flag
CAN_RF0FF_FLAG: FIFO0 full flag
CAN_RF0OF_FLAG: FIFO0 overflow flag
CAN_RF1MN_FLAG: FIFO1 non-empty flag
CAN_RF1FF_FLAG: FIFO1 full flag
CAN_RF1OF_FLAG: FIFO1 overflow flag
CAN_QDZIF_FLAG: Quit Sleep mode flag
CAN_EDZC_FLAG: Enter Sleep mode flag
CAN_TMEF_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

Example

```
/* check receive fifo 0 message num interrupt flag */
if(can_interrupt_flag_get(CAN1, CAN_RF0MN_FLAG) != RESET)
{
}
```

5.4.25 can_flag_clear function

The table below describes the function can_flag_clear.

Table 96. can_flag_clear function

Name	Description
Function name	can_flag_clear
Function prototype	void can_flag_clear(can_type* can_x, uint32_t can_flag);
Function description	Clear the selected CAN flag
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	*can_flag:* indicates the selected flag Refer to “can_flag” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

can_flag:

This is used to clear the selected flag, including:

CAN_EAF_FLAG: Error active flag
CAN_EPF_FLAG: Error passive flag
CAN_BOF_FLAG: Bus-off flag
CAN_ETR_FLAG: Error type record (non-zero Error type flag)
CAN_EOIF_FLAG: Error occur interrupt flag
CAN_TM0TCF_FLAG: Mailbox 0 transmission complete flag
CAN_TM1TCF_FLAG: Mailbox 1 transmission complete flag
CAN_TM2TCF_FLAG: Mailbox 2 transmission complete flag
CAN_RF0FF_FLAG: FIFO0 full flag
CAN_RF0OF_FLAG: FIFO0 overflow flag
CAN_RF1FF_FLAG: FIFO1 full flag
CAN_RF1OF_FLAG: FIFO1 overflow flag
CAN_QDZIF_FLAG: Quit Sleep mode flag
CAN_EDZC_FLAG: Enter Sleep mode flag
CAN_TMEF_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

Note: The CAN_RF0MN_FLAG (FIFO0 non-empty flag) and CAN_RF1MN_FLAG (FIFO1 non-empty flag) have no clear operations since both are defined by software.

Example:

```
/* clear receive fifo 0 overflow flag */
can_flag_clear (CAN1, CAN_RF1OF_FLAG);
```

5.5 CRC calculation unit (CRC)

The CRC register structure crc_type is defined in the “at32f413_crc.h”.

```
/**
  * @brief type define crc register all
  */
typedef struct
{
     ...

} crc_type;
```

The table below gives a list of the CRC registers.

Table 97. Summary of CRC registers

Register	Description
dt	Data register
cdt	General-purpose data register
ctrl	Control register
idt	Initialization register
poly	Polynomial generator

The table below gives a list of the CRC library functions.

Table 98. Summary of CRC library functions

Function name	Description
crc_data_reset	Data register reset
crc_one_word_calculate	Calulate the CRC value using combination of a new 32-bit data and the previous CRC value
crc_block_calculate	Write a data block in sequence to go through CRC check and return the calculated result
crc_data_get	Get the currently calculated CRC result
crc_common_data_set	Configure common registers
crc_common_date_get	Get the value of common registers
crc_init_data_set	Set the CRC initialization register
crc_reverse_input_data_set	Set CRC input data bit reverse type
crc_reverse_output_data_set	Set CRC output data reverse type
crc_poly_value_set	Set polynomial value
crc_poly_value_get	Get polynomial value
crc_poly_size_set	Set polynomial valid width
crc_poly_size_get	Get polynomial valid width

5.5.1 crc_data_reset function

The table below describes the function crc_data_reset.

Table 99. crc_data_reset function

Name	Description
Function name	crc_data_reset
Function prototype	void crc_data_reset(void);
Function description	When the data register is reset, the value of the initialization register is added into the data register as an initiai value. The default reset value is 0xFFFFFFFF.
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reset crc data register */
crc_data_reset();
```

5.5.2 crc_one_word_calculate function

The table below describes the function crc_one_word_calculate.

Table 100. crc_one_word_calculate function

Name	Description
Function name	crc_one_word_calculate
Function prototype	uint32_t crc_one_word_calculate(uint32_t data);
Function description	Calculate the CRC value using a combination of a new 32-bit data and the previous CRC value.
Input parameter 1	data: input a 32-bit data
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: return CRC calculation result
Required preconditions	NA
Called functions	NA

Example:

```
/* calculate and return result */
uint32_t data = 0x12345678, result = 0;
result = crc_one_word_calculate (data);
```

5.5.3 crc_block_calculate function

The table below describes the function crc_block_calculate.

Table 101. crc_block_calculate function

Name	Description
Function name	crc_block_calculate
Function prototype	uint32_t crc_block_calculate(uint32_t *pbuffer, uint32_t length);
Function description	Input a data block in sequence to go through CRC calculation and return a result
Input parameter 1	pbuffer: point to the data block pending for CRC check
Input parameter 2	length: data block length pending for CRC check, in terms of 32-bit
Output parameter	NA
Return value	uint32_t: return CRC calculation result
Required preconditions	NA
Called functions	NA

Example:

```
/* calculate and return result */
uint32_t pbuffer[2] = {0x12345678, 0x87654321};
uint32_t result = 0;
result = crc_block_calculate (pbuffer, 2);
```

5.5.4 crc_data_get function

The table below describes the function crc_data_get.

Table 102. crc_data_get function

Name	Description
Function name	crc_data_get
Function prototype	uint32_t crc_data_get(void);
Function description	Return the current CRC calculation result
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: return CRC calculation result
Required preconditions	NA
Called functions	NA

Example:

```
/* get result */
uint32_t result = 0;
result = crc_data_get ();
```

5.5.5 crc_common_data_set function

The table below describes the function crc_common_data_set.

Table 103. crc_common_data_set function

Name	Description
Function name	crc_common_data_set
Function prototype	void crc_common_data_set(uint8_t cdt_value);
Function description	Configure common data register
Input parameter 1	cdt_value: 8-bitommon data that can be used as temporary storage data
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set common data */
crc_common_data_set (0x88);
```

5.5.6 crc_common_data_get function

The table below describes the function crc_common_data_get.

Table 104. crc_common_data_get function

Name	Description
Function name	crc_common_data_get
Function prototype	uint8_t crc_common_data_get(void);
Function description	Return the value of the common data register
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	uint8_t: return the value of the previously programmed common data register
Required preconditions	NA
Called functions	NA

Example:

```
/* get common data */
uint8_t cdt_value = 0;
cdt_value = crc_common_data_get ();
```

5.5.7 crc_init_data_set function

The table below describes the function crc_init_data_set.

Table 105. crc_init_data_set function

Name	Description
Function name	crc_init_data_set
Function prototype	void crc_init_data_set(uint32_t value);
Function description	Set the value of the CRC initialization register
Input parameter 1	value: the value of the CRC initialization register
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

After the value of the CRC initialization register is programmed, the CRC data register is updated with this value whenever the crc_data_reset function is called.

Example:

```
/* set initial data */
uint32_t init_value = 0x11223344;
crc_init_data_set (init_value);
```

5.5.8 crc_reverse_input_data_set function

The table below describes the function crc_reverse_input_data_set.

Table 106. crc_reverse_input_data_set function

Name	Description
Function name	crc_reverse_input_data_set
Function prototype	void crc_reverse_input_data_set(crc_reverse_input_type value);
Function description	Define the CRC input data bit reverse type
Input parameter 1	value: input data bit reverse type
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

Define the reverse type of input data bit.

CRC_REVERSE_INPUT_NO_AFFECTE: No effect
CRC_REVERSE_INPUT_BY_BYTE: Byte reverse
CRC_REVERSE_INPUT_BY_HALFWORD: Half-word reverse
CRC_REVERSE_INPUT_BY_WORD: Word reverse

Example:

```
/* set input data reversing type */
crc_reverse_input_data_set(CRC_REVERSE_INPUT_BY_WORD);
```

5.5.9 crc_reverse_output_data_set function

The table below describes the function crc_reverse_output_data_set.

Table 107. crc_reverse_output_data_set function

Name	Description
Function name	crc_reverse_output_data_set
Function prototype	void crc_reverse_output_data_set(crc_reverse_output_type value);
Function description	Define the CRC output data reverse type
Input parameter 1	value: output data bit reverse type
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

Define the reverse type of output data bit.

CRC_REVERSE_OUTPUT_NO_AFFECTE: No effect

CRC_REVERSE_OUTPUT_DATA: Word reverse

Example:

```
/* set output data reversing type */
crc_reverse_output_data_set (CRC_REVERSE_OUTPUT_DATA);
```

5.5.10 crc_poly_value_set function

The table below describes the function crc_poly_value_set.

Table 108. crc_poly_value_set function

Name	Description
Function name	crc_poly_value_set
Function prototype	void crc_poly_value_set(uint32_t value);
Function description	Set CRC polynomial value
Input parameter 1	value: polynomial value
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example

```
/* set poly value */
crc_poly_value_set(0x12345671);
```

5.5.11 crc_poly_value_get function

The table below describes the function crc_poly_value_get.

Table 109. crc_poly_value_get function

Name	Description
Function name	crc_poly_value_get
Function prototype	uint32_t crc_poly_value_get(void);
Function description	Get CRC polynomial value
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: return polynomial value
Required preconditions	NA
Called functions	NA

Example

```
/* get poly value */
uint32_t poly = 0;
poly = crc_poly_value_get();
```

5.5.12 crc_poly_size_set function

The table below describes the function crc_poly_size_set.

Table 110. crc_poly_size_set function

Name	Description
Function name	crc_poly_size_set
Function prototype	void crc_poly_size_set(crc_poly_size_type size);
Function description	Set CRC polynomial valid width
Input parameter 1	size: polynomial valid width
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

size

Define the valid width of polynomial.

CRC_POLY_SIZE_32B: 32-bit
CRC_POLY_SIZE_16B: 16-bit
CRC_POLY_SIZE_8B: 8-bit
CRC_POLY_SIZE_7B: 7-bit

Example

```
/* set poly size 32-bit */
crc_poly_size_set(CRC_POLY_SIZE_32B);
```

5.5.13 crc_poly_size_get function

The table below describes the function crc_poly_size_get.

Table 111. crc_poly_size_get function

Name	Description
Function name	crc_poly_size_get
Function prototype	crc_poly_size_type crc_poly_size_get(void);
Function description	Get CRC polynomial valid width
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	crc_poly_size_type: polynomial valid width
Required preconditions	NA
Called functions	NA

crc_poly_size_type

Define the valid width of polynomial.

CRC_POLY_SIZE_32B: 32-bit

CRC_POLY_SIZE_16B: 16-bit

CRC_POLY_SIZE_8B: 8-bit

CRC_POLY_SIZE_7B: 7-bit

Example

```
/* get poly size */
crc_poly_size_type size;
size = crc_poly_size_get();
```

5.6 Clock and reset management (CRM)

5.6.1 crm_reset function

The table below describes the function crm_reset.

Table 112. crm_reset function

Name	Description
Function name	crm_reset
Function prototype	void crm_reset(void);
Function description	Reset the clock reset management register and control status
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

1. This function does not change the HICKTRIM[5:0] in the CRM_CTRL register.
2. Modifying the function does not reset the CRM_BPDC and CRM_CTRLSTS registers.

Example:

```
/* reset crm */
crm_reset();
```

5.6.2 crm_lext_bypass function

The table below describes the function crm_lext_bypass.

Table 113. crm_lext_bypass function

Name	Description
Function name	crm_lext_bypass
Function prototype	void crm_lext_bypass(confirm_state new_state);
Function description	Configure low-speed external clock bypass
Input parameter 1	new_state: Enable bypass (TRUE), disable bypass (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	The LEXT configuration must be done before being enabled.
Called functions	NA

Example:

```
/* enable lext bypass mode */
crm_lext_bypass(TRUE);
```

5.6.3 crm_hext_bypass function

The table below describes the function crm_hext_bypass.

Table 114. crm_hext_bypass function

Name	Description
Function name	crm_hext_bypass
Function prototype	void crm_hext_bypass(confirm_state new_state);
Function description	Configure high-speed external clock bypass
Input parameter 1	new_state: Enable bypass (TRUE), disable bypass (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	The HEXT configuration must be done before being enabled.
Called functions	NA

Example:

```
/* enable hext bypass mode */
crm_hext_bypass(TRUE);
```

5.6.4 crm_flag_get function

The table below describes the function crm_flag_get.

Table 115. crm_flag_get function

Name	Description
Function name	crm_flag_get
Function prototype	flag_status crm_flag_get(uint32_t flag);
Function description	Check if the selected flag has been set.
Input parameter 1	flag: flag selection
Input parameter 2	NA
Output parameter	NA
Return value	flag_status: indicates the status of the selected flag (SET or RESET)
Required preconditions	NA
Called functions	NA

flag

Select a flag to read, including:

CRM_HICK_STABLE_FLAG: HICK clock stable flag
CRM_HEXT_STABLE_FLAG: HEXT clock stable flag
CRM_PLL_STABLE_FLAG: PLL clock stable flag
CRM_LEXT_STABLE_FLAG: LEXT clock stable flag
CRM_LICK_STABLE_FLAG: LICK clock stable flag
CRM_NRST_RESET_FLAG: NRST pin reset flag
CRM_POR_RESET_FLAG: Power-on/low voltage reset flag
CRM_SW_RESET_FLAG: Software reset flag
CRM_WDT_RESET_FLAG: Watchdog reset flag
CRM_WWDT_RESET_FLAG: Window watchdog reset flag
CRM_LOWPOWER_RESET_FLAG: Low-power consumption reset flag

CRM_LICK_READY_INT_FLAG: LICK clock ready interrupt flag
CRM_LEXT_READY_INT_FLAG: LEXT clock ready interrupt flag
CRM_HICK_READY_INT_FLAG: HICK clock ready interrupt flag
CRM_HEXT_READY_INT_FLAG: HEXT clock ready interrupt flag
CRM_PLL_READY_INT_FLAG: PLL clock ready interrupt flag
CRM_CLOCK_FAILURE_INT_FLAG: Clock failure interrupt flag

Example:

```
/* wait till pll is ready */
while(crm_flag_get(CRM_PLL_STABLE_FLAG) != SET)
{
}
```

5.6.5 crm_interrupt_flag_get function

The table below describes the function crm_interrupt_flag_get.

Table 116. crm_interrupt_flag_get function

Name	Description
Function name	crm_interrupt_flag_get
Function prototype	flag_status crm_interrupt_flag_get(uint32_t flag);
Function description	Check if the selected flag has been set.
Input parameter 1	flag: flag selection Refer to the “flag” description below for details.
Input parameter 2	NA
Output parameter	NA
Return value	flag_status: indicates the status of the selected flag (SET or RESET)
Required preconditions	NA
Called functions	NA

flag

Select a flag to read, including:

CRM_LICK_READY_INT_FLAG: LICK clock ready interrupt flag
CRM_LEXT_READY_INT_FLAG: LEXT clock ready interrupt flag
CRM_HICK_READY_INT_FLAG: HICK clock ready interrupt flag
CRM_HEXT_READY_INT_FLAG: HEXT clock ready interrupt flag
CRM_PLL_READY_INT_FLAG: PLL clock ready interrupt flag
CRM_CLOCK_FAILURE_INT_FLAG: Clock failure interrupt flag

Example

```
/* check pll ready interrupt flag */
if(crm_interrupt_flag_get(CRM_PLL_READY_INT_FLAG) != RESET)
{
}
```

5.6.6 crm_hext_stable_wait function

The table below describes the function crm_hext_stable_wait.

Table 117. crm_hext_stable_wait function

Name	Description
Function name	crm_hext_stable_wait
Function prototype	error_status crm_hext_stable_wait(void);
Function description	Wait for HEXT to activate and become stable
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	error_status: Return the status of HEXT (SUCCESS or ERROR).
Required preconditions	NA
Called functions	NA

Example:

```
/* wait till hext is ready */
while(crm_hext_stable_wait() == ERROR)
{
}
```

5.6.7 crm_hick_clock_trimming_set function

The table below describes the function crm_hick_clock_trimming_set.

Table 118. crm_hick_clock_trimming_set function

Name	Description
Function name	crm_hick_clock_trimming_set
Function prototype	void crm_hick_clock_trimming_set(uint8_t trim_value);
Function description	Trim HICK clock
Input parameter 1	trim_value: trimming value. Default value is 0x20, and configurable range is from 0 to 0x3F.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set trimming value */
crm_hick_clock_trimming_set(0x1F);
```

5.6.8 crm_hick_clock_calibration_set function

The table below describes the function crm_hick_clock_calibration_set.

Table 119. crm_hick_clock_calibration_set function

Name	Description
Function name	crm_hick_clock_calibration_set
Function prototype	void crm_hick_clock_calibration_set(uint8_t cali_value);
Function description	Set HICK clock calibration value
Input parameter 1	cali_value: calibration compensation value. The factory gate value is the default value, and its configurable range is from 0 to 0xFF
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set trimming value */
crm_hick_clock_trimming_set(0x80);
```

5.6.9 crm_periph_clock_enable function

The table below describes the function crm_periph_clock_enable.

Table 120. crm_periph_clock_enable function

Name	Description
Function name	crm_periph_clock_enable
Function prototype	void crm_periph_clock_enable(crm_periph_clock_type value, confirm_state new_state);
Function description	Enable peripheral clock
Input parameter 1	value: defines peripheral clock type
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

The crm_periph_clock_type is defined in the at32f413_crm.h.

The naming rule of this parameter is: CRM_peripheral_PERIPH_CLOCK.

CRM_DMA1_PERIPH_CLOCK: DMA1 peripheral clock

CRM_DMA2_PERIPH_CLOCK: DMA2 peripheral clock

…

CRM_PWC_PERIPH_CLOCK: PWC peripheral clock

Example:

```
/* enable gpioa periph clock */
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

5.6.10 crm_periph_reset function

The table below describes the function crm_periph_reset.

Table 121. crm_periph_reset function

Name	Description
Function name	crm_periph_reset
Function prototype	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);
Function description	Reset peripherals
Input parameter 1	value: peripheral reset type
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

This indicates the selected peripheral.

The crm_periph_reset_type is defined in the at32f413_crm.h.

The naming rule of this parameter is: CRM_peripheral_PERIPH_RESET.

CRM_DMA1_PERIPH_RESET: DMA1 peripheral reset

CRM_DMA2_PERIPH_RESET: DMA2 peripheral reset

…

CRM_PWC_PERIPH_RESET: PWC peripheral reset

Example:

```
/* reset gpioa periph */
crm_periph_reset(CRM_GPIOA_PERIPH_RESET, TRUE);
```

5.6.11 crm_periph_sleep_mode_clock_enable function

The table below describes the function crm_periph_sleep_mode_clock_enable.

Table 122. crm_periph_sleep_mode_clock_enable function

Name	Description
Function name	crm_periph_sleep_mode_clock_enable
Function prototype	void crm_periph_sleep_mode_clock_enable(crm_periph_clock_sleepmd_type value, confirm_state new_state);
Function description	Enable peripheral clock in sleep mode
Input parameter 1	value: indicates peripheral clock type in sleep mode
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

It indicates the selected peripheral.

The crm_periph_clock_sleepmd_type is defined in the at32f413_crm.h.

The naming rule of this parameter is: CRM_ peripheral_PERIPH_CLOCK_SLEEP_MODE.

CRM_SRAM_PERIPH_RESET: SRAM sleep mode clock definition
CRM_FLASH_PERIPH_RESET: FLASH sleep mode clock definition

Example:

```
/* disable flash clock when entry sleep mode */
crm_periph_sleep_mode_clock_enable (CRM_FLASH_PERIPH_CLOCK_SLEEP_MODE, FALSE);
```

5.6.12 crm_clock_source_enable function

The table below describes the function crm_clock_source_enable.

Table 123. crm_clock_source_enable function

Name	Description
Function name	crm_clock_source_enable
Function prototype	void crm_clock_source_enable(crm_clock_source_type source, confirm_state new_state);
Function description	Enable clock source
Input parameter 1	source: Clock source type
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

source

Clock source selection

CRM_CLOCK_SOURCE_HICK: HICK
CRM_CLOCK_SOURCE_HEXT: HEXT
CRM_CLOCK_SOURCE_PLL: PLL
CRM_CLOCK_SOURCE_LEXT: LEXT
CRM_CLOCK_SOURCE_LICK: LICK

Example:

```
/* enable hext */
crm_clock_source_enable (CRM_CLOCK_SOURCE_HEXT, FALSE);
```

5.6.13 crm_flag_clear function

The table below describes the function crm_flag_clear

Table 124. crm_flag_clear function

Name	Description
Function name	crm_flag_clear
Function prototype	void crm_flag_clear(uint32_t flag);
Function description	Clear the selected flags
Input parameter 1	flag: indicates the flag to clear
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

Select a flag to clear

CRM_NRST_RESET_FLAG:	NRST pin reset flag
CRM_POR_RESET_FLAG:	Power-on/low-voltage reset flag
CRM_SW_RESET_FLAG:	Software reset flag
CRM_WDT_RESET_FLAG:	Watchdog reset flag
CRM_WWDT_RESET_FLAG:	Window watchdog reset flag
CRM_LOWPOWER_RESET_FLAG:	Low-power reset flag
CRM_ALL_RESET_FLAG:	All reset flags
CRM_LICK_READY_INT_FLAG:	LICK clock ready interrupt flag
CRM_LEXT_READY_INT_FLAG:	LEXT clock ready interrupt flag
CRM_HICK_READY_INT_FLAG:	HICK clock ready interrupt flag
CRM_HEXT_READY_INT_FLAG:	HEXT clock ready interrupt flag
CRM_PLL_READY_INT_FLAG:	PLL clock ready interrupt flag
CRM_CLOCK_FAILURE_INT_FLAG:	Clock failure interrupt flag

Example:

```
/* clear clock failure detection flag */
crm_flag_clear(CRM_CLOCK_FAILURE_INT_FLAG);
```

5.6.14 crm_rtc_clock_select function

The table below describes the function crm_rtc_clock_select.

Table 125. crm_rtc_clock_select function

Name	Description
Function name	crm_rtc_clock_select
Function prototype	void crm_rtc_clock_select(crm_rtc_clock_type value);
Function description	Select RTC clock source
Input parameter 1	value: indicates RTC clock source type
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

RTC clock source selection

CRM_RTC_CLOCK_NOCLK: No clock source for RTC

CRM_RTC_CLOCK_LEXT: LEXT selected as RTC clock

CRM_RTC_CLOCK_LICK: LICK selected as RTC clock

CRM_RTC_CLOCK_HEXT_DIV: HEXT/128 selected as RTC clock

Example:

```
/* config lext as rtc clock */
crm_rtc_clock_select (CRM_RTC_CLOCK_LEXT);
```

5.6.15 crm_rtc_clock_enable function

The table below describes the function crm_rtc_clock_enable.

Table 126. crm_rtc_clock_enable function

Name	Description
Function name	crm_rtc_clock_enable
Function prototype	void crm_rtc_clock_enable(confirm_state new_state);
Function description	Enable RTC clock
Input parameter 1	new_state: enable (TRUE) or disable (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable rtc clock */
crm_rtc_clock_enable (TRUE);
```

5.6.16 crm_ahb_div_set function

The table below describes the function crm_ahb_div_set.

Table 127. crm_ahb_div_set function

Name	Description
Function name	crm_ahb_div_set
Function prototype	void crm_ahb_div_set(crm_ahb_div_type value);
Function description	Configure AHB clock division
Input parameter 1	value: indicates the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_AHB_DIV_1: SCLK/1 used as AHB clock
CRM_AHB_DIV_2: SCLK/2 used as AHB clock
CRM_AHB_DIV_4: SCLK/4 used as AHB clock
CRM_AHB_DIV_8: SCLK/8 used as AHB clock
CRM_AHB_DIV_16: SCLK/16 used as AHB clock
CRM_AHB_DIV_64: SCLK/64 used as AHB clock
CRM_AHB_DIV_128: SCLK/128 used as AHB clock
CRM_AHB_DIV_256: SCLK/256 used as AHB clock
CRM_AHB_DIV_512: SCLK/512 used as AHB clock

Example:

```
/* config ahbclk */
crm_ahb_div_set(CRM_AHB_DIV_1);
```

5.6.17 crm_apb1_div_set function

The table below describes the function crm_apb1_div_set.

Table 128. crm_apb1_div_set function

Name	Description
Function name	crm_apb1_div_set
Function prototype	void crm_apb1_div_set(crm_apb1_div_type value);
Function description	Configure APB1 clock division
Input parameter 1	value: indicates the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_APB1_DIV_1: AHB/1 used as APB1 clock
CRM_APB1_DIV_2: AHB/2 used as APB1 clock

CRM_APB1_DIV_4: AHB/4 used as APB1 clock
CRM_APB1_DIV_8: AHB/8 used as APB1 clock
CRM_APB1_DIV_16: AHB/16 used as APB1 clock

Example:

```
/* config apb1clk */
crm_apb1_div_set(CRM_APB1_DIV_2);
```

5.6.18 crm_apb2_div_set function

The table below describes the function crm_apb2_div_set.

Table 129. crm_apb2_div_set function

Name	Description
Function name	crm_apb2_div_set
Function prototype	void crm_apb2_div_set(crm_apb2_div_type value);
Function description	Configure APB2 clock division
Input parameter 1	value: indicates the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_APB2_DIV_1: AHB/1 used as APB2 clock
CRM_APB2_DIV_2: AHB/2 used as APB2 clock
CRM_APB2_DIV_4: AHB/4 used as APB2 clock
CRM_APB2_DIV_8: AHB/8 used as APB2 clock
CRM_APB2_DIV_16: AHB/16 used as APB2 clock

Example:

```
/* config apb2clk */
crm_apb2_div_set(CRM_APB2_DIV_2);
```

5.6.19 crm_adc_clock_div_set function

The table below describes the function crm_adc_clock_div_set.

Table 130. crm_adc_clock_div_set function

Name	Description
Function name	crm_adc_clock_div_set
Function prototype	void crm_adc_clock_div_set(crm_adc_div_type div_value);
Function description	Configure ADC clock division
Input parameter 1	div_value: indicates the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

div_value

CRM_ADC_DIV_2: APB/2 used as ADC clock
CRM_ADC_DIV_4: APB/4 used as ADC clock
CRM_ADC_DIV_6: APB/6 used as ADC clock
CRM_ADC_DIV_8: APB/8 used as ADC clock
CRM_ADC_DIV_12: APB/12 used as ADC clock
CRM_ADC_DIV_16: APB/16 used as ADC clock

Example:

```
/* config adc div 4 */
crm_adc_clock_div_set (CRM_ADC_DIV_4);
```

5.6.20 crm_usb_clock_div_set function

The table below describes the function crm_usb_clock_div_set.

Table 131. crm_usb_clock_div_set function

Name	Description
Function name	crm_usb_clock_div_set
Function prototype	void crm_usb_clock_div_set(crm_usb_div_type div_value);
Function description	Configure PLL clock division
Input parameter 1	div_value: indicates the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

div_value

CRM_USB_DIV_1_5: PLL/1.5 used as USB clock
CRM_USB_DIV_1: PLL/1 used as USB clock
CRM_USB_DIV_2_5: PLL/2.5 used as USB clock
CRM_USB_DIV_2: PLL/2 used as USB clock
CRM_USB_DIV_3_5: PLL/3.5 used as USB clock
CRM_USB_DIV_3: PLL/3 used as USB clock
CRM_USB_DIV_4: PLL/4 used as USB clock

Example:

```
/* config usb div 2 */
crm_usb_clock_div_set (CRM_USB_DIV_2);
```

5.6.21 crm_clock_failure_detection_enable function

The table below describes the function crm_clock_failure_detection_enable.

Table 132. crm_clock_failure_detection_enable function

Name	Description
Function name	crm_clock_failure_detection_enable
Function prototype	void crm_clock_failure_detection_enable(confirm_state new_state);
Function description	Enable clock failure detection
Input parameter 1	new_state: enable (TRUE) or disable (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable clock failure detection */
crm_clock_failure_detection_enable(TRUE);
```

5.6.22 crm_battery_powered_domain_reset function

The table below describes the function crm_battery_powered_domain_reset.

Table 133. crm_battery_powered_domain_reset function

Name	Description
Function name	crm_battery_powered_domain_reset
Function prototype	void crm_battery_powered_domain_reset(confirm_state new_state);
Function description	Reset battery powered domain
Input parameter 1	new_state: reset (TRUE) or not reset (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

When it comes to resetting battery powered domain, it is usually necessary to reset battery powered domain through TRUE operation and then disable battery powered domain reset through FALSE operation after the completion of reset.

Example:

```
/* reset battery powered domain */
crm_battery_powered_domain_reset (TRUE);
```

5.6.23 crm_pll_config function

The table below describes the function crm_pll_config.

Table 134. crm_pll_config function

Name	Description
Function name	crm_pll_config
Function prototype	void crm_pll_config(crm_pll_clock_source_type clock_source, crm_pll_mult_type mult_value, crm_pll_output_range_type pll_range);
Function description	Configure PLL clock source and frequency multiplication factor
Input parameter 1	clock_source: clock source for PLL frequency multiplication
Input parameter 2	mult_value: frequency multiplication factor
Input parameter 3	pll_range: configure PLL clock output range (≤72 MHz or >72 MHz)
Output parameter	NA
Return value	NA
Required preconditions	PLL clock source must be enabled and stable before configuring and enabling PLL.
Called functions	NA

clock_source

CRM_PLL_SOURCE_HICK: HICK is selected as PLL clock source

CRM_PLL_SOURCE_HEXT: HEXT is selected as PLL clock source

CRM_PLL_SOURCE_HEXT_DIV: Divided HEXT is selected as PLL clock source

mult_value

CRM_PLL_MULT_2: PLL output x 2

CRM_PLL_MULT_3: PLL output x 3

…

CRM_PLL_MULT_63: PLL output x 63

CRM_PLL_MULT_64: PLL output x 64

pll_range

CRM_PLL_OUTPUT_RANGE_LE72MHZ: Configure when PLL clock output ≤72 MHz

CRM_PLL_OUTPUT_RANGE_GT72MHZ: Configure when PLL clock output >72 MHz

Example:

```
/* config pll clock resource */
crm_pll_config(CRM_PLL_SOURCE_HEXT_DIV, CRM_PLL_MULT_60,
CRM_PLL_OUTPUT_RANGE_GT72MHZ);
```

5.6.24 crm_sysclk_switch function

The table below describes the function crm_sysclk_switch.

Table 135. crm_sysclk_switch function

Name	Description
Function name	crm_sysclk_switch
Function prototype	void crm_sysclk_switch(crm_sclk_type value);
Function description	Switch system clock source
Input parameter 1	value: indicates the clock source for system clock
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_SCLK_HICK: HICK is used as system clock

CRM_SCLK_HEXT: HECT is used as system clock

CRM_SCLK_PLL: PLL is used as system clock

Example:

```
/* select pll as system clock source */
crm_sysclk_switch(CRM_SCLK_PLL);
```

5.6.25 crm_sysclk_switch_status_get function

The table below describes the function crm_sysclk_switch_status_get.

Table 136. crm_sysclk_switch_status_get function

Name	Description
Function name	crm_sysclk_switch_status_get
Function prototype	crm_sclk_type crm_sysclk_switch_status_get(void);
Function description	Get the clock source of system clock
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	crm_sclk_type: return the clock source of system clock
Required preconditions	NA
Called functions	NA

Example:

```
/* wait till pll is used as system clock source */
while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL)
{
}
```

5.6.26 crm_clocks_freq_get function

The table below describes the function crm_clocks_freq_get.

Table 137. crm_clocks_freq_get function

Name	Description
Function name	crm_clocks_freq_get
Function prototype	void crm_clocks_freq_get(crm_clocks_freq_type *clocks_struct);
Function description	Get clock frequency
Input parameter 1	clocks_struct: crm_clocks_freq_type pointer, including clock frequency
Input parameter 2	NA
Output parameter	NA
Return value	crm_sclk_type: return the clock source for system clock
Required preconditions	NA
Called functions	NA

crm_clocks_freq_type

The crm_clocks_freq_type is defined in the at32f413_crm.h.

```
typedef struct
{
    uint32_t        sclk_freq;
    uint32_t        ahb_freq;
    uint32_t        apb2_freq;
    uint32_t        apb1_freq;
    uint32_t        adc_freq;
} crm_clocks_freq_type;
```

sclk_freq

Get the system clock frequency, in Hz

ahb_freq

Get the clock frequency of AHB, in Hz

apb2_freq

Get the clock frequency of APB2, in Hz

apb1_freq

Get the clock frequency of APB1, in Hz

adc_freq

Get the clock frequency of ADC, in Hz

Example:

```
/* get frequency */
crm_clocks_freq_type clocks_struct;
crm_clocks_freq_get(&clocks_struct);
```

5.6.27 crm_clock_out_set function

The table below describes the function crm_clock_out_set.

Table 138. crm_clock_out_set function

Name	Description
Function name	crm_clock_out_set
Function prototype	void crm_clock_out_set(crm_clkout_select_type clkout);
Function description	Select clock source output on clkout pin
Input parameter 1	clkout: clock source output on clkout pin
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* config PA8 output pll/4 */
crm_clock_out_set(CRM_CLKOUT_PLL_DIV_4);
```

5.6.28 crm_interrupt_enable function

The table below describes the function crm_interrupt_enable.

Table 139. crm_interrupt_enable function

Name	Description
Function name	crm_interrupt_enable
Function prototype	void crm_interrupt_enable(uint32_t crm_int, confirm_state new_state);
Function description	Enable interrupts
Input parameter 1	crm_int: indicates the selected crm interrupt
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

crm_int

CRM_LICK_STABLE_INT: LICK stable interrupt
CRM_LEXT_STABLE_INT: LEXT stable interrupt
CRM_HICK_STABLE_INT: HICK stable interrupt
CRM_HEXT_STABLE_INT: HEXT stable interrupt
CRM_PLL_STABLE_INT: PLL clock stable interrupt
CRM_CLOCK_FAILURE_INT: Clock failure interrupt

Example:

```
/* enable pll stable interrupt */
crm_interrupt_enable (CRM_PLL_STABLE_INT);
```

5.6.29 crm_auto_step_mode_enable function

The table below describes the function crm_auto_step_mode_enable.

Table 140. crm_auto_step_mode_enable function

Name	Description
Function name	crm_auto_step_mode_enable
Function prototype	void crm_auto_step_mode_enable(confirm_state new_state);
Function description	Enable auto step-by-step mode
Input parameter 1	new_state: enable (TRUE) or disable (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable auto step mode */
crm_auto_step_mode_enable(TRUE);
```

5.6.30 crm_usb_interrupt_remapping_set function

The table below describes the function crm_usb_interrupt_remapping_set.

Table 141. crm_usb_interrupt_remapping_set function

Name	Description
Function name	crm_usb_interrupt_remapping_set
Function prototype	void crm_usb_interrupt_remapping_set(crm_usb_int_map_type int_remap);
Function description	Configure USB interrupt remapping
Input parameter 1	int_remap: USB interrupt selection
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

int_remap

CRM_USB_INT19_INT20: USB uses the 19^{th} and 20^{th} interrupt

CRM_USB_INT73_INT74: USB uses the 73^{rd} and 74^{th} interrupt

Example:

```
/* config usb IRQ number with 73/74 */
crm_usb_interrupt_remapping_set (CRM_USB_INT73_INT74);
```

5.6.31 crm_hick_sclk_frequency_select function

The table below describes the function crm_hick_sclk_frequency_select.

Table 142. crm_hick_sclk_frequency_select function

Name	Description
Function name	crm_hick_sclk_frequency_select
Function prototype	void crm_hick_sclk_frequency_select(crm_hick_sclk_frequency_type value);
Function description	Select 8M or 48M system clock frequency when HICK is used as system clock
Input parameter 1	value: 8M or 48M HICK
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_HICK_SCLK_8MHZ: 8 MHz HICK used as system clock

CRM_HICK_SCLK_48MHZ: 48 MHz HICK used as system clock

Example:

```
/* config sysclk with hick 48mhz */
crm_hick_sclk_frequency_select (CRM_HICK_SCLK_48MHZ);
```

5.6.32 crm_usb_clock_source_select function

The table below describes the function crm_usb_clock_source_select.

Table 143. crm_usb_clock_source_select function

Name	Description
Function name	crm_usb_clock_source_select
Function prototype	void crm_usb_clock_source_select(crm_usb_clock_source_type value);
Function description	Select PLL or HICK (48M) as USB clock source
Input parameter 1	value: PLL or HICK (48M)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_USB_CLOCK_SOURCE_PLL: PLL is used as USB clock source

CRM_USB_CLOCK_SOURCE_HICK: HICK is used as USB clock source

Example:

```
/* select hick48 as usb clock */
crm_usb_clock_source_select (CRM_USB_CLOCK_SOURCE_HICK);
```

5.6.33 crm_clkout_to_tmr10_enable function

The table below describes the function crm_clkout_to_tmr10_enable.

Table 144. crm_clkout_to_tmr10_enable function

Name	Description
Function name	crm_clkout_to_tmr10_enable
Function prototype	void crm_clkout_to_tmr10_enable(confirm_state new_state);
Function description	Enable the clkout to tmr10 channel 1
Input parameter 1	new_state: enable (TRUE) or disable (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* config clkout internal connect to tmr10 channel1 */
crm_clkout_to_tmr10_enable (TRUE);
```

5.6.34 crm_clkout_div_set function

The table below describes the function crm_clkout_div_set.

Table 145. crm_clkout_div_set function

Name	Description
Function name	crm_clkout_div_set
Function prototype	void crm_clkout_div_set(crm_clkout_div_type clkout_div);
Function description	Clock frequency division on clkout pin
Input parameter 1	clkout_div: clkout frequency division
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_CLKOUT_DIV_1: Clock output divided by 1
CRM_CLKOUT_DIV_2: Clock output divided by 2
CRM_CLKOUT_DIV_4: Clock output divided by 4
CRM_CLKOUT_DIV_8: Clock output divided by 8
CRM_CLKOUT_DIV_16: Clock output divided by 16
CRM_CLKOUT_DIV_64: Clock output divided by 64
CRM_CLKOUT_DIV_128: Clock output divided by 128
CRM_CLKOUT_DIV_256: Clock output divided by 256
CRM_CLKOUT_DIV_512: Clock output divided by 512

Example:

```
/* config clkout division */
crm_clkout_div_set(CRM_CLKOUT_DIV_1);
```

5.7 Debug

The DEBUG register structure debug_type is defined in the “at32f413_debug.h”.

```
/**
  * @brief type define debug register all
  */
typedef struct
{
      ...

} debug_type;
```

The table below gives a list of the DEBUG registers.

Table 146. Summary of DEBUG registers

Register	Description
idcode	Device ID
ctrl	Control register

The table below gives a list of the DEBUG library functions.

Table 147. Summary of DEBUG library functions

Function name	Description
debug_device_id_get	Read device idcode
debug_periph_mode_set	Peripheral debug mode configuration

5.7.1 debug_device_id_get function

The table below describes the function debug_device_id_get.

Table 148. debug_device_id_get function

Name	Description
Function name	debug_device_id_get
Function prototype	uint32_t debug_device_id_get(void);
Function description	Read device idcode
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	Return 32-bit idcode
Required preconditions	NA
Called functions	NA

Example:

```
/* get idcode */
uint32_t idcode = 0;
idcode = debug_device_id_get();
```

5.7.2 debug_periph_mode_set function

The table below describes the function debug_periph_mode_set.

Table 149. debug_periph_mode_set function

Name	Description
Function name	debug_periph_mode_set
Function prototype	void debug_periph_mode_set(uint32_t periph_debug_mode, confirm_state new_state);
Function description	Select a peripheral/mode to debug
Input parameter 1	periph_debug_mode: select a peripheral or mode
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

periph_debug_mode

Select a peripheral or mode to debug

DEBUG_SLEEP: Debug in Sleep mode
DEBUG_DEEPSLEEP: Debug in Deepsleep mode
DEBUG_STANDBY: Debug in Standby mode
DEBUG_WDT_PAUSE: Watchdog pause control bit
DEBUG_WWDT_PAUSE: Window watchdog pause control bit
DEBUG_TMR1_PAUSE: TMR1 pause control bit
DEBUG_TMR2_PAUSE: TMR2 pause control bit
DEBUG_TMR3_PAUSE: TMR3 pause control bit
DEBUG_TMR4_PAUSE: TMR4 pause control bit
DEBUG_TMR5_PAUSE: TMR5 pause control bit
DEBUG_TMR8_PAUSE: TMR8 pause control bit
DEBUG_TMR9_PAUSE: TMR9 pause control bit
DEBUG_TMR10_PAUSE: TMR10 pause control bit
DEBUG_TMR11_PAUSE: TMR11 pause control bit
DEBUG_I2C1_SMBUS_TIMEOUT: I2C1 SMBUS TIMEOUT pause control bit
DEBUG_I2C2_SMBUS_TIMEOUT: I2C2 SMBUS TIMEOUT pause control bit
DEBUG_CAN1_PAUSE: CAN1 receive register pause control bit

Example:

```
/* enable tmr1 debug mode */
debug_periph_mode_set(DEBUG_TMR1_PAUSE, TRUE);
```

5.8 DMA controller

The DMA register structure dma_type is defined in the “at32f413_dma.h”.

```
/**
  * @brief type define dma register
  */
typedef struct
{
      …

} dma_type;
```

The DMA channel register structure dma_channel_type is defined in the “at32f413_dma.h”.

```
/**
  * @brief type define dma channel register all
  */
typedef struct
{
      …

} dma_channel_type;
```

The table below gives a list of the DMA registers.

Table 150. Summary of DMA registers

Register	Description
dma_sts	DMA status register
dma_clr	DMA status clear register
dma_c1ctrl	DMA channel 1 configuration register
dma_c1dtcnt	DMA channel 1 number of data register
dma_c1paddr	DMA channel 1 peripheral address register
dma_c1maddr	DMA channel 1 memory address register
dma_c2ctrl	DMA channel 2 configuration register
dma_c2dtcnt	DMA channel 2 number of data register
dma_c2paddr	DMA channel 2 peripheral address register
dma_c2maddr	DMA channel 2 memory address register
dma_c3ctrl	DMA channel 3 configuration register
dma_c3dtcnt	DMA channel 3 number of data register
dma_c3paddr	DMA channel 3 peripheral address register
dma_c3maddr	DMA channel 3 memory address register
dma_c4ctrl	DMA channel 4 configuration register
dma_c4dtcnt	DMA channel 4 number of data register
dma_c4paddr	DMA channel 4 peripheral address register
dma_c4maddr	DMA channel 4 memory address register
dma_c5ctrl	DMA channel 5 configuration register
dma_c5dtcnt	DMA channel 5 number of data register

Register	Description
dma_c5paddr	DMA channel 5 peripheral address register
dma_c5maddr	DMA channel 5 memory address register
dma_c6ctrl	DMA channel 6 configuration register
dma_c6dtcnt	DMA channel 6 number of data register
dma_c6paddr	DMA channel 6 peripheral address register
dma_c6maddr	DMA channel 6 memory address register
dma_c7ctrl	DMA channel 7 configuration register
dma_c7dtcnt	DMA channel 7 number of data register
dma_c7paddr	DMA channel 7 peripheral address register
dma_c7maddr	DMA channel 7 memory address register
dma_src_sel0	Channel source register 0
dma_src_sel1	Channel source register 1

The table below gives a list of the DMA library functions.

Table 151. Summary of DMA library functions

Function name	Description
dma_default_para_init	Initialize parameters of the dma_init_struct
dma_init	Initialize the selected DMA channel
dma_reset	Reset the selected DMA channel
dma_data_number_set	Set the number of data transfer of a given channel
dma_data_number_get	Get the number of data transfer of a given channel
dma_interrupt_enable	Enable DMA channel interrupt
dma_channel_enable	Enable DMA channel
dma_flexible_config	Configure flexible DMA request mapping
dma_flag_get	Get the flag of DMA channels
dma_flag_clear	Clear the flag of DMA channels

5.8.1 dma_default_para_init function

The table below describes the function dma_default_para_init.

Table 152. dma_default_para_init function

Name	Description
Function name	dma_default_para_init
Function prototype	void dma_default_para_init(dma_init_type* dma_init_struct);
Function description	Initialize parameters in the dma_init_struct
Input parameter 1	dma_init_struct: dma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of dma_init_struct members.

Table 153. dma_init_struct default values

Member	Default value
peripheral_base_addr	0x0
memory_base_addr	0x0
direction	DMA_DIR_PERIPHERAL_TO_MEMORY
buffer_size	0x0
peripheral_inc_enable	FALSE
memory_inc_enable	FALSE
peripheral_data_width	DMA_PERIPHERAL_DATA_WIDTH_BYTE
memory_data_width	DMA_MEMORY_DATA_WIDTH_BYTE
loop_mode_enable	FALSE
priority	DMA_PRIORITY_LOW

Example:

```
/* dma init config with its default value */
dma_init_type dma_init_struct = {0};
dma_default_para_init(&dma_init_struct);
```

5.8.2 dma_init function

The table below describes the function dma_init.

Table 154. dma_init function

Name	Description
Function name	dma_init
Function prototype	void dma_init(dma_channel_type* dmax_channely, dma_init_type* dma_init_struct)
Function description	Initialize the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	dma_init_struct: dma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_init_type structure

The dma_init_type is defined in the at32f413_dma.h.

```
typedef struct
{
    uint32_t                        peripheral_base_addr;
    uint32_t                        memory_base_addr;
    dma_dir_type                    direction;
    uint16_t                        buffer_size;
    confirm_state                   peripheral_inc_enable;
    confirm_state                   memory_inc_enable;
   dma_peripheral_data_size_type    peripheral_data_width;
```

```
    dma_memory_data_size_type           memory_data_width;
    confirm_state                       loop_mode_enable;
    dma_priority_level_type             priority;
} dma_init_type;
```

peripheral_base_addr

Set the peripheral address of a DMA channel

memory_base_addr

Set the memory address of a DMA channel.

direction

Set the transfer direction of a DMA channel

DMA_DIR_PERIPHERAL_TO_MEMORY: Peripheral to memory

DMA_DIR_MEMORY_TO_PERIPHERAL: Memory to peripheral

DMA_DIR_MEMORY_TO_MEMORY: Memory to memory

buffer_size

Set the number of data transfer of a DMA channel.

peripheral_inc_enable

Enable/disable DMA channel peripheral address auto increment.

FALSE: Peripheral address is not incremented

TRUE: Peripheral address is incremented

memory_inc_enable

Enable/disable DMA channel memory address auto increment.

FALSE: Memory address is not incremented

TRUE: Memory address is incremented

peripheral_data_width

Set DMA peripheral data width.

DMA_PERIPHERAL_DATA_WIDTH_BYTE: Byte

DMA_PERIPHERAL_DATA_WIDTH_HALFWORD: Half-word

DMA_PERIPHERAL_DATA_WIDTH_WORD: Word

memory_data_width

Set DMA memory data width.

DMA_MEMORY_DATA_WIDTH_BYTE: Byte

DMA_MEMORY_DATA_WIDTH_HALFWORD: Half-word

DMA_MEMORY_DATA_WIDTH_WORD: Word

loop_mode_enable

Set DMA loop mode.

FALSE: DMA single mode

TRUE: DMA loop mode

priority

Set DMA channel priority.

DMA_PRIORITY_LOW: Low

DMA_PRIORITY_MEDIUM: Medium

DMA_PRIORITY_HIGH: High

DMA_PRIORITY_VERY_HIGH: Very high

Example:

```
dma_init_type dma_init_struct = {0};
/* dma2 channel1 configuration */
dma_init_struct.buffer_size = BUFFER_SIZE;
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
dma_init_struct.memory_base_addr = (uint32_t)src_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)0x4001100C;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
dma_init(DMA2_CHANNEL1, &dma_init_struct);
```

5.8.3 dma_reset function

The table below describes the function dma_reset.

Table 155. dma_reset function

Name	Description
Function name	dma_reset
Function prototype	void dma_reset(dma_channel_type* dmax_channely);
Function description	Reset the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reset dma2 channel1 */
dma_reset(DMA2_CHANNEL1);
```

5.8.4 dma_data_number_set function

The table below describes the function dma_data_number_set.

Table 156. dma_data_number_set function

Name	Description
Function name	dma_ data_number_set
Function prototype	void dma_data_number_set(dma_channel_type* dmax_channely, uint16_t data_number);
Function description	Set the number of data transfer of the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	data_number: indicates the number of data transfer, up to 65535
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set dma2 channel1 data count is 0x100*/
dma_data_number_set(DMA2_CHANNEL1, 0x100);
```

5.8.5 dma_data_number_get function

The table below describes the function dma_data_number_get.

Table 157. dma_data_number_get function

Name	Description
Function name	dma_data_number_get
Function prototype	uint16_t dma_data_number_get(dma_channel_type* dmax_channely);
Function description	Get the number of data transfer of the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Output parameter	NA
Return value	Get the number of data transfer of a DMA channel
Required preconditions	NA
Called functions	NA

Example:

```
/* get dma2 channel1 data count*/
uint16_t data_counter;
data_counter = dma_data_number_set(DMA2_CHANNEL1);
```

5.8.6 dma_interrupt_enable function

The table below describes the function dma_interrupt_enable.

Table 158. dma_interrupt_enable function

Name	Description
Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(dma_channel_type* dmax_channely, uint32_t dma_int, confirm_state new_state);
Function description	Enable DMA channel interrupt
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	dma_int: interrupt source selection
Input parameter 3	new_state: interrupt enable/disable
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_int

Select DMA interrupt source

DMA_FDT_INT: Transfer complete interrupt

DMA_HDT_INT: Half transfer complete interrupt

DMA_DTERR_INT: Transfer error interrupt

new_state

Enable or disable DMA channel interrupt

FALSE: Disabled

TRUE: Enabled

Example:

```
/* enable dma2 channel1 transfer full data intterrupt */
dma_interrupt_enable(DMA2_CHANNEL1, DMA_FDT_INT, TRUE);
```

5.8.7 dma_channel_enable function

The table below describes the function dma_interrupt_enable

Table 159. dma_channel_enable function

Name	Description
Function name	dma_channel_enable
Function prototype	void dma_channel_enable(dma_channel_type* dmax_channely, confirm_state new_state);
Function description	Enable the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	new_state: Enable or disable the selected DMA channel
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable DMA channels

FALSE: Disabled

TRUE: Enabled

Example:

```
/* enable dma channel */
dma_channel_enable(DMA2_CHANNEL1, TRUE);
```

5.8.8 dma_flexible_config function

The table below describes the function dma_flexible_config.

Table 160. dma_flexible_config function

Name	Description
Function name	dma_flexible_config
Function prototype	void dma_flexible_config(dma_type* dma_x, uint8_t flex_channelx, dma_flexible_request_type flexible_request);
Function description	Configure flexible DMA request mapping
Input parameter 1	dma_x: select DMAx, x=1 or 2
Input parameter 2	flex_channelx: FLEX_CHANNELx defines a DMA channel number, x=1…7
Input parameter 3	flexible_request: channel request ID
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flexible_request

The table below shows the DMA channel request ID.

Table 161. DMA channel request source ID

Request source ID	Description	Request source ID	Description
0x01	DMA_FLEXIBLE_ADC1	0x0A	DMA_FLEXIBLE_SPI1_TX
0x09	DMA_FLEXIBLE_SPI1_RX	0x0C	DMA_FLEXIBLE_SPI2_TX
0x0B	DMA_FLEXIBLE_SPI2_RX	0x1A	DMA_FLEXIBLE_UART1_TX
0x19	DMA_FLEXIBLE_UART1_RX	0x1C	DMA_FLEXIBLE_UART2_TX
0x1B	DMA_FLEXIBLE_UART2_RX	0x1E	DMA_FLEXIBLE_UART3_TX
0x1D	DMA_FLEXIBLE_UART3_RX	0x20	DMA_FLEXIBLE_UART4_TX
0x1F	DMA_FLEXIBLE_UART4_RX	0x22	DMA_FLEXIBLE_UART5_TX
0x21	DMA_FLEXIBLE_UART5_RX	0x2A	DMA_FLEXIBLE_I2C1_TX
0x29	DMA_FLEXIBLE_I2C1_RX	0x2C	DMA_FLEXIBLE_I2C2_TX
0x2B	DMA_FLEXIBLE_I2C2_RX	0x36	DMA_FLEXIBLE_TMR1_HALL
0x31	DMA_FLEXIBLE_SDIO1	0x38	DMA_FLEXIBLE_TMR1_CH1
0x35	DMA_FLEXIBLE_TMR1_TRIG	0x3A	DMA_FLEXIBLE_TMR1_CH3
0x37	DMA_FLEXIBLE_TMR1_OVERFLOW	0x3D	DMA_FLEXIBLE_TMR2_TRIG
0x39	DMA_FLEXIBLE_TMR1_CH2	0x40	DMA_FLEXIBLE_TMR2_CH1
0x3B	DMA_FLEXIBLE_TMR1_CH4	0x42	DMA_FLEXIBLE_TMR2_CH3
0x3F	DMA_FLEXIBLE_TMR2_OVERFLOW	0x45	DMA_FLEXIBLE_TMR3_TRIG
0x41	DMA_FLEXIBLE_TMR2_CH2	0x48	DMA_FLEXIBLE_TMR3_CH1
0x43	DMA_FLEXIBLE_TMR2_CH4	0x4A	DMA_FLEXIBLE_TMR3_CH3
0x47	DMA_FLEXIBLE_TMR3_OVERFLOW	0x4D	DMA_FLEXIBLE_TMR4_TRIG
0x49	DMA_FLEXIBLE_TMR3_CH2	0x50	DMA_FLEXIBLE_TMR4_CH1
0x4B	DMA_FLEXIBLE_TMR3_CH4	0x52	DMA_FLEXIBLE_TMR4_CH3
0x4F	DMA_FLEXIBLE_TMR4_OVERFLOW	0x55	DMA_FLEXIBLE_TMR5_TRIG
0x51	DMA_FLEXIBLE_TMR4_CH2	0x58	DMA_FLEXIBLE_TMR5_CH1
0x53	DMA_FLEXIBLE_TMR4_CH4	0x5A	DMA_FLEXIBLE_TMR5_CH3
0x57	DMA_FLEXIBLE_TMR5_OVERFLOW	0x6D	DMA_FLEXIBLE_TMR8_TRIG
0x59	DMA_FLEXIBLE_TMR5_CH2	0x6F	DMA_FLEXIBLE_TMR8_OVERFLOW
0x5B	DMA_FLEXIBLE_TMR5_CH4	0x71	DMA_FLEXIBLE_TMR8_CH2
0x6E	DMA_FLEXIBLE_TMR8_HALL	0x73	DMA_FLEXIBLE_TMR8_CH4
0x70	DMA_FLEXIBLE_TMR8_CH1	0x72	DMA_FLEXIBLE_TMR8_CH3

Example:

```
/* tmr2 flexible function enable */
dma_flexible_config(DMA2, FLEX_CHANNEL1, DMA_FLEXIBLE_TMR2_OVERFLOW);
```

5.8.9 dma_flag_get function

The table below describes the function dma_flag_get.

Table 162. dma_flag_get function

Name	Description
Function name	dma_flag_get
Function prototype	flag_status dma_flag_get(uint32_t dmax_flag);
Function description	Get the flag of the selected DMA channel
Input parameter 1	*dmax_flag*: select the desired flag
Output parameter	NA
Return value	flag_status: indicates whether the desired flag is set or not
Required preconditions	NA
Called functions	NA

dmax_flag

The dmax_flag is used for flag section, including:

DMA1_GL1_FLAG: DMA1 channel 1 global flag
DMA1_FDT1_FLAG: DMA1 channel 1 transfer complete flag
DMA1_HDT1_FLAG: DMA1 channel 1 half transfer complete flag
DMA1_DTERR1_FLAG: DMA1 channel 1 transfer error flag
DMA1_GL2_FLAG: DMA1 channel 2 global flag
DMA1_FDT2_FLAG: DMA1 channel 2 transfer complete flag
DMA1_HDT2_FLAG: DMA1 channel 2 half transfer complete flag
DMA1_DTERR2_FLAG: DMA1 channel 2 transfer error flag
DMA1_GL3_FLAG: DMA1 channel 3 global flag
DMA1_FDT3_FLAG: DMA1 channel 3 transfer complete flag
DMA1_HDT3_FLAG: DMA1 channel 3 half transfer complete flag
DMA1_DTERR3_FLAG: DMA1 channel 3 transfer error flag
DMA1_GL4_FLAG: DMA1 channel 4 global flag
DMA1_FDT4_FLAG: DMA1 channel 4 transfer complete flag
DMA1_HDT4_FLAG: DMA1 channel 4 half transfer complete flag
DMA1_DTERR4_FLAG: DMA1 channel 4 transfer error flag
DMA1_GL5_FLAG: DMA1 channel 5 global flag
DMA1_FDT5_FLAG: DMA1 channel 5 transfer complete flag
DMA1_HDT5_FLAG: DMA1 channel 5 half transfer complete flag
DMA1_DTERR5_FLAG: DMA1 channel 5 transfer error flag
DMA1_GL6_FLAG: DMA1 channel 6 global flag
DMA1_FDT6_FLAG: DMA1 channel 6 transfer complete flag
DMA1_HDT6_FLAG: DMA1 channel 6 half transfer complete flag
DMA1_DTERR6_FLAG: DMA1 channel 6 transfer error flag
DMA1_GL7_FLAG: DMA1 channel 7 global flag
DMA1_FDT7_FLAG: DMA1 channel 7 transfer complete flag
DMA1_HDT7_FLAG: DMA1 channel 7 half transfer complete flag
DMA1_DTERR7_FLAG: DMA1 channel 7 transfer error flag
DMA2_GL1_FLAG: DMA2 channel 1 global flag
DMA2_FDT1_FLAG: DMA2 channel 1 transfer complete flag
DMA2_HDT1_FLAG: DMA2 channel 1 half transfer complete flag

DMA2_DTERR1_FLAG:	DMA2 channel 1 transfer error flag
DMA2_GL2_FLAG:	DMA2 channel 2 global flag
DMA2_FDT2_FLAG:	DMA2 channel 2 transfer complete flag
DMA2_HDT2_FLAG:	DMA2 channel 2 half transfer complete flag
DMA2_DTERR2_FLAG:	DMA2 channel 2 transfer error flag
DMA2_GL3_FLAG:	DMA2 channel 3 global flag
DMA2_FDT3_FLAG:	DMA2 channel 3 transfer complete flag
DMA2_HDT3_FLAG:	DMA2 channel 3 half transfer complete flag
DMA2_DTERR3_FLAG:	DMA2 channel 3 transfer error flag
DMA2_GL4_FLAG:	DMA2 channel 4 global flag
DMA2_FDT4_FLAG:	DMA2 channel 4 transfer complete flag
DMA2_HDT4_FLAG:	DMA2 channel 4 half transfer complete flag
DMA2_DTERR4_FLAG:	DMA2 channel 4 transfer error flag
DMA2_GL5_FLAG:	DMA2 channel 5 global flag
DMA2_FDT5_FLAG:	DMA2 channel 5 transfer complete flag
DMA2_HDT5_FLAG:	DMA2 channel 5 half transfer complete flag
DMA2_DTERR5_FLAG:	DMA2 channel 5 transfer error flag
DMA2_GL6_FLAG:	DMA2 channel 6 global flag
DMA2_FDT6_FLAG:	DMA2 channel 6 transfer complete flag
DMA2_HDT6_FLAG:	DMA2 channel 6 half transfer complete flag
DMA2_DTERR6_FLAG:	DMA2 channel 6 transfer error flag
DMA2_GL7_FLAG:	DMA2 channel 7 global flag
DMA2_FDT7_FLAG:	DMA2 channel 7 transfer complete flag
DMA2_HDT7_FLAG:	DMA2 channel 7 half transfer complete flag
DMA2_DTERR7_FLAG:	DMA2 channel 7 transfer error flag

flag_status

RESET: Flag is reset

SET: Flag is set

Example:

```
if(dma_flag_get(DMA2_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.8.10 dma_flag_clear function

The table below describes the function dma_flag_clear.

Table 163. dma_flag_clear function

Name	Description
Function name	dma_flag_clear
Function prototype	void dma_flag_clear(uint32_t dmax_flag);
Function description	Clear the selected flag
Input parameter 1	*dmax_flag*: a flag that needs to be cleared
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dmax_flag

The dmax_flag is used to select the desired flag, including:

DMA1_GL1_FLAG: DMA1 channel 1 global flag
DMA1_FDT1_FLAG: DMA1 channel 1 transfer complete flag
DMA1_HDT1_FLAG: DMA1 channel 1 half transfer complete flag
DMA1_DTERR1_FLAG: DMA1 channel 1 transfer error flag
DMA1_GL2_FLAG: DMA1 channel 2 global flag
DMA1_FDT2_FLAG: DMA1 channel 2 transfer complete flag
DMA1_HDT2_FLAG: DMA1 channel 2 half transfer complete flag
DMA1_DTERR2_FLAG: DMA1 channel 2 transfer error flag
DMA1_GL3_FLAG: DMA1 channel 3 global flag
DMA1_FDT3_FLAG: DMA1 channel 3 transfer complete flag
DMA1_HDT3_FLAG: DMA1 channel 3 half transfer complete flag
DMA1_DTERR3_FLAG: DMA1 channel 3 transfer error flag
DMA1_GL4_FLAG: DMA1 channel 4 global flag
DMA1_FDT4_FLAG: DMA1 channel 4 transfer complete flag
DMA1_HDT4_FLAG: DMA1 channel 4 half transfer complete flag
DMA1_DTERR4_FLAG: DMA1 channel 4 transfer error flag
DMA1_GL5_FLAG: DMA1 channel 5 global flag
DMA1_FDT5_FLAG: DMA1 channel 5 transfer complete flag
DMA1_HDT5_FLAG: DMA1 channel 5 half transfer complete flag
DMA1_DTERR5_FLAG: DMA1 channel 5 transfer error flag
DMA1_GL6_FLAG: DMA1 channel 6 global flag
DMA1_FDT6_FLAG: DMA1 channel 6 transfer complete flag
DMA1_HDT6_FLAG: DMA1 channel 6 half transfer complete flag
DMA1_DTERR6_FLAG: DMA1 channel 6 transfer error flag
DMA1_GL7_FLAG: DMA1 channel 7 global flag
DMA1_FDT7_FLAG: DMA1 channel 7 transfer complete flag
DMA1_HDT7_FLAG: DMA1 channel 7 half transfer complete flag
DMA1_DTERR7_FLAG: DMA1 channel 7 transfer error flag
DMA2_GL1_FLAG: DMA2 channel 1 global flag
DMA2_FDT1_FLAG: DMA2 channel 1 transfer complete flag
DMA2_HDT1_FLAG: DMA2 channel 1 half transfer complete flag

DMA2_DTERR1_FLAG:	DMA2 channel 1 transfer error flag
DMA2_GL2_FLAG:	DMA2 channel 2 global flag
DMA2_FDT2_FLAG:	DMA2 channel 2 transfer complete flag
DMA2_HDT2_FLAG:	DMA2 channel 2 half transfer complete flag
DMA2_DTERR2_FLAG:	DMA2 channel 2 transfer error flag
DMA2_GL3_FLAG:	DMA2 channel 3 global flag
DMA2_FDT3_FLAG:	DMA2 channel 3 transfer complete flag
DMA2_HDT3_FLAG:	DMA2 channel 3 half transfer complete flag
DMA2_DTERR3_FLAG:	DMA2 channel 3 transfer error flag
DMA2_GL4_FLAG:	DMA2 channel 4 global flag
DMA2_FDT4_FLAG:	DMA2 channel 4 transfer complete flag
DMA2_HDT4_FLAG:	DMA2 channel 4 half transfer complete flag
DMA2_DTERR4_FLAG:	DMA2 channel 4 transfer error flag
DMA2_GL5_FLAG:	DMA2 channel 5 global flag
DMA2_FDT5_FLAG:	DMA2 channel 5 transfer complete flag
DMA2_HDT5_FLAG:	DMA2 channel 5 half transfer complete flag
DMA2_DTERR5_FLAG:	DMA2 channel 5 transfer error flag
DMA2_GL6_FLAG:	DMA2 channel 6 global flag
DMA2_FDT6_FLAG:	DMA2 channel 6 transfer complete flag
DMA2_HDT6_FLAG:	DMA2 channel 6 half transfer complete flag
DMA2_DTERR6_FLAG:	DMA2 channel 6 transfer error flag
DMA2_GL7_FLAG:	DMA2 channel 7 global flag
DMA2_FDT7_FLAG:	DMA2 channel 7 transfer complete flag
DMA2_HDT7_FLAG:	DMA2 channel 7 half transfer complete flag
DMA2_DTERR7_FLAG:	DMA2 channel 7 transfer error flag

Example:

```
if(dma_flag_get(DMA2_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    dma_flag_clear(DMA2_FDT1_FLAG);
}
```

5.9 External interrupt/event controller (EXINT)

The EXINT register structure exint_type is defined in the “at32f413_exint.h”.

```
/**
  * @brief type define exint register all
  */
typedef struct
{
    …
} exint_type;
```

The table below gives a list of the EXINT registers.

Table 164. Summary of EXINT registers

Register	Description
inten	Interrupt enable register
evten	Event enable register
polcfg1	Polarity configuration register 1
polcfg2	Polarity configuration register 2
swtrg	Software trigger register
intsts	Interrupt status register

The table below gives a list of the EXINT library functions.

Table 165. Summary of EXINT library functions

Function name	Description
exint_reset	Reset all EXINT registers to their reset values
exint_default_para_init	Configure the EXINT initial structure with the initial value
exint_init	Initialize EXINT
exint_flag_clear	Clear the selected EXINT interrupt flag
exint_flag_get	Read the selected EXINT interrupt flag
exint_software_interrupt_event_generate	Software interrupt event generation
exint_interrupt_enable	Enable the selected EXINT interrupt
exint_event_enable	Enable the selected EXINT event

5.9.1 exint_reset function

The table below describes the function exint_reset.

Table 166. exint_reset function

Name	Description
Function name	exint_reset
Function prototype	void exint_reset(void);
Function description	Reset all EXINT registers to their reset values
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
exint_reset ();
```

5.9.2 exint_default_para_init function

The table below describes the function exint_default_para_init.

Table 167. exint_default_para_init function

Name	Description
Function name	exint_default_para_init
Function prototype	void exint_default_para_init(exint_init_type *exint_struct);
Function description	Configure the EXINT initial structure with the initial value
Input parameter 1	exint_struct: *exint_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of exint_init_type before starting.
Called functions	NA

Example:

```
exint_init_type exint_init_struct;
exint_default_para_init(&exint_init_struct);
```

5.9.3 exint_init function

The table below describes the function exint_init.

Table 168. exint_init function

Name	Description
Function name	exint_init
Function prototype	void exint_init(exint_init_type *exint_struct);
Function description	Initialize EXINT
Input parameter 1	*exint_init_type:* exint_init_struct pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of exint_init_type before starting.
Called functions	NA

The exint_init_type is defined in the at32f413_exint.h.

```
typedef struct
{
    exint_line_mode_type            line_mode;
    uint32_t                        line_select;
    exint_polarity_config_type      line_polarity;
    confirm_state                   line_enable;
} exint_init_type;
```

line_mode

Select event mode or interrupt mode

EXINT_LINE_INTERRUPUT: Interrupt mode

EXINT_LINE_EVENT: Event mode

line_select

Line selection

EXINT_LINE_NONE: No line

EXINT_LINE_0: line0

EXINT_LINE_1: line1

…

EXINT_LINE_18: line18

line_polarity

Trigger edge selection

EXINT_TRIGGER_RISING_EDGE: Rising edge

EXINT_TRIGGER_FALLING_EDGE: Falling edge

EXINT_TRIGGER_BOTH_EDGE: Rising/falling edge

line_enable

Enable/disable line

FALSE: Disable line

TRUE: Enable line

Example:

```
exint_init_type exint_init_struct;
exint_default_para_init(&exint_init_struct);
```

```
exint_init_struct.line_enable = TRUE;
exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
exint_init_struct.line_select = EXINT_LINE_0;
exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
exint_init(&exint_init_struct);
```

5.9.4 exint_flag_clear function

The table below describes the function exint_flag_clear.

Table 169. exint_flag_clear function

Name	Description
Function name	exint_flag_clear
Function prototype	void exint_flag_clear(uint32_t exint_line);
Function description	Clear the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to *line_select* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_flag_clear(EXINT_LINE_0);
```

5.9.5 exint_flag_get function

The table below describes the function exint_flag_get.

Table 170. exint_flag_get function

Name	Description
Function name	exint_flag_get
Function prototype	flag_status exint_flag_get(uint32_t exint_line);
Function description	Get the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to the *line_select* for details.
Output parameter	NA
Return value	flag_status: indicates the status of the selected flag This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
flag_status status = RESET;
status = exint_flag_get(EXINT_LINE_0);
```

5.9.6 exint_interrupt_flag_get function

The table below describes the function exint_interrupt_flag_get.

Table 171. exint_interrupt_flag_get function

Name	Description
Function name	exint_interrupt_flag_get
Function prototype	flag_status exint_interrupt_flag_get(uint32_t exint_line)
Function description	Get the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to the *line_select* for details.
Output parameter	NA
Return value	flag_status: indicates the status of the selected flag This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example

```
flag_status status = RESET;
status = exint_interrupt_flag_get (EXINT_LINE_0);
```

5.9.7 exint_software_interrupt_event_generate function

The table below describes the function exint_software_interrupt_event_generate.

Table 172. exint_software_interrupt_event_generate function

Name	Description
Function name	exint_software_interrupt_event_generate
Function prototype	void exint_software_interrupt_event_generate(uint32_t exint_line);
Function description	Generate software interrupt event
Input parameter	exint_line: line selection Refer to the *line_select* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_software_interrupt_event_generate (EXINT_LINE_0);
```

5.9.8 exint_interrupt_enable function

The table below describes the function exint_interrupt_enable.

Table 173. exint_interrupt_enable function

Name	Description
Function name	exint_interrupt_enable
Function prototype	void exint_interrupt_enable(uint32_t exint_line, confirm_state new_state);
Function description	Enable the selected EXINT interrupt
Input parameter 1	exint_line: line selection Refer to the *line_select* for details.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_interrupt_enable (EXINT_LINE_0);
```

5.9.9 exint_event_enable function

The table below describes the function exint_event_enable.

Table 174. exint_event_enable function

Name	Description
Function name	exint_event_enable
Function prototype	void exint_event_enable(uint32_t exint_line, confirm_state new_state);
Function description	Enable the selected EXINT event
Input parameter 1	exint_line: line selection Refer to the *line_select* for details.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_event_enable (EXINT_LINE_0);
```

5.10 Flash memory controller (FLASH)

The FLASH register structure flash_type is defined in the “at32f413_flash.h”.

```
/**
  * @brief type define flash register all
  */
typedef struct
{
       …
} flash_type;
```

The table below gives a list of the FLASH registers.

Table 175. Summary of FLASH registers

Register	Description
flash_psr	Flash performance select register
flash_unlock	Flash unlock register
flash_usd_unlock	Flash user system data unlock register
flash_sts	Flash status register
flash_ctrl	Flash control register
flash_addr	Flash address register
flash_usd	User system data register
flash_epps	Erase/program protection status register
flash_unlock3	Flash unlock register 3
flash_select	Flash select register
flash_sts3	Flash status register 3
flash_ctrl3	Flash control register 3
flash_addr3	Flash address register 3
flash_da	Flash decryption address register
slib_sts0	Flash security library status register 0
slib_sts1	Flash security library status register 1
slib_pwd_clr	Flash security library password clear register
slib_misc_sts	Security library additional status register
slib_set_pwd	Security library password setting register
slib_set_range	Security library address setting register
slib_unlock	Security library unlock register
flash_crc_ctrl	Flash CRC check control register
flash_crc_chkr	Flash CRC check result register

The table below gives a list of the FLASH library functions.

Table 176. Summary of FLASH library functions

Function name	Description
flash_flag_get	Get flag status
flash_flag_clear	Clear flag
flash_operation_status_get	Get operation staus (Flash memory)
flash_spim_operation_status_get	Get operation status (external memory)
flash_operation_wait_for	Wait for operation complete (Flash memory)
flash_spim_operation_wait_for	Wait for operation complete (external memory)
flash_unlock	Unlock flash memory
flash_spim_unlock	Unlock external memory
flash_lock	Lock flash memory bank
flash_spim_lock	Lock external memory
flash_sector_erase	Erase Flash sector
flash_internal_all_erase	Erase internal Flash memory
flash_spim_all_erase	Erase external memory
flash_user_system_data_erase	Erase user system data
flash_word_program	Flash word programming
flash_halfword_program	Flash half-word programming
flash_byte_program	Flash byte programming
flash_user_system_data_program	User system data programming
flash_epp_set	Erase/programming protection configuration
flash_epp_status_get	Get erase/programming protection status
flash_fap_enable	Configure Flash access protection
flash_fap_status_get	Get Flash access protection status
flash_ssb_set	System configuration byte configuration
flash_ssb_status_get	Get system configuration byte configuration status
flash_interrupt_enable	Flash interrupt configuration
flash_spim_model_select	Select external memory model
flash_spim_encryption_range_set	Configure external memory encryption range
flash_slib_enable	Enable security library
flash_slib_disable	Disable security library
flash_slib_remaining_count_get	Get sLib remaining count
flash_slib_state_get	Get sLib status
flash_slib_start_sector_get	Get sLib start sector
flash_slib_datastart_sector_get	Get sLib data start sector
flash_slib_end_sector_get	Get sLib end sector
flash_crc_calibrate	Flash CRC verify

5.10.1 flash_flag_get function

The table below describes the function flash_flag_get.

Table 177. flash_flag_get function

Name	Description
Function name	flash_flag_get
Function prototype	flag_status flash_flag_get(uint32_t flash_flag);
Function description	Get flag status
Input parameter	flash_flag: flag selection
Output parameter	NA
Return value	flag_status: indicates the flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flash_flag

Flag selection

FLASH_OBF_FLAG: Flash operation busy
FLASH_ODF_FLAG: Flash operation complete
FLASH_PRGMERR_FLAG: Flash programming error
FLASH_EPPERR_FLAG: Flash erase error
FLASH_SPIM_OBF_FLAG: External memory operation busy
FLASH_SPIM_ODF_FLAG: External memory operation complete
FLASH_SPIM_PRGMERR_FLAG: External memory programming error
FLASH_SPIM_EPPERR_FLAG: External memory erase error
FLASH_USDERR_FLAG: User system data area error

Example:

```
flag_status status;
status = flash_flag_get (FLASH_ODF_FLAG);
```

5.10.2 flash_flag_clear function

The table below describes the function flash_flag_clear.

Table 178. flash_flag_clear function

Name	Description
Function name	flash_flag_clear
Function prototype	void flash_flag_clear(uint32_t flash_flag);
Function description	Clear flag
Input parameter	flash_flag: flag selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flash_flag

Flag selection

FLASH_ODF_FLAG: Flash operation complete

FLASH_PRGMERR_FLAG: Flash programming error
FLASH_EPPERR_FLAG: Flash erase error
FLASH_SPIM_ODF_FLAG: External memory operation complete
FLASH_SPIM_PRGMERR_FLAG: External memory programming error
FLASH_SPIM_EPPERR_FLAG: External memory erase error

Example:

```
flash_flag_clear(FLASH_ODF_FLAG);
```

5.10.3 flash_operation_status_get function

The table below describes the function flash_operation_status_get

Table 179. flash_operation_status_get function

Name	Description
Function name	flash_operation_status_get
Function prototype	flash_status_type flash_operation_status_get(void);
Function description	Get Flash operation status
Input parameter	NA
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

flash_status_type

FLASH_OPERATE_BUSY: Operate busy
FLASH_PROGRAM_ERROR: Programming error
FLASH_EPP_ERROR: Erase/program protection error
FLASH_OPERATE_DONE: Flash operation complete
FLASH_OPERATE_TIMEOUT: Flash operation timeout

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_operation_status_get();
```

5.10.4 flash_spim_operation_status_get function

The table below describes the function flash_spim_operation_status_get.

Table 180. flash_spim_operation_status_get function

Name	Description
Function name	flash_spim_operation_status_get
Function prototype	flash_status_type flash_spim_operation_status_get (void);
Function description	Get external memory operation status
Input parameter	NA
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_spim_operation_status_get();
```

5.10.5 flash_operation_wait_for function

The table below describes the function flash_operation_wait_for.

Table 181. flash_operation_wait_for function

Name	Description
Function name	flash_operation_wait_for
Function prototype	flash_status_type flash_operation_wait_for(uint32_t time_out);
Function description	Wait for Flash operation
Input parameter	time_out: wait timeout The wait timeout value is defined in the flash.h file; refer to *flash_time_out*.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

flash_time_out

ERASE_TIMEOUT: Erase timeout
PROGRAMMING_TIMEOUT: Programming timeout
SPIM_ERASE_TIMEOUT: External memory erase timeout
SPIM_PROGRAMMING_TIMEOUT: External memory programming timeout
OPERATION_TIMEOUT: General operation timeout

Example:

```
/* wait for operation to be completed */
status = flash_operation_wait_for(PROGRAMMING_TIMEOUT);
```

5.10.6 flash_spim_operation_wait_for function

The table below describes the function flash_spim_operation_wait_for.

Table 182. flash_spim_operation_wait_for function

Name	Description
Function name	flash_spim_operation_wait_for
Function prototype	flash_status_type flash_spim_operation_wait_for(uint32_t time_out);
Function description	Wait for external memory operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file; refer to *flash_time_out*.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
/* wait for operation to be completed */
status = flash_spim_operation_wait_for(PROGRAMMING_TIMEOUT);
```

5.10.7 flash_unlock function

The table below describes the function flash_unlock.

Table 183. flash_unlock function

Name	Description
Function name	flash_unlock
Function prototype	void flash_unlock(void);
Function description	Unlock Flash memory controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_unlock();
```

5.10.8 flash_spim_unlock function

The table below describes the function flash_spim_unlock

Table 184. flash_spim_unlock function

Name	Description
Function name	flash_spim_unlock
Function prototype	void flash_spim_unlock(void);
Function description	Unlock external memory controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_spim_unlock();
```

5.10.9 flash_lock function

The table below describes the function flash_lock.

Table 185. flash_lock function

Name	Description
Function name	flash_lock
Function prototype	void flash_lock(void);
Function description	Lock Flash memory controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_lock();
```

5.10.10 flash_spim_lock function

The table below describes the function flash_spim_lock.

Table 186. flash_spim_lock function

Name	Description
Function name	flash_spim_lock
Function prototype	void flash_spim_lock(void);
Function description	Lock external memory controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_spim_lock();
```

5.10.11 flash_sector_erase function

The table below describes the function flash_sector_erase.

Table 187. flash_sector_erase function

Name	Description
Function name	flash_sector_erase
Function prototype	flash_status_type flash_sector_erase(uint32_t sector_address);
Function description	Erase data in the selected Flash sector address
Input parameter	sector_address: select the Flash sector address to be erased, usually Flash sector start address
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_sector_erase(0x08001000);
```

5.10.12 flash_internal_all_erase function

The table below describes the function flash_internal_all_erase.

Table 188. flash_internal_all_erase function

Name	Description
Function name	flash_internal_all_erase
Function prototype	flash_status_type flash_internal_all_erase(void);
Function description	Erase internal Flash data
Input parameter	NA
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_internal_all_erase();
```

5.10.13 flash_spim_all_erase function

The table below describes the function flash_spim_all_erase.

Table 189. flash_spim_all_erase function

Name	Description
Function name	flash_spim_all_erase
Function prototype	flash_status_type flash_spim_all_erase(void);
Function description	Erase external memory data
Input parameter	NA
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_spim_unlock();
status = flash_spim_all_erase();
```

5.10.14 flash_user_system_data_erase function

The table below describes the function flash_user_system_data_erase.

Table 190. flash_user_system_data_erase function

Name	Description
Function name	flash_user_system_data_erase
Function prototype	flash_status_type flash_user_system_data_erase(void);
Function description	Erase user system data
Input parameter	NA
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Note: As this function remains in FAP state, it only erases data except FAP in the user system data area.

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_user_system_data_erase();
```

5.10.15 flash_word_program function

The table below describes the function flash_word_program

Table 191. flash_word_program function

Name	Description
Function name	flash_word_program
Function prototype	flash_status_type flash_word_program(uint32_t address, uint32_t data);
Function description	Write one word data to a given address
Input parameter 1	address: programmed address, word-aligned
Input parameter 2	data: programmed data
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	The programming operation can be allowed only when data in the address are all 0xFF.
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
uint32_t i;
flash_unlock();
status = flash_sector_erase(0x08001000);
if(status = FLASH_OPERATE_DONE)
{
  /* program 256 words */
  for(I = 0; I < 256; i++)
  {
    status = flash_word_program(0x08001000 + i*4, i);
```

```
    }
}
```

5.10.16 flash_halfword_program function

The table below describes the function flash_halfword_program.

Table 192. flash_halfword_program function

Name	Description
Function name	flash_halfword_program
Function prototype	flash_status_type flash_halfword_program(uint32_t address, uint16_t data);
Function description	Write a half-word data to a given address
Input parameter 1	address: programmed address, half-word-aligned
Input parameter 2	data: programmed data
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	The programming operation can be allowed only when data in the address are all 0xFF.
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
uint32_t i;
flash_unlock();
status = flash_sector_erase(0x08001000);
if(status = FLASH_OPERATE_DONE)
{
  /* program 256 halfwords */
  for(I = 0; I < 256; i++)
  {
    status = flash_halfword_program(0x08001000 + i*2, (uint16_t)i);
  }
}
```

5.10.17 flash_byte_program function

The table below describes the function flash_byte_program.

Table 193. flash_byte_program function

Name	Description
Function name	flash_byte_program
Function prototype	flash_status_type flash_byte_program(uint32_t address, uint8_t data);
Function description	Program a byte data to a given address
Input parameter 1	address: programmed address
Input parameter 2	data: programmed data
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	The programming operation can be allowed only when data in the address are all 0xFF.
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
uint32_t i;
flash_unlock();
status = flash_sector_erase(0x08001000);
if(status = FLASH_OPERATE_DONE)
{
  /* program 256 bytes */
  for(I = 0; I < 256; i++)
  {
    status = flash_byte_program(0x08001000 + i*2, (uint8_t)i);
  }
}
```

5.10.18 flash_user_system_data_program function

The table below describes the function flash_user_system_data_program.

Table 194. flash_user_system_data_program function

Name	Description
Function name	flash_user_system_data_program
Function prototype	flash_status_type flash_user_system_data_program (uint32_t address, uint8_t data);
Function description	Program a byte data to a given address in the user system data area
Input parameter 1	address: programmed address
Input parameter 2	data: programmed data
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	The programming operation can be allowed only when data and its inverse data in the user system data area are all 0xFF.
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_user_system_data_erase();
if(status = FLASH_OPERATE_DONE)
{
  /* program user system data */
  status = flash_user_system_data_program(0x1FFFF804, 0x55);
}
```

5.10.19 flash_epp_set function

The table below describes the function flash_epp_set.

Table 195. flash_epp_set function

Name	Description
Function name	flash_epp_set
Function prototype	flash_status_type flash_epp_set(uint32_t *sector_bits);
Function description	Enable erase programming protection
Input parameter	*sector_bits: Erase programming protection sector address pointer. Each bit protects 4 KB sectors, and the last bit protects the remaining sectors. Setting this bit to 1 enables sector protection.
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
uint32_t epp_val;
flash_unlock();
status = flash_user_system_data_erase();
if(status = FLASH_OPERATE_DONE)
{
  epp_val = 0x00000001;
  /* program epp */
  status = flash_epp_set(&epp_val);
}
```

5.10.20 flash_epp_status_get function

The table below describes the function flash_epp_status_get.

Table 196. flash_epp_status_get function

Name	Description
Function name	flash_epp_status_get
Function prototype	void flash_epp_status_get(uint32_t *sector_bits);
Function description	Get the status of erase programming protection
Input parameter	NA
Output parameter	*sector_bits: Erase programming protection sector address pointer. Each bit protects 4KB sectors, and the last bit protects the remaining sectors. Setting this bit to 1 enables sector protection.
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t epp_val;
/* get epp status */
flash_epp_status_get(&epp_val);
```

5.10.21 flash_fap_enable function

The table below describes the function flash_fap_enable.

Table 197. flash_fap_enable function

Name	Description
Function name	flash_fap_enable
Function prototype	flash_status_type flash_fap_enable(confirm_state new_state);
Function description	Enable Flash access protection
Input parameter	new_state: Flash access protection status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Note: This function will erase the whole user system data area. If there were data programmed in the user system data area before calling this function, they have to be re-programmed after calling this function.

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_fap_enable(TRUE);
```

5.10.22 flash_fap_status_get function

The table below describes the function flash_fap_status_get.

Table 198. flash_fap_status_get function

Name	Description
Function name	flash_fap_status_get
Function prototype	flag_status flash_fap_status_get(void);
Function description	Get the status of Flash access protection
Input parameter	NA
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
flag_status status;
status = flash_fap_status_get();
```

5.10.23 flash_ssb_set function

The table below describes the function flash_ssb_set.

Table 199. flash_ssb_set function

Name	Description
Function name	flash_ssb_set
Function prototype	flash_status_type flash_ssb_set(uint8_t usd_ssb);
Function description	Configure system setting bytes
Input parameter	usd_ssb: system setting byte value is a combination of the selected data from all data group; refer to *ssb_data_define* for details.
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

ssb_data_define

type 1:

USD_WDT_ATO_DISABLE: Watchdog auto-start disabled

USD_WDT_ATO_ENABLE: Watchdog auto-start enabled

type 2:

USD_DEPSLP_NO_RST: No reset occurs when entering Deepsleep mode

USD_DEPSLP_RST: Reset occurs when entering Deepsleep mode

type 3:

USD_STDBY_NO_RST: No reset occurs when entering Standby mode

USD_STDBY_RST: Reset occurs when entering Standby mode

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
```

```
status = flash_user_system_data_erase();
if(status = FLASH_OPERATE_DONE)
{
  status = flash_ssb_set(USD_WDT_ATO_DISABLE | USD_DEPSLP_NO_RST | USD_STDBY_RST);
}
```

5.10.24 flash_ssb_status_get function

The table below describes the function flash_ssb_status_get.

Table 200. flash_ssb_status_get function

Name	Description
Function name	flash_ssb_status_get
Function prototype	uint8_t flash_ssb_status_get(void);
Function description	Get the status of system setting bytes
Input parameter	NA
Output parameter	NA
Return value	Return system setting byte value. Refer to *ssb_data_define* for details.
Required preconditions	NA
Called functions	NA

Example:

```
uint8_t ssb_val;
ssb_val = flash_ssb_status_get();
```

5.10.25 flash_interrupt_enable function

The table below describes the function flash_interrupt_enable.

Table 201. flash_interrupt_enable function

Name	Description
Function name	flash_interrupt_enable
Function prototype	void flash_interrupt_enable(uint32_t flash_int, confirm_state new_state);
Function description	Enable Flash interrupts
Input parameter 1	flash_int: Flash interrupt type. Refer to *flash_interrupt_***type** for details.
Input parameter 2	new_state: interrupt status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flash_interrupt_type

FLASH_ERR_INT: Flash error interrupt

FLASH_ODF_INT: Flash operation complete interrupt

FLASH_SPIM_ERR_INT: External memory error interrupt

FLASH_SPIM_ODF_INT: External memory operation complete interrupt

Example:

```
flash_interrupt_enable(FLASH_ERR_INT | FLASH_ODF_INT, TRUE);
```

5.10.26 flash_spim_model_select function

The table below describes the function flash_spim_model_select.

Table 202. flash_spim_model_select function

Name	Description
Function name	flash_spim_model_select
Function prototype	void flash_spim_model_select(flash_spim_model_type mode);
Function description	Select external memory type
Input parameter	mode: external memory type; refer to *flash_spim_mode_type* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flash_spim_mode_type

FLASH_SPIM_MODEL1: Model 1

FLASH_SPIM_MODEL2: Model 2

Example:

```
flash_spim_model_select(FLASH_SPIM_MODEL1);
```

5.10.27 flash_spim_encryption_range_set function

The table below describes the function flash_spim_encryption_range_set.

Table 203. flash_spim_encryption_range_set function

Name	Description
Function name	flash_spim_encryption_range_set
Function prototype	void flash_spim_encryption_range_set(uint32_t decode_address);
Function description	Set external memory data encryption range
Input parameter	decode_address: encryption address, word-aligned; data before this address is in ciphertext
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_spim_encryption_range_set(0x08401000);
```

5.10.28 flash_slib_enable function

The table below describes the function flash_slib_enable.

Table 204. flash_slib_enable function

Name	Description
Function name	flash_slib_enable
Function prototype	flash_status_type flash_slib_enable(uint32_t pwd, uint16_t start_sector, uint16_t data_start_sector, uint16_t end_sector);
Function description	Enable security library (sLib) and its address range
Input parameter 1	pwd: The sLib data are saved as ciphertext, associated with encrypted computing. A correct password is entered in order to unlock encryption.
Input parameter 2	start_sector: sLib start sector number
Input parameter 3	data_start_sector: sLib data area instruction start sector number
Input parameter 4	end_sector: sLib end sector number
Output parameter	NA
Return value	Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
status = flash_slib_enable(0x12345678, 0x04 , 0x05, 0x06);
```

5.10.29 flash_slib_disable function

The table below describes the function flash_slib_disable.

Table 205. flash_slib_disable function

Name	Description
Function name	flash_slib_disable
Function prototype	error_status flash_slib_disable(uint32_t pwd);
Function description	Disable security library (sLib)
Input parameter	pwd: sLib password. it must be entered correctly, otherwise it is not allowed to enter until reset.
Output parameter	NA
Return value	Return error status This parameter can be ERROE or SUCCESS.
Required preconditions	NA
Called functions	NA

Note: Successful calling of this function will erase the whole internal Flash memory.

Example:

```
error_status status;
status = flash_slib_disable(0x12345678);
```

5.10.30 flash_slib_remaining_count_get function

The table below describes the function flash_slib_remaining_count_get.

Table 206. flash_slib_remaining_count_get function

Name	Description
Function name	flash_slib_remaining_count_get
Function prototype	uint32_t flash_slib_remaining_count_get(void);
Function description	Get the sLib remaining count
Input parameter	NA
Output parameter	NA
Return value	Return the sLib remaining count
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t num;
num = flash_slib_remaining_count_get();
```

5.10.31 flash_slib_state_get function

The table below describes the function flash_slib_state_get.

Table 207. flash_slib_state_get function

Name	Description
Function name	flash_slib_state_get
Function prototype	flag_status flash_slib_state_get(void);
Function description	Get the status of sLib
Input parameter	NA
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
flag_status status;
status = flash_slib_state_get();
```

5.10.32 flash_slib_start_sector_get function

The table below describes the function flash_slib_start_sector_get.

Table 208. flash_slib_start_sector_get function

Name	Description
Function name	flash_slib_start_sector_get
Function prototype	uint16_t flash_slib_start_sector_get(void);
Function description	Get the start sector number of sLib
Input parameter	NA
Output parameter	NA
Return value	Return the start sector number of sLib
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t num;
num = flash_slib_start_sector_get();
```

5.10.33 flash_slib_datastart_sector_get function

The table below describes the function flash_slib_datastart_sector_get.

Table 209. flash_slib_datastart_sector_get function

Name	Description
Function name	flash_slib_datastart_sector_get
Function prototype	uint16_t flash_slib_datastart_sector_get(void);
Function description	Get the start sector number of sLib data area
Input parameter	NA
Output parameter	NA
Return value	Return the start sector number of sLib data area
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t num;
num = flash_slib_datastart_sector_get();
```

5.10.34 flash_slib_end_sector_get function

The table below describes the function flash_slib_end_sector_get.

Table 210. flash_slib_end_sector_get function

Name	Description
Function name	flash_slib_end_sector_get
Function prototype	uint16_t flash_slib_end_sector_get(void);
Function description	Get the end sector number of sLib
Input parameter	NA
Output parameter	NA
Return value	Return the end sector number of sLib
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t num;
num = flash_slib_end_sector_get();
```

5.10.35 flash_crc_calibrate function

The table below describes the function flash_crc_calibrate.

Table 211. flash_crc_calibrate function

Name	Description
Function name	flash_crc_calibrate
Function prototype	uint32_t flash_crc_calibrate(uint32_t start_sector, uint32_t sector_cnt);
Function description	Enable Flash CRC check
Input parameter 1	start_sector: CRC check start sector
Input parameter 2	sector_cnt: CRC check sector count
Output parameter	NA
Return value	Return CRC calculation result
Required preconditions	NA
Called functions	NA

Note: The sector set to go through CRC check is only allowed to be on a single area, rather than on both security library and common area.

Example:

```
uint32_t crc_val;
crc_val = flash_crc_calibrate(0, 10);
```

5.11 General-purpose I/Os and multiplexed I/Os (GPIO/IOMUX)

The GPIO and IOMUX register structure gpio_type and iomux_type are defined in the "at32f413_gpio.h".

```
/**
  * @brief type define gpio register all
  */
typedef struct
{


} gpio_type;

/**
  * @brief type define iomux register all
  */
typedef struct
{


} iomux_type;
```

The table below gives a list of the GPIO registers.

Table 212. Summary of GPIO registers

Register	Description
cfglr	GPIO configuration register low
cfghr	GPIO configuration register high
idt	GPIO input data register
odt	GPIO output data register
scr	GPIO set/clear register
clr	GPIO clear register
wpr	GPIO write protection register

The table below gives a list of the IOMUX registers.

Table 213. Summary of IOMUX registers

Register	Description
evtout	Event output control register
remap	IOMUX remap register
exintc1	IOMUX external interrupt configure register 1
exintc2	IOMUX external interrupt configure register 2
exintc3	IOMUX external interrupt configure register 3
exintc4	IOMUX external interrupt configure register 4
remap2	IOMUX remap register 2
remap3	IOMUX remap register 3
remap4	IOMUX remap register 4

Register	Description
remap5	IOMUX remap register 5
remap6	IOMUX remap register 6
remap7	IOMUX remap register 7

The table below gives a list of the GPIO and IOMUX library functions.

Table 214. Summary of GPIO and IOMUX library functions

Function name	Description
gpio_reset	GPIO is reset by CRM reset register
gpio_iomux_reset	IOMUX is reset by CRM reset register
gpio_init	Initialize GPIO peripherals
gpio_default_para_init	Initialize GPIO default parameters
gpio_input_data_bit_read	Read GPIO input data bit
gpio_input_data_read	Read GPIO input data
gpio_output_data_bit_read	Read GPIO output data bit
gpio_output_data_read	Read GPIO output data
gpio_bits_set	Set GPIO bits
gpio_bits_reset	Reset GPIO bits
gpio_bits_write	Write GPIO bits
gpio_port_write	Write GPIO ports
gpio_pin_wp_config	Configure GPIO pin write protection
gpio_event_output_config	Configure GPIO event output feature
gpio_event_output_enable	Enable/disable GPIO event output feature
gpio_pin_remap_config	Configure GPIO pin multiplexed function
gpio_exint_line_config	Configure GPIO external interrupt line

5.11.1 gpio_reset function

The table below describes the function gpio_reset.

Table 215. gpio_reset function

Name	Description
Function name	gpio_reset
Function prototype	void gpio_reset(gpio_type *gpio_x);
Function description	GPIO is reset by CRM reset register
Input parameter	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
gpio_reset(GPIOA);
```

5.11.2 gpio_iomux_reset function

The table below describes the function gpio_iomux_reset.

Table 216. gpio_iomux_reset function

Name	Description
Function name	gpio_iomux_reset
Function prototype	void gpio_iomux_reset ();
Function description	IOMUX is reset by CRM reset register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
gpio_iomux_reset();
```

5.11.3 gpio_init function

The table below describes the function gpio_init.

Table 217. gpio_init function

Name	Description
Function name	gpio_init
Function prototype	void gpio_init(gpio_type *gpio_x, gpio_init_type *gpio_init_struct);
Function description	Initialize GPIO peripheral
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	gpio_init_struct: gpio_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

gpio_init_type structure

The gpio_init_type is defined in the at32f413_gpio.h.

```
typedef struct
{
    uint32_t                gpio_pins;
    gpio_output_type        gpio_out_type;
    gpio_pull_type          gpio_pull;
    gpio_mode_type          gpio_mode;
    gpio_drive_type         gpio_drive_strength;
} gpio_init_type;
```

gpio_pins

Select a GPIO pin.

GPIO_PINS_0: GPIO pin 0

GPIO_PINS_1: GPIO pin 1

GPIO_PINS_2: GPIO pin 2
GPIO_PINS_3: GPIO pin 3
GPIO_PINS_4: GPIO pin 4
GPIO_PINS_5: GPIO pin 5
GPIO_PINS_6: GPIO pin 6
GPIO_PINS_7: GPIO pin 7
GPIO_PINS_8: GPIO pin 8
GPIO_PINS_9: GPIO pin 9
GPIO_PINS_10: GPIO pin 10
GPIO_PINS_11: GPIO pin 11
GPIO_PINS_12: GPIO pin 12
GPIO_PINS_13: GPIO pin 13
GPIO_PINS_14: GPIO pin 14
GPIO_PINS_15: GPIO pin 15

gpio_out_type

Set GPIO output type.

GPIO_OUTPUT_PUSH_PULL: GPIO push-pull
GPIO_OUTPUT_OPEN_DRAIN: GPIOopen-drain

gpio_pull

Set GPIO pull-up or pull-down.

GPIO_PULL_NONE: No GPIO pull-up/pull-down
GPIO_PULL_UP: GPIO pull-up
GPIO_PULL_DOWN: GPIO pull-down

gpio_mode

Set GPIO mode.

GPIO_MODE_INPUT: GPIO input mode
GPIO_MODE_OUTPUT: GPIO output mode
GPIO_MODE_MUX: GPIO multiplexed mode
GPIO_MODE_ANALOG: GPIO analog mode

gpio_drive_strength

Set GPIO drive capability.

GPIO_DRIVE_STRENGTH_STRONGER: Strong drive strength
GPIO_DRIVE_STRENGTH_MODERATE: Moderate drive strength

Example:

```
gpio_init_type gpio_init_struct;
gpio_init_struct.gpio_pins = GPIO_PINS_0;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOA, &gpio_init_struct);
```

5.11.4 gpio_default_para_init function

The table below describes the function gpio_default_para_init.

Table 218. gpio_default_para_init function

Name	Description
Function name	gpio_default_para_init
Function prototype	void gpio_default_para_init(gpio_init_type *gpio_init_struct);
Function description	Initialize GPIO default parameters.
Input parameter	gpio_init_struct: gpio_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the gpio_init_struct.

Table 219. gpio_init_struct default values

Member	Default value
gpio_pins	GPIO_PINS_ALL
gpio_mode	GPIO_MODE_INPUT
gpio_out_type	GPIO_OUTPUT_PUSH_PULL
gpio_pull	GPIO_PULL_NONE
gpio_drive_strength	GPIO_DRIVE_STRENGTH_STRONGER

Example:

```
gpio_init_type gpio_init_struct;
gpio_default_para_init(&gpio_init_struct);
```

5.11.5 gpio_input_data_bit_read function

The table below describes the function gpio_input_data_bit_read.

Table 220. gpio_input_data_bit_read function

Name	Description
Function name	gpio_input_data_bit_read
Function prototype	flag_status gpio_input_data_bit_read(gpio_type *gpio_x, uint16_t pins);
Function description	Read GPIO input port pins
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Output parameter	NA
Return value	Return GPIO input pin status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_input_data_bit_read(GPIOA, GPIO_PINS_0);
```

5.11.6 gpio_input_data_read function

The table below describes the function gpio_input_data_read.

Table 221. gpio_input_data_read function

Name	Description
Function name	gpio_input_data_read
Function prototype	uint16_t gpio_input_data_read(gpio_type *gpio_x);
Function description	Read GPIO input ports
Input parameter	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Output parameter	NA
Return value	Return GPIO input port status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_input_data_read(GPIOA);
```

5.11.7 gpio_output_data_bit_read function

The table below describes the function gpio_output_data_bit_read.

Table 222. gpio_output_data_bit_read function

Name	Description
Function name	gpio_output_data_bit_read
Function prototype	uint16_t gpio_output_data_bit_read(gpio_type *gpio_x);
Function description	Read GPIO output port pin
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Output parameter	NA
Return value	Return GPIO output pin status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_output_data_bit_read(GPIOA, GPIO_PINS_0);
```

5.11.8 gpio_output_data_read function

The table below describes the function gpio_output_data_read.

Table 223. gpio_output_data_read function

Name	Description
Function name	gpio_output_data_read
Function prototype	uint16_t gpio_output_data_read(gpio_type *gpio_x);
Function description	Read GPIO output port
Input parameter	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Output parameter	NA
Return value	Return GPIO output port status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_output_data_read(GPIOA);
```

5.11.9 gpio_bits_set function

The table below describes the function gpio_bits_set.

Table 224. gpio_bits_set function

Name	Description
Function name	gpio_bits_set
Function prototype	void gpio_bits_set(gpio_type *gpio_x, uint16_t pins);
Function description	Set GPIO pins
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_bits_set(GPIOA, GPIO_PINS_0);
```

5.11.10 gpio_bits_reset function

The table below describes the function gpio_bits_reset.

Table 225. gpio_bits_reset function

Name	Description
Function name	gpio_bits_reset
Function prototype	void gpio_bits_reset(gpio_type *gpio_x, uint16_t pins);
Function description	Reset GPIO pins
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	pins: indicates the GPIO pin; refer to *gpio_pins* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_bits_reset(GPIOA, GPIO_PINS_0);
```

5.11.11 gpio_bits_write function

The table below describes the function gpio_bits_write.

Table 226. gpio_bits_write function

Name	Description
Function name	gpio_bits_write
Function prototype	void gpio_bits_write(gpio_type *gpio_x, uint16_t pins, confirm_state bit_state);
Function description	Write GPIO pins
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	pins: indicates the GPIO pin; refer to *gpio_pins* for details.
Input parameter 3	bit_state: indicates the GPIO pin value; it can be 1 (TRUE) or 0 (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_bits_write(GPIOA, GPIO_PINS_0, TRUE);
```

5.11.12 gpio_port_write function

The table below describes the function gpio_port_write.

Table 227. gpio_port_write function

Name	Description
Function name	gpio_port_write
Function prototype	void gpio_port_write(gpio_type *gpio_x, uint16_t port_value);
Function description	Write GPIO ports
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	port_value: indicates the port value to write This parameter can be 0x0000~0xFFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_port_write(GPIOA, 0xFFFF);
```

5.11.13 gpio_pin_wp_config function

The table below describes the function gpio_pin_wp_config.

Table 228. gpio_pin_wp_config function

Name	Description
Function name	gpio_pin_wp_config
Function prototype	void gpio_pin_wp_config(gpio_type *gpio_x, uint16_t pins);
Function description	Configure GPIO pin write protection
Input parameter 1	gpio_x: select a GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.
Input parameter 2	pins: indicates the GPIO pin; refer to *gpio_pins* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_pin_wp_config(GPIOA, GPIO_PINS_0);
```

5.11.14 gpio_event_output_config function

The table below describes the function gpio_event_output_config.

Table 229. gpio_event_output_config function

Name	Description
Function name	gpio_event_output_config
Function prototype	void gpio_event_output_config(gpio_port_source_type gpio_port_source, gpio_pins_source_type gpio_pin_source);
Function description	Configure GPIO event output feature
Input parameter 1	gpio_port_source: indicates the GPIO port
Input parameter 2	gpio_pin_source: indicates the GPIO pin
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

gpio_port_source

Set a GPIO port.

GPIO_PORT_SOURCE_GPIOA: GPIO port A
GPIO_PORT_SOURCE_GPIOB: GPIO port B
GPIO_PORT_SOURCE_GPIOC: GPIO port C
GPIO_PORT_SOURCE_GPIOD: GPIO port D
GPIO_PORT_SOURCE_GPIOF: GPIO port F

gpio_pin_source

Set a GPIO pin.

GPIO_PINS_SOURCE0: GPIO pin 0
GPIO_PINS_SOURCE1: GPIO pin 1
GPIO_PINS_SOURCE2: GPIO pin 2
GPIO_PINS_SOURCE3: GPIO pin 3
GPIO_PINS_SOURCE4: GPIO pin 4
GPIO_PINS_SOURCE5: GPIO pin 5
GPIO_PINS_SOURCE6: GPIO pin 6
GPIO_PINS_SOURCE7: GPIO pin 7
GPIO_PINS_SOURCE8: GPIO pin 8
GPIO_PINS_SOURCE9: GPIO pin 9
GPIO_PINS_SOURCE10: GPIO pin 10
GPIO_PINS_SOURCE11: GPIO pin 11
GPIO_PINS_SOURCE12: GPIO pin 12
GPIO_PINS_SOURCE13: GPIO pin 13
GPIO_PINS_SOURCE14: GPIO pin 14
GPIO_PINS_SOURCE15: GPIO pin 15

Example:

```
gpio_event_output_config(GPIO_PORT_SOURCE_GPIOA, GPIO_PINS_SOURCE0);
```

5.11.15 gpio_event_output_enable function

The table below describes the function gpio_event_output_enable.

Table 230. gpio_event_output_enable function

Name	Description
Function name	gpio_event_output_enable
Function prototype	void gpio_event_output_enable(confirm_state new_state);
Function description	Enable/disable GPIO event output feature
Input parameter	new_state: indicates the GPIO event output status Enable (TRUE) or disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_event_output_enable(TRUE);
```

5.11.16 gpio_pin_remap_config function

The table below describes the function gpio_pin_remap_config.

Table 231. gpio_pin_remap_config function

Name	Description
Function name	gpio_pin_remap_config
Function prototype	void gpio_pin_remap_config(uint32_t gpio_remap, confirm_state new_state);
Function description	Configure GPIO pin multiplexed function
Input parameter 1	gpio_remap: indicates a GPIO peripheral
Input parameter 2	new_state: GPIO pin multiplexed status Enable (TRUE) or disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

gpio_remap

Select a GPIO peripheral, including (for details, refer to the Reference Manual):

SPI1_MUX_01: spi1_cs/i2s1_ws(pa15), spi1_sck/i2s1_ck(pb3), spi1_miso(pb4), spi1_mosi/i2s1_sd(pb5), i2s1_mck(pb0)

I2C1_MUX: i2c1_scl(pb8), i2c1_sda(pb9)

…

SWJTAG_GMUX_100: SWJ and JTAG are disabled (jtag-dp + sw-dp)

PD01_GMUX: pd0/pd1 mapped on osc_in/osc_out

Example:

```
gpio_pin_remap_config(SPI1_MUX_01, TRUE);
```

5.11.17 gpio_exint_line_config function

The table below describes the function gpio_exint_line_config.

Table 232. gpio_exint_line_config function

Name	Description
Function name	gpio_exint_line_config
Function prototype	void gpio_exint_line_config(gpio_port_source_type gpio_port_source, gpio_pins_source_type gpio_pin_source);
Function description	Configure GPIO external interrupt line
Input parameter 1	gpio_port_source: indicates a GPIO port
Input parameter 2	gpio_pin_source: indicates a GPIO pin
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

gpio_port_source

Set a GPIO port. Refer to *gpio_port_source* for details.

gpio_pin_source

Set a GPIO pin. Refer to *gpio_pin_source* for details.

Example:

```
gpio_exint_line_config(GPIO_PORT_SOURCE_GPIOA, GPIO_PINS_SOURCE0);
```

5.12 I2C interfaces

The I2C register structure i2c_type is defined in the “at32f413_i2c.h”.

```
/**
  * @brief type define i2c register all
  */
typedef struct
{


} i2c_type;
```

The table below gives a list of the I2C registers.

Table 233. Summary of I2C registers

Register	Description
ctrl1	I2C control register 1
ctrl2	I2C control register 2
oaddr1	I2C own address register 1
oaddr2	I2C own address register 2
dt	I2C data register
sts1	I2C status register 1
sts2	I2C status register 2
clkctrl	I2C clock control register
tmrise	I2C clock rise time register

The table below gives a list of the I2C library functions.

Table 234. Summary of I2C library functions

Function name	Description
i2c_reset	I2C peripheral reset
i2c_software_reset	I2C software reset
i2c_init	Initialize I2C and set bus speed
i2c_own_address1_set	Set I2C own address 1
i2c_own_address2_set	Set I2C own address 2
i2c_own_address2_enable	Enable I2C own address 2
i2c_smbus_enable	Enable Smbus mode
i2c_enable	Enable I2C
i2c_fast_mode_duty_set	Set fast mode duty cycle
i2c_clock_stretch_enable	Enable clock stretching capability
i2c_ack_enable	Enable ACK response
i2c_master_receive_ack_set	Set master receive mode acknowledge control
i2c_pec_position_set	Set PEC position in smbus mode and master receive mode
i2c_general_call_enable	Enable general call (broadcast address enable)
i2c_arp_mode_enable	Enable SMBus ARP address
i2c_smbus_mode_set	Set SMBus device mode
i2c_smbus_alert_set	SetSMBus alert pin level

i2c_pec_transmit_enable	Enable PEC transmission
i2c_pec_calculate_enable	Enable PEC calculation
i2c_pec_value_get	Get the current PEC value
i2c_dma_end_transfer_set	Set DMA transfer end indication
i2c_dma_enable	Enable DMA transfer
i2c_interrupt_enable	Enable I2C interrupt
i2c_start_generate	Generate start condition
i2c_stop_generate	Generate stop condition
i2c_7bit_address_send	Send 7-bit slave address
i2c_data_send	Send data
i2c_data_receive	Receive data
i2c_flag_get	Get flag
i2c_flag_clear	Clear flag

Table 235. Summary of I2C application-layer library functions

Function name	Description
i2c_config	I2C application initialization
i2c_lowlevel_init	I2C low-layer initialization
i2c_wait_end	I2C wait data transmit complete
i2c_wait_flag	I2C wait flag
i2c_master_transmit	I2C master transmits data (polling mode)
i2c_master_receive	I2C master receives data (polling mode)
i2c_slave_transmit	I2C slave transmits data (polling mode)
i2c_slave_receive	I2C slave receives data (polling mode)
i2c_master_transmit_int	I2C master transmits data (interrupt mode)
i2c_master_receive_int	I2C master receives data (interrupt mode)
i2c_slave_transmit_int	I2C slave transmits data (interrupt mode)
i2c_slave_receive_int	I2C slave receives data (interrupt mode)
i2c_master_transmit_dma	I2C master transmits data (DMA mode)
i2c_master_receive_dma	I2C master receives data (DMA mode)
i2c_slave_transmit_dma	I2C slave transmits data (DMA mode)
i2c_slave_receive_dma	I2C slave receives data (DMA mode)
i2c_memory_write	I2C writes data to EEPROM (polling mode)
i2c_memory_write_int	I2C writes data to EEPROM (interrupt mode)
i2c_memory_write_dma	I2C writes data to EEPROM (DMA mode)
i2c_memory_read	I2C reads data from EEPROM (polling mode)
i2c_memory_read_int	I2C reads data from EEPROM (interrupt mode)
i2c_memory_read_dma	I2C reads data from EEPROM (DMA mode)
i2c_evt_irq_handler	I2C event interrupt function
i2c_err_irq_handler	I2C error interrupt function
i2c_dma_tx_irq_handler	I2C DMA Tx interrupt function
i2c_dma_rx_irq_handler	I2C DMA Rx interrupt function

5.12.1 i2c_reset function

The table below describes the function i2c_reset.

Table 236. i2c_reset function

Name	Description
Function name	i2c_reset
Function prototype	void i2c_reset(i2c_type *i2c_x)
Function description	Reset all I2C registers to their reset values through CRM (clock and reset management)
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);

Example:

```
i2c_reset(I2C1);
```

5.12.2 i2c_software_reset function

The table below describes the function i2c_software_reset.

Table 237. i2c_software_reset function

Name	Description
Function name	i2c_software_reset
Function prototype	void i2c_software_reset(i2c_type *i2c_x, confirm_state new_state);
Function description	Reset I2C registers by software, obtaining the same result of i2c_reset(i2c_type *i2c_x)
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: software reset status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_software_reset(I2C1, TRUE);
i2c_software_reset(I2C1, FALSE);
```

5.12.3 i2c_init function

The table below describes the function i2c_init.

Table 238. i2c_init function

Name	Description
Function name	i2c_init
Function prototype	void i2c_init(i2c_type *i2c_x, i2c_fsmode_duty_cycle_type duty, uint32_t speed);
Function description	Set I2C bus speed and duty cycle in fast mode
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	duty: SCL bus duty cycle in fast mode Refer to the “duty” description below for details.
Input parameter 3	speed: bus speed, in Hz
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

duty

Set SCL bus duty cycle in fast mode (bus speed ≥400 kHz).

I2C_FSMODE_DUTY_2_1: 2: 1

I2C_FSMODE_DUTY_16_9: 16: 9

Example:

```
i2c_init(I2C1, I2C_FSMODE_DUTY_2_1, 100000);
```

5.12.4 i2c_own_address1_set function

The table below describes the function i2c_own_address1_set.

Table 239. i2c_own_address1_set function

Name	Description
Function name	i2c_own_address1_set
Function prototype	void i2c_own_address1_set(i2c_type *i2c_x, i2c_address_mode_type mode, uint16_t address);
Function description	Set I2C own address 1
Input parameter 1	mode: I2C own address 1 mode Refer to the “mode” description below for details.
Input parameter 2	address: I2C own address 1
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

mode

Set I2C own address 1 mode.

I2C_ADDRESS_MODE_7BIT: 7-bit address mode

I2C_ADDRESS_MODE_10BIT: 10-bit address mode

Example:

```
i2c_own_address1_set(I2C1, I2C_ADDRESS_MODE_7BIT, 0xA0);
```

5.12.5 i2c_own_address2_set function

The table below describes the function i2c_own_address2_set.

Table 240. i2c_own_address2_set function

Name	Description
Function name	i2c_own_address2_set
Function prototype	void i2c_own_address2_set(i2c_type *i2c_x, uint8_t address);
Function description	Set I2C own address 2. This address is valid after the own address 2 is enabled. Note that only 7-bit address is supported, not 10-bit address.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	address: own address 2
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_own_address2_set(I2C1, 0xB0);
```

5.12.6 i2c_own_address2_enable function

The table below describes the function i2c_own_address2_enable.

Table 241. i2c_own_address2_enable function

Name	Description
Function name	i2c_own_address2_enable
Function prototype	void i2c_own_address2_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable I2C own address 2. The own address 2 is vaid after it is enabled. This function is used together with the i2c_own_address2_set.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: enable/disable own address 2 This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_own_address2_enable(I2C1, TRUE);
```

5.12.7 i2c_smbus_enable function

The table below describes the function i2c_smbus_enable.

Table 242. i2c_smbus_enable function

Name	Description
Function name	i2c_smbus_enable
Function prototype	void i2c_smbus_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable SMBus mode. It is I2C mode by default after power-on reset.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: enable/disable SMBus mode This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_enable(I2C1, TRUE);
```

5.12.8 i2c_enable function

The table below describes the function i2c_enable.

Table 243. i2c_enable function

Name	Description
Function name	i2c_enable
Function prototype	void i2c_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable I2C
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: enable/disable I2C This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_enable(I2C1, TRUE);
```

5.12.9 i2c_fast_mode_duty_set function

The table below describes the function i2c_fast_mode_duty_set.

Table 244. i2c_fast_mode_duty_set function

Name	Description
Function name	i2c_fast_mode_duty_set
Function prototype	void i2c_fast_mode_duty_set(i2c_type *i2c_x, i2c_fsmode_duty_cycle_type duty);
Function description	Set the ratio of SCL bus duty cycle in fast mode.. This function has the same function as “duty” in the void i2c_init(i2c_type *i2c_x, i2c_fsmode_duty_cycle_type duty, uint32_t speed).
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	duty: SCL bus duty cycle in fast mode Refer to the “duty” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

duty

Set SCL bus duty cycle in fast mode (bus speed ≥400 kHz).

I2C_FSMODE_DUTY_2_1: 2: 1

I2C_FSMODE_DUTY_16_9: 16: 9

Example:

```
i2c_fast_mode_duty_set(I2C1, I2C_FSMODE_DUTY_2_1);
```

5.12.10 i2c_clock_stretch_enable function

The table below describes the function i2c_clock_stretch_enable.

Table 245. i2c_clock_stretch_enable function

Name	Description
Function name	i2c_clock_stretch_enable
Function prototype	void i2c_clock_stretch_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable clock stretching mode. This function is applicable to slave mode only. In most cases, enabling the clock stretching mode is recommended in order to prevent slave from having no sufficient time to receive or send data due to slow process speed, which causes a loss of data. It should be noted that the host must be able to support clock stretching function before using this mode by slave. For example, some hosts based on IO analog are not equipped with the clock stretching capability.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: enable/disable clock stretching mode This parameter can be TRUE or FALSE.
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_clock_stretch_enable(I2C1, TRUE);
```

5.12.11 i2c_ack_enable function

The table below describes the function i2c_ack_enable

Table 246. i2c_ack_enable function

Name	Description
Function name	i2c_ack_enable
Function prototype	void i2c_ack_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	This function is used to enable ACK or NACK of each byte in master and slave mode. For ACK information on I2C communication protocol, refer to I2C protocol or AT32 reference manual.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: indicates ACK response status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_ack_enable(I2C1, TRUE);
```

5.12.12 i2c_master_receive_ack_set function

The table below describes the function i2c_master_receive_ack_set.

Table 247. i2c_master_receive_ack_set function

Name	Description
Function name	i2c_master_receive_ack_set
Function prototype	void i2c_master_receive_ack_set(i2c_type *i2c_x, i2c_master_ack_type pos)
Function description	Master receive mode acknowledge control. In master receive mode, it is used to set the void i2c_ack_enable(i2c_type *i2c_x, confirm_state new_state) control bit, which is used when the number of bytes to receive is equal to 2 so as to ensure that the host responds to ACK in time.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	pos: ACKEN effective bit Refer to the “pos” description below for details.
Output parameter	NA
Return value	NA

Name	Description
Required preconditions	NA
Called functions	NA

pos

Set ACKEN control bit.

I2C_MASTER_ACK_CURRENT: ACKEN bit controls ACK of the current byte being transferred

I2C_MASTER_ACK_NEXT: ACKEN bit controls ACK of the next byte to be transferred

Example:

```
i2c_master_receive_ack_set(I2C1, TRUE);
```

5.12.13 i2c_pec_position_set function

The table below describes the function i2c_pec_position_set.

Table 248. i2c_pec_position_set function

Name	Description
Function name	i2c_pec_position_set
Function prototype	void i2c_pec_position_set(i2c_type *i2c_x, i2c_pec_position_type pos);
Function description	Set PEC position in SMBus mode and master receive mode. This bit is used only when the number of bytes to receive is equal to 2 so as to ensure that the host receives PEC and responds to NACK in time.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	pos: PEC position Refer to the “pos” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pos

Set ACKEN control bit.

I2C_PEC_POSITION_CURRENT: Current byte is PEC

I2C_PEC_POSITION_NEXT: The next byte is PEC

Example:

```
i2c_pec_position_set(I2C1, I2C_PEC_POSITION_CURRENT);
```

5.12.14 i2c_general_call_enable function

The table below describes the function i2c_general_call_enable.

Table 249. i2c_general_call_enable function

Name	Description
Function name	i2c_general_call_enable
Function prototype	void i2c_general_call_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable broadcast address. After enabled, broadcast address 0x00 is responded.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: Broadcast address enable state This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_general_call_enable(I2C1, TRUE);
```

5.12.15 i2c_arp_mode_enable function

The table below describes the function i2c_arp_mode_enable.

Table 250. i2c_arp_mode_enable function

Name	Description
Function name	i2c_arp_mode_enable
Function prototype	void i2c_arp_mode_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable SMBus ARP address. After it is enabled, SMBus host: response to host address 0001000x SMBus device: response to default device address 0001100x Refer to SMBUS protocol for details about ARP protocol.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2
Input parameter 2	new_state: ARP address status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_arp_mode_enable(I2C1, TRUE);
```

5.12.16 i2c_smbus_mode_set function

The table below describes the function i2c_smbus_mode_set.

Table 251. i2c_smbus_mode_set function

Name	Description
Function name	i2c_smbus_mode_set
Function prototype	void i2c_smbus_mode_set(i2c_type *i2c_x, i2c_smbus_mode_set_type mode);
Function description	Select SMBus device mode (SMBus host or SMBus device).
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	mode: SMBus device mode Refer to the “mode” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

mode

Set SMBus device mode.

I2C_SMBUS_MODE_DEVICE: SMBus device

I2C_SMBUS_MODE_HOST: SMBus host

Example:

```
i2c_smbus_mode_set(I2C1, I2C_SMBUS_MODE_HOST);
```

5.12.17 i2c_smbus_alert_set function

The table below describes the function i2c_smbus_alert_set.

Table 252. i2c_smbus_alert_set function

Name	Description
Function name	i2c_smbus_alert_set
Function prototype	void i2c_smbus_alert_set(i2c_type *i2c_x, i2c_smbus_alert_set_type level);
Function description	Set SMBus alert pin level high or low.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2
Input parameter 2	level: SMBus alert pin level Refer to the “level” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

level

Set SMBus alert pin level.

I2C_SMBUS_ALERT_LOW: Low level

I2C_SMBUS_ALERT_HIGH: High level

Example:

```
i2c_smbus_alert_set(I2C1, I2C_SMBUS_ALERT_LOW);
```

5.12.18 i2c_pec_transmit_enable function

The table below describes the function i2c_pec_transmit_enable.

Table 253. i2c_pec_transmit_enable function

Name	Description
Function name	i2c_pec_transmit_enable
Function prototype	void i2c_pec_transmit_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable PEC transmission (transmit/receive PEC). Once this function is called, PEC is transmitted or received immediately.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: enable/disable PEC transmission This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_pec_transmit_enable(I2C1, TRUE);
```

5.12.19 i2c_pec_calculate_enable function

The table below describes the function i2c_pec_calculate_enable.

Table 254. i2c_pec_calculate_enable function

Name	Description
Function name	i2c_pec_calculate_enable
Function prototype	void i2c_pec_calculate_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable PEC calculation.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: enable/disable PEC calculation This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_pec_calculate_enable(I2C1, TRUE);
```

5.12.20 i2c_pec_value_get function

The table below describes the function i2c_pec_value_get.

Table 255. i2c_pec_value_get function

Name	Description
Function name	i2c_pec_value_get
Function prototype	uint8_t i2c_pec_value_get(i2c_type *i2c_x);
Function description	Get the current PEC value
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Output parameter	uint8_t: current PEC value
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
Pec_value = i2c_pec_value_get(I2C1);
```

5.12.21 i2c_dma_end_transfer_set function

The table below describes the function i2c_dma_end_transfer_set.

Table 256. i2c_dma_end_transfer_set function

Name	Description
Function name	i2c_dma_end_transfer_set
Function prototype	void i2c_dma_end_transfer_set(i2c_type *i2c_x, confirm_state new_state);
Function description	DMA transfer end indication.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: indicates whether it is the last data This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_dma_end_transfer_set(I2C1, TRUE);
```

5.12.22 i2c_dma_enable function

The table below describes the function i2c_dma_enable.

Table 257. i2c_dma_enable function

Name	Description
Function name	i2c_dma_enable
Function prototype	void i2c_dma_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable DMA transfer
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	new_state: enable/disable DMA transfer This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_dma_enable(I2C1, TRUE);
```

5.12.23 i2c_interrupt_enable function

The table below describes the function i2c_interrupt_enable.

Table 258. i2c_interrupt_enable function

Name	Description
Function name	i2c_interrupt_enable
Function prototype	void i2c_interrupt_enable(i2c_type *i2c_x, uint16_t source, confirm_state new_state)
Function description	Enable I2C interrupt
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	source: interrupt source Refer to the “source” description below for details.
Input parameter 3	new_state: enable/disable interrupt This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

source

Select an interrupt source.

I2C_DATA_INT: Data interrupt

I2C_EV_INT: Event interrupt

I2C_ERR_INT: Error interrupt

Example:

```
i2c_interrupt_enable(I2C1, I2C_DATA_INT, TRUE);
```

5.12.24 i2c_start_generate function

The table below describes the function i2c_start_generate.

Table 259. i2c_start_generate function

Name	Description
Function name	i2c_start_generate
Function prototype	void i2c_start_generate(i2c_type *i2c_x);
Function description	Generate start condition (for the master)
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_start_generate(I2C1);
```

5.12.25 i2c_stop_generate function

The table below describes the function i2c_stop_generate.

Table 260. i2c_stop_generate function

Name	Description
Function name	i2c_stop_generate
Function prototype	void i2c_stop_generate(i2c_type *i2c_x);
Function description	Generate stop condition.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_stop_generate(I2C1);
```

5.12.26 i2c_7bit_address_send function

The table below describes the function i2c_7bit_address_send.

Table 261. i2c_7bit_address_send function

Name	Description
Function name	i2c_7bit_address_send
Function prototype	void i2c_7bit_address_send(i2c_type *i2c_x, uint8_t address, i2c_direction_type direction);
Function description	Send 7-bit slave address (for the master)
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	address: slave address
Input parameter 3	direction: data transfer direction Refer to the "direction" description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

direction

Select data transfer direction.

I2C_DIRECTION_TRANSMIT: Master transmit

I2C_DIRECTION_RECEIVE: Master receive

Example:

```
i2c_7bit_address_send(I2C1, 0xB0, I2C_DIRECTION_TRANSMIT);
```

5.12.27 i2c_data_send function

The table below describes the function i2c_data_send.

Table 262. i2c_data_send function

Name	Description
Function name	i2c_data_send
Function prototype	void i2c_data_send(i2c_type *i2c_x, uint8_t data);
Function description	Send data
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	data: data to be sent
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_data_send(I2C1, 0x55);
```

5.12.28 i2c_data_receive function

The table below describes the function i2c_data_receive.

Table 263. i2c_data_receive function

Name	Description
Function name	i2c_data_receive
Function prototype	uint8_t i2c_data_receive(i2c_type *i2c_x);
Function description	Receive data
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Output parameter	uint8_t: data to be received
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
data_value = i2c_data_receive(I2C1);
```

5.12.29 i2c_flag_get function

The table below describes the function i2c_flag_get.

Table 264. i2c_flag_get function

Name	Description
Function name	i2c_flag_get
Function prototype	flag_status i2c_flag_get(i2c_type *i2c_x, uint32_t flag);
Function description	Get flag status
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	flag: flag selection Refer to the following “flag” descriptions for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This bit is used to select a flag to get its status. Optional parameters are below:

I2C_STARTF_FLAG: Start condition generation complete flag
I2C_ADDR7F_FLAG: 0~7 bit address match flag
I2C_TDC_FLAG: Data transfer complete flag
I2C_ADDRHF_FLAG: Master 9~8 bit address header match flag
I2C_STOPF_FLAG: Stop condition generation complete flag
I2C_RDBF_FLAG: Receive data buffer full flag
I2C_TDBE_FLAG: Transmit data buffer empty flag
I2C_BUSERR_FLAG: Bus error flag
I2C_ARLOST_FLAG: Arbitration lost flag

I2C_ACKFAIL_FLAG: Acknowledge failure flag
I2C_OUF_FLAG: Overload / underload flag
I2C_PECERR_FLAG: PEC receive error flag
I2C_TMOUT_FLAG: SMBus timeout flag
I2C_ALERTF_FLAG: SMBus alert flag
I2C_TRMODE_FLAG: Transmission mode
I2C_BUSYF_FLAG: Bus busy flag
I2C_DIRF_FLAG: Transfer direction flag
I2C_GCADDRF_FLAG: General call address reception flag
I2C_DEVADDRF_FLAG: SMBus device address reception flag
I2C_HOSTADDRF_FLAG: SMBus host address reception flag
I2C_ADDR2_FLAG: Received address 2 flag

Example:

```
i2c_flag_get(I2C1, I2C_STARTF_FLAG);
```

5.12.30 i2c_interrupt_flag_get function

The table below describes the function i2c_interrupt_flag_get.

Table 265. i2c_interrupt_flag_get function

Name	Description
Function name	i2c_interrupt_flag_get
Function prototype	flag_status i2c_interrupt_flag_get(i2c_type *i2c_x, uint32_t flag);
Function description	Get flag status and judge the corresponding interrupt enable bit
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	flag: flag selection Refer to the following “flag” descriptions for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This bit is used to select a flag to get its status. Optional parameters are below:

I2C_STARTF_FLAG: Start condition generation complete flag
I2C_ADDR7F_FLAG: 0~7 bit address match flag
I2C_TDC_FLAG: Data transfer complete flag
I2C_ADDRHF_FLAG: Master 9~8 bit address header match flag
I2C_STOPF_FLAG: Stop condition generation complete flag
I2C_RDBF_FLAG: Receive data buffer full flag
I2C_TDBE_FLAG: Transmit data buffer empty flag
I2C_BUSERR_FLAG: Bus error flag
I2C_ARLOST_FLAG: Arbitration lost flag
I2C_ACKFAIL_FLAG: Acknowledge failure flag
I2C_OUF_FLAG: Overload / underload flag
I2C_PECERR_FLAG: PEC receive error flag

I2C_TMOUT_FLAG: SMBus timeout flag
I2C_ALERTF_FLAG: SMBus alert flag

Example

```
i2c_interrupt_flag_get(I2C1, I2C_STARTF_FLAG);
```

5.12.31 i2c_flag_clear function

The table below describes the function i2c_flag_clear.

Table 266. i2c_flag_clear function

Name	Description
Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(i2c_type *i2c_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1 or I2C2.
Input parameter 2	flag: flag to be cleared Refer to the “flag” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

This bit is used to select a flag, including

I2C_BUSERR_FLAG: Bus error flag
I2C_ARLOST_FLAG: Arbitration lost flag
I2C_ACKFAIL_FLAG: Acknowledge failure flag
I2C_OUF_FLAG: Overload / underload flag
I2C_PECERR_FLAG: PEC receive error flag
I2C_TMOUT_FLAG: SMBus timeout flag
I2C_ALERTF_FLAG: SMBus alert flag
I2C_ADDR7F_FLAG: 0~7 bit address match flag
I2C_STOPF_FLAG: Stop condition generation complete flag

Example:

```
i2c_flag_clear(I2C1, I2C_ACKFAIL_FLAG);
```

5.12.32 i2c_config function

The table below describes the function i2c_config.

Table 267. i2c_config function

Name	Description
Function name	i2c_config
Function prototype	void i2c_config(i2c_handle_type* hi2c);
Function description	I2C initialization function used to initialize I2C. Call the function i2c_lowlevel_init() to initialize I2C peripherals, GPIO, DMA, interrupts and others.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type.*
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void i2c_lowlevel_init(i2c_handle_type* hi2c);

i2c_handle_type* hi2c

The i2c_handle_type is defined in the i2c_application.h.

```
typedef struct
{
    i2c_type                *i2cx;
    uint8_t                 *pbuff;
    __IO uint16_t           pcount;
    __IO uint32_t           mode;
    __IO uint32_t           timeout;
    __IO uint32_t           status;
    __IO i2c_status_type    error_code;
    dma_channel_type        *dma_tx_channel;
    dma_channel_type        *dma_rx_channel;
    dma_init_type           dma_init_struct;
}i2c_handle_type;
```

i2cx

Select an I2C peripheral from I2C1 or I2C2.

pbuff

An array of data to be sent or received

pcount

The number of data to be sent or received

mode

I2C communication mode. It is used in internal state machine. Users don’t care.

timeout

Communications timeout

status

Transfer status. It is used in internal state machine. Users don’t care.

error_code

This bit is used to enumerate error code in the i2c_status_type. When a communication error occurs, it logs the corresponding error code.

I2C_OK:	Communication OK
I2C_ERR_STEP_1:	Step 1 error
I2C_ERR_STEP_2:	Step 2 error
I2C_ERR_STEP_3:	Step 3 error
I2C_ERR_STEP_4:	Step 4 error
I2C_ERR_STEP_5:	Step 5 error
I2C_ERR_STEP_6:	Step 6 error
I2C_ERR_STEP_7:	Step 7 error
I2C_ERR_STEP_8:	Step 8 error
I2C_ERR_STEP_9:	Step 9 error
I2C_ERR_STEP_10:	Step 10 error
I2C_ERR_STEP_11:	Step 11 error
I2C_ERR_STEP_12:	Step 12 error
I2C_ERR_START:	START condition send error
I2C_ERR_ADDR10:	10bit address header (bit9~8) send error
I2C_ERR_ADDR:	Address send error
I2C_ERR_STOP:	STOP condition send error
I2C_ERR_ACKFAIL:	Acknowledge error
I2C_ERR_TIMEOUT:	Timeout error
I2C_ERR_INTERRUPT:	Enter an interrupt when an error event occurred

dma_tx_channel

I2C transmit DMA channel

dma_rx_channel

I2C receive DMA channel

dma_init_struct

DMA initialization structure

Example:

```
i2c_handle_type hi2c;
hi2c.i2cx = I2C1;
i2c_config(&hi2c);
```

5.12.33 i2c_lowlevel_init function

The table below describes the function i2c_lowlevel_init.

Table 268. i2c_lowlevel_init function

Name	Description
Function name	i2c_lowlevel_init
Function prototype	void i2c_lowlevel_init(i2c_handle_type* hi2c);
Function description	I2C lower-level initialization callback function. It is called in the i2c_config to initialize I2C peripherals, GPIO, DMA, interrupts and others. It requires users to implement I2C initialization inside the function.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type.*
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void i2c_lowlevel_init(i2c_handle_type* hi2c)
{
  if(hi2c->i2cx == I2C1)
  {
    Implement I2C1 initialization
  }
  else if(hi2c->i2cx == I2C2)
  {
    Implement I2C2 initialization
  }
}
```

5.12.34 i2c_wait_end function

The table below describes the function i2c_wait_end.

Table 269. i2c_wait_end function

Name	Description
Function name	i2c_wait_end
Function prototype	i2c_status_type i2c_wait_end(i2c_handle_type* hi2c, uint32_t timeout);
Function description	Wait for the end of communication. This function is used in DMA and interrupt transfer modes as they are non-blocking functions and can thus be used to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
if (i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF) != I2C_OK)
{
   error_handler(i2c_status);
}

/* wait for the end of communication */
if(i2c_wait_end(&hi2c, 0xFFFFFFFF) != I2C_OK)
{
   error_handler(i2c_status);
}
```

5.12.35 i2c_wait_flag function

The table below describes the function i2c_wait_flag.

Table 270. i2c_wait_flag function

Name	Description
Function name	i2c_wait_flag
Function prototype	i2c_status_type i2c_wait_flag(i2c_handle_type* hi2c, uint32_t flag, uint32_t event_check, uint32_t timeout)
Function description	Wait for a flag to be set or reset Only BUSFY flag is "wait for a flag to be reset", and others are "wait for a flag to be set"
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 3	event_check: check if the event has occurred or not while waiting for a flag Refer to the "event_check" descriptions below for details.
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

flag

Select a flag to wait for.

I2C_STARTF_FLAG:	Start condition generation complete flag
I2C_ADDR7F_FLAG:	0~7 bit address match flag
I2C_TDC_FLAG:	Data transfer complete flag
I2C_ADDRHF_FLAG:	Master 9~8 bit address head match flag
I2C_STOPF_FLAG:	Stop condition generation complete flag
I2C_RDBF_FLAG:	Receive data buffer full flag
I2C_TDBE_FLAG:	Transmit data buffer empty flag
I2C_BUSERR_FLAG:	Bus error flag
I2C_ARLOST_FLAG:	Arbitration lost flag
I2C_ACKFAIL_FLAG:	Acknowledge failure flag
I2C_OUF_FLAG:	Overload / underload flag
I2C_PECERR_FLAG:	PEC receive error flag
I2C_TMOUT_FLAG:	SMBus timeout flag
I2C_ALERTF_FLAG:	SMBus alert flag
I2C_TRMODE_FLAG:	Transmission mode
I2C_BUSYF_FLAG:	Bus busy flag
I2C_DIRF_FLAG:	Transfer direction flag
I2C_GCADDRF_FLAG:	General call address reception flag
I2C_DEVADDRF_FLAG:	SMBus device address reception flag
I2C_HOSTADDRF_FLAG:	SMBus host address reception flag
I2C_ADDR2_FLAG:	Received address 2 flag

event_check

Check if the event has occurred or not while waiting for a flag.

I2C_EVENT_CHECK_NONE: None

I2C_EVENT_CHECK_ACKFAIL: Check ACKFAIL event

I2C_EVENT_CHECK_STOP: Check STOP event

Example:

```
i2c_wait_flag(&hi2c, I2C_BUSYF_FLAG, I2C_EVENT_CHECK_NONE, 0xFFFFFFFF);
```

5.12.36 i2c_master_transmit function

The table below describes the function i2c_master_transmit.

Table 271. i2c_master_transmit function

Name	Description
Function name	i2c_master_transmit
Function prototype	i2c_status_type i2c_master_transmit(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master sends data (polling mode). This is a blocking function, and so I2C transfer ends after the function is executed.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address of to-be-sent data
Input parameter 4	size: the size of data to be sent
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.12.37 i2c_master_receive function

The table below describes the function i2c_master_receive.

Table 272. i2c_master_receive function

Name	Description
Function name	i2c_master_receive
Function prototype	i2c_status_type i2c_master_receive(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (polling mode). This function is a blocking type. After the execution is done, so does I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address to receive data
Input parameter 4	size: number of data to receive
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.12.38 i2c_slave_transmit function

The table below describes the function i2c_slave_transmit.

Table 273. i2c_slave_transmit function

Name	Description
Function name	i2c_slave_transmit
Function prototype	i2c_status_type i2c_slave_transmit(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (polling mode). This function is a blocking type. In other words, after the function execution is done, so does I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	pdata: array address of data to be sent
Input parameter 3	size: number of data to be sent
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.12.39 i2c_slave_receive function

The table below describes the function i2c_slave_receive.

Table 274. i2c_slave_receive function

Name	Description
Function name	i2c_slave_receive
Function prototype	i2c_status_type i2c_slave_receive(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (polling mode). This function is a blocking type. In other words, after the function execution is done, so does I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	pdata: array address to receive data
Input parameter 3	size: number of data to be received
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.12.40 i2c_master_transmit_int function

The table below describes the function i2c_master_transmit_int.

Table 275. i2c_master_transmit_int function

Name	Description
Function name	i2c_master_transmit_int
Function prototype	i2c_status_type i2c_master_transmit_int(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master sends data (interrupt mode). This function is a non-blocking type. In other words, after the function execution is done, I2C transfer has not completed yet. In this case, it is possible to call the i2c_wait_end() to wait for the completion of communication.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address of data to be sent
Input parameter 4	size: number of data to be sent
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit_int(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.12.41 i2c_master_receive_int function

The table below describes the function i2c_master_receive_int.

Table 276. i2c_master_receive_int function

Name	Description
Function name	i2c_master_receive_int
Function prototype	i2c_status_type i2c_master_receive_int(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address to receive data
Input parameter 4	size: number of data to be received
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive_int(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.12.42 i2c_slave_transmit_int function

The table below describes the function i2c_slave_transmit_int.

Table 277. i2c_slave_transmit_int function

Name	Description
Function name	i2c_slave_transmit_int
Function prototype	i2c_status_type i2c_slave_transmit_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (through interrupt mode). This function operates in non-blocking mode. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	pdata: array address of data to be sent
Input parameter 3	size: number of data to be sent
Input parameter 4	timeout: wait timeout
Output parameter	NA

Name	Description
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit_int(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.12.43 i2c_slave_receive_int function

The table below describes the function i2c_slave_receive_int.

Table 278. i2c_slave_receive_int function

Name	Description
Function name	i2c_slave_receive_int
Function prototype	i2c_status_type i2c_slave_receive_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() for the completion of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	pdata: array address to receive data
Input parameter 3	size: number of data to be received
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive_int(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.12.44 i2c_master_transmit_dma function

The table below describes the function i2c_master_transmit_dma.

Table 279. i2c_master_transmit_dma function

Name	Description
Function name	i2c_master_transmit_dma
Function prototype	i2c_status_type i2c_master_transmit_dma(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master sends data (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address of data to be sent
Input parameter 4	size: number of data to be sent
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.12.45 i2c_master_receive_dma function

The table below describes the function i2c_master_receive_dma.

Table 280. i2c_master_receive_dma function

Name	Description
Function name	i2c_master_receive_dma
Function prototype	i2c_status_type i2c_master_receive_dma(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address to receive data
Input parameter 4	size: number of data to be received
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive_dma(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.12.46 i2c_slave_transmit_dma function

The table below describes the function i2c_slave_transmit_dma.

Table 281. i2c_slave_transmit_dma function

Name	Description
Function name	i2c_slave_transmit_dma
Function prototype	i2c_status_type i2c_slave_transmit_dma(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	pdata: array address of data to be sent
Input parameter 3	size: number of data to be sent
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit_dma(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.12.47 i2c_slave_receive_dma function

The table below describes the function i2c_slave_receive_dma.

Table 282. i2c_slave_receive_dma function

Name	Description
Function name	i2c_slave_receive_dma
Function prototype	i2c_status_type i2c_slave_receive_dma(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to *i2c_handle_type* for details.
Input parameter 2	pdata: array address to receive data
Input parameter 3	size: number of data to be received
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive_dma(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.12.48 i2c_memory_write function

The table below describes the function i2c_memory_write.

Table 283. i2c_memory_write function

Name	Description
Function name	i2c_memory_write
Function prototype	i2c_status_type i2c_memory_write(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write data to EEPROM (through polling mode). This function is a blocking type. In other words, after the function execution is done, so does I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be sent
Input parameter 6	size: number of data to be sent
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

mem_address_width

EEPROM memory address width

I2C_MEM_ADDR_WIDIH_8: 8-bit address width

I2C_MEM_ADDR_WIDIH_16: 16-bit address width

Example:

```
i2c_memory_write(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

5.12.49 i2c_memory_write_int function

The table below describes the function i2c_memory_write_int.

Table 284. i2c_memory_write_int function

Name	Description
Function name	i2c_memory_write_int
Function prototype	i2c_status_type i2c_memory_write_int(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write EEPROM (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be sent
Input parameter 6	size: number of data to be sent
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

mem_address_width

EEPROM memory address width

I2C_MEM_ADDR_WIDIH_8: 8-bit address bit

I2C_MEM_ADDR_WIDIH_16: 16-bit address bit

Example:

```
i2c_memory_write_int(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

5.12.50 i2c_memory_write_dma function

The table below describes the function i2c_memory_write_dma.

Table 285. i2c_memory_write_dma function

Name	Description
Function name	i2c_memory_write_dma
Function prototype	i2c_status_type i2c_memory_write_dma(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write EEPROM (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be sent
Input parameter 6	size: number of data to be sent
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

mem_address_width

EEPROM memory address width

I2C_MEM_ADDR_WIDIH_8: 8-bit address width

I2C_MEM_ADDR_WIDIH_16: 16-bit address width

Example:

```
i2c_memory_write_dma(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

5.12.51 i2c_memory_read function

The table below describes the function i2c_memory_read

Table 286. i2c_memory_read function

Name	Description
Function name	i2c_memory_read
Function prototype	i2c_status_type i2c_memory_read(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (through polling mode). This function is a blocking type. In other words, after the function execution is done, so does I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM date memory address
Input parameter 5	pdata: array address of data to be read
Input parameter 6	size: number of data to be read
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

mem_address_width

EEPROM memory address width

I2C_MEM_ADDR_WIDIH_8: 8-bit address width

I2C_MEM_ADDR_WIDIH_16: 16-bit address width

Example:

```
i2c_memory_read(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

5.12.52 i2c_memory_read_int function

The table below describes the function i2c_memory_read_int.

Table 287. i2c_memory_read_int function

Name	Description
Function name	i2c_memory_read_int
Function prototype	i2c_status_type i2c_memory_read_int(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be read
Input parameter 6	size: number of data to be read
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

mem_address_width

EEPROM memory address width

I2C_MEM_ADDR_WIDIH_8: 8-bit address width

I2C_MEM_ADDR_WIDIH_16: 16-bit address width

Example:

```
i2c_memory_read_int(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

5.12.53 i2c_memory_read_dma function

The table below describes the function i2c_memory_read_dma.

Table 288. i2c_memory_read_dma function

Name	Description
Function name	i2c_memory_read_dma
Function prototype	i2c_status_type i2c_memory_read_dma(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be read
Input parameter 6	size: number of data to be read
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.12.32 for details.
Required preconditions	NA
Called functions	NA

mem_address_width

EEPROM memory address width

I2C_MEM_ADDR_WIDIH_8: 8-bit address width

I2C_MEM_ADDR_WIDIH_16: 16-bit address width

Example:

```
i2c_memory_read_dma(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

5.12.54 i2c_evt_irq_handler function

The table below describes the function i2c_evt_irq_handler.

Table 289. i2c_evt_irq_handler function

Name	Description
Function name	i2c_evt_irq_handler
Function prototype	void i2c_evt_irq_handler(i2c_handle_type* hi2c);
Function description	Event interrupt function. It is used to handle I2C event interrupt.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void I2C1_EVT_IRQHandler(void)
{
    i2c_evt_irq_handler(&hi2c);
}
```

5.12.55 i2c_err_irq_handler function

The table below describes the function i2c_err_irq_handler.

Table 290. i2c_err_irq_handler function

Name	Description
Function name	i2c_err_irq_handler
Function prototype	void i2c_err_irq_handler(i2c_handle_type* hi2c);
Function description	Error interrupt function. It is used to handle I2C error interrupt.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void I2C1_ERR_IRQHandler(void)
{
    i2c_err_irq_handler(&hi2c);
}
```

5.12.56 i2c_dma_tx_irq_handler function

The table below describes the function i2c_dma_tx_irq_handler.

Table 291. i2c_dma_tx_irq_handler function

Name	Description
Function name	i2c_dma_tx_irq_handler
Function prototype	void i2c_dma_tx_irq_handler(i2c_handle_type* hi2c);
Function description	DMA transmit interrupt function. It is used to handle DMA transmit interrupt.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void DMA1_Channel6_IRQHandler(void)
{
    i2c_dma_rx_irq_handler(&hi2c);
}
```

5.12.57 i2c_dma_rx_irq_handler function

The table below describes the function i2c_dma_rx_irq_handler.

Table 292. i2c_dma_rx_irq_handler function

Name	Description
Function name	i2c_dma_rx_irq_handler
Function prototype	void i2c_dma_rx_irq_handler(i2c_handle_type* hi2c);
Function description	DMA receive interrupt function. It is used to handle DMA receive interrupt.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void DMA1_Channel7_IRQHandler(void)
{
    i2c_dma_tx_irq_handler(&hi2c);
}
```

5.13 Nested vectored interrupt controller (NVIC)

The NVIC register structure NVIC_Type is defined in the “core_cm4.h”.

```
/**
  * @brief Structure type to access the Nested Vectored Interrupt Controller (NVIC).
  */
typedef struct
{
      ……
} NVIC_Type;
```

The table below gives a list of the NVIC registers.

Table 293. Summary of PWC registers

Register	Description
iser	Interrupt enable set register
icer	Interrupt enable clear register
ispr	Interrupt suspend set register
icpr	Interrupt suspend clear register
iabr	Interrupt activate bit register
ip	Interrupt priority register
stir	Software trigger interrupt register

The table below gives a list of the NVIC library functions.

Table 294. Summary of PWC library functions

Function name	Description
nvic_system_reset	System software reset
nvic_irq_enable	NVIC interrupt enable and priority enable
nvic_irq_disable	NVIC interrupt disable
nvic_priority_group_config	NVIC interrupt priority grouping configuration
nvic_vector_table_set	NVIC interrupt vector table base address and offset address configuration
nvic_lowpower_mode_config	NVIC low-power mode configuration

5.13.1 nvic_system_reset function

The table below describes the function nvic_system_reset.

Table 295. nvic_system_reset function

Name	Description
Function name	nvic_system_reset
Function prototype	void nvic_system_reset(void)
Function description	System software reset
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SystemReset()

Example:

```
/* system reset */
nvic_system_reset();
```

5.13.2 nvic_irq_enable function

The table below describes the function nvic_irq_enable.

Table 296. nvic_irq_enable function

Name	Description
Function name	nvic_irq_enable
Function prototype	void nvic_irq_enable(IRQn_Type irqn, uint32_t preempt_priority, uint32_t sub_priority)
Function description	NVIC interrupt enable and priority configuration
Input parameter 1	irqn: interrupt vector selection Refer to *irqn* for details.
Input parameter 2	preempt_priority: set preemption priority This parameter cannot be greater than the highest preemption priority defined in the NVIC_PRIORITY_GROUP_x.
Input parameter 3	sub_priority: set response priority This parameter cannot be greater than the highest response priority defined in the NVIC_PRIORITY_GROUP_x.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SetPriority() NVIC_EnableIRQ()

irqn

The irqn is used to select interrupt vectors, including

WWDT_IRQn: Window timer interrupt

PVM_IRQn: PVM interrupt linked to EXINT

……

USBFS_MAPL_IRQn: USBFS remap low priority interrupt

DMA2_Channel6_7_IRQn: DMA2 channel 6 and DMA2 channel 7 global interrupts

Example:

```
/* enable nvic irq */
nvic_irq_enable(ADC1_2_IRQn, 0, 0);
```

5.13.3 nvic_irq_disable function

The table below describes the function nvic_irq_disable.

Table 297. nvic_irq_disable function

Name	Description
Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(IRQn_Type irqn)
Function description	NVIC interrupt enable
Input parameter	irqn: select interrupt vector. Refer to *irqn* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_DisableIRQ()

Example:

```
/* disable nvic irq */
nvic_irq_disable(ADC1_2_IRQn);
```

5.13.4 nvic_priority_group_config function

The table below describes the function nvic_priority_group_config.

Table 298. nvic_priority_group_config function

Name	Description
Function name	nvic_priority_group_config
Function prototype	void nvic_priority_group_config(nvic_priority_group_type priority_group)
Function description	NVIC interrupt priority grouping configuration
Input parameter	priority_group: select interrupt priority group This parameter can be any enumerated value in the nvic_priority_group_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SetPriorityGrouping()

priority_group

The priority_group is used to select priority group from the parameters below:

NVIC_PRIORITY_GROUP_0:

Priority group 0 (0 bit for preemption priority, and 4 bits for response priority)

NVIC_PRIORITY_GROUP_1:

Priority group 1 (1 bit for preemption priority, and 3 bits for response priority)

NVIC_PRIORITY_GROUP_2:

Priority group 2 (2 bits for preemption priority, and 2 bits for response priority)

NVIC_PRIORITY_GROUP_3:

Priority group 3 (3 bits for preemption priority, and 1 bit for response priority)

NVIC_PRIORITY_GROUP_4:

Priority group 4 (4 bits for preemption priority, and 0 bit for response priority)

Example:

```
/* config nvic priority group */
```

```
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
```

5.13.5 nvic_vector_table_set function

The table below describes the function nvic_vector_table_set.

Table 299. nvic_vector_table_set function

Name	Description
Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t base, uint32_t offset)
Function description	Set NVIC interrupt vector table base address and offset address
Input parameter 1	base: base address of interrupt vector table The base address can be set in RAM or FLASH.
Input parameter 2	offset: offset address of interrupt vector table This parameter defines the start address of interrupt vector table, so it must be set to a multiple of 0x200.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

base

The base is used to select the base address of interrupt vector table, including:

NVIC_VECTTAB_RAM: Interrupt vector table base address is located in RAM

NVIC_VECTTAB_FLASH: Interrupt vector table base address is located in FLASH

Example:

```
/* config vector table offset */
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x4000);
```

5.13.6 nvic_lowpower_mode_config function

The table below describes the function nvic_lowpower_mode_config.

Table 300. nvic_lowpower_mode_config function

Name	Description
Function name	nvic_lowpower_mode_config
Function prototype	void nvic_lowpower_mode_config(nvic_lowpower_mode_type lp_mode, confirm_state new_state)
Function description	Configure NVIC low-power mode
Input parameter 1	lp_mode: select low-power modes This parameter can be any enumerated value in the nvic_lowpower_mode_type.
Input parameter 2	new_state: indicates the pre-configured status of battery powered domain This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

lp_mode

The lp_mode is used to select low-power modes, including:

NVIC_LP_SEVONPEND:

Send wakeup event upon interrupt suspend (this option is usually used in conjunction with WFE)

NVIC_LP_SLEEPDEEP:

Deepsleep mode control bit (enable or disable core clock)

NVIC_LP_SLEEPONEXIT:

Sleep mode entry when system leaves the lowest-priority interrupt

Example:

```
/* enable sleep-on-exit feature */
nvic_lowpower_mode_config(NVIC_LP_SLEEPONEXIT, TRUE);
```

5.14 Power controller (PWC)

The PWC register structure pwc_type is defined in the “at32f413_pwc.h”.

```
/**
  * @brief type define pwc register all
  */
typedef struct
{
    ……
} pwc_type;
```

The table below gives a list of the PWC registers.

Table 301. Summary of PWC registers

Register	Description
ctrl	Power control register
ctrlsts	Power control/status register

The table below gives a list of the PWC library functions.

Table 302. Summary of PWC library functions

Function name	Description
pwc_reset	Reset PWC registers to their reset values
pwc_battery_powered_domain_access	Enable battery powered domain access
pwc_pvm_level_select	Select PVM threshold
pwc_power_voltage_monitor_enable	Enable voltage monitor
pwc_wakeup_pin_enable	Enable standby-mode wakeup pin
pwc_flag_clear	Clear flag
pwc_flag_get	Get flag
pwc_sleep_mode_enter	Enter Sleep mode
pwc_deep_sleep_mode_enter	Enter Deepsleep mode
pwc_standby_mode_enter	Enter Standby mode

5.14.1 pwc_reset function

The table below describes the function pwc_reset.

Table 303. pwc_reset function

Name	Description
Function name	pwc_reset
Function prototype	void pwc_reset(void)
Function description	Reset all PWC registers to their reset values.
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset()

Example:

```
/* deinitialize pwc */
pwc_reset();
```

5.14.2 pwc_battery_powered_domain_access function

The table below describes the function pwc_battery_powered_domain_access.

Table 304. pwc_battery_powered_domain_access function

Name	Description
Function name	pwc_battery_powered_domain_access
Function prototype	void pwc_battery_powered_domain_access(confirm_state new_state)
Function description	Battery powered domain access enable
Input parameter	new_state: indicates the pre-configured status of battery powered domain This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the battery-powered domain write operations */
pwc_battery_powered_domain_access(TRUE);
```

Note: Access to battery powered domain (such as, RTC) is allowed only after enabling it through this function.

5.14.3 pwc_pvm_level_select function

The table below describes the function pwc_pvm_level_select.

Table 305. pwc_pvm_level_select function

Name	Description
Function name	pwc_pvm_level_select
Function prototype	void pwc_pvm_level_select(pwc_pvm_voltage_type pvm_voltage)
Function description	Select PVM threshold
Input parameter	pvm_voltage: indicates the selected PVM threshold This parameter can be any enumerated value in the pwc_pvm_voltage_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pvm_voltage

The pvm_voltage is used to select a PVM threshold from the optional parameters below:

PWC_PVM_VOLTAGE_2V3: PVM threshold is 2.3V

PWC_PVM_VOLTAGE_2V4: PVM threshold is 2.4V

PWC_PVM_VOLTAGE_2V5: PVM threshold is 2.5V

PWC_PVM_VOLTAGE_2V6: PVM threshold is 2.6V

PWC_PVM_VOLTAGE_2V7: PVM threshold is 2.7V

PWC_PVM_VOLTAGE_2V8: PVM threshold is 2.8V

PWC_PVM_VOLTAGE_2V9: PVM threshold is 2.9V

Example:

```
/* set the threshold voltage to 2.9v */
pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V9);
```

5.14.4 pwc_power_voltage_monitor_enable function

The table below describes the function pwc_power_voltage_monitor_enable.

Table 306. pwc_power_voltage_monitor_enable function

Name	Description
Function name	pwc_power_voltage_monitor_enable
Function prototype	void pwc_power_voltage_monitor_enable(confirm_state new_state)
Function description	Enable power voltage monitor (PVM)
Input parameter	new_state: indicates the pre-configured status of PVM This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable power voltage monitor */
pwc_power_voltage_monitor_enable(TRUE);
```

5.14.5 pwc_wakeup_pin_enable function

The table below describes the function pwc_wakeup_pin_enable.

Table 307. pwc_wakeup_pin_enable function

Name	Description
Function name	pwc_wakeup_pin_enable
Function prototype	void pwc_wakeup_pin_enable(uint32_t pin_num, confirm_state new_state)
Function description	Enable Standby wakeup pin
Input parameter 1	pin_num: select a standby wakeup pin This parameter can be any pin that is capable of waking up from Standby mode.
Input parameter 2	new_state: indicates the pre-configured status of Standby wakeup pins This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pin_num

The pin_num is used to select Standby-mode wakeup pin, including:

PWC_WAKEUP_PIN_1: Standby wakeup pin 1 (corresponding GPIO is PA0)

Example:

```
/* enable wakeup pin - pa0 */
```

```
pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);
```

5.14.6 pwc_flag_clear function

The table below describes the function pwc_flag_clear.

Table 308. pwc_flag_clear function

Name	Description
Function name	pwc_flag_clear
Function prototype	void pwc_flag_clear(uint32_t pwc_flag)
Function description	Clear flag
Input parameter	pwc_flag: to-be-cleared flag Refer to *pwc_flag* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_flag

The pwc_flag is used to select a flag from the optional parameters below:

PWC_WAKEUP_FLAG: Standby wakeup event

PWC_STANDBY_FLAG: Standby mode entry

PWC_PVM_OUTPUT_FLAG: PVM output (this parameter cannot be cleared by software)

Example:

```
/* wakeup event flag clear */
pwc_flag_clear(PWC_WAKEUP_FLAG);
```

5.14.7 pwc_flag_get function

The table below describes the function pwc_flag_get.

Table 309. pwc_flag_get function

Name	Description
Function name	pwc_flag_get
Function prototype	flag_status pwc_flag_get(uint32_t pwc_flag)
Function description	Get flag status
Input parameter	pwc_flag: select a flag. Refer to *pwc_flag* for details.
Output parameter	NA
Return value	flag_status: indicates flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* check if wakeup event flag is set */
if(pwc_flag_get(PWC_WAKEUP_FLAG) != RESET)
```

5.14.8 pwc_sleep_mode_enter function

The table below describes the function pwc_sleep_mode_enter.

Table 310. pwc_sleep_mode_enter function

Name	Description
Function name	pwc_sleep_mode_enter
Function prototype	void pwc_sleep_mode_enter(pwc_sleep_enter_type pwc_sleep_enter)
Function description	Enter Sleep mode
Input parameter	pwc_sleep_enter: select a command to enter Sleep mode This parameter can be any enumerated value in the pwc_sleep_enter_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_sleep_enter

The pwc_sleep_enter is used to select a command to enter Sleep mode from the optional parameters below:

PWC_SLEEP_ENTER_WFI: Enter Sleep mode by WFI

PWC_SLEEP_ENTER_WFE: Enter Sleep mode by WFE

Example:

```
/* enter sleep mode */
pwc_sleep_mode_enter(PWC_SLEEP_ENTER_WFI);
```

5.14.9 pwc_deep_sleep_mode_enter function

The table below describes the function pwc_deep_sleep_mode_enter.

Table 311. pwc_deep_sleep_mode_enter function

Name	Description
Function name	pwc_deep_sleep_mode_enter
Function prototype	void pwc_deep_sleep_mode_enter(pwc_deep_sleep_enter_type pwc_deep_sleep_enter)
Function description	Enter Deepsleep mode
Input parameter	pwc_deep_sleep_enter: select a command to enter Deepsleep mode This parameter can be any enumerated value in the pwc_deep_sleep_enter_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_deep_sleep_enter

The pwc_deep_sleep_enter is used to select a command to enter Deepsleep mode, including:

PWC_DEEP_SLEEP_ENTER_WFI: Enter Deepsleep mode by WFI

PWC_DEEP_SLEEP_ENTER_WFE: Enter Deepsleep mode by WFE

Example:

```
/* enter deep sleep mode */
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);
```

5.14.10 pwc_standby_mode_enter function

The table below describes the function pwc_standby_mode_enter.

Table 312. pwc_standby_mode_enter function

Name	Description
Function name	pwc_standby_mode_enter
Function prototype	void pwc_standby_mode_enter(void)
Function description	Enter Standby mode
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enter standby mode */
pwc_standby_mode_enter();
```

5.15 Real-time clock (RTC)

The RTC register structure rtc_type is defined in the “at32f413_rtc.h”.

```
/**
  * @brief type define rtc register all
  */
typedef struct
{


} rtc_type;
```

The table below gives a list of the RTC registers.

Table 313. Summary of RTC registers

Register	Description
ctrlh	RTC control register high
ctrll	RTC control register low
divh	RTC divider register high
divl	RTC divider register low
divcnth	RTC divider counter register high
divcntl	RTC divider counter register low
cnth	RTC counter value register high
cntl	RTC counter value register low
tah	RTC alarm register high
tal	RTC alarm register low

The table below gives a list of the RTC library functions.

Table 314. Summary of RTC library functions

Function name	Description
rtc_counter_set	Set RTC counter value
rtc_counter_get	Get RTC counter value
rtc_divider_set	Set RTC divider
rtc_divider_get	Get RTC division value
rtc_alarm_set	Set RTC clock
rtc_interrupt_enable	Enable RTC interrupt
rtc_flag_get	Get RTC flag
rtc_flag_clear	Clear RTC flag
rtc_wait_config_finish	Wait for RTC configuration complete
rtc_wait_update_finish	Wait for RTC update complete

5.15.1 rtc_counter_set function

The table below describes the function rtc_counter_set.

Table 315. rtc_counter_set function

Name	Description
Function name	rtc_counter_set
Function prototype	void rtc_counter_set(uint32_t counter_value);
Function description	Set RTC counter value
Input parameter 1	counter_value: RTC counter value, range: 0~0xFFFFFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
rtc_counter_set(0x00000008);
```

5.15.2 rtc_counter_get function

The table below describes the function rtc_counter_get.

Table 316. rtc_counter_get function

Name	Description
Function name	rtc_counter_get
Function prototype	uint32_t rtc_counter_get(void);
Function description	Get RTC counter value
Input parameter 1	NA
Output parameter	NA
Return value	uint32_t: current counter value, generally incremented by 1
Required preconditions	NA
Called functions	NA

Example:

```
value = rtc_counter_get();
```

5.15.3 rtc_divider_set function

The table below describes the function rtc_divider_set.

Table 317. rtc_divider_set function

Name	Description
Function name	rtc_divider_set
Function prototype	void rtc_divider_set(uint32_t div_value);
Function description	Set RTC divider
Input parameter 1	div_value: RTC division value, range: 0~0x000FFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
rtc_divider_set(32767);
```

5.15.4 rtc_divider_get function

The table below describes the function rtc_divider_get.

Table 318. rtc_divider_get function

Name	Description
Function name	rtc_divider_get
Function prototype	uint32_t rtc_divider_get(void);
Function description	Get RTC division value
Input parameter 1	NA
Output parameter	NA
Return value	uint32_t: current division value
Required preconditions	NA
Called functions	NA

Example:

```
value = rtc_divider_get();
```

5.15.5 rtc_alarm_set function

The table below describes the function rtc_alarm_set.

Table 319. rtc_alarm_set function

Name	Description
Function name	rtc_alarm_set
Function prototype	void rtc_alarm_set(uint32_t alarm_value);
Function description	Set RTC alarm
Input parameter 1	alarm_value: RTC alarm, range: 0~0xFFFFFFFF The alarm event occurs when the current RTC counter value reaches the RTC alarm value.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
rtc_alarm_set(0x00000006);
```

5.15.6 rtc_interrupt_enable function

The table below describes the function rtc_interrupt_enable.

Table 320. rtc_interrupt_enable function

Name	Description
Function name	rtc_interrupt_enable
Function prototype	void rtc_interrupt_enable(uint16_t source, confirm_state new_state);
Function description	Enable interrupt
Input parameter 1	source: interrupt source Refer to the “source” below for details.
Input parameter 2	new_state: enable/disable interrupt This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

source

It is used to select an interrupt source.

RTC_TS_INT: Second clock interrupt

RTC_TA_INT: Alarm interrupt

RTC_OVF_INT: Counter overflow interrupt

Example:

```
rtc_interrupt_enable(RTC_TS_INT, TRUE);
```

5.15.7 rtc_flag_get function

The table below describes the function rtc_flag_get.

Table 321. rtc_flag_get function

Name	Description
Function name	rtc_flag_get
Function prototype	flag_status rtc_flag_get(uint16_t flag);
Function description	Get flag status
Input parameter 1	flag: flag selection Refer to the “flag” description below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

It is used to select a flag, including:

RTC_TS_FLAG: Second clock flag

RTC_TA_FLAG: Alarm flag

RTC_OVF_FLAG: Counter value overflow flag

RTC_UPDF_FLAG: Time update flag

RTC_CFGF_FLAG: RTC register configuration complete flag

Example:

```
rtc_flag_get(RTC_TS_FLAG);
```

5.15.8 rtc_interrupt_flag_get function

The table below describes the function rtc_interrupt_flag_get.

Table 322. rtc_interrupt_flag_get function

Name	Description
Function name	rtc_interrupt_flag_get
Function prototype	flag_status rtc_interrupt_flag_get(uint16_t flag);
Function description	Get flag status and judge the corresponding interrupt enable bit
Input parameter 1	flag flag selection Refer to the “flag” description below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

It is used to select a flag, including:

RTC_TS_FLAG: Second clock flag

RTC_TA_FLAG: Alarm flag

RTC_OVF_FLAG: Counter value overflow flag

Example

```
rtc_interrupt_flag_get(RTC_TS_FLAG);
```

5.15.9 rtc_flag_clear function

The table below describes the function rtc_flag_clear.

Table 323. rtc_flag_clear function

Name	Description
Function name	rtc_flag_clear
Function prototype	void rtc_flag_clear(uint16_t flag);
Function description	Clear flag
Input parameter 1	flag: to-be-cleared flag Refer to the “flag” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

It is used to select a flag, including:

RTC_TS_FLAG: Second clock flag

RTC_TA_FLAG: Alarm flag

RTC_OVF_FLAG: Counter value overflow flag

RTC_UPDF_FLAG: Time update flag

Example:

```
rtc_flag_clear(RTC_TS_FLAG);
```

5.15.10 rtc_wait_config_finish function

The table below describes the function rtc_wait_config_finish.

Table 324. rtc_wait_config_finish function

Name	Description
Function name	rtc_wait_config_finish
Function prototype	void rtc_wait_config_finish(void);
Function description	Wait for RTC configuration complete
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
rtc_wait_config_finish();
```

5.15.11 rtc_wait_update_finish function

The table below describes the function rtc_wait_update_finish.

Table 325. rtc_wait_update_finish function

Name	Description
Function name	rtc_wait_update_finish
Function prototype	void rtc_wait_update_finish(void);
Function description	Wait for RTC update complete
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
rtc_wait_update_finish();
```

5.16 SDIO interface (SDIO)

The SDIO register structure crm_type is defined in the “at32f413_sdio.h”.

```
/**
  * @brief type define sdio register all
  */
typedef struct
{
    ...

} sdio_type;
```

The table below gives a list of the SDIO registers.

Table 326. Summary of SDIO registers

Register	Description
pwrctrl	Power control register
clkctrl	Clock control register
arg	Argument register
cmd	Command register
rspcmd	Command response register
rsp1	Response register1
rsp2	Response register 2
rsp3	Response register 3
rsp4	Response register 4
dttmr	Data timer register
dtlen	Data length register
dtctrl	Data control register
dtcntr	Data counter register
sts	Status register
intclr	Clear interrupt register
inten	Interrupt mask register
bufcntr	BUF counter register
buf	Data BUF register

The table below gives a list of the SDIO library functions.

Table 327. Summary of SDIO library functions

Function name	Description
sdio_reset	Reset SDIO peripheral registers and control status
sdio_power_set	Configure controller power status
sdio_power_status_get	Get controller power status
sdio_clock_config	Configure clock parameters
sdio_bus_width_config	Configure bus width
sdio_clock_bypass	Enable clock bypass mode
sdio_power_saving_mode_enable	Enable controller power-saving mode
sdio_flow_control_enable	Enable flow control mode

sdio_clock_enable	Enable clock
sdio_dma_enable	Enable DMA
sdio_interrupt_enable	Enable interrupts
sdio_flag_get	Get the flag
sdio_flag_clear	Clear the flag
sdio_command_config	Configure command argument
sdio_command_state_machine_enable	Enable command state machine
sdio_command_response_get	Get response command index
sdio_response_get	Get card command response
sdio_data_config	Configure data paramters
sdio_data_state_machine_enable	Enable data state machine
sdio_data_counter_get	Get the counter of to-be-sent data
sdio_data_read	Read one-WORD data from the receive FIFO
sdio_buffer_counter_get	Get the counter of data to be written into BUF or read from BUF
sdio_data_write	Write one-WORD data to the transmit FIFO
sdio_read_wait_mode_set	Configure read wait mode
sdio_read_wait_start	Read wait start
sdio_read_wait_stop	Read wait stop
sdio_io_function_enable	Enable IO function mode
sdio_io_suspend_command_set	Enable suspend command in IO function mode

5.16.1 sdio_reset function

The table below describes the function sdio_reset.

Table 328. sdio_reset function

Name	Description
Function name	sdio_reset
Function prototype	void sdio_reset(sdio_type *sdio_x);
Function description	Reset SDIO peripheral registers and control status
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reset sdio */
sdio_reset(SDIO1);
```

5.16.2 sdio_power_set function

The table below describes the function sdio_power_set.

Table 329. sdio_power_set function

Name	Description
Function name	sdio_power_set
Function prototype	void sdio_power_set(sdio_type *sdio_x, sdio_power_state_type power_state);
Function description	Configure the controller power status
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	power_state: controller power status
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

power_state

It is used to set the controller power status.

SDIO_POWER_ON: Power ON

SDIO_POWER_OFF: Power OFF

Example:

```
/* sdio power on */
sdio_power_set(SDIO1, SDIO_POWER_ON);
```

5.16.3 sdio_power_status_get function

The table below describes the function sdio_power_status_get.

Table 330. sdio_power_status_get function

Name	Description
Function name	sdio_power_status_get
Function prototype	sdio_power_state_type sdio_power_status_get(sdio_type *sdio_x);
Function description	Get the controller power status
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	sdio_power_state_type: controller power status
Required preconditions	NA
Called functions	NA

Example:

```
/* check power status */
if(sdio_power_status_get(SDIO1) == SDIO_POWER_OFF)
{
  return SD_REQ_NOT_APPLICABLE;
}
```

5.16.4 sdio_clock_config function

The table below describes the function sdio_clock_config.

Table 331. sdio_clock_config function

Name	Description
Function name	sdio_clock_config
Function prototype	void sdio_clock_config(sdio_type *sdio_x, uint16_t clk_div, sdio_edge_phase_type clk_edg);
Function description	Configure clock parameters
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	clk_div: clock division, range: 0~0x3FF
Input parameter 2	clk_edg: clock edge configuration
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

clk_edg

Clock edge selection

SDIO_CLOCK_EDGE_RISING: Clock rising edge

SDIO_CLOCK_EDGE_FALLING: Clock falling edge

Example:

```
/* config sdio clock divide and edge phase */
sdio_clock_config(SDIO1, 0x2, SDIO_CLOCK_EDGE_FALLING);
```

5.16.5 sdio_bus_width_config function

The table below describes the function sdio_bus_width_config.

Table 332. sdio_bus_width_config function

Name	Description
Function name	sdio_bus_width_config
Function prototype	void sdio_bus_width_config(sdio_type *sdio_x, sdio_bus_width_type width);
Function description	Configure bus width
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	width: selected bus width
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

width

Data bus width selection

SDIO_BUS_WIDTH_D1: 1-bit data bus width

SDIO_BUS_WIDTH_D4: 4-bit data bus width

SDIO_BUS_WIDTH_D8: 8-bit data bus width

Example:

```
/* config sdio bus width */
```

```
sdio_bus_width_config(SDIOx, SDIO_BUS_WIDTH_D1);
```

5.16.6 sdio_clock_bypass function

The table below describes the function sdio_clock_bypass.

Table 333. sdio_clock_bypass function

Name	Description
Function name	sdio_clock_bypass
Function prototype	void sdio_clock_bypass(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable clock bypass mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state; enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* disable clock bypass */
sdio_clock_bypass(SDIO1, FALSE);
```

5.16.7 sdio_power_saving_mode_enable function

The table below describes the function sdio_power_saving_mode_enable.

Table 334. sdio_power_saving_mode_enable function

Name	Description
Function name	sdio_power_saving_mode_enable
Function prototype	void sdio_power_saving_mode_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable controller power-saving mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* disable power saving mode */
sdio_power_saving_mode_enable(SDIO1, FALSE);
```

5.16.8 sdio_flow_control_enable function

The table below describes the function sdio_flow_control_enable.

Table 335. sdio_flow_control_enable function

Name	Description
Function name	sdio_flow_control_enable
Function prototype	void sdio_flow_control_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable flow control mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* disable flow control */
sdio_flow_control_enable(SDIO1, FALSE);
```

5.16.9 sdio_clock_enable function

The table below describes the function sdio_clock_enable.

Table 336. sdio_clock_enable function

Name	Description
Function name	sdio_clock_enable
Function prototype	void sdio_clock_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable clock
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable to output sdio_ck */
sdio_clock_enable(SDIO1, TRUE);
```

5.16.10 sdio_dma_enable function

The table below describes the function sdio_dma_enable.

Table 337. sdio_dma_enable function

Name	Description
Function name	sdio_dma_enable
Function prototype	void sdio_dma_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable DMA
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable sdio dma */
sdio_dma_enable(SDIO1, TRUE);
```

5.16.11 sdio_interrupt_enable function

The table below describes the function sdio_interrupt_enable.

Table 338. crm_flag_clear function

Name	Description
Function name	sdio_interrupt_enable
Function prototype	void sdio_interrupt_enable(sdio_type *sdio_x, uint32_t int_opt, confirm_state new_state);
Function description	Enable interrupts
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	int_opt: selected interrupt type
Input parameter 3	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

int_opt

Interrupt type selection

SDIO_CMDFAIL_INT: Command CRC fail interrupt
SDIO_DTFAIL_INT: Data CRC fail interrupt
SDIO_CMDTIMEOUT_INT: Command timeout interrupt
SDIO_DTTIMEOUT_INT: Data timeout interrupt
SDIO_TXERRU_INT: TxBUF underrun error interrupt
SDIO_RXERRO_INT: RxBUF overrun error interrupt
SDIO_CMDRSPCMPL_INT: Command response received interrupt
SDIO_CMDCMPL_INT: Command sent interrupt
SDIO_DTCMP_INT: Data transfer complete interrupt

SDIO_SBITERR_INT: Start bit error interrupt
SDIO_DTBLKCMPL_INT: Data block transfer complete interrupt
SDIO_DOCMD_INT: Command acting interrupt
SDIO_DOTX_INT: Data transmit acting interrupt
SDIO_DORX_INT: Data receive acting interrupt
SDIO_TXBUFH_INT: TxBUF half empty interrupt
SDIO_RXBUFH_INT: RxBUF half empty interrupt
SDIO_TXBUFF_INT: TxBUF full interrupt
SDIO_RXBUFF_INT: RxBUF full interrupt
SDIO_TXBUFE_INT: TxBUF empty interrupt
SDIO_RXBUFE_INT: RxBUF empty interrupt
SDIO_TXBUF_INT: Data available in TxBUF interrupt
SDIO_RXBUF_INT: Data available in RxBUF interrupt
SDIO_SDIOIF_INT: SD I/O mode received interrupt

Example:

```
/* disable interrupt */
sdio_interrupt_enable(SDIO1, (SDIO_DTFAIL_INT    | SDIO_DTTIMEOUT_INT | \
                              SDIO_DTCMP_INT | SDIO_TXBUFH_INT | SDIO_RXBUFH_INT      | \
                              SDIO_TXERRU_INT| SDIO_RXERRO_INT | SDIO_SBITERR_INT), FALSE);
```

5.16.12 sdio_flag_get function

The table below describes the function sdio_flag_get.

Table 339. sdio_flag_get function

Name	Description
Function name	sdio_flag_get
Function prototype	flag_status sdio_flag_get(sdio_type *sdio_x, uint32_t flag);
Function description	Get the flag
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	flag: selected interrupt type
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

Flag selection

SDIO_CMDFAIL_FLAG: Command CRC fail flag
SDIO_DTFAIL_FLAG: Data CRC fail flag
SDIO_CMDTIMEOUT_FLAG: Command timeout flag
SDIO_DTTIMEOUT_FLAG: Data timeout flag
SDIO_TXERRU_FLAG: TxBUF underrun error flag
SDIO_RXERRO_FLAG: RxBUF overrun error flag
SDIO_CMDRSPCMPL_FLAG: Command response received flag
SDIO_CMDCMPL_FLAG: Command sent flag
SDIO_DTCMP_FLAG: Data transfer complete flag

SDIO_SBITERR_FLAG: Start bit error flag
SDIO_DTBLKCMPL_FLAG: Data block transfer complete flag
SDIO_DOCMD_FLAG: Command acting flag
SDIO_DOTX_FLAG: Data transmit acting flag
SDIO_DORX_FLAG: Data receive acting flag
SDIO_TXBUFH_FLAG: TxBUF half-empty flag
SDIO_RXBUFH_FLAG: RxBUF half-empty flag
SDIO_TXBUFF_FLAG: TxBUF full flag
SDIO_RXBUFF_FLAG: RxBUF full flag
SDIO_TXBUFE_FLAG: TxBUF empty flag
SDIO_RXBUFE_FLAG: RxBUF empty flag
SDIO_TXBUF_FLAG: Data available in TxBUF flag
SDIO_RXBUF_FLAG: Data available in RxBUF flag
SDIO_SDIOIF_FLAG: SD I/O mode received flag

Example:

```
/* check dttimeout flag */
if(sdio_flag_get(SDIOx, SDIO_DTTIMEOUT_FLAG) != RESET)
{
}
```

5.16.13 sdio_interrupt_flag_get function

The table below describes the function sdio_interrupt_flag_get.

Table 340. sdio_interrupt_flag_get function

Name	Description
Function name	sdio_interrupt_flag_get
Function prototype	flag_status sdio_interrupt_flag_get(sdio_type *sdio_x, uint32_t flag);
Function description	Check if the selected interrupt flag is set or not
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	flag: selected interrupt type
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

Flag selection

SDIO_CMDFAIL_FLAG: Command CRC fail flag
SDIO_DTFAIL_FLAG: Data CRC fail flag
SDIO_CMDTIMEOUT_FLAG: Command timeout flag
SDIO_DTTIMEOUT_FLAG: Data timeout flag
SDIO_TXERRU_FLAG: TxBUF underrun error flag
SDIO_RXERRO_FLAG: RxBUF overrun error flag
SDIO_CMDRSPCMPL_FLAG: Command response received flag
SDIO_CMDCMPL_FLAG: Command sent flag
SDIO_DTCMP_FLAG: Data transfer complete flag

SDIO_SBITERR_FLAG: Start bit error flag
SDIO_DTBLKCMPL_FLAG: Data block transfer complete flag
SDIO_DOCMD_FLAG: Command acting flag
SDIO_DOTX_FLAG: Data transmit acting flag
SDIO_DORX_FLAG: Data receive acting flag
SDIO_TXBUFH_FLAG: TxBUF half-empty flag
SDIO_RXBUFH_FLAG: RxBUF half-empty flag
SDIO_TXBUFF_FLAG: TxBUF full flag
SDIO_RXBUFF_FLAG: RxBUF full flag
SDIO_TXBUFE_FLAG: TxBUF empty flag
SDIO_RXBUFE_FLAG: RxBUF empty flag
SDIO_TXBUF_FLAG: Data available in TxBUF flag
SDIO_RXBUF_FLAG: Data available in RxBUF flag
SDIO_SDIOIF_FLAG: SD I/O mode received flag

Example

```
/* check dttimeout interrupt flag */
if(sdio_interrupt_flag_get(SDIOx, SDIO_DTTIMEOUT_FLAG) != RESET)
{
}
```

5.16.14 sdio_flag_clear function

The table below describes the function sdio_flag_clear.

Table 341. sdio_flag_clear function

Name	Description
Function name	sdio_flag_clear
Function prototype	void sdio_flag_clear(sdio_type *sdio_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	flag: selected interrupt type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

Flag selection

SDIO_CMDFAIL_FLAG: Command CRC fail flag
SDIO_DTFAIL_FLAG: Data CRC fail flag
SDIO_CMDTIMEOUT_FLAG: Command timeout flag
SDIO_DTTIMEOUT_FLAG: Data timeout flag
SDIO_TXERRU_FLAG: TxBUF underrun error flag
SDIO_RXERRO_FLAG: RxBUFoverrun error flag
SDIO_CMDRSPCMPL_FLAG: Command response received flag
SDIO_CMDCMPL_FLAG: Command transfer complete flag
SDIO_DTCMP_FLAG: Data transfer complete flag
SDIO_SBITERR_FLAG: Start bit error flag

SDIO_DTBLKCMPL_FLAG: Data block transfer complete flag
SDIO_SDIOIF_FLAG: SD I/O mode received flag

Example:

```
/* clear flags */
#define SDIO_STATIC_FLAGS                    ((uint32_t)0x000005FF)
sdio_flag_clear(SDIO1, SDIO_STATIC_FLAGS);
```

5.16.15 sdio_command_config function

The table below describes the function sdio_command_config.

Table 342. sdio_command_config function

Name	Description
Function name	sdio_command_config
Function prototype	void sdio_command_config(sdio_type *sdio_x, sdio_command_struct_type *command_struct);
Function description	Configure command argument
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	command_struct: sdio_command_struct_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

command_struct

sdio_command_struct_type is defined in the at32f413_sdio.h.

```
typedef struct
{
  uint32_t                argument;
  uint8_t                 cmd_index;
  sdio_reponse_type       rsp_type;
  sdio_wait_type          wait_type;
} sdio_command_struct_type;
```

argument

Command argument is sent to a card as part of a command. It is dependent on the command type.

cmd_index

Command index

rsp_type

Response type, dependent on the command type, including:

SDIO_RESPONSE_NO: No response
SDIO_RESPONSE_SHORT: Short response
SDIO_RESPONSE_LONG: Long response

wait_type

Wait type, dependent on the command type, including

SDIO_WAIT_FOR_NO: No wiat
SDIO_WAIT_FOR_INT: Wait for interrupt request
SDIO_WAIT_FOR_PEND: Wait for end of transfer

Example:

```
/* send cmd16, set block length */
sdio_command_struct_type sdio_command_init_struct;
sdio_command_init_struct.argument = (uint32_t)8;
sdio_command_init_struct.cmd_index = SD_CMD_SET_BLOCKLEN;
sdio_command_init_struct.rsp_type = SDIO_RESPONSE_SHORT;
sdio_command_init_struct.wait_type = SDIO_WAIT_FOR_NO;
/* sdio command config */
sdio_command_config(SDIOx, &sdio_command_init_struct);
```

5.16.16 sdio_command_state_machine_enable function

The table below describes the function sdio_command_state_machine_enable.

Table 343. sdio_command_state_machine_enable function

Name	Description
Function name	sdio_command_state_machine_enable
Function prototype	void sdio_command_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable command state machine
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable ccsm */
sdio_command_state_machine_enable(SDIO1, TRUE);
```

5.16.17 sdio_command_response_get function

The table below describes the function sdio_command_response_get.

Table 344. sdio_command_response_get function

Name	Description
Function name	sdio_command_response_get
Function prototype	uint8_t sdio_command_response_get(sdio_type *sdio_x);
Function description	Get response command index
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	uint8_t: response command index
Required preconditions	NA
Called functions	NA

Example:

```
/* get response of command index */
```

```
uint8_t rsp_cmd = 0;
rsp_cmd = sdio_command_response_get(SDIO1);
```

5.16.18 sdio_response_get function

The table below describes the function sdio_response_get.

Table 345. sdio_response_get function

Name	Description
Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(sdio_type *sdio_x, sdio_rsp_index_type reg_index);
Function description	Get card command response
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	reg_index: response register number (1/2/3/4)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

reg_div

Response register selection

SDIO_RSP1_INDEX: Response register 1

SDIO_RSP2_INDEX: Response register 2

SDIO_RSP3_INDEX: Response register 3

SDIO_RSP4_INDEX: Response register 4

Example:

```
/* get response register1 */
response = sdio_response_get(SDIO1, SDIO_RSP1_INDEX);
```

5.16.19 sdio_data_config function

The table below describes the function sdio_data_config.

Table 346. sdio_data_config function

Name	Description
Function name	sdio_data_config
Function prototype	void sdio_data_config(sdio_type *sdio_x, sdio_data_struct_type *data_struct);
Function description	Configure data paremeters
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	data_struct: sdio_data_struct_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

data_struct

sdio_data_struct_type is defined in the at32f413_sdio.h.

typedef struct

{

```
    uint32_t                            timeout;
    uint32_t                            data_length;
    sdio_block_size_type                block_size;
    sdio_transfer_mode_type             transfer_mode;
    sdio_transfer_direction_type        transfer_direction;
} sdio_data_struct_type;
```

timeout

Data transfer timeout, with the bus clock as the counting base

data_length

Length of the to-be-sent data

block_size

Block size, including

SDIO_DATA_BLOCK_SIZE_1B: 1-bit
SDIO_DATA_BLOCK_SIZE_2B: 2-bit
SDIO_DATA_BLOCK_SIZE_4B: 4-bit
SDIO_DATA_BLOCK_SIZE_8B: 8-bit
SDIO_DATA_BLOCK_SIZE_16B: 16-bit
SDIO_DATA_BLOCK_SIZE_32B: 32-bit
SDIO_DATA_BLOCK_SIZE_64B: 64-bit
SDIO_DATA_BLOCK_SIZE_128B: 128-bit
SDIO_DATA_BLOCK_SIZE_256B: 256-bit
SDIO_DATA_BLOCK_SIZE_512B: 512-bit
SDIO_DATA_BLOCK_SIZE_1024B: 1024-bit
SDIO_DATA_BLOCK_SIZE_2048B: 2048-bit
SDIO_DATA_BLOCK_SIZE_4096B: 4096-bit
SDIO_DATA_BLOCK_SIZE_8192B: 8192-bit
SDIO_DATA_BLOCK_SIZE_16384B: 16384-bit

transfer_mode

Data transfer mode selection

SDIO_DATA_BLOCK_TRANSFER: Data block mode
SDIO_DATA_STREAM_TRANSFER: Stream mode

transfer_direction

Data transfer direction selection

SDIO_DATA_TRANSFER_TO_CARD: Controller-to-card
SDIO_DATA_TRANSFER_TO_CONTROLLER: Card-to-controller

Example:

```
sdio_data_struct_type sdio_data_init_struct;
sdio_data_init_struct.block_size = SDIO_DATA_BLOCK_SIZE_512B;
sdio_data_init_struct.data_length = 8 ;
sdio_data_init_struct.timeout = SD_DATATIMEOUT ;
sdio_data_init_struct.transfer_direction = SDIO_DATA_TRANSFER_TO_CARD;
sdio_data_init_struct.transfer_mode = SDIO_DATA_BLOCK_TRANSFER;
/* config sdio data */
sdio_data_config(SDIO1, &sdio_data_init_struct);
```

5.16.20 sdio_data_state_machine_enable function

The table below describes the function sdio_data_state_machine_enable.

Table 347. sdio_data_state_machine_enable function

Name	Description
Function name	sdio_data_state_machine_enable
Function prototype	void sdio_data_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable data state machine
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state; enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable dcsm */
sdio_data_state_machine_enable(SDIO1, TRUE);
```

5.16.21 sdio_data_counter_get function

The table below describes the function sdio_data_counter_get.

Table 348. sdio_data_counter_get function

Name	Description
Function name	sdio_data_counter_get
Function prototype	uint32_t sdio_data_counter_get(sdio_type *sdio_x);
Function description	Get the counter of to-be-sent data
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: the counter of to-be-sent data
Required preconditions	NA
Called functions	NA

Example:

```
/* get data counter */
uint32_t count = 0;
count = sdio_data_counter_get (SDIO1);
```

5.16.22 sdio_data_read function

The table below describes the function sdio_data_read.

Table 349. sdio_data_read function

Name	Description
Function name	sdio_data_read
Function prototype	uint32_t sdio_data_read(sdio_type *sdio_x);
Function description	Read one-WORD data from the receive FIFO
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Input parameter 3	NA
Output parameter	NA
Return value	uint32_t: one-WORD data
Required preconditions	NA
Called functions	NA

Example:

```
/* read data */
uint32_t data = 0;
data = sdio_data_read(SDIO1);
```

5.16.23 sdio_buffer_counter_get function

The table below describes the function sdio_buffer_counter_get

Table 350. sdio_buffer_counter_get functi.on

Name	Description
Function name	sdio_buffer_counter_get
Function prototype	uint32_t sdio_buffer_counter_get(sdio_type *sdio_x);
Function description	Get the counter of data to be written into BUF or read from BUF
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* get buffer count */
uint32_t count = 0;
count = sdio_buffer_counter_get(SDIO1);
```

5.16.24 sdio_data_write function

The table below describes the function sdio_data_write.

Table 351. sdio_data_write function

Name	Description
Function name	sdio_data_write
Function prototype	void sdio_data_write(sdio_type *sdio_x, uint32_t data);
Function description	Write one-WORD data to the transmit FIFO
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* write data */
uin32_t data = 0x11223344;
sdio_data_write(SDIO1, data);
```

5.16.25 sdio_read_wait_mode_set function

The table below describes the function sdio_read_wait_mode_set.

Table 352. sdio_read_wait_mode_set function

Name	Description
Function name	sdio_read_wait_mode_set
Function prototype	void sdio_read_wait_mode_set(sdio_type *sdio_x, sdio_read_wait_mode_type mode);
Function description	Configure read wait mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	mode: read wait mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

mode

SDIO_READ_WAIT_CONTROLLED_BY_D2: Read wait is controlled by DATA Line2

SDIO_READ_WAIT_CONTROLLED_BY_CK: Read wait is controlled by clocke line

Example:

```
/* config read wait mode */
sdio_read_wait_mode_set(SDIO1, SDIO_READ_WAIT_CONTROLLED_BY_D2);
```

5.16.26 sdio_read_wait_start function

The table below describes the function sdio_read_wait_start.

Table 353. sdio_read_wait_start function

Name	Description
Function name	sdio_read_wait_start
Function prototype	void sdio_read_wait_start(sdio_type *sdio_x, confirm_state new_state);
Function description	Read wait start
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Calling this function indicates start of read wait; when this function is called to disable wait state, it indicates that no action occurs.

Example:

```
/* start read wait mode */
sdio_read_wait_start (SDIO1, TRUE);
```

5.16.27 sdio_read_wait_stop function

The table below describes the function sdio_read_wait_stop.

Table 354. sdio_read_wait_stop function

Name	Description
Function name	sdio_read_wait_stop
Function prototype	void sdio_read_wait_stop(sdio_type *sdio_x, confirm_state new_state);
Function description	Read wait stop
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Calling this function indicates stop of read wait; when this function is called to disable wait state, it indicates that the read wait is in progress.

Example:

```
/* stop read wait mode */
sdio_read_wait_stop (SDIO1, TRUE);
```

5.16.28 sdio_io_function_enable function

The table below describes the function sdio_io_function_enable.

Table 355. sdio_io_function_enable function

Name	Description
Function name	sdio_io_function_enable
Function prototype	void sdio_io_function_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable IO function mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable sdio IO mode */
sdio_io_function_enable (SDIO1, TRUE);
```

5.16.29 sdio_io_suspend_command_set function

The table below describes the function sdio_io_suspend_command_set.

Table 356. sdio_io_suspend_command_set function

Name	Description
Function name	sdio_io_suspend_command_set
Function prototype	void sdio_io_suspend_command_set(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable suspend command in IO function mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* send suspend command */
sdio_io_suspend_command_set (SDIO1, TRUE);
```

5.17 Serial peripheral port (SPI)/I²S

The SPI register structure spi_type is defined in the "at32f413_spi.h".

```
/**
  * @brief type define spi register all
  */
typedef struct
{
    …
} spi_type;
```

The table below gives a list of the SPI registers.

Table 357. Summary of SPI registers

Register	Description
ctrl1	SPI control register 1
ctrl2	SPI control register 2
sts	SPI status register
dt	SPI data register
cpoly	SPI CRC register
rcrc	SPI RxCRC register
tcrc	SPI TxCRC register
i2sctrl	SPI_I2S register
i2sclkp	SPI_I2S prescaler register

The table below gives a list of the SPI library functions.

Table 358. Summary of SPI library functions

Function name	Description
spi_i2s_reset	Reset SPI/I²S registers to their reset values
spi_default_para_init	Configure the SPI initialization structure with an initial value
spi_init	Initialize SPI
spi_crc_next_transmit	Next data transfer is CRC command
spi_crc_polynomial_set	SPI CRC polynomial configuration
spi_crc_polynomial_get	Get SPI CRC polynomial
spi_crc_enable	Enable SPI CRC
spi_crc_value_get	Get CRC result of SPI receive/transmit
spi_hardware_cs_output_enable	Enable hardware CS output
spi_software_cs_internal_level_set	Set software CS internal level
spi_frame_bit_num_set	Set the number of frame bits
spi_half_duplex_direction_set	Set transfer direction of single-wire bidirectional half-duplex mode
spi_enable	Enable SPI
i2s_default_para_init	Set an initial value for the I²S initialization structure
i2s_init	Initialize I²S
i2s_enable	Enable I²S
spi_i2s_interrupt_enable	Enable SPI/I²S interrupts

spi_i2s_dma_transmitter_enable	Enable SPI/I²S DMA transmit
spi_i2s_dma_receiver_enable	Enable SPI/I²S DMA receive
spi_i2s_data_transmit	SPI/I²S transmits data
spi_i2s_data_receive	SPI/I²S receives data
spi_i2s_flag_get	Get SPI/I²S flags
spi_i2s_flag_clear	Clear SPI/I²S flags

5.17.1 spi_i2s_reset function

The table below describes the function spi_i2s_reset.

Table 359. spi_i2s_reset function

Name	Description
Function name	spi_i2s_reset
Function prototype	void spi_i2s_reset(spi_type *spi_x);
Function description	Reset SPI/I²S registers to their reset values
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
spi_i2s_reset (SPI1);
```

5.17.2 spi_default_para_init function

The table below describes the function spi_default_para_init.

Table 360. spi_default_para_init function

Name	Description
Function name	spi_default_para_init
Function prototype	void spi_default_para_init(spi_init_type* spi_init_struct);
Function description	Set an initial value for the SPI initialization structure
Input parameter 1	spi_init_struct: *spi_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of spi_init_type before starting.
Called functions	NA

Example:

```
spi_init_type spi_init_struct;
spi_default_para_init (&spi_init_struct);
```

5.17.3 spi_init function

The table below describes the function spi_init.

Table 361. spi_init function

Name	Description
Function name	spi_init
Function prototype	void spi_init(spi_type* spi_x, spi_init_type* spi_init_struct);
Function description	Initialize SPI
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	spi_init_struct: *spi_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of spi_init_type before starting.
Called functions	NA

The spi_init_type is defined in the at32f413_spi.h.

```
typedef struct
{
    spi_transmission_mode_type          transmission_mode;
    spi_master_slave_mode_type          master_slave_mode;
    spi_mclk_freq_div_type              mclk_freq_division;
    spi_first_bit_type                  first_bit_transmission;
    spi_frame_bit_num_type              frame_bit_num;
    spi_clock_polarity_type             clock_polarity;
    spi_clock_phase_type                clock_phase;
    spi_cs_mode_type                    cs_mode_selection;
} spi_init_type;
```

spi_transmission_mode

SPI transmission mode selection

SPI_TRANSMIT_FULL_DUPLEX: Two-wire unidirectional full-duplex mode
SPI_TRANSMIT_SIMPLEX_RX: Two-wire unidirectional receive-only mode
SPI_TRANSMIT_HALF_DUPLEX_RX: Single-wire bidirectional receive-only mode
SPI_TRANSMIT_HALF_DUPLEX_TX: Single-wire bidirectional transmit-only mode

master_slave_mode

Master/slave mode selection

SPI_MODE_SLAVE: Slave mode
SPI_MODE_MASTER: Master mode

mclk_freq_division

Frequency division factor selection

SPI_MCLK_DIV_2: Divided by 2
SPI_MCLK_DIV_4: Divided by 4
SPI_MCLK_DIV_8: Divided by 8
SPI_MCLK_DIV_16: Divided by 16
SPI_MCLK_DIV_32: Divided by 32
SPI_MCLK_DIV_64: Divided by 64

SPI_MCLK_DIV_128:	Divided by 128
SPI_MCLK_DIV_256:	Divided by 256
SPI_MCLK_DIV_512:	Divided by 512
SPI_MCLK_DIV_1024:	Divided by 1024

first_bit_transmission

SPI MSB-first/LSB-first selection

SPI_FIRST_BIT_MSB:	MSB-first
SPI_FIRST_BIT_LSB:	LSB-first

frame_bit_num

Set the number of bits in a frame

SPI_FRAME_8BIT:	8-bit data in a frame
SPI_FRAME_16BIT:	16-bit data in a frame

clock_polarity

Select clock polarity

SPI_CLOCK_POLARITY_LOW:	Clock output low in idle state
SPI_CLOCK_POLARITY_HIGH:	Clock output high in idle state

clock_phase

Select clock phase

SPI_CLOCK_PHASE_1EDGE:	Sample on the first clock edge
SPI_CLOCK_PHASE_2EDGE:	Sample on the second clock edge

cs_mode_selection

Select CS mode

SPI_CS_HARDWARE_MODE:	Hardware CS mode
SPI_CS_SOFTWARE_MODE:	Software CS mode

Example:

```
spi_init_type spi_init_struct;
spi_default_para_init(&spi_init_struct);
spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;
spi_init_struct.master_slave_mode = SPI_MODE_MASTER;
spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;
spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_MSB;
spi_init_struct.frame_bit_num = SPI_FRAME_16BIT;
spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;
spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;
spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;
spi_init(SPI1, &spi_init_struct);
```

5.17.4 spi_crc_next_transmit function

The table below describes the function spi_crc_next_transmit.

Table 362. spi_crc_next_transmit function

Name	Description
Function name	spi_crc_next_transmit
Function prototype	void spi_crc_next_transmit(spi_type* spi_x);
Function description	The next data to be sent is CRC command
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
spi_crc_next_transmit (SPI1);
```

5.17.5 spi_crc_polynomial_set function

The table below describes the function spi_crc_polynomial_set.

Table 363. spi_crc_polynomial_set function

Name	Description
Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(spi_type* spi_x, uint16_t crc_poly);
Function description	Set SPI CRC polynomial
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	crc_poly: CRC polynomial Value range: 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/*set spi crc polynomial value */
spi_crc_polynomial_set (SPI1, 0x07);
```

5.17.6 spi_crc_polynomial_get function

The table below describes the function spi_crc_polynomial_get.

Table 364. spi_crc_polynomial_get function

Name	Description
Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(spi_type* spi_x);
Function description	Get SPI CRC polynomial
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Output parameter	NA
Return value	CRC polynomial Value range: 0x0000~0xFFFF
Required preconditions	NA
Called functions	NA

Example:

```
/*get spi crc polynomial value */
uint16_t crc_poly;
crc_poly = spi_crc_polynomial_get (SPI1);
```

5.17.7 spi_crc_enable function

The table below describes the function spi_crc_enable.

Table 365. spi_crc_enable function

Name	Description
Function name	spi_crc_enable
Function prototype	void spi_crc_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI CRC
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	new_state: enabled or disabled This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* spi crc enable */
spi_crc_enable (SPI1, TRUE);
```

5.17.8 spi_crc_value_get function

The table below describes the function spi_crc_value_get.

Table 366. spi_crc_value_get function

Name	Description
Function name	spi_crc_value_get
Function prototype	uint16_t spi_crc_value_get(spi_type* spi_x, spi_crc_direction_type crc_direction);
Function description	Get SPI receive/transmit CRC result
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	*crc_direction*: Select receive/transmit CRC
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

crc_direction

Select receive/transmit CRC

SPI_CRC_RX: Receive CRC

SPI_CRC_TX: Transmit CRC

Example:

```
/* get spi rx & tx crc enable */
uint16_t spi_rx_crc, spi_tx_crc;
spi_rx_crc = spi_crc_value_get (SPI1, SPI_CRC_RX);
spi_tx_crc = spi_crc_value_get (SPI1, SPI_CRC_TX);
```

5.17.9 spi_hardware_cs_output_enable function

The table below describes the function spi_hardware_cs_output_enable.

Table 367. spi_hardware_cs_output_enable function

Name	Description
Function name	spi_hardware_cs_output_enable
Function prototype	void spi_hardware_cs_output_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable hardware CS output
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	new_state: enabled or disabled This parameter can FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	This setting is applicable to SPI master mode only.
Called functions	NA

Example:

```
/* enable the hardware cs output */
```

```
spi_hardware_cs_output_enable (SPI1, TRUE);
```

5.17.10 spi_software_cs_internal_level_set function

The table below describes the function spi_software_cs_internal_level_set.

Table 368. spi_software_cs_internal_level_set function

Name	Description
Function name	spi_software_cs_internal_level_set
Function prototype	void spi_software_cs_internal_level_set(spi_type* spi_x, spi_software_cs_level_type level);
Function description	Set software CS internal level
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	*level*: set software CS internal level
Output parameter	NA
Return value	NA
Required preconditions	1. This setting is applicable to software CS mode only 2. In master mode, the “level” value must be “SPI_SWCS_INTERNAL_LEVEL_HIGHT”.
Called functions	NA

level

Set software CS internal level

SPI_SWCS_INTERNAL_LEVEL_LOW: Software CS internal low level

SPI_SWCS_INTERNAL_LEVEL_HIGHT: Software CS internal high level

Example:

```
/* set the internal level high */
spi_software_cs_internal_level_set (SPI1, SPI_SWCS_INTERNAL_LEVEL_HIGHT);
```

5.17.11 spi_frame_bit_num_set function

The table below describes the function spi_frame_bit_num_set.

Table 369. spi_frame_bit_num_set function

Name	Description
Function name	spi_frame_bit_num_set
Function prototype	void spi_frame_bit_num_set(spi_type* spi_x, spi_frame_bit_num_type bit_num);
Function description	Set the number of bits in a frame
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	*bit_num*: Set the number of bits in a frame
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

bit_num

Set the number of bits in a frame

SPI_FRAME_8BIT: 8-bit data in a frame

SPI_FRAME_16BIT: 16-bit data in a frame

Example:

```
/* set the data frame bit num as 8 */
spi_frame_bit_num_set (SPI1, SPI_FRAME_8BIT);
```

5.17.12 spi_half_duplex_direction_set function

The table below describes the function spi_half_duplex_direction_set.

Table 370. spi_half_duplex_direction_set function

Name	Description
Function name	spi_half_duplex_direction_set
Function prototype	void spi_half_duplex_direction_set(spi_type* spi_x, spi_half_duplex_direction_type direction);
Function description	Set the transfer direction of single-wire bidirectional half-duplex mode
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	*direction*: transfer direction
Output parameter	NA
Return value	NA
Required preconditions	This setting is applicable to the single-wire bidirectional half-duplex mode only.
Called functions	NA

direction

Transfer direction

SPI_HALF_DUPLEX_DIRECTION_RX: Receive

SPI_HALF_DUPLEX_DIRECTION_TX: Transmit

Example:

```
/* set the data transmission direction as transmit */
spi_half_duplex_direction_set (SPI1, SPI_HALF_DUPLEX_DIRECTION_TX);
```

5.17.13 spi_enable function

The table below describes the function spi_enable.

Table 371. spi_enable function

Name	Description
Function name	spi_enable
Function prototype	void spi_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	new_state: enabled or disabled This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable spi */
spi_enable (SPI1, TRUE);
```

5.17.14 i2s_default_para_init function

The table below describes the function i2s_default_para_init.

Table 372. i2s_default_para_init function

Name	Description
Function name	i2s_default_para_init
Function prototype	void i2s_default_para_init(i2s_init_type* i2s_init_struct);
Function description	Set an initial value for the I^{2}S initialization structure
Input parameter 1	i2s_init_struct: *spi_i2s_flag* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of i2s_init_type before starting.
Called functions	NA

Example:

```
i2s_init_type i2s_init_struct;
i2s_default_para_init (&i2s_init_struct);
```

5.17.15 i2s_init function

The table below describes the function i2s_init.

Table 373. i2s_init function

Name	Description
Function name	i2s_init
Function prototype	void i2s_init(spi_type* spi_x, i2s_init_type* i2s_init_struct);
Function description	Initialize I^2S
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	i2s_init_struct: *spi_i2s_flag* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of i2s_init_type before starting.
Called functions	NA

The i2s_init_type is defined in the at32f413_spi.h.

```
typedef struct
{
    i2s_operation_mode_type          operation_mode;
    i2s_audio_protocol_type          audio_protocol;
    i2s_audio_sampling_freq_type     audio_sampling_freq;
    i2s_data_channel_format_type     data_channel_format;
    i2s_clock_polarity_type          clock_polarity;
    confirm_state                    mclk_output_enable;
} i2s_init_type;
```

operation_mode

I^2S transfer mode

I2S_MODE_SLAVE_TX: I2S slave transmit

I2S_MODE_SLAVE_RX: I2S slave receive

I2S_MODE_MASTER_TX: I2S master transmit

I2S_MODE_MASTER_RX: I2S master receive

audio_protocol

I^2S audio protocol standards

I2S_AUDIO_PROTOCOL_PHILLIPS: Phillips

I2S_AUDIO_PROTOCOL_MSB: MSB aligned (left-aligned)

I2S_AUDIO_PROTOCOL_LSB: LSB aligned (right-aligned)

I2S_AUDIO_PROTOCOL_PCM_SHORT: PCM short frame synchronization

I2S_AUDIO_PROTOCOL_PCM_LONG: PCM long frame synchronization

audio_sampling_freq

I^2S audio sampling frequency.

I2S_AUDIO_FREQUENCY_DEFAULT:

Kept at its reset value (sampling frequency changes with SCLK)

I2S_AUDIO_FREQUENCY_8K: I2S sampling frequency 8K

I2S_AUDIO_FREQUENCY_11_025K: I2S sampling frequency 11.025K

I2S_AUDIO_FREQUENCY_16K: I2S sampling frequency 16K

I2S_AUDIO_FREQUENCY_22_05K: I2S sampling frequency 22.05K
I2S_AUDIO_FREQUENCY_32K: I2S sampling frequency 32K
I2S_AUDIO_FREQUENCY_44_1K: I2S sampling frequency 44.1K
I2S_AUDIO_FREQUENCY_48K: I2S sampling frequency 48K
I2S_AUDIO_FREQUENCY_96K: I2S sampling frequency 96K
I2S_AUDIO_FREQUENCY_192K: I2S sampling frequency 192K

data_channel_format

I^2S data/channel bits format

I2S_DATA_16BIT_CHANNEL_16BIT: 16-bit data, 16-bit channel
I2S_DATA_16BIT_CHANNEL_32BIT: 16-bit data, 32-bit channel
I2S_DATA_24BIT_CHANNEL_32BIT: 24-bit data, 32-bit channel
I2S_DATA_32BIT_CHANNEL_32BIT: 32-bit data, 32-bit channel

clock_polarity

I^2S clock polarity

I2S_CLOCK_POLARITY_LOW: Clock output low in idle state
I2S_CLOCK_POLARITY_HIGH: Clock output high in idle state

mclk_output_enable

Enable mclk clock output

This parameter can be FALSE or TURE.

Example:

```
i2s_init_type i2s_init_struct;
i2s_default_para_init(&i2s_init_struct);
i2s_init_struct.audio_protocol = I2S_AUDIO_PROTOCOL_PHILLIPS;
i2s_init_struct.data_channel_format = I2S_DATA_16BIT_CHANNEL_32BIT;
i2s_init_struct.mclk_output_enable = FALSE;
i2s_init_struct.audio_sampling_freq = I2S_AUDIO_FREQUENCY_48K;
i2s_init_struct.clock_polarity = I2S_CLOCK_POLARITY_LOW;
i2s_init_struct.operation_mode = I2S_MODE_MASTER_TX;
i2s_init(SPI2, &i2s_init_struct);
```

5.17.16 i2s_enable function

The table below describes the function i2s_enable.

Table 374. i2s_enable function

Name	Description
Function name	i2s_enable
Function prototype	void i2s_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable I^2S
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable i2s*/
i2s_enable (SPI1, TRUE);
```

5.17.17 spi_i2s_interrupt_enable function

The table below describes the function spi_i2s_interrupt_enable.

Table 375. spi_i2s_interrupt_enable function

Name	Description
Function name	spi_i2s_interrupt_enable
Function prototype	void spi_i2s_interrupt_enable(spi_type* spi_x, uint32_t spi_i2s_int, confirm_state new_state);
Function description	Enable SPI/I²Sinterrupts
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	*spi_i2s_int*: select SPI interrupts
Input parameter 3	new_state: Enable or disable This parameter can be FALSE or TURE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

spi_i2s_int

SPI/I²S interrupt selection

SPI_I2S_ERROR_INT: SPI/I²Serror interrupts (including CRC error, overflow error, underflow error and mode error)

SPI_I2S_RDBF_INT: Receive data buffer full

SPI_I2S_TDBE_INT: Transmit data buffer empty

Example:

```
/* enable the specified spi/i2s interrupts */
spi_i2s_interrupt_enable (SPI1, SPI_I2S_ERROR_INT);
spi_i2s_interrupt_enable (SPI1, SPI_I2S_RDBF_INT);
spi_i2s_interrupt_enable (SPI1, SPI_I2S_TDBE_INT);
```

5.17.18 spi_i2s_dma_transmitter_enable function

The table below describes the function spi_i2s_dma_transmitter_enable.

Table 376. spi_i2s_dma_transmitter_enable function

Name	Description
Function name	spi_i2s_dma_transmitter_enable
Function prototype	void spi_i2s_dma_transmitter_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI/I^{2}S DMA transmitter
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	new_state: enabled or disabled This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable spi transmitter dma */
spi_i2s_dma_transmitter_enable (SPI1, TRUE);
```

5.17.19 spi_i2s_dma_receiver_enable function

The table below describes the function spi_i2s_dma_receiver_enable.

Table 377. spi_i2s_dma_receiver_enable function

Name	Description
Function name	spi_i2s_dma_receiver_enable
Function prototype	void spi_i2s_dma_receiver_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI/I^{2}S DMA receiver
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	new_state: enabled or disabled This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable spi dma transmitter */
spi_i2s_dma_transmitter_enable (SPI1, TRUE);
```

5.17.20 spi_i2s_data_transmit function

The table below describes the function spi_i2s_data_transmit.

Table 378. spi_i2s_data_transmit function

Name	Description
Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(spi_type* spi_x, uint16_t tx_data);
Function description	SPI/I²S sends data
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	tx_data: data to send Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for 16-bit bit in a frame): 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* spi data transmit */
uint16_t tx_data = 0x6666;
spi_i2s_data_transmit (SPI1, tx_data);
```

5.17.21 spi_i2s_data_receive function

The table below describes the function spi_i2s_data_receive.

Table 379. spi_i2s_data_receive function

Name	Description
Function name	spi_i2s_data_receive
Function prototype	uint16_t spi_i2s_data_receive(spi_type* spi_x);
Function description	SPI/I²S receives data
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Output parameter	rx_data: data to receive Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for 16-bit bit in a frame): 0x0000~0xFFFF
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* spi data receive */
uint16_t rx_data = 0;
rx_data = spi_i2s_data_receive (SPI1);
```

5.17.22 spi_i2s_flag_get function

The table below describes the function spi_i2s_flag_get.

Table 380. spi_i2s_flag_get function

Name	Description
Function name	spi_i2s_flag_get
Function prototype	flag_status spi_i2s_flag_get(spi_type* spi_x, uint32_t spi_i2s_flag);
Function description	Get SPI/I²S flags
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	spi_i2s_flag: flag selection Refer to the "spi_i2s_flag" description below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

spi_i2s_flag

SPI/I²S is used to select a flag from the optional parameters below:

SPI_I2S_RDBF_FLAG: SPI/I²S receive data buffer full
SPI_I2S_TDBE_FLAG: SPI/I²S transmit data buffer empty
I2S_ACS_FLAG: I2S audio channel state (indicating left/right channel)
I2S_TUERR_FLAG: I2S transmitter underload error
SPI_CCERR_FLAG: SPI CRC error
SPI_MMERR_FLAG: SPI master mode error
SPI_I2S_ROERR_FLAG: SPI/I²S receiver overflow error
SPI_I2S_BF_FLAG: SPI/I²S busy

Example:

```
/* get receive data buffer full flag */
flag_status status;
status = spi_i2s_flag_get(SPI1, SPI_I2S_RDBF_FLAG);
```

5.17.23 spi_i2s_interrupt_flag_get function

The table below describes the function spi_i2s_interrupt_flag_get.

Table 381. spi_i2s_interrupt_flag_get function

Name	Description
Function name	spi_i2s_interrupt_flag_get
Function prototype	flag_status spi_i2s_interrupt_flag_get(spi_type* spi_x, uint32_t spi_i2s_flag);
Function description	Get SPI/I^{2}S interrupt flag
Input parameter 1	spi_x: : selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	spi_i2s_flag: flag selection Refer to the “spi_i2s_flag” description below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

spi_i2s_flag

SPI/I^{2}S is used to select a flag from the optional parameters below:

SPI_I2S_RDBF_FLAG: SPI/I^{2}S receive data buffer full

SPI_I2S_TDBE_FLAG: SPI/I^{2}S transmit data buffer empty

I2S_TUERR_FLAG: I2S transmitter underload error

SPI_CCERR_FLAG: SPI CRC error

SPI_MMERR_FLAG: SPI master mode error

SPI_I2S_ROERR_FLAG: SPI/I^{2}S receiver overflow error

Example

```
/* get receive data buffer full flag */
flag_status status;
status = spi_i2s_interrupt_flag_get(SPI1, SPI_I2S_RDBF_FLAG);
```

5.17.24 spi_i2s_flag_clear function

The table below describes the function spi_i2s_flag_clear.

Table 382. spi_i2s_flag_clear function

Name	Description
Function name	spi_i2s_flag_clear
Function prototype	void spi_i2s_flag_clear(spi_type* spi_x, uint32_t spi_i2s_flag)
Function description	Clear SPI/I^{2}S flags
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1 or SPI2.
Input parameter 2	*spi_i2s_flag:* select a flag to clear Refer to the “spi_i2s_flag” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

spi_i2s_flag:

SPI/I^{2}S is used for flag selection, including:

SPI_I2S_RDBF_FLAG: SPI/I^{2}S receive data buffer full

I2S_TUERR_FLAG: I2S transmitter underload error

SPI_CCERR_FLAG: SPI CRC error

SPI_MMERR_FLAG: SPI master mode error

SPI_I2S_ROERR_FLAG: SPI/I^{2}S receiver overflow error

Note: The SPI_I2S_TDBE_FLAG (SPI/I2S transmit data buffer empty), I2S_ACS_FLAG (I2S audio channel state) and SPI_I2S_BF_FLAG (SPI/I2S busy) are all set and cleared by hardware to indicate communication state, without the intervention of software.

Example:

```
/* clear receive data buffer full flag */
spi_i2s_flag_clear (SPI1, SPI_I2S_RDBF_FLAG);
```

5.18 SysTick

The SysTick register structure SysTick_Type is defined in the “core_cm4.h”.

```
typedef struct
{
    ...

} SysTick_Type;
```

The table below gives a list of the SysTick registers.

Table 383. Summary of SysTick registers

Register	Description
ctrl	Control status register
load	Reload value register
val	Current counter value register
calib	Calibration register

The table below gives a list of the SysTick library functions.

Table 384. Summary of SysTick library functions

Function name	Description
systick_clock_source_config	Configure SysTick clock sources
SysTick_Config	Configure SysTick counter reload value and interrupts

5.18.1 systick_clock_source_config function

The table below describes the function systick_clock_source_config.

Table 385. systick_clock_source_config function

Name	Description
Function name	systick_clock_source_config
Function prototype	void systick_clock_source_config(systick_clock_source_type source);
Function description	Configure SysTick clock source
Input parameter 1	source: systick clock source
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

source

SYSTICK_CLOCK_SOURCE_AHBCLK_DIV8: AHB/8 as SysTick clock

SYSTICK_CLOCK_SOURCE_AHBCLK_NODIV: AHB as SysTick clock

Example:

```
/* config systick clock source */
systick_clock_source_config(SYSTICK_CLOCK_SOURCE_AHBCLK_NODIV);
```

5.18.2 SysTick_Config function

The table below describes the function SysTick_Config.

Table 386. SysTick_Config function

Name	Description
Function name	SysTick_Config
Function prototype	uint32_t SysTick_Config(uint32_t ticks);
Function description	Configure SysTick counter reload value and enable interrupt
Input parameter 1	ticks: SysTick counter interrupt reload value
Input parameter 2	
Output parameter	NA
Return value	Return the setting status of this function, success (0) or failure (1)
Required preconditions	NA
Called functions	NA

Example:

```
/* config systick reload value and enable interrupt */
SysTick_Config(1000);
```

5.19 Timers (TMR)

The TMR register structure tmr_type is defined in the “at32f413_tmr.h”.

```
/**
  * @brief type define tmr register all
  */
typedef struct
{


} tmr_type;
```

The table below gives a list of the TMR registers.

Table 387. Summary of TMR registers

Register	Description
ctrl1	TMR control register 1
ctrl2	TMR control register 2
stctrl	TMR slave timer control register
iden	TMR DMA/interrupt enable register
ists	TMR interrupt status register
swevt	TMR software event register
cm1	TMR channel mode register 1
cm2	TMR channel mode register 2
cctrl	TMR channel control register
cval	TMR counter value register
div	TMR divider
pr	TMR period register
rpr	TMR repetition period register
c1dt	TMR channel 1 data register
c2dt	TMR channel 2 data register
c3dt	TMR channel 3 data register
c4dt	TMR channel 4 data register
brk	TMR break register
dmactrl	TMR DMA control register
dmadt	TMR DMA data register

The table below gives a list of the TMR library functions.

Table 388. Summary of TMR library functions

Function name	Description
tmr_reset	TMR is reset by CRM reset register
tmr_counter_enable	Enable/disable TMR counter
tmr_output_default_para_init	Initialize TMR output default parameters
tmr_input_default_para_init	Initialize TMR input default parameters
tmr_brkdt_default_para_init	Initialize TMR brkdt default parameters

tmr_base_init	Initialize TMR period and division
tmr_clock_source_div_set	Set TMR clock source frequency division factor
tmr_cnt_dir_set	Set TMR counter direction
tmr_repetition_counter_set	Set repetition period register
tmr_counter_value_set	Set TMR counter value
tmr_counter_value_get	Get TMR counter value
tmr_div_value_set	Set TMR division value
tmr_div_value_get	Get TMR division value
tmr_output_channel_config	Configure TMR output channels
tmr_output_channel_mode_select	Select TMR output channel mode
tmr_period_value_set	Set TMR period value
tmr_period_value_get	Get TMR period value
tmr_channel_value_set	Set TMR channel value
tmr_channel_value_get	Get TMR channel value
tmr_period_buffer_enable	Enable/disable TMR periodic buffer
tmr_output_channel_buffer_enable	Enable/disable TMR output channel buffer
tmr_output_channel_immediately_set	TMR output channel enable immediately
tmr_output_channel_switch_set	Set TMR output channel switch
tmr_one_cycle_mode_enable	Enable/disable TMR one-cycle mode
tmr_32_bit_function_enable	Enable/disable TMR 32-bit function (plus mode)
tmr_overflow_request_source_set	Select TMR overflow event source
tmr_overflow_event_disable	Enable/disable TMR overflow event generation
tmr_input_channel_init	Initialize TMR input channel
tmr_channel_enable	Enable/disable TMR channel
tmr_input_channel_filter_set	Set TMR input channel filter
tmr_pwm_input_config	Configure TMR pwm input
tmr_channel1_input_select	Select TMR channel 1 input
tmr_input_channel_divider_set	Set TMR input channel divider
tmr_primary_mode_select	Select TMR master mode
tmr_sub_mode_select	Select TMR slave timer mode
tmr_channel_dma_select	Select TMR channel DMA request source
tmr_hall_select	Select TMR hall mode
tmr_channel_buffer_enable	Enable/disable TMR channel buffer
tmr_trigger_input_select	Select TMR slave timer trigger input
tmr_sub_sync_mode_set	Set TMR slave timer synchronization mode
tmr_dma_request_enable	Enable/disable TMR DMA request
tmr_interrupt_enable	Enable/disable TMR interrupt
tmr_flag_get	Get TMR flags
tmr_flag_clear	Clear TMR flags
tmr_event_sw_trigger	Software trigger TMR event
tmr_output_enable	Enable/disable TMR output
tmr_internal_clock_set	Set TMR internal clock
tmr_output_channel_polarity_set	Set TMR output channel polarity
tmr_external_clock_config	Set TMR external clock
tmr_external_clock_mode1_config	Set TMR external clock mode 1

tmr_external_clock_mode2_config	Set TMR external clock mode 2
tmr_encoder_mode_config	Set TMR encoder mode
tmr_force_output_set	Set TMR forced output
tmr_dma_control_config	Set TMR DMA control
tmr_brkdt_config	Set TMR break mode and dead-time

5.19.1 tmr_reset function

The table below describes the function tmr_reset.

Table 389. tmr_reset function

Name	Description
Function name	tmr_reset
Function prototype	void tmr_reset(tmr_type *tmr_x);
Function description	TMR is reset by CRM reset register.
Input parameter	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
tmr_reset(TMR1);
```

5.19.2 tmr_counter_enable function

The table below describes the function tmr_counter_enable.

Table 390. tmr_counter_enable function

Name	Description
Function name	tmr_counter_enable
Function prototype	void tmr_counter_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR counter
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11
Input parameter 2	new_state: indicates counter status, ON (TRUE) or OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_counter_enable(TMR1, TRUE);
```

5.19.3 tmr_output_default_para_init function

The table below describes the function tmr_output_default_para_init.

Table 391. tmr_output_default_para_init function

Name	Description
Function name	tmr_output_default_para_init
Function prototype	void tmr_output_default_para_init(tmr_output_config_type *tmr_output_struct);
Function description	Initialize tmr output default parameters
Input parameter	tmr_output_struct: tmr_output_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the tmr_output_struct.

Table 392. tmr_output_struct default values

Member	Default value
oc_mode	TMR_OUTPUT_CONTROL_OFF
oc_idle_state	FALSE
occ_idle_state	FALSE
oc_polarity	TMR_OUTPUT_ACTIVE_HIGH
occ_polarity	TMR_OUTPUT_ACTIVE_HIGH
oc_output_state	FALSE
occ_output_state	FALSE

Example:

```
tmr_output_config_type tmr_output_struct;
tmr_output_default_para_init(&tmr_output_struct);
```

5.19.4 tmr_input_default_para_init function

The table below describes the function tmr_input_default_para_init.

Table 393. tmr_input_default_para_init function

Name	Description
Function name	tmr_input_default_para_init
Function prototype	void tmr_input_default_para_init(tmr_input_config_type *tmr_input_struct);
Function description	Initialize TMR input default parameters
Input parameter	tmr_input_struct: tmr_input_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the tmr_input_struct.

Table 394. tmr_input_struct default values

Member	Default value
input_channel_select	TMR_SELECT_CHANNEL_1
input_polarity_select	TMR_INPUT_RISING_EDGE
input_mapped_select	TMR_CC_CHANNEL_MAPPED_DIRECT
input_filter_value	0x0

Example:

```
tmr_input_config_type tmr_input_struct;
tmr_input_default_para_init(&tmr_input_struct);
```

5.19.5 tmr_brkdt_default_para_init function

The table below describes the function tmr_brkdt_default_para_init.

Table 395. tmr_brkdt_default_para_init function

Name	Description
Function name	tmr_brkdt_default_para_init
Function prototype	void tmr_brkdt_default_para_init(tmr_brkdt_config_type *tmr_brkdt_struct);
Function description	Initialize TMR brkdt default parameters
Input parameter	tmr_brkdt_struct: tmr_brkdt_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the tmr_brkdt_struct.

Table 396. tmr_brkdt_struct default values

Member	Default value
deadtime	0x0
brk_polarity	TMR_BRK_INPUT_ACTIVE_LOW
wp_level	TMR_WP_OFF
auto_output_enable	FALSE
fcsoen_state	FALSE
fcsodis_state	FALSE
brk_enable	FALSE

Example:

```
tmr_brkdt_config_type tmr_brkdt_struct;
tmr_brkdt_default_para_init(&tmr_brkdt_struct);
```

5.19.6 tmr_base_init function

The table below describes the function tmr_base_init

Table 397. tmr_base_init function

Name	Description
Function name	tmr_base_init
Function prototype	void tmr_base_init(tmr_type* tmr_x, uint32_t tmr_pr, uint32_t tmr_div);
Function description	Initialize TMR period and division
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_pr: timer period value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-bit timer
Input parameter 3	tmr_div: timer division value, 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_base_init(TMR1, 0xFFFF, 0xFFFF);
```

5.19.7 tmr_clock_source_div_set function

The table below describes the function tmr_clock_source_div_set.

Table 398. tmr_clock_source_div_set function

Name	Description
Function name	tmr_clock_source_div_set
Function prototype	void tmr_clock_source_div_set(tmr_type *tmr_x, tmr_clock_division_type tmr_clock_div);
Function description	Set TMR clock source division
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_clock_div: timer clock source frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_clock_div

Select TMR clock source frequency division factor

TMR_CLOCK_DIV1: Divided by 1

TMR_CLOCK_DIV2: Divided by 2

TMR_CLOCK_DIV4: Divided by 4

Example:

```
tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV4);
```

5.19.8 tmr_cnt_dir_set function

The table below describes the function tmr_cnt_dir_set.

Table 399. tmr_cnt_dir_set function

Name	Description
Function name	tmr_cnt_dir_set
Function prototype	void tmr_cnt_dir_set(tmr_type *tmr_x, tmr_count_mode_type tmr_cnt_dir);
Function description	Set TMR counter direction
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_cnt_dir: timer counting direction
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_cnt_dir

Select timer counting direction.

TMR_COUNT_UP: Up counting

TMR_COUNT_DOWN: Down counting

TMR_COUNT_TWO_WAY_1: Center-aligned mode (up/down counting) 1

TMR_COUNT_TWO_WAY_2: Center-aligned mode (up/down counting)2

TMR_COUNT_TWO_WAY_3: Center-aligned mode (up/down counting)3

Example:

```
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
```

5.19.9 tmr_repetition_counter_set function

The table below describes the function tmr_repetition_counter_set.

Table 400. tmr_repetition_counter_set function

Name	Description
Function name	tmr_repetition_counter_set
Function prototype	void tmr_repetition_counter_set(tmr_type *tmr_x, uint8_t tmr_rpr_value);
Function description	Set repetition period register (rpr)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1 or TMR8.
Input parameter 2	tmr_rpr_value: timer repetition period value, 0x00~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_repetition_counter_set(TMR1, 0x10);
```

5.19.10 tmr_counter_value_set function

The table below describes the function tmr_counter_value_set.

Table 401. tmr_counter_value_set function

Name	Description
Function name	tmr_counter_value_set
Function prototype	void tmr_counter_value_set(tmr_type *tmr_x, uint32_t tmr_cnt_value);
Function description	Set TMR counter value
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_cnt_value: timer counter value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-bit timer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_counter_value_set(TMR1, 0xFFFF);
```

5.19.11 tmr_counter_value_get function

The table below describes the function tmr_counter_value_get.

Table 402. tmr_counter_value_get function

Name	Description
Function name	tmr_counter_value_get
Function prototype	uint32_t tmr_counter_value_get(tmr_type *tmr_x);
Function description	Get TMR counter value
Input parameter	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Output parameter	NA
Return value	Timer counter value
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t counter_value;
counter_value = tmr_counter_value_get(TMR1);
```

5.19.12 tmr_div_value_set function

The table below describes the function tmr_div_value_set.

Table 403. tmr_div_value_set function

Name	Description
Function name	tmr_div_value_set
Function prototype	void tmr_div_value_set(tmr_type *tmr_x, uint32_t tmr_div_value);
Function description	Set TMR frequency division value
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_div_value: timer frequency division value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-bit timer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_div_value_set(TMR1, 0xFFFF);
```

5.19.13 tmr_div_value_get function

The table below describes the function tmr_div_value_get.

Table 404. tmr_div_value_get function

Name	Description
Function name	tmr_div_value_get
Function prototype	uint32_t tmr_div_value_get(tmr_type *tmr_x);
Function description	Get TMR frequency division value
Input parameter	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Output parameter	NA
Return value	Timer frequency division value
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t div_value;
div_value = tmr_div_value_get(TMR1);
```

5.19.14 tmr_output_channel_config function

The table below describes the function tmr_output_channel_config.

Table 405. tmr_output_channel_config function

Name	Description
Function name	tmr_output_channel_config
Function prototype	void tmr_output_channel_config(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_output_config_type *tmr_output_struct);
Function description	Configure TMR output channels
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	tmr_output_struct: tmr_output_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Channel 1

TMR_SELECT_CHANNEL_2: Channel 2

TMR_SELECT_CHANNEL_3: Channel 3

TMR_SELECT_CHANNEL_4: Channel 4

tmr_output_config_type structure

tmr_output_config_type is defined in the at32f413_tmr.h.

```
typedef struct
{
  tmr_output_control_mode_type      oc_mode;
  confirm_state                     oc_idle_state;
  confirm_state                     occ_idle_state;
  tmr_output_polarity_type          oc_polarity;
  tmr_output_polarity_type          occ_polarity;
  confirm_state                     oc_output_state;
  confirm_state                     occ_output_state;
} tmr_output_config_type;
```

oc_mode

Set output channel mode, that is, to configure channel original signals (CxORAW).

TMR_OUTPUT_CONTROL_OFF: Disconnect channel output (CxOUT) from CxORAW

TMR_OUTPUT_CONTROL_HIGH: CxORAW high

TMR_OUTPUT_CONTROL_LOW: CxORAW low

TMR_OUTPUT_CONTROL_SWITCH: Switch CxORAW level

TMR_OUTPUT_CONTROL_FORCE_LOW: CxORAW forced low

TMR_OUTPUT_CONTROL_FORCE_HIGH: CxORAW forced high

TMR_OUTPUT_CONTROL_PWM_MODE_A: PWM mode A

TMR_OUTPUT_CONTROL_PWM_MODE_B: PWM mode B

oc_idle_state

Set output channel idle state

FALSE: Output channel idle state is 0

TRUE: Output channel idle state is 1

occ_idle_state

Set complementary output channel idle state

FALSE: Complementary output channel idle state is 0

TRUE: Complementary output channel idle state is 1

oc_polarity

Set the polarity of output channels

TMR_OUTPUT_ACTIVE_HIGH: Active high

TMR_OUTPUT_ACTIVE_LOW: Active low

occ_polarity

Set the polarity of complementary output channels

TMR_OUTPUT_ACTIVE_HIGH: Active high

TMR_OUTPUT_ACTIVE_LOW: Active low

oc_output_state

Set the state of output channels

FALSE: Output channel OFF

TRUE: Output channel ON

occ_output_state

Set the state of complementary output channels

FALSE: Complementary output channel OFF

TRUE: Complementary output channel ON

Example:

```
tmr_output_config_type tmr_output_struct;
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_OFF;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = TRUE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.occ_idle_state = TRUE;
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
```

5.19.15 tmr_output_channel_mode_select function

The table below describes the function tmr_output_channel_mode_select.

Table 406. tmr_output_channel_mode_select function

Name	Description
Function name	tmr_output_channel_mode_select
Function prototype	void tmr_output_channel_mode_select(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_output_control_mode_type oc_mode);
Function description	Select TMR output channel mode
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	oc_mode: output mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Timer channel 1
TMR_SELECT_CHANNEL_2: Timer channel 2
TMR_SELECT_CHANNEL_3: Timer channel 3
TMR_SELECT_CHANNEL_4: Timer channel 4

oc_mode

Set output channel mode, that is, to configure channel original signals (CxORAW).

TMR_OUTPUT_CONTROL_OFF: Disconnect channel output (CxOUT) from CxORAW
TMR_OUTPUT_CONTROL_HIGH: CxORAW high
TMR_OUTPUT_CONTROL_LOW: CxORAW low
TMR_OUTPUT_CONTROL_SWITCH: Switch CxORAW level
TMR_OUTPUT_CONTROL_FORCE_LOW: CxORAW forced low
TMR_OUTPUT_CONTROL_FORCE_HIGH: CxORAW forced high
TMR_OUTPUT_CONTROL_PWM_MODE_A: PWM mode A
TMR_OUTPUT_CONTROL_PWM_MODE_B: PWM mode B

Example:

```
tmr_output_channel_mode_select(TMR1, TMR_SELECT_CHANNEL_1, TMR_OUTPUT_CONTROL_SWITCH);
```

5.19.16 tmr_period_value_set function

The table below describes the function tmr_period_value_set.

Table 407. tmr_period_value_set function

Name	Description
Function name	tmr_period_value_set
Function prototype	void tmr_period_value_set(tmr_type *tmr_x, uint32_t tmr_pr_value);
Function description	Set TMR period value
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_pr_value: timer period value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-bit timer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_period_value_set(TMR1, 0xFFFF);
```

5.19.17 tmr_period_value_get function

The table below describes the function tmr_period_value_get.

Table 408. tmr_period_value_get function

Name	Description
Function name	tmr_period_value_get
Function prototype	uint32_t tmr_period_value_get(tmr_type *tmr_x);
Function description	Get TMR period value
Input parameter	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Output parameter	NA
Return value	Timer period value
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t pr_value;
pr_value = tmr_period_value_get(TMR1);
```

5.19.18 tmr_channel_value_set function

The table below describes the function tmr_channel_value_set.

Table 409. tmr_channel_value_set function

Name	Description
Function name	tmr_channel_value_set
Function prototype	void tmr_channel_value_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, uint32_t tmr_channel_value);
Function description	Set TMR channel value
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	tmr_channel_value: timer channel value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-bit timer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 0xFFFF);
```

5.19.19 tmr_channel_value_get function

The table below describes the function tmr_channel_value_get.

Table 410. tmr_channel_value_get function

Name	Description
Function name	tmr_channel_value_get
Function prototype	uint32_t tmr_channel_value_get(tmr_type *tmr_x, tmr_channel_select_type tmr_channel);
Function description	Get TMR channel value
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Output parameter	Timer channel value
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
uint32_t ch_value;
ch_value = tmr_channel_value_get(TMR1, TMR_SELECT_CHANNEL_1);
```

5.19.20 tmr_period_buffer_enable function

The table below describes the function tmr_period_buffer_enable.

Table 411. tmr_period_buffer_enable function

Name	Description
Function name	tmr_period_buffer_enable
Function prototype	void tmr_period_buffer_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR period buffer
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	new_state: indicates the status of period buffer. It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_period_buffer_enable(TMR1, TRUE);
```

5.19.21 tmr_output_channel_buffer_enable function

The table below describes the function tmr_output_channel_buffer_enable.

Table 412. tmr_output_channel_buffer_enable function

Name	Description
Function name	tmr_output_channel_buffer_enable
Function prototype	void tmr_output_channel_buffer_enable(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable or disable TMR output channel buffer
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of output channel buffer; enable (TRUE)or disable (FALSE)
Output parameter	NA
Return value	NA

Name	Description
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_output_channel_buffer_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.19.22 tmr_output_channel_immediately_set function

The table below describes the function tmr_output_channel_immediately_set.

Table 413. tmr_output_channel_immediately_set function

Name	Description
Function name	tmr_output_channel_immediately_set
Function prototype	void tmr_output_channel_immediately_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable TMR output channel immediately
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of output channel enable. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_output_channel_immediately_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.19.23 tmr_output_channel_switch_set function

The table below describes the function tmr_output_channel_switch_set.

Table 414. tmr_output_channel_switch_set function

Name	Description
Function name	tmr_output_channel_switch_set
Function prototype	void tmr_output_channel_switch_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Set TMR output channel switch
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of output channel switch. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_output_channel_switch_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.19.24 tmr_one_cycle_mode_enable function

The table below describes the function tmr_one_cycle_mode_enable.

Table 415. tmr_one_cycle_mode_enable function

Name	Description
Function name	tmr_one_cycle_mode_enable
Function prototype	void tmr_one_cycle_mode_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR one-cycle mode
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	new_state: indicates the status of one-cycle mode. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_one_cycle_mode_enable(TMR1, TRUE);
```

5.19.25 tmr_32_bit_function_enable function

The table below describes the function tmr_32_bit_function_enable.

Table 416. tmr_32_bit_function_enable function

Name	Description
Function name	tmr_32_bit_function_enable
Function prototype	void tmr_32_bit_function_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR 32-bit feature (plus mode)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR2 or TMR5.
Input parameter 2	new_state: the status of 32-bit mode This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_32_bit_function_enable(TMR2, TRUE);
```

5.19.26 tmr_overflow_request_source_set function

The table below describes the function tmr_overflow_request_source_set.

Table 417. tmr_overflow_request_source_set function

Name	Description
Function name	tmr_overflow_request_source_set
Function prototype	void tmr_overflow_request_source_set(tmr_type *tmr_x, confirm_state new_state);
Function description	Select TMR overflow event sources
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	new_state: indicates the overflow event source
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Select an overflow event source

FALSE: Counter overflow, OVFSWTR being set, overflow event from slave mode timer controller

TRUE: Counter overflow only

Example:

```
tmr_overflow_request_source_set(TMR1, TRUE);
```

5.19.27 tmr_overflow_event_disable function

The table below describes the function tmr_overflow_event_disable.

Table 418. tmr_overflow_event_disable function

Name	Description
Function name	tmr_overflow_event_disable
Function prototype	void tmr_overflow_event_disable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR overflow event generation
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	new_state: indicates the status of overflow event generation
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Select the status of overflow event generation

FALSE: Enable overflow event generation, which can be generated from the following:

- Counter overflow
- Set OVFSWTR=1
- Overflow event from slave mode timer controller

TRUE: Disable overflow event generation

Example:

```
tmr_overflow_event_disable(TMR1, TRUE);
```

5.19.28 tmr_input_channel_init function

The table below describes the function tmr_input_channel_init.

Table 419. tmr_input_channel_init function

Name	Description
Function name	tmr_input_channel_init
Function prototype	void tmr_input_channel_init(tmr_type *tmr_x, tmr_input_config_type *input_struct, tmr_channel_input_divider_type divider_factor);
Function description	Initialize TMR input channels
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	input_struct: tmr_input_config_type pointer
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_input_config_type structure

tmr_input_config_type is defined in the at32f413_tmr.h.

```
typedef struct
{
    tmr_channel_select_type              input_channel_select;
    tmr_input_polarity_type              input_polarity_select;
    tmr_input_direction_mapped_type      input_mapped_select;
    uint8_t                              input_filter_value;
} tmr_input_config_type;
```

input_channel_select

Select a TMR input channel

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

input_polarity_select

Select the polarity of input channels

TMR_INPUT_RISING_EDGE: Rising edge

TMR_INPUT_FALLING_EDGE: Falling edge

TMR_INPUT_BOTH_EDGE: Both edges (Rising edge and Falling edge)

input_mapped_select

Select input channel mapping

TMR_CC_CHANNEL_MAPPED_DIRECT:

TMR input channel 1,2,3 and 4 are linked to C1IRAW, C2IRAW, C3IRAW and C4IRAW respectively.

TMR_CC_CHANNEL_MAPPED_INDIRECT:

TMR input channel 1,2,3 and 4 are linked to C2IRAW, C1IRAW, C4IRAW and C3IRAW respectively.

TMR_CC_CHANNEL_MAPPED_STI:

TMR input channel is mapped on STI

input_filter_value

Select an input channel filter value, ranging from 0x00 to 0x0F

divider_factor

Select input channel frequency division factor

TMR_CHANNEL_INPUT_DIV_1: Divided by 1

TMR_CHANNEL_INPUT_DIV_2: Divided by 2

TMR_CHANNEL_INPUT_DIV_4: Divided by 4

TMR_CHANNEL_INPUT_DIV_8: Divided by 8

Example:

```
tmr_input_config_type tmr_input_config_struct;
tmr_input_config_struct.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_input_config_struct.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_input_config_struct.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_input_config_struct.input_filter_value = 0x00;
tmr_input_channel_init(TMR1, &tmr_input_config_struct, TMR_CHANNEL_INPUT_DIV_1);
```

5.19.29 tmr_channel_enable function

The table below describes the function tmr_channel_enable.

Table 420. tmr_channel_enable function

Name	Description
Function name	tmr_channel_enable
Function prototype	void tmr_channel_enable(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable or disable TMR channels
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of timer channel. It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Set TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1
TMR_SELECT_CHANNEL_1C: Complementary channel 1
TMR_SELECT_CHANNEL_2: Timer channel 2
TMR_SELECT_CHANNEL_2C: Complementary channel 2
TMR_SELECT_CHANNEL_3: Timer channel 3
TMR_SELECT_CHANNEL_3C: Complementary channel 3
TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_channel_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.19.30 tmr_input_channel_filter_set function

The table below describes the function tmr_input_channel_filter_set.

Table 421. tmr_input_channel_filter_set function

Name	Description
Function name	tmr_input_channel_filter_set
Function prototype	void tmr_input_channel_filter_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, uint16_t filter_value);
Function description	Set TMR input channel filter
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	filter_value: set channel filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Set TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1
TMR_SELECT_CHANNEL_2: Timer channel 2
TMR_SELECT_CHANNEL_3: Timer channel 3
TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_input_channel_filter_set(TMR1, TMR_SELECT_CHANNEL_1, 0x0F);
```

5.19.31 tmr_pwm_input_config function

The table below describes the function tmr_pwm_input_config.

Table 422. tmr_pwm_input_config function

Name	Description
Function name	tmr_pwm_input_config
Function prototype	void tmr_pwm_input_config(tmr_type *tmr_x, tmr_input_config_type *input_struct, tmr_channel_input_divider_type divider_factor);
Function description	Configure TMR pwm input
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	input_struct: tmr_input_config_type pointer
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

input_struct

tmr_input_config_type pointer; refer to

tmr_input_config_type for details.

divider_factor

Select input channel frequency division factor.

TMR_CHANNEL_INPUT_DIV_1: Divided by 1

TMR_CHANNEL_INPUT_DIV_2: Divided by 2

TMR_CHANNEL_INPUT_DIV_4: Divided by 4

TMR_CHANNEL_INPUT_DIV_8: Divided by 8

Example:

```
tmr_input_config_type tmr_ic_init_structure;
tmr_ic_init_structure.input_filter_value = 0;
tmr_ic_init_structure.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_ic_init_structure.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_ic_init_structure.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_pwm_input_config(TMR1, &tmr_ic_init_structure, TMR_CHANNEL_INPUT_DIV_1);
```

5.19.32 tmr_channel1_input_select function

The table below describes the function tmr_channel1_input_select.

Table 423. tmr_channel1_input_select function

Name	Description
Function name	tmr_channel1_input_select
Function prototype	void tmr_channel1_input_select(tmr_type *tmr_x, tmr_channel1_input_connected_type ch1_connect);
Function description	Select TMR channel 1 input
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5 or TMR8.
Input parameter 2	ch1_connect: channel 1 input selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

ch1_connect

Select channel 1 input

TMR_CHANEL1_CONNECTED_C1IRAW: CH1 pin is connected to C1IRAW

TMR_CHANEL1_2_3_CONNECTED_C1IRAW_XOR: Connect XOR results CH1, CH2 and CH3 pins to C1IRAW

Example:

```
tmr_channel1_input_select(TMR1, TMR_CHANEL1_2_3_CONNECTED_C1IRAW_XOR);
```

5.19.33 tmr_input_channel_divider_set function

The table below describes the function tmr_input_channel_divider_set.

Table 424. tmr_input_channel_divider_set function

Name	Description
Function name	tmr_input_channel_divider_set
Function prototype	void tmr_input_channel_divider_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_channel_input_divider_type divider_factor);
Function description	Set TMR input channel divider
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1
TMR_SELECT_CHANNEL_2: Timer channel 2
TMR_SELECT_CHANNEL_3: Timer channel 3
TMR_SELECT_CHANNEL_4: Timer channel 4

divider_factor

Select input channel frequency division factor.

TMR_CHANNEL_INPUT_DIV_1: Divided by 1
TMR_CHANNEL_INPUT_DIV_2: Divided by 2
TMR_CHANNEL_INPUT_DIV_4: Divided by 4
TMR_CHANNEL_INPUT_DIV_8: Divided by 8

Example:

```
tmr_input_channel_divider_set(TMR1, TMR_SELECT_CHANNEL_1, TMR_CHANNEL_INPUT_DIV_2);
```

5.19.34 tmr_primary_mode_select function

The table below describes the function tmr_primary_mode_select.

Table 425. tmr_primary_mode_select function

Name	Description
Function name	tmr_primary_mode_select
Function prototype	void tmr_primary_mode_select(tmr_type *tmr_x, tmr_primary_select_type primary_mode);
Function description	Select TMR primary (master) mode
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5 or TMR8.
Input parameter 2	primary_mode: master mode
Output parameter	NA
Return value	NA
Required preconditions	NA

Name	Description
Called functions	NA

primary_mode

Select primary mode, that is, master timer output signal selection.

TMR_PRIMARY_SEL_RESET: Reset
TMR_PRIMARY_SEL_ENABLE: Enable
TMR_PRIMARY_SEL_OVERFLOW: Overflow
TMR_PRIMARY_SEL_COMPARE: Compare pulse
TMR_PRIMARY_SEL_C1ORAW: C1ORAW
TMR_PRIMARY_SEL_C2ORAW: C2ORAW
TMR_PRIMARY_SEL_C3ORAW: C3ORAW
TMR_PRIMARY_SEL_C4ORAW: C4ORAW

Example:

```
tmr_primary_mode_select(TMR1, TMR_PRIMARY_SEL_RESET);
```

5.19.35 tmr_sub_mode_select function

The table below describes the function tmr_sub_mode_select.

Table 426. tmr_sub_mode_select function

Name	Description
Function name	tmr_sub_mode_select
Function prototype	void tmr_sub_mode_select(tmr_type *tmr_x, tmr_sub_mode_select_type sub_mode);
Function description	Select TMR slave timer mode
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR9.
Input parameter 2	sub_mode: slave timer mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

primary_mode

Select slave timer modes.

TMR_SUB_MODE_DIABLE: Disable
TMR_SUB_ENCODER_MODE_A: Encoder mode A
TMR_SUB_ENCODER_MODE_B: Encoder mode B
TMR_SUB_ENCODER_MODE_C: Encoder mode C
TMR_SUB_RESET_MODE: Reset
TMR_SUB_HANG_MODE: Suspend
TMR_SUB_TRIGGER_MODE: Trigger
TMR_SUB_EXTERNAL_CLOCK_MODE_A: External clock A

Example:

```
tmr_sub_mode_select(TMR1, TMR_SUB_HANG_MODE);
```

5.19.36 tmr_channel_dma_select function

The table below describes the function tmr_channel_dma_select.

Table 427. tmr_channel_dma_select function

Name	Description
Function name	tmr_channel_dma_select
Function prototype	void tmr_channel_dma_select(tmr_type *tmr_x, tmr_dma_request_source_type cc_dma_select);
Function description	Select TMR channel DMA request source
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR9.
Input parameter 2	cc_dma_select: TMR channel DMA request source
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

cc_dma_select

Select DMA request source for TMR channels.

TMR_DMA_REQUEST_BY_CHANNEL: DMA request upon a channel event (CxIF = 1)

TMR_DMA_REQUEST_BY_OVERFLOW: DMA request upon an overflow event (OVFIF = 1)

Example:

```
tmr_channel_dma_select(TMR1, TMR_DMA_REQUEST_BY_OVERFLOW);
```

5.19.37 tmr_hall_select function

The table below describes the function tmr_hall_select.

Table 428. tmr_hall_select function

Name	Description
Function name	tmr_hall_select
Function prototype	void tmr_hall_select(tmr_type *tmr_x, confirm_state new_state);
Function description	Select TMR hall mode
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1 or TMR8.
Input parameter 2	new_state: indicates the status of TMR hall mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Select the status of TMR hall mode in order to refresh channel control bit.

FALSE: Refresh channel control bit through HALL

TRUE: Refresh channel control bit through HALL or the rising edge of TRGIN

Example:

```
tmr_hall_select(TMR1, TRUE);
```

5.19.38 tmr_channel_buffer_enable function

The table below describes the function tmr_channel_buffer_enable.

Table 429. tmr_channel_buffer_enable function

Name	Description
Function name	tmr_channel_buffer_enable
Function prototype	void tmr_channel_buffer_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR channel buffer
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1 or TMR8.
Input parameter 2	new_state: indicates the status of TMR channel buffer. It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_channel_buffer_enable(TMR1, TRUE);
```

5.19.39 tmr_trigger_input_select function

The table below describes the function tmr_trigger_input_select.

Table 430. tmr_trigger_input_select function

Name	Description
Function name	tmr_trigger_input_select
Function prototype	void tmr_trigger_input_select(tmr_type *tmr_x, sub_tmr_input_sel_type trigger_select);
Function description	Select TMR slave timer trigger input
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR9.
Input parameter 2	trigger_select: TMR slave timer trigger input
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

trigger_select

Select TMR slave timer trigger input.

TMR_SUB_INPUT_SEL_IS0: Internal input 0
TMR_SUB_INPUT_SEL_IS1: Internal input 1
TMR_SUB_INPUT_SEL_IS2: Internal input 2
TMR_SUB_INPUT_SEL_IS3: Internal input 3
TMR_SUB_INPUT_SEL_C1INC: C1IRAW input detection
TMR_SUB_INPUT_SEL_C1DF1: Filter input channel 1
TMR_SUB_INPUT_SEL_C2DF2: Filter input channel 2
TMR_SUB_INPUT_SEL_EXTIN: External input channel EXT

Example:

```
tmr_trigger_input_select(TMR1, TMR_SUB_INPUT_SEL_IS0);
```

5.19.40 tmr_sub_sync_mode_set function

The table below describes the function tmr_sub_sync_mode_set.

Table 431. tmr_sub_sync_mode_set function

Name	Description
Function name	tmr_sub_sync_mode_set
Function prototype	void tmr_sub_sync_mode_set(tmr_type *tmr_x, confirm_state new_state);
Function description	Set TMR slave timer synchronization mode
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR9.
Input parameter 2	new_state: indicates the status of TMR slave timer synchronization mode It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_sub_sync_mode_set(TMR1, TRUE);
```

5.19.41 tmr_dma_request_enable function

The table below describes the function tmr_dma_request_enable.

Table 432. tmr_dma_request_enable function

Name	Description
Function name	tmr_dma_request_enable
Function prototype	void tmr_dma_request_enable(tmr_type *tmr_x, tmr_dma_request_type dma_request, confirm_state new_state);
Function description	Enable or disable TMR DMA request
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	dma_request: DMA request
Input parameter 3	new_state: indicates the status of DMA request It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_request

Select a DMA request.

TMR_OVERFLOW_DMA_REQUEST: Overflow event DMA request

TMR_C1_DMA_REQUEST: Channel 1 DMA request

TMR_C2_DMA_REQUEST: Channel 2 DMA request
TMR_C3_DMA_REQUEST: Channel 3 DMA request
TMR_C4_DMA_REQUEST: Channel 4 DMA request
TMR_HALL_DMA_REQUEST: HALL event DMA request
TMR_TRIGGER_DMA_REQUEST: Trigger event DMA request

Example:

```
tmr_dma_request_enable(TMR1, TMR_OVERFLOW_DMA_REQUEST, TRUE);
```

5.19.42 tmr_interrupt_enable function

The table below describes the function tmr_interrupt_enable.

Table 433. tmr_interrupt_enable function

Name	Description
Function name	tmr_interrupt_enable
Function prototype	void tmr_interrupt_enable(tmr_type *tmr_x, uint32_t tmr_interrupt, confirm_state new_state);
Function description	Enable or disable TMR interrupts
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_interrupt: TMR interrupts
Input parameter 3	new_state: indicates the status of TMR interrupts It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_interrupt

Select a TMR interrupt

TMR_OVF_INT: Overflow event interrupt
TMR_C1_INT: Channel 1 event interrupt
TMR_C2_INT: Channel 2 event interrupt
TMR_C3_INT: Channel 3 event interrupt
TMR_C4_INT: Channel 4 event interrupt
TMR_HALL_INT: HALL event interrupt
TMR_TRIGGER_INT: Trigger event interrupt
TMR_BRK_INT: Break event interrupt

Example:

```
tmr_interrupt_enable(TMR1, TMR_OVF_INT, TRUE);
```

5.19.43 tmr_interrupt_flag_get function

The table below describes the function tmr_interrupt_flag_get.

Table 434. tmr_interrupt_flag_get function

Name	Description
Function name	tmr_interrupt_flag_get
Function prototype	flag_status tmr_interrupt_flag_get (tmr_type *tmr_x, uint32_t tmr_flag);
Function description	Get interrupt flag status
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_flag: flag selection Refer to the “tmr_flag” description for details .
Output parameter	NA
Return value	flag_status: status of flag Return SET or RESET.
Required preconditions	NA
Called functions	NA

tmr_flag

This is used for flag selection, including

TMR_OVF_FLAG: Overflow interrupt flag
TMR_C1_FLAG: Channel 1 interrupt flag
TMR_C2_FLAG: Channel 2 interrupt flag
TMR_C3_FLAG: Channel 3 interrupt flag
TMR_C4_FLAG: Channel 4 interrupt flag
TMR_HALL_FLAG: HALL interrupt flag
TMR_TRIGGER_FLAG: Trigger interrupt flag
TMR_BRK_FLAG: Break interrupt flag

Example

```
if(tmr_interrupt_flag_get (TMR1, TMR_OVF_FLAG) != RESET)
```

5.19.44 tmr_flag_get function

The table below describes the function tmr_flag_get.

Table 435. tmr_flag_get function

Name	Description
Function name	tmr_flag_get
Function prototype	flag_status tmr_flag_get(tmr_type *tmr_x, uint32_t tmr_flag);
Function description	Get flag status
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_flag: flag selection Refer to the “tmr_flag” description for details .
Output parameter	NA
Return value	flag_status: status of flag Return SET or RESET.
Required preconditions	NA
Called functions	NA

tmr_flag

This is used for flag selection, including

TMR_OVF_FLAG: Overflow interrupt flag
TMR_C1_FLAG: Channel 1 interrupt flag
TMR_C2_FLAG: Channel 2 interrupt flag
TMR_C3_FLAG: Channel 3 interrupt flag
TMR_C4_FLAG: Channel 4 interrupt flag
TMR_HALL_FLAG: HALL interrupt flag
TMR_TRIGGER_FLAG: Trigger interrupt flag
TMR_BRK_FLAG: Break interrupt flag
TMR_C1_RECAPTURE_FLAG: Channel 1 recapture flag
TMR_C2_RECAPTURE_FLAG: Channel 2 recapture flag
TMR_C3_RECAPTURE_FLAG: Channel 3 recapture flag
TMR_C4_RECAPTURE_FLAG: Channel 4 recapture flag

Example:

```
if(tmr_flag_get(TMR1, TMR_OVF_FLAG) != RESET)
```

5.19.45 tmr_flag_clear function

The table below describes the function tmr_flag_clear.

Table 436. tmr_flag_clear function

Name	Description
Function name	tmr_flag_clear
Function prototype	void tmr_flag_clear(tmr_type *tmr_x, uint32_t tmr_flag);
Function description	Clear flag
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_flag: flag selection Refer to the “tmr_flag” description for details .
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_flag_clear(TMR1, TMR_OVF_FLAG);
```

5.19.46 tmr_event_sw_trigger function

The table below describes the function tmr_event_sw_trigger.

Table 437. tmr_event_sw_trigger function

Name	Description
Function name	tmr_event_sw_trigger
Function prototype	void tmr_event_sw_trigger(tmr_type *tmr_x, tmr_event_trigger_type tmr_event);
Function description	Software triggers TMR event
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_event: select a TMR event
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_event

Set TMR events triggered by software

TMR_OVERFLOW_SWTRIG: Overflow event
TMR_C1_SWTRIG: Channel 1 event
TMR_C2_SWTRIG: Channel 2 event
TMR_C3_SWTRIG: Channel 3 event
TMR_C4_SWTRIG: Channel 4 event
TMR_HALL_SWTRIG: HALL event
TMR_TRIGGER_SWTRIG: Trigger event

TMR_BRK_SWTRIG: Break event

Example:

```
tmr_event_sw_trigger(TMR1, TMR_OVERFLOW_SWTRIG);
```

5.19.47 tmr_output_enable function

The table below describes the function tmr_output_enable.

Table 438. tmr_output_enable function

Name	Description
Function name	tmr_output_enable
Function prototype	void tmr_output_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR output
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1 or TMR8.
Input parameter 2	new_state: TMR output status It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_output_enable(TMR1, TRUE);
```

5.19.48 tmr_internal_clock_set function

The table below describes the function tmr_internal_clock_set.

Table 439. tmr_internal_clock_set function

Name	Description
Function name	tmr_internal_clock_set
Function prototype	void tmr_internal_clock_set(tmr_type *tmr_x);
Function description	Set TMR internal clock
Input parameter	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR9.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_internal_clock_set(TMR1);
```

5.19.49 tmr_output_channel_polarity_set function

The table below describes the function tmr_output_channel_polarity_set.

Table 440. tmr_output_channel_polarity_set function

Name	Description
Function name	tmr_output_channel_polarity_set
Function prototype	void tmr_output_channel_polarity_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_polarity_active_type oc_polarity);
Function description	Set TMR output channel polarity
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	oc_polarity: output channel polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1
TMR_SELECT_CHANNEL_1C: Complementary channel 1
TMR_SELECT_CHANNEL_2: Timer channel 2
TMR_SELECT_CHANNEL_2C: Complementary channel 2
TMR_SELECT_CHANNEL_3: Timer channel 3
TMR_SELECT_CHANNEL_3C: Complementary channel 3
TMR_SELECT_CHANNEL_4: Timer channel 4

oc_polarity

Select TMR channel polarity.

TMR_POLARITY_ACTIVE_HIGH: Active high
TMR_POLARITY_ACTIVE_LOW: Active low

Example:

```
tmr_output_channel_polarity_set(TMR1, TMR_SELECT_CHANNEL_1, TMR_POLARITY_ACTIVE_HIGH);
```

5.19.50 tmr_external_clock_config function

The table below describes the function tmr_external_clock_config.

Table 441. tmr_external_clock_config function

Name	Description
Function name	tmr_external_clock_config
Function prototype	void tmr_external_clock_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5 or TMR8.
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

es_divide

Set TMR external signal frequency division factor

TMR_ES_FREQUENCY_DIV_1: Divided by 1

TMR_ES_FREQUENCY_DIV_2: Divided by 2

TMR_ES_FREQUENCY_DIV_4: Divided by 4

TMR_ES_FREQUENCY_DIV_8: Divided by 8

es_polarity

Select TMR external signal polarity

TMR_ES_POLARITY_NON_INVERTED: High or rising edge

TMR_ES_POLARITY_INVERTED: Low or falling edge

Example:

```
tmr_external_clock_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED, 0x0F);
```

5.19.51 tmr_external_clock_mode1_config function

The table below describes the function tmr_external_clock_mode1_config.

Table 442. tmr_external_clock_mode1_config function

Name	Description
Function name	tmr_external_clock_mode1_config
Function prototype	void tmr_external_clock_mode1_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock mode 1 (corresponding to external mode A in the reference manual)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5 or TMR8.
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

es_divide

Set TMR external signal frequency division factor; refer to *es_divide* for details.

es_polarity

Set TMR external signal polarity, refer to *es_polarity* for details.

Example:

```
tmr_external_clock_mode1_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED,
0x0F);
```

5.19.52 tmr_external_clock_mode2_config function

The table below describes the function tmr_external_clock_mode2_config.

Table 443. tmr_external_clock_mode2_config function

Name	Description
Function name	tmr_external_clock_mode2_config
Function prototype	void tmr_external_clock_mode2_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock mode 2 (corresponding to external mode B in the reference manual)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5 or TMR8.
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

es_divide

Set TMR external signal frequency division factor; refer to *es_divide* for details.

es_polarity

Set TMR external signal polarity; refer to *es_polarity* for details.

Example:

```
tmr_external_clock_mode2_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED, 0x0F);
```

5.19.53 tmr_encoder_mode_config function

The table below describes the function tmr_encoder_mode_config.

Table 444. tmr_encoder_mode_config function

Name	Description
Function name	tmr_encoder_mode_config
Function prototype	void tmr_encoder_mode_config(tmr_type *tmr_x, tmr_encoder_mode_type encoder_mode, tmr_input_polarity_type ic1_polarity, tmr_input_polarity_type ic2_polarity);
Function description	Configure TMR encoder mode
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5 or TMR8.
Input parameter 2	encoder_mode: encoder mode
Input parameter 3	ic1_polarity: input channel 1 polarity
Input parameter 4	Ic2_polarity: input channel 2 polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

encoder_mode

Select a TMR encoder mode

TMR_ENCODER_MODE_A: Encoder mode A

TMR_ENCODER_MODE_B: Encoder mode B

TMR_ENCODER_MODE_C: Encoder mode C

ic1_polarity

Select TMR input channel 1 polarity

TMR_INPUT_RISING_EDGE: Rising edge

TMR_INPUT_FALLING_EDGE: Falling edge

TMR_INPUT_BOTH_EDGE: Both edges (Rising edge and Falling edge)

ic2_polarity

Select TMR input channel 2 polarity

TMR_INPUT_RISING_EDGE: Rising edge

TMR_INPUT_FALLING_EDGE: Falling edge

TMR_INPUT_BOTH_EDGE: Both edges (Rising edge and Falling edge)

Example:

```
tmr_encoder_mode_config(TMR1, TMR_ENCODER_MODE_A, TMR_INPUT_RISING_EDGE,
TMR_INPUT_RISING_EDGE);
```

5.19.54 tmr_force_output_set function

The table below describes the function tmr_force_output_set.

Table 445. tmr_force_output_set function

Name	Description
Function name	tmr_force_output_set
Function prototype	void tmr_force_output_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_force_output_type force_output);
Function description	Set TMR forced output
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10 or TMR11.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	force_output: forced output level
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

force_output

Forced output level of output channels

TMR_FORCE_OUTPUT_HIGH: CxORAW forced high

TMR_FORCE_OUTPUT_LOW: CxORAW forced low

Example:

```
tmr_force_output_set(TMR1, TMR_SELECT_CHANNEL_1, TMR_FORCE_OUTPUT_HIGH);
```

5.19.55 tmr_dma_control_config function

The table below describes the function tmr_dma_control_config.

Table 446. tmr_dma_control_config function

Name	Description
Function name	tmr_dma_control_config
Function prototype	void tmr_dma_control_config(tmr_type *tmr_x, tmr_dma_transfer_length_type dma_length, tmr_dma_address_type dma_base_address);
Function description	Configure TMR DMA control
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5 or TMR8.
Input parameter 2	dma_length: DMA transfer length
Input parameter 3	dma_base_address: DMA transfer offset address
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_length

Set DAM transfer bytes, including

TMR_DMA_TRANSFER_1BYTE: 1 byte

TMR_DMA_TRANSFER_2BYTES: 2 bytes

TMR_DMA_TRANSFER_3BYTES: 3 bytes

…

TMR_DMA_TRANSFER_17BYTES: 17 bytes

TMR_DMA_TRANSFER_18BYTES: 18 bytes

dma_base_address

Set DMA transfer offset address, starting from TMR control register 1, including

TMR_CTRL1_ADDRESS

TMR_CTRL2_ADDRESS

TMR_STCTRL_ADDRESS

TMR_IDEN_ADDRESS

TMR_ISTS_ADDRESS

TMR_SWEVT_ADDRESS

TMR_CM1_ADDRESS

TMR_CM2_ADDRESS

TMR_CCTRL_ADDRESS

TMR_CVAL_ADDRESS

TMR_DIV_ADDRESS

TMR_PR_ADDRESS

TMR_RPR_ADDRESS

TMR_C1DT_ADDRESS

TMR_C2DT_ADDRESS

TMR_C3DT_ADDRESS

TMR_C4DT_ADDRESS

TMR_BRK_ADDRESS

TMR_DMACTRL_ADDRESS

Example:

```
tmr_dma_control_config(TMR1, TMR_DMA_TRANSFER_8BYTES, TMR_CTRL1_ADDRESS);
```

5.19.56 tmr_brkdt_config function

The table below describes the function tmr_brkdt_config.

Table 447. tmr_brkdt_config function

Name	Description
Function name	tmr_brkdt_config
Function prototype	void tmr_brkdt_config(tmr_type *tmr_x, tmr_brkdt_config_type *brkdt_struct);
Function description	Configure TMR break mode and dead time
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1 or TMR8.
Input parameter 2	brkdt_struct: tmr_brkdt_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_brkdt_config_type structure

tmr_brkdt_config_type is defined in the at32f413_tmr.h.

```
typedef struct
{
    uint8_t                     deadtime;
    tmr_brk_polarity_type       brk_polarity;
    tmr_wp_level_type           wp_level;
    confirm_state               auto_output_enable;
    confirm_state               fcsoen_state;
    confirm_state               fcsodis_state;
    confirm_state               brk_enable;
} tmr_brkdt_config_type;
```

deadtime

Set dead time, ranging from 0x00 to 0xFF

brk_polarity

Select break input polarity

TMR_BRK_INPUT_ACTIVE_LOW: Active low

TMR_BRK_INPUT_ACTIVE_HIGH: Active high

wp_level

Set write protection level

TMR_WP_OFF: Write protection OFF

TMR_WP_LEVEL_3:

Level 3 write protection, protecting the bits below

- TMRx_BRK: DTC, BRKEN, BRKV and AOEN
- TMRx_CTRL2: CxIOS and CxCIOS

TMR_WP_LEVEL_2:

Level 2 write protection, protecting the bits below in addition to level -3 protected bits:

- TMRx_CCTRL: CxP and CxCP

– TMRx_BRK: FCSODIS and FCSOEN

TMR_WP_LEVEL_1:

Level 1 write protection, protecting the bits below in addition to level -2 protected bits:

– TMRx_CMx: CxOCTRL and CxOBEN

auto_output_enable

Enable auto output, Enable (TRUE) or Disable (FALSE)

fcsoen_state

Indicates the frozen status when main output is ON. It is used to configure the status of complementary output channels when timer is OFF and output is enabled (OEN=1).

FALSE: Disable CxOUT/CxCOUT output

TRUE: Enable CxOUT/CxCOUT output, inactive level

fcsodis_state

Indicates the frozen status when main output is OFF. It is used to configure the status of complementary output channels when timer is OFF and output is disabled (OEN=0).

FALSE: Disable CxOUT/CxCOUT output

TRUE: Enable CxOUT/CxCOUT output, idle level

brk_enable

Enable break feature, Enable (TRUE) or Disable (FALSE).

Example:

```
tmr_brkdt_config_type tmr_brkdt_config_struct;
tmr_brkdt_config_struct.brk_enable = TRUE;
tmr_brkdt_config_struct.auto_output_enable = TRUE;
tmr_brkdt_config_struct.deadtime = 0;
tmr_brkdt_config_struct.fcsodis_state = TRUE;
tmr_brkdt_config_struct.fcsoen_state = TRUE;
tmr_brkdt_config_struct.brk_polarity = TMR_BRK_INPUT_ACTIVE_HIGH;
tmr_brkdt_config_struct.wp_level = TMR_WP_OFF;
tmr_brkdt_config(TMR1, &tmr_brkdt_config_struct);
```

5.20 Universal synchronous/asynchronous receiver/transmitter (USART)

USART register structure usart_type is defined in the "at32f413_usart.h".

```
/**
  * @brief type define usart register all
  */
typedef struct
{
    ...

} usart_type;
```

The table below gives a list of the USART registers.

Table 448. Summary of USART registers

Register	Description
sts	Status register
dt	Data register
baudr	Baud rate register
ctrl1	Control register 1
ctrl2	Control register 2
ctrl3	Control register 3
gdiv	Guard time and divider register

The table below gives a list of the USART library functions.

Table 449. Summary of USART library functions

Function name	Description
usart_reset	Reset USART peripheral registers
usart_init	Set baud rate, data bits and stop bits
usart_parity_selection_config	Parity selection
usart_enable	Enable USART peripherals
usart_transmitter_enable	Enable USART transmitter
usart_receiver_enable	Enable USART receiver
usart_clock_config	Set clock polarity and phases for synchronization
usart_clock_enable	Set clock output for synchronization
usart_interrupt_enable	Enable interrupts
usart_dma_transmitter_enable	Enable DMA transmitter
usart_dma_receiver_enable	Enable DMA receiver
usart_wakeup_id_set	Set wakeup ID
usart_wakeup_mode_set	Set wakeup mode
usart_receiver_mute_enable	Enable receiver mute mode
usart_break_bit_num_set	Set break frame length
usart_lin_mode_enable	Enable LIN mode
usart_data_transmit	Data transmit

usart_data_receive	Data receive
usart_break_send	Send break frame
usart_smartcard_guard_time_set	Set smartcard guard time
usart_irda_smartcard_division_set	Set infrared and smartcard division
usart_smartcard_mode_enable	Enable smartcard mode
usart_smartcard_nack_set	Enable smartcard NACK
usart_single_line_halfduplex_select	Enable single-wire half-duplex mode
usart_irda_mode_enable	Enable infrared mode
usart_irda_low_power_enable	Enable infrared low-power mode
usart_hardware_flow_control_set	Enable hardware flow control
usart_flag_get	Get flag
usart_flag_clear	Clear flag

5.20.1 usart_reset function

The table below describes the function usart_reset.

Table 450. usart_reset function

Name	Description
Function name	usart_reset
Function prototype	void usart_reset(usart_type* usart_x);
Function description	Reset USART peripheral registers
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset

Example:

```
/* reset usart1 */
usart_reset(USART1);
```

5.20.2 usart_init function

The table below describes the function usart_init.

Table 451. usart_init function

Name	Description
Function name	usart_init
Function prototype	void usart_init(usart_type* usart_x, uint32_t baud_rate, usart_data_bit_num_type data_bit, usart_stop_bit_num_type stop_bit);
Function description	Set baud rate, data bits and stop bits.
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	baud_rate: baud rate for serial interfaces
Input parameter 3	data_bit: data bit width for serial interfaces
Input parameter 4	stop_bit: stop bit width for serial interfaces

Name	Description
Output parameter	NA
Return value	NA
Required preconditions	This operation can be allowed only when external low-speed clock is disabled.
Called functions	NA

data_bit

Select data bit size for serial interface communication

USART_DATA_8BITS: 8-bit

USART_DATA_9BITS: 9-bit

stop_bit

Select stop bit size for serial interface communication

USART_STOP_1_BIT: 1-bit

USART_STOP_0_5_BIT: 0.5-bit

USART_STOP_2_BIT: 2-bit

USART_STOP_1_5_BIT: 1.5-bit

Example:

```
/* configure uart param */
usart_init(USART1, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
```

5.20.3 usart_parity_selection_config function

The table below describes the function usart_parity_selection_config.

Table 452. usart_parity_selection_config function

Name	Description
Function name	usart_parity_selection_config
Function prototype	void usart_parity_selection_config(usart_type* usart_x, usart_parity_selection_type parity);
Function description	Parity selection
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	parity: parity mode for serial interface communication
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

parity

Select parity mode for serial interface communication

USART_PARITY_NONE: No parity

USART_PARITY_EVEN: Even

USART_PARITY_ODD: Odd

Example:

```
/* config usart even parity */
usart_parity_selection_config(USART1, USART_PARITY_EVEN);
```

5.20.4 usart_enable function

The table below describes the function usart_enable.

Table 453. usart_enable function

Name	Description
Function name	usart_enable
Function prototype	void usart_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: Enable (TRUE) and disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable usart1 */
usart_enable(USART1, TRUE);
```

5.20.5 usart_transmitter_enable function

The table below describes the function usart_transmitter_enable.

Table 454. usart_transmitter_enable function

Name	Description
Function name	usart_transmitter_enable
Function prototype	void usart_transmitter_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART transmitter
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: Enable (TRUE) and disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable usart1 transmitter */
usart_transmitter_enable(USART1, TRUE);
```

5.20.6 usart_receiver_enable function

The table below describes the function usart_receiver_enable.

Table 455. usart_receiver_enable function

Name	Description
Function name	usart_receiver_enable
Function prototype	void usart_receiver_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART receiver
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: Enable (TRUE) and disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable usart1 receiver */
usart_ receiver _enable(USART1, TRUE);
```

5.20.7 usart_clock_config function

The table below describes the function usart_clock_config.

Table 456. usart_clock_config function

Name	Description
Function name	usart_clock_config
Function prototype	void usart_clock_config(usart_type* usart_x, usart_clock_polarity_type clk_pol, usart_clock_phase_type clk_pha, usart_lbcp_type clk_lb);
Function description	Configure clock polarity and phase for synchronization feature
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	clk_pol: clock polarity for synchronization
Input parameter 3	clk_pha: clock phase for synchronization
Input parameter 4	clk_lb: selects whether to output clock on the last bit (upper bit) of data sent through synchronization feature
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

clk_pol

Clock polarity selection

USART_CLOCK_POLARITY_LOW: Low

USART_CLOCK_POLARITY_HIGH: High

clk_pha

Clock phase selection

USART_CLOCK_PHASE_1EDGE: 1^{st} edge

USART_CLOCK_PHASE_2EDGE: 2^{nd} edge

clk_lb

Select whether to output clock on the last bit of data

USART_CLOCK_LAST_BIT_NONE: No clock output

USART_CLOCK_LAST_BIT_OUTPUT: Clock output

Example:

```
/* config synchronous mode */
usart_clock_config(USART1, USART_CLOCK_POLARITY_HIGH, USART_CLOCK_PHASE_2EDGE,
USART_CLOCK_LAST_BIT_OUTPUT);
```

5.20.8 usart_clock_enable function

The table below describes the function usart_clock_enable.

Table 457. usart_clock_enable function

Name	Description
Function name	usart_clock_enable
Function prototype	void usart_clock_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable clock output
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable clock */
usart_clock_enable(USART1, TRUE);
```

5.20.9 usart_interrupt_enable function

The table below describes the function usart_interrupt_enable.

Table 458. usart_interrupt_enable function

Name	Description
Function name	usart_interrupt_enable
Function prototype	void usart_interrupt_enable(usart_type* usart_x, uint32_t usart_int, confirm_state new_state);
Function description	Enable interrupts
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	usart_int: interrupt type
Input parameter 3	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

usart_int

Select a peripheral interrupt

USART_IDLE_INT: Bus idle

USART_RDBF_INT: Receive data buff full
USART_TDC_INT: Transmit data complete
USART_TDBE_INT: Transmit data buff empty
USART_PERR_INT: Parity error
USART_BF_INT: Break frame receive
USART_ERR_INT: Error interrupt
USART_CTSCF_INT: CTS (Clear To Send) change

Example:

```
/* enable usart1 transmit complete interrupt */
usart_interrupt_enable (USART1, USART_TDC_INT, TRUE);
```

5.20.10 usart_dma_transmitter_enable function

The table below describes the function usart_dma_transmitter_enable.

Table 459. usart_dma_transmitter_enable function

Name	Description
Function name	usart_dma_transmitter_enable
Function prototype	void usart_dma_transmitter_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable DMA transmitter
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable dma transmitter */
usart_dma_transmitter_enable (USART1, TRUE);
```

5.20.11 usart_dma_receiver_enable function

The table below describes the function usart_dma_receiver_enable.

Table 460. usart_dma_receiver_enable function

Name	Description
Function name	usart_dma_receiver_enable
Function prototype	void usart_dma_receiver_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable DMA receiver
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable dma receiver */
usart_dma_receiver_enable (USART1, TRUE);
```

5.20.12 usart_wakeup_id_set function

The table below describes the function usart_wakeup_id_set.

Table 461. usart_wakeup_id_set function

Name	Description
Function name	usart_wakeup_id_set
Function prototype	void usart_wakeup_id_set(usart_type* usart_x, uint8_t usart_id);
Function description	Set wakeup ID
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	usart_id: wakeup ID
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* config wakeup id */
usart_wakeup_id_set (USART1, 0x88);
```

5.20.13 usart_wakeup_mode_set function

The table below describes the function usart_wakeup_mode_set.

Table 462. usart_wakeup_mode_set function

Name	Description
Function name	usart_wakeup_mode_set
Function prototype	void usart_wakeup_mode_set(usart_type* usart_x, usart_wakeup_mode_type wakeup_mode);
Function description	Set wakeup mode
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	wakeup_mode: wakeup mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

wakeup_mode

Set wakeup mode to wake up from silent state.

USART_WAKEUP_BY_IDLE_FRAME: Woke up by idle frame

USART_WAKEUP_BY_MATCHING_ID: Woke up by ID matching

Example:

```
/* config usart1 wakeup mode */
usart_wakeup_mode_set (USART1, USART_WAKEUP_BY_MATCHING_ID);
```

5.20.14 usart_receiver_mute_enable function

The table below describes the function usart_receiver_mute_enable.

Table 463. usart_receiver_mute_enable function

Name	Description
Function name	usart_receiver_mute_enable
Function prototype	void usart_receiver_mute_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART receiver mute mode
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* config receiver mute */
usart_receiver_mute_enable (USART1, TRUE);
```

5.20.15 usart_break_bit_num_set function

The table below describes the function usart_break_bit_num_set.

Table 464. usart_break_bit_num_set function

Name	Description
Function name	usart_break_bit_num_set
Function prototype	void usart_break_bit_num_set(usart_type* usart_x, usart_break_bit_num_type break_bit);
Function description	Set USART break frame length
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	break_bit: break frame length type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

break_bit

Set break frame length.

USART_BREAK_10BITS: 10 bits

USART_BREAK_11BITS: 11 bits

Example:

```
/* config break frame length 10bits */
usart_break_bit_num_set (USART1, USART_BREAK_10BITS);
```

5.20.16 usart_lin_mode_enable function

The table below describes the function usart_lin_mode_enable.

Table 465. usart_lin_mode_enable function

Name	Description
Function name	usart_lin_mode_enable
Function prototype	void usart_lin_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable LIN mode
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable usart1 lin mode */
usart_lin_mode_enable (USART1, TRUE);
```

5.20.17 usart_data_transmit function

The table below describes the function usart_data_transmit.

Table 466. usart_data_transmit function

Name	Description
Function name	usart_data_transmit
Function prototype	void usart_data_transmit(usart_type* usart_x, uint16_t data);
Function description	Transmit data
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	data: data to send
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* transmit data */
uint16_t data = 0x88;
usart_data_transmit (USART1, data);
```

5.20.18 usart_data_receive function

The table below describes the function usart_data_receive.

Table 467. usart_data_receive function

Name	Description
Function name	usart_data_receive
Function prototype	uint16_t usart_data_receive(usart_type* usart_x);

Name	Description
Function description	Receive data
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	NA
Output parameter	NA
Return value	uint16_t: return the received data
Required preconditions	NA
Called functions	NA

Example:

```
/* receive data */
uint16_t data = 0;
data = usart_data_receive (USART1);
```

5.20.19 usart_break_send function

The table below describes the function usart_break_send.

Table 468. usart_break_send function

Name	Description
Function name	usart_break_send
Function prototype	void usart_break_send(usart_type* usart_x);
Function description	Send break frame
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* send break frame */
usart_break_send (USART1);
```

5.20.20 usart_smartcard_guard_time_set function

The table below describes the function usart_smartcard_guard_time_set.

Table 469. usart_smartcard_guard_time_set function

Name	Description
Function name	usart_smartcard_guard_time_set
Function prototype	void usart_smartcard_guard_time_set(usart_type* usart_x, uint8_t guard_time_val);
Function description	Set smartcard guard time
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	guard_time_val: guard time, 0x00~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA

Name	Description
Called functions	NA

Example:

```
/* usart guard time set to 2 bit */
usart_smartcard_guard_time_set(USART1, 0x2);
```

5.20.21 usart_irda_smartcard_division_set function

The table below describes the function usart_irda_smartcard_division_set.

Table 470. usart_irda_smartcard_division_set function

Name	Description
Function name	usart_irda_smartcard_division_set
Function prototype	void usart_irda_smartcard_division_set(usart_type* usart_x, uint8_t div_val);
Function description	Infrared and smartcard frequency division settings
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	div_val: division value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* usart clock set to (apbclk / (2 * 20)) */
usart_irda_smartcard_division_set(USART1, 20);
```

5.20.22 usart_smartcard_mode_enable function

The table below describes the function usart_smartcard_mode_enable.

Table 471. usart_smartcard_mode_enable function

Name	Description
Function name	usart_smartcard_mode_enable
Function prototype	void usart_smartcard_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable smartcode mode
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the smartcard mode */
usart_smartcard_mode_enable(USART1, TRUE);
```

5.20.23 usart_smartcard_nack_set function

The table below describes the function usart_smartcard_nack_set.

Table 472. usart_smartcard_nack_set function

Name	Description
Function name	usart_smartcard_nack_set
Function prototype	void usart_smartcard_nack_set(usart_type* usart_x, confirm_state new_state);
Function description	Enable smartcard NACK
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the nack transmission */
usart_smartcard_nack_set(USART1, TRUE);
```

5.20.24 usart_single_line_halfduplex_select function

The table below describes the function usart_single_line_halfduplex_select.

Table 473. usart_single_line_halfduplex_select function

Name	Description
Function name	usart_single_line_halfduplex_select
Function prototype	void usart_single_line_halfduplex_select(usart_type* usart_x, confirm_state new_state);
Function description	Enable single-wire half-duplex mode
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable halfduplex */
usart_single_line_halfduplex_select(USART1, TRUE);
```

5.20.25 usart_irda_mode_enable function

The table below describes the function usart_irda_mode_enable.

Table 474. usart_irda_mode_enable function

Name	Description
Function name	usart_irda_mode_enable
Function prototype	void usart_irda_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable infrared mode
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable irda mode */
usart_irda_mode_enable(USART1, TRUE);
```

5.20.26 usart_irda_low_power_enable function

The table below describes the function usart_irda_low_power_enable.

Table 475. usart_irda_low_power_enable function

Name	Description
Function name	usart_irda_low_power_enable
Function prototype	void usart_irda_low_power_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable infrared low-power mode
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable irda lowpower mode */
usart_irda_low_power_enable (USART1, TRUE);
```

5.20.27 usart_hardware_flow_control_set function

The table below describes the function usart_hardware_flow_control_set.

Table 476. usart_hardware_flow_control_set function

Name	Description
Function name	usart_hardware_flow_control_set
Function prototype	void usart_hardware_flow_control_set(usart_type* usart_x,usart_hardware_flow_control_type flow_state);
Function description	Set peripheral hardware flow control
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	flow_state: flow control type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flow_state

USART_HARDWARE_FLOW_NONE: No hardware flow control
USART_HARDWARE_FLOW_RTS: RTS
USART_HARDWARE_FLOW_CTS: CTS
USART_HARDWARE_FLOW_RTS_CTS: RTS and CTS

Example:

```
/* hardware flow set none */
usart_hardware_flow_control_set (USART1, USART_HARDWARE_FLOW_NONE);
```

5.20.28 usart_flag_get function

The table below describes the function usart_flag_get.

Table 477. usart_flag_get function

Name	Description
Function name	usart_flag_get
Function prototype	flag_status usart_flag_get(usart_type* usart_x, uint32_t flag);
Function description	Get flag status
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	flag: flag
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

flag

USART_CTSCF_FLAG: CTS (Clear To Send) change flag
USART_BFF_FLAG: Break frame receive flag
USART_TDBE_FLAG: Transmit buff empty flag
USART_TDC_FLAG: Transmit complete flag
USART_RDBF_FLAG: Receive data buff full flag
USART_IDLEF_FLAG: Idle frame flag

USART_ROERR_FLAG: Receive overflow flag
USART_NERR_FLAG: Noise error flag
USART_FERR_FLAG: Frame error flag
USART_PERR_FLAG: Parity error flag

Example:

```
/* wait data transmit complete flag */
while(usart_flag_get (USART1, USART_TDC_FLAG) == RESET);
```

5.20.29 usart_interrupt_flag_get function

The table below describes the function usart_interrupt_flag_get.

Table 478. usart_interrupt_flag_get function

Name	Description
Function name	usart_interrupt_flag_get
Function prototype	flag_status usart_interrupt_flag_get(usart_type* usart_x, uint32_t flag);
Function description	Check if the selected flag is set or not
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2…
Input parameter 2	flag: flag selection
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

flag

USART_CTSCF_FLAG: CTS (Clear To Send) change flag
USART_BFF_FLAG: Break frame receive flag
USART_TDBE_FLAG: Transmit buff empty flag
USART_TDC_FLAG: Transmit complete flag
USART_RDBF_FLAG: Receive data buff full flag
USART_IDLEF_FLAG: Idle frame flag
USART_ROERR_FLAG: Receive overflow flag
USART_NERR_FLAG: Noise error flag
USART_FERR_FLAG: Frame error flag
USART_PERR_FLAG: Parity error flag

Example

```
/* check received data flag */
if(usart_interrupt_flag_get(USART1, USART_RDBF_FLAG) != RESET)
{
}
```

5.20.30 usart_flag_clear function

The table below describes the function usart_flag_clear.

Table 479. usart_flag_clear function

Name	Description
Function name	usart_flag_clear
Function prototype	void usart_flag_clear(usart_type* usart_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	usart_x: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	flag: to-be-cleared flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

USART_CTSCF_FLAG: CTS (Clear To Send) change flag
USART_BFF_FLAG: Break frame receive flag
USART_TDC_FLAG: Transmit complete flag
USART_RDBF_FLAG: Receive data buff full flag

Example:

```
/* clear data transmit complete flag */
usart_flag_clear (USART1, USART_TDC_FLAG );
```

5.21 Watchdog timer (WDT)

The WDT register structure wdt_type is defined in the "at32f413_wdt.h".

```
/**
  * @brief type define wdt register all
  */
typedef struct
{


} wdt_type;
```

The table below gives a list of the WDT registers.

Table 480. Summary of WDT registers

Register	Description
cmd	Command register
div	Divider register
rld	Reload register
sts	Status register

The table below gives a list of the WDT library functions.

Table 481. Summary of WDT library functions

Function name	Description
wdt_enable	Enable watchdog
wdt_counter_reload	Reload counter
wdt_reload_value_set	Set reload value
wdt_divider_set	Set division value
wdt_register_write_enable	Unlock WDT_DIV and WDT_RLD register write protection
wdt_flag_get	Get flag

5.21.1 wdt_enable function

The table below describes the function wdt_enable.

Table 482. wdt_enable function

Name	Description
Function name	wdt_enable
Function prototype	void wdt_enable(void);
Function description	Enable watchdog
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_enable();
```

5.21.2 wdt_counter_reload function

The table below describes the function wdt_counter_reload.

Table 483. wdt_counter_reload function

Name	Description
Function name	wdt_counter_reload
Function prototype	void wdt_counter_reload(void);
Function description	Reload counter
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_counter_reload();
```

5.21.3 wdt_reload_value_set function

The table below describes the function wdt_reload_value_set.

Table 484. wdt_reload_value_set function

Name	Description
Function name	wdt_reload_value_set
Function prototype	void wdt_reload_value_set(uint16_t reload_value);
Function description	Set reload value
Input parameter 1	reload_value: reload value, 0x000~0xFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_reload_value_set(0xFFF);
```

5.21.4 wdt_divider_set function

The table below describes the function wdt_divider_set.

Table 485. wdt_divider_set function

Name	Description
Function name	wdt_divider_set
Function prototype	void wdt_divider_set(wdt_division_type division);
Function description	Set division value
Input parameter 1	division: watchdog division value Refer to the following “division” descriptions for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

division

Select watchdog division value

WDT_CLK_DIV_4: Divided by 4
WDT_CLK_DIV_8: Divided by 8
WDT_CLK_DIV_16: Divided by 16
WDT_CLK_DIV_32: Divided by 32
WDT_CLK_DIV_64: Divided by 64
WDT_CLK_DIV_128: Divided by 128
WDT_CLK_DIV_256: Divided by 256

Example:

```
wdt_divider_set(WDT_CLK_DIV_4);
```

5.21.5 wdt_register_write_enable function

The table below describes the function wdt_register_write_enable.

Table 486. wdt_register_write_enable function

Name	Description
Function name	wdt_register_write_enable
Function prototype	void wdt_register_write_enable(confirm_state new_state);
Function description	Unlock WDT_DIV and WDT_RLD write protection
Input parameter 1	new_state: unlock register write protection This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_register_write_enable(TRUE);
```

5.21.6 wdt_flag_get function

The table below describes the function wdt_flag_get.

Table 487. wdt_flag_get function

Name	Description
Function name	wdt_flag_get
Function prototype	flag_status wdt_flag_get(uint16_t wdt_flag);
Function description	Get flag status
Input parameter 1	flag: flag selection Refer to the “flag” des cription for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This is used for flag selection, including

WDT_DIVF_UPDATE_FLAG: Division value update complete

WDT_RLDF_UPDATE_FLAG: Reload value update complete

Example:

```
wdt_flag_get(WDT_DIVF_UPDATE_FLAG);
```

5.22 Window watchdog timer (WWDT)

The WWDT register structure wwdt_type is defined in the "at32f413_wwdt.h".

```
/**
  * @brief type define wwdt register all
  */
typedef struct
{


} wwdt_type;
```

The table below gives a list of the WWDT registers.

Table 488. Summary of WWDT registers

Register	Description
ctrl	Control register
cfg	Configuration register
sts	Status register

The table below gives a list of the WWDT library functions.

Table 489. Summary of WWDT library functions

Function name	Description
wwdt_reset	Reset window watchdog registers
wwdt_divider_set	Set divider
wwdt_flag_clear	Clear reload counter interrupt flag
wwdt_enable	Enable WWDT
wwdt_interrupt_enable	Enable reload counter interrupt
wwdt_flag_get	Get flag
wwdt_counter_set	Set counter value
wwdt_window_counter_set	Set window value

5.22.1 wwdt_reset function

The table below describes the function wwdt_reset.

Table 490. wwdt_reset function

Name	Description
Function name	wwdt_reset
Function prototype	void wwdt_reset(void);
Function description	Reset window watchdog registers to their initial values
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);

Example:

```
wwdt_reset();
```

5.22.2 wwdt_divider_set function

The table below describes the function wwdt_divider_set.

Table 491. wwdt_divider_set function

Name	Description
Function name	wwdt_divider_set
Function prototype	void wwdt_divider_set(wwdt_division_type division);
Function description	Set divider
Input parameter 1	division: WWDT division value Refer to the following "division" descriptions for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

division

Select WWDT division value

WWDT_PCLK1_DIV_4096: Divided by 4096

WWDT_PCLK1_DIV_8192: Divided by 8192

WWDT_PCLK1_DIV_16384: Divided by 16384

WWDT_PCLK1_DIV_32768: Divided by 32768

Example:

```
wwdt_divider_set(WWDT_PCLK1_DIV_4096);
```

5.22.3 wwdt_enable function

The table below describes the function wwdt_enable.

Table 492. wwdt_enable function

Name	Description
Function name	wwdt_enable
Function prototype	void wwdt_enable(uint8_t wwdt_cnt);
Function description	Enable WWDT
Input parameter 1	wwdt_cnt: WWDT counter initial value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_enable(0x7F);
```

5.22.4 wwdt_interrupt_enable function

The table below describes the function wwdt_interrupt_enable.

Table 493. wwdt_interrupt_enable function

Name	Description
Function name	wwdt_interrupt_enable
Function prototype	void wwdt_interrupt_enable(void);
Function description	Enable reload counter interrupt
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_interrupt_enable();
```

5.22.5 wwdt_counter_set function

The table below describes the function wwdt_counter_set.

Table 494. wwdt_counter_set function

Name	Description
Function name	wwdt_counter_set
Function prototype	void wwdt_counter_set(uint8_t wwdt_cnt);
Function description	Set counter value
Input parameter 1	wwdt_cnt: WWDT counter value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_counter_set(0x7F);
```

5.22.6 wwdt_window_counter_set function

The table below describes the function wwdt_window_counter_set.

Table 495. wwdt_window_counter_set function

Name	Description
Function name	wwdt_window_counter_set
Function prototype	void wwdt_window_counter_set(uint8_t window_cnt);
Function description	Set window value
Input parameter 1	wwdt_cnt: WWDT window value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA

Name	Description
Called functions	NA

Example:

```
wwdt_window_counter_set(0x6F);
```

5.22.7 wwdt_flag_get function

The table below describes the function wwdt_flag_get.

Table 496. wwdt_flag_get function

Name	Description
Function name	wwdt_flag_get
Function prototype	flag_status wwdt_flag_get(void);
Function description	Get reload counter interrupt flag
Input parameter 1	NA
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_flag_get();
```

5.22.8 wwdt_interrupt_flag_get function

The table below describes the function wwdt_interrupt_flag_get.

Table 497. wwdt_interrupt_flag_get function

Name	Description
Function name	wwdt_interrupt_flag_get
Function prototype	flag_status wwdt_ interrupt_flag_get(void);
Function description	Get reload counter interrupt flag and judge the corresponding interrupt enable bit
Input parameter 1	NA
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example

```
wwdt_ interrupt_flag_get();
```

5.22.9 wwdt_flag_clear function

The table below describes the function wwdt_flag_clear.

Table 498. wwdt_flag_clear function

Name	Description
Function name	wwdt_flag_clear
Function prototype	void wwdt_flag_clear(void);
Function description	Clear reload counter interrupt flag
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_flag_clear();
```

6 Precautions

6.1 Device model replacement

While replacing the device part number in an existing project or demo with another one, it is necessary to check the macro definitions corresponding to the device defined in Table 1 before replacement. The subsequent sections give a detailed description of how to replace a device in KEIL and IAR environments (Just taking the at32f403avgt7 as an example as other devices share similar operations).

There are two steps to get this happen:

1. By changing device
2. By changing macro definition

6.1.1 KEIL environment

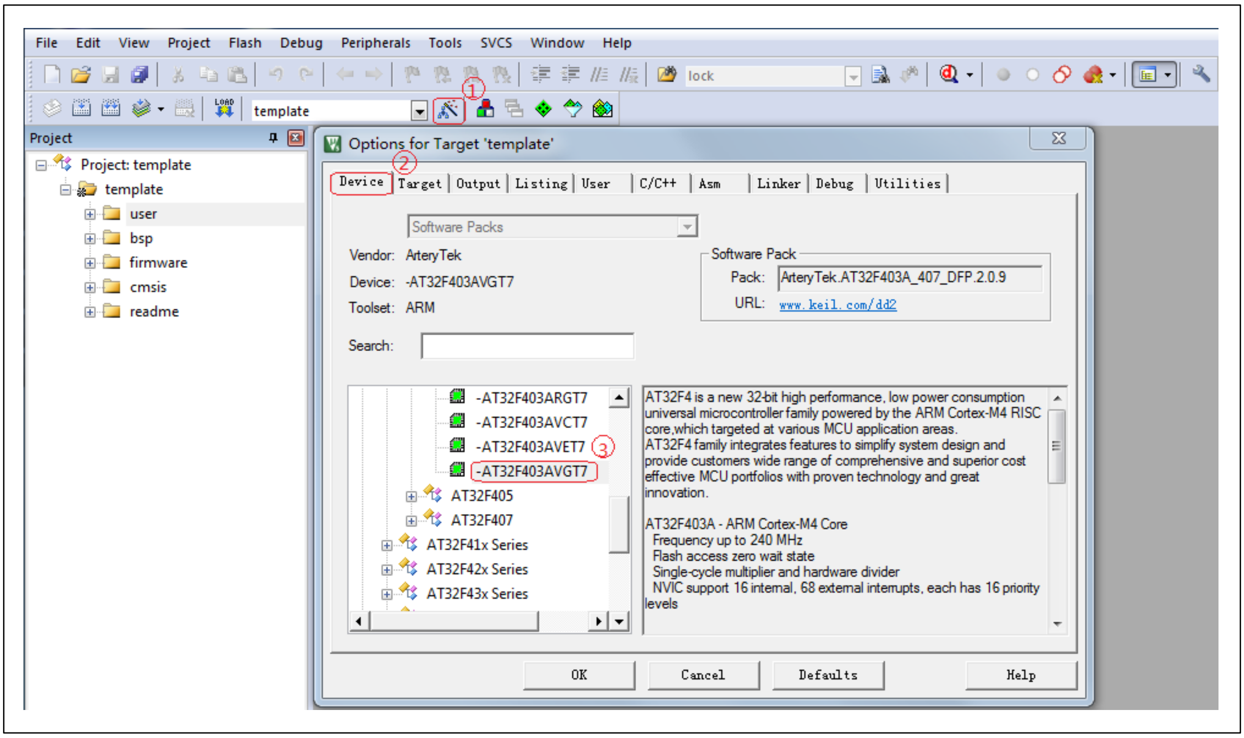
Follow the steps and illustration below for device replacement in Keil environment.

① Click on magic wand “Options for Target”

② Click on “Device”

③ Select the desired device part number

Figure 29. Change device part number in Keil



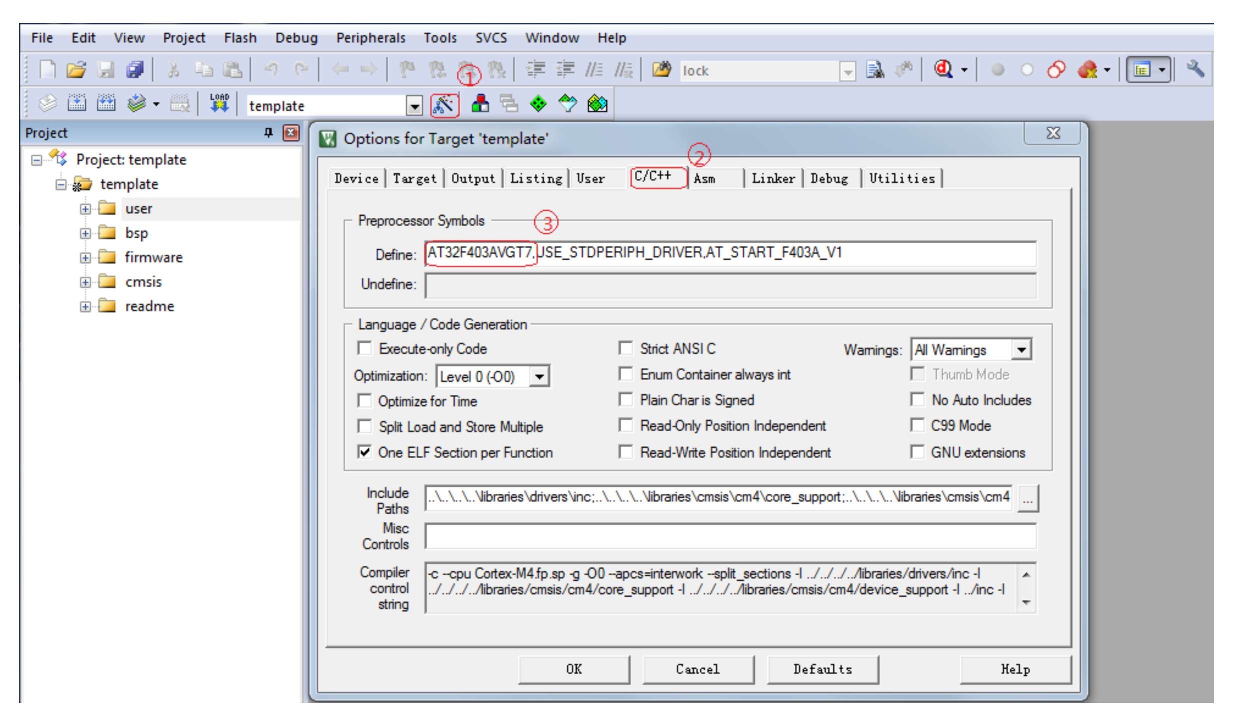
Follow the steps and illustration below to change macro definition.

① Click on magic wand “Options for Target”

② Click on “C/C++”

③ Delete the original macro definition in “Define” box, and write the desired one corresponding to the selected device part number based on Table 1.

Figure 30. Change macro definition in Keil



6.1.2 IAR environment

Follow the steps and illustration below for device replacement in IAR environment:

① Right click on the file name, and select “Options…”

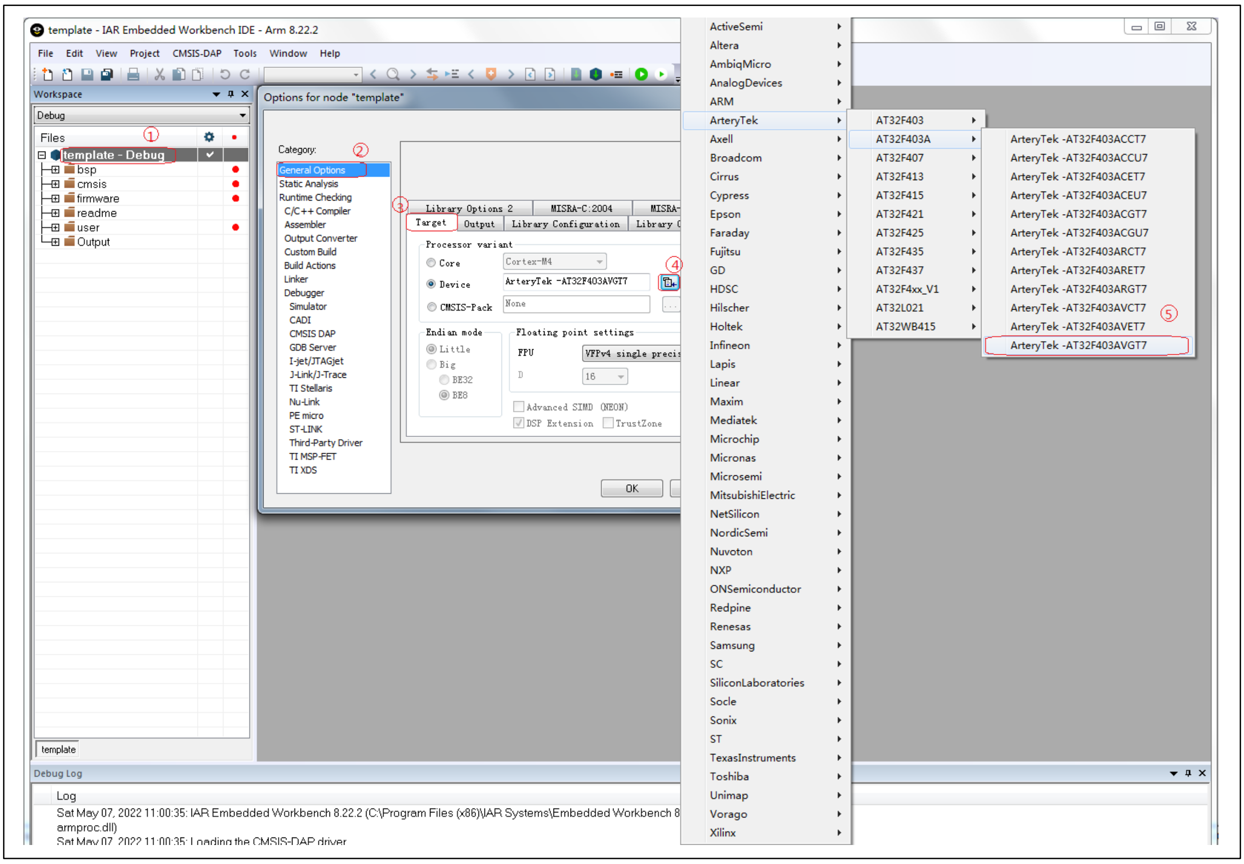
② Select “General Options”

③ Select “Target”

④ Click on the check box

⑤ Select the desired device part number

Figure 31. Change device part number in IAR



Follow the steps and illustration below to change macro definition in IAR environment:

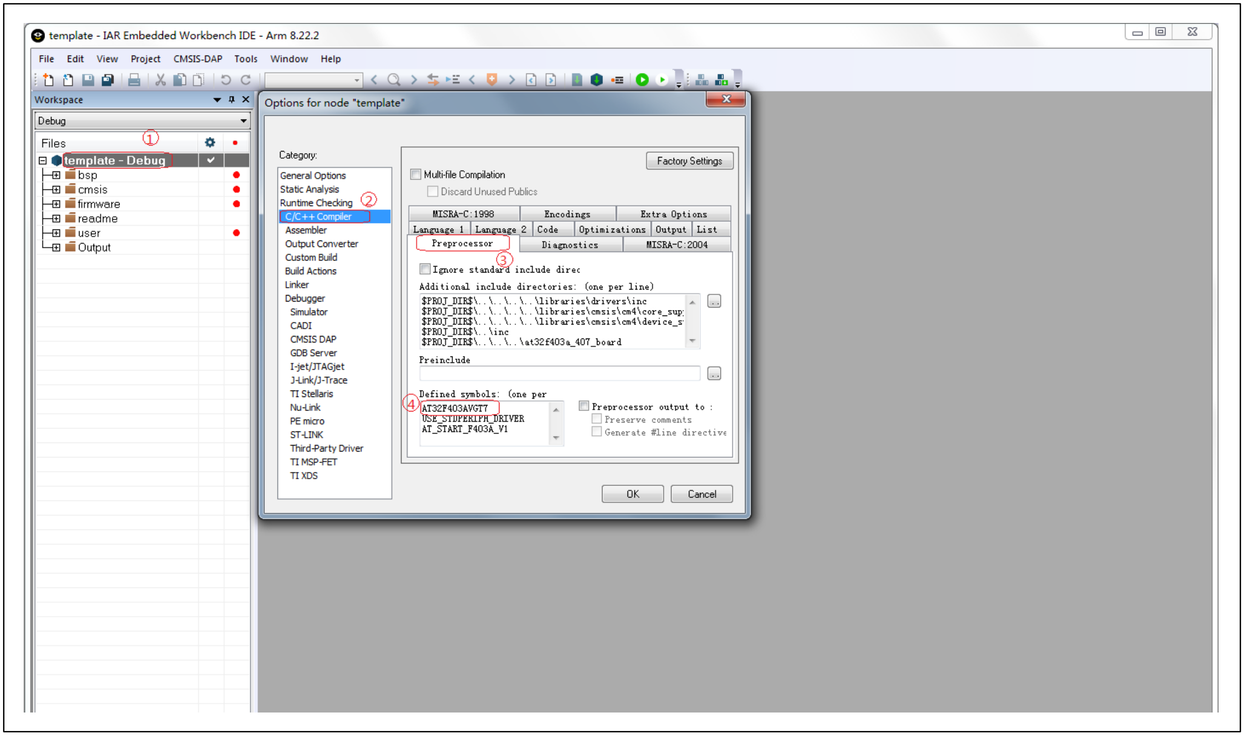
① Right click on the file name, and select “Options…”

② Select “C/C++ Compiler”

③ Click on “Preprocessor”

④ Delete the original macro definition in “Defined symbols” column, and write the desired one corresponding to the selected device part number based on Table 1.

Figure 32. Change macro definition in IAR



6.2 Unable to identify IC by JLink software in Keil

In special circumstances, the Keil project compiled by an engineer is unknown to the J-Link software even if it can be compiled by other engineers and identified by ICP software. For example, some warnings like below will be displayed.

Figure 33. Error warning 1

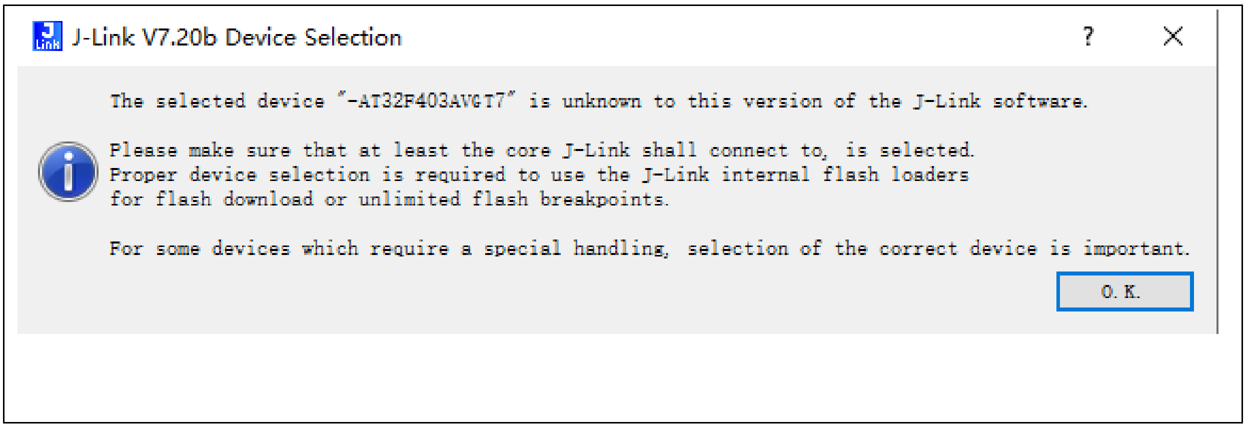


Figure 34. Error warning 2

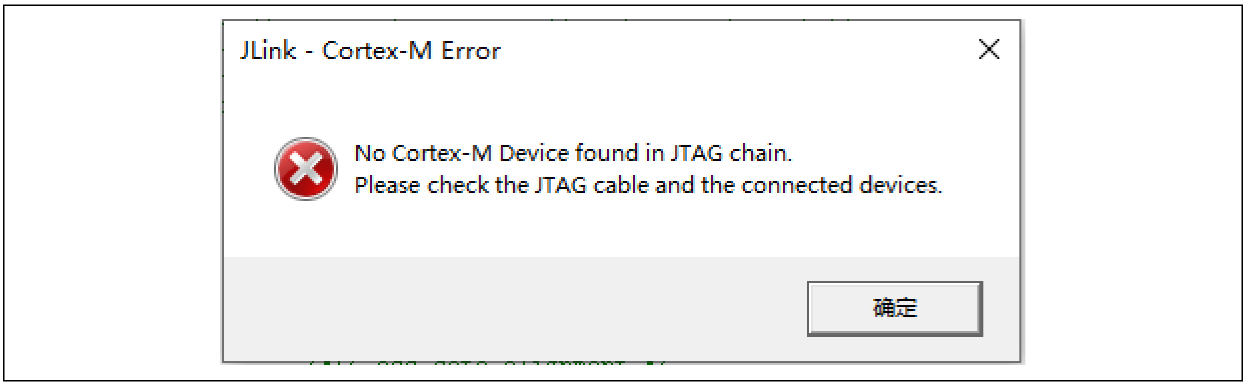
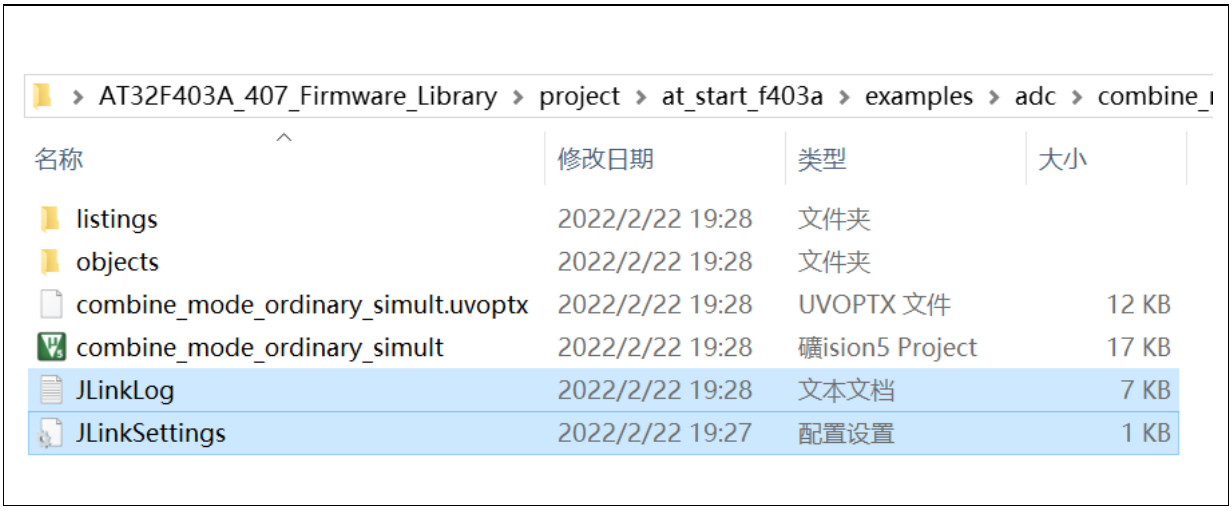


Figure 35. Error warning 3

How to solve this problem?

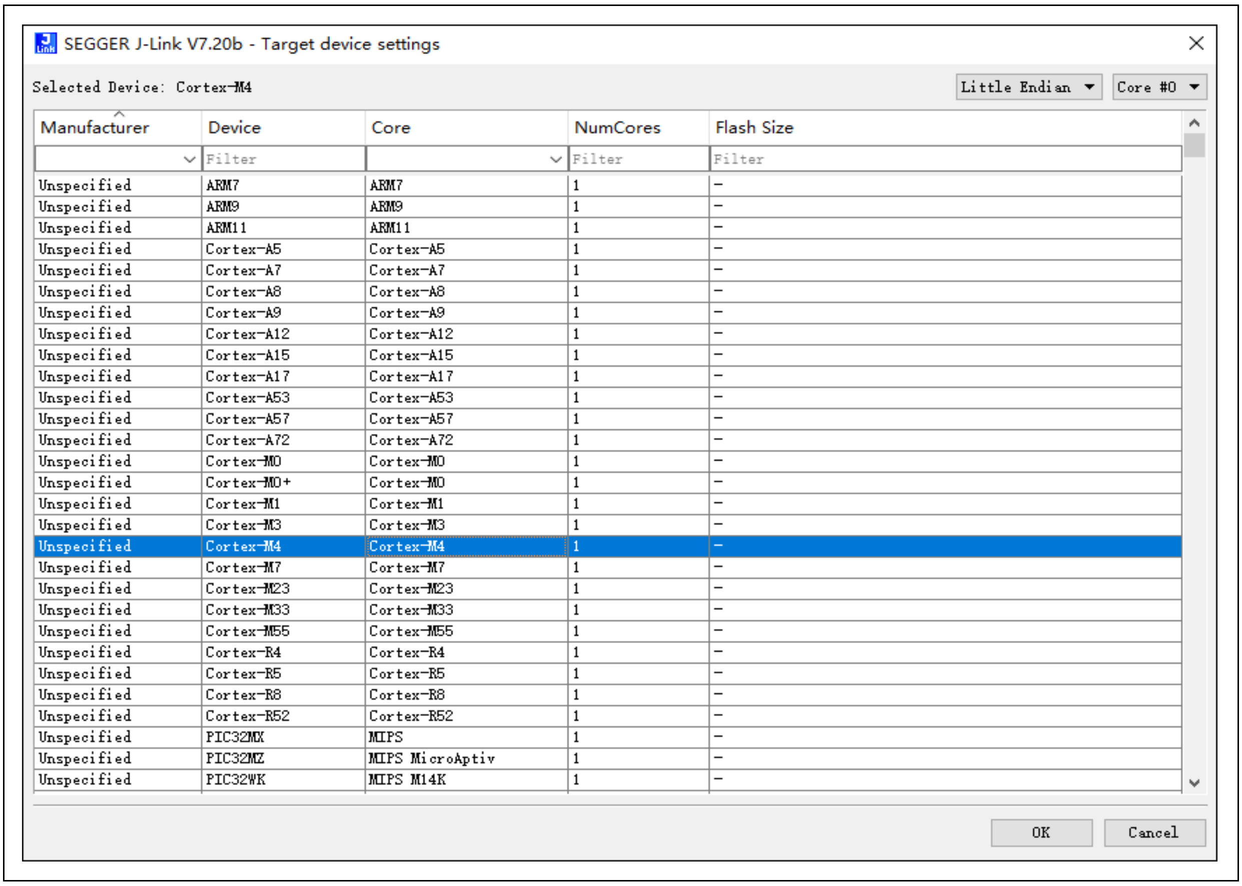
Step 1: Find "JLinkLog" and "JLinkSettings" files according to project path, and delete them.

Figure 36. JLinkLog and JLinkSettings



Step 2: Click on magic wand, go to "Debug", select "Unspecified Cortex–M4".

Figure 37. Unspecified Cortex–M4



6.3 How to change HEXT crystal

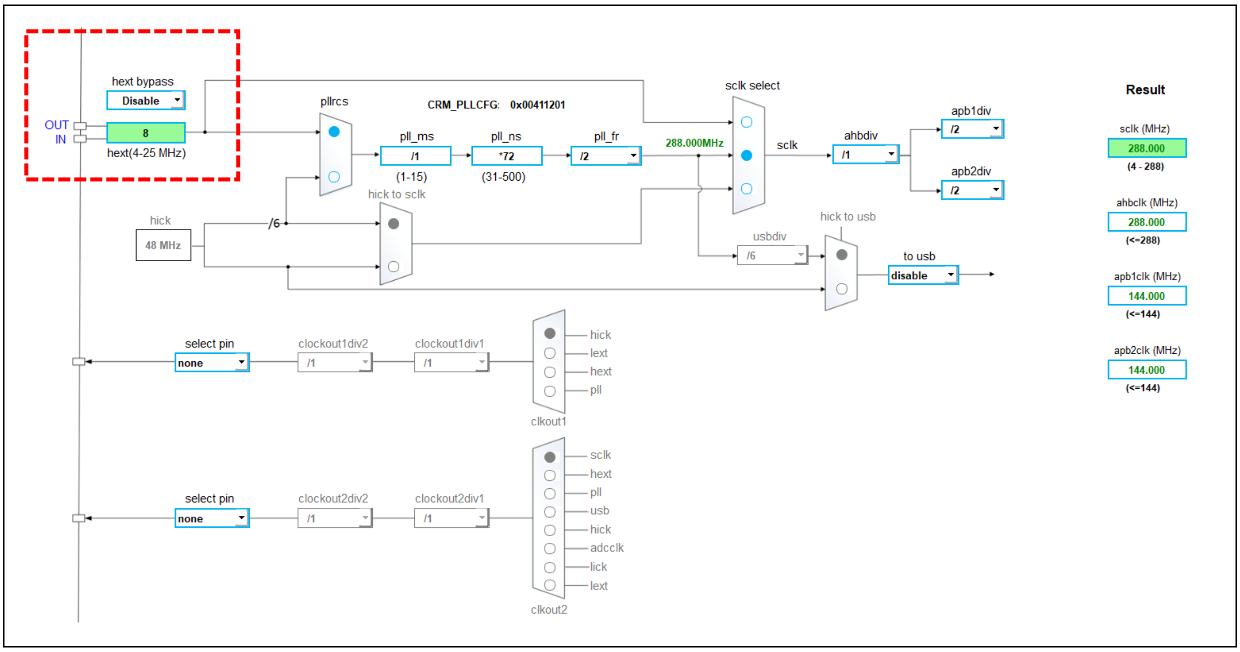
All examples used in BSP implements frequency multiplication based on 8 MHz external highspeed crystal oscillator on the evaluation board. If a non-8 MHz external crystal is used in actual scenarios, it is necessary to modify clock configuration in BSP to allow for accurate and stable clock frequency.

Therefore, the “AT32_New_Clock_Configuration” tool is specially developed by Artery to generate the desired BSP system clock code file, including external clock source, frequency division factor, frequency multiplication factor, clock source selection and other parameters, marked in red in Figure 38. After the completion of parameter configuration, it is ready to generate code file, avoiding complicated operations involved in code modification.

The users simply need to replace the original one in BSP demo with the newly generated clock code file (at32f4xx_clock.c/ at32f4xx_clock.h/ at32f4xx_conf.h) and call the function system_clock_config in main function.

Also, it is necessary to replace the macro definition HEXT_VALUE in the at32f4xx_conf.h. Taking the AT32F403A as an example, the HEXT_VALUE of the at32f403a_407_conf.h is defined as:
#define HEXT_VALUE   ((uint32_t)8000000) /*!< value of the high speed exernal crystal in hz */
Figure 38 shows the window of AT32_New_Clock_Configuration tool.

Figure 38. AT32_New_Clock_Configuration window



For more information on the AT32_New_Clock_Configuration, please refer to the corresponding Application Note shown in the table below, which are all available from the official website of Artery.

Table 499. Clock configuration guideline

Part number	Application note
AT32F403A/407 clock configuration	AN0082
AT32F435/437 clock configuration	AN0084
AT32F421 clock configuration	AN0116
AT32F415 clock configuration	AN0117
AT32F413 clock configuration	AN0118
AT32F425 clock configuration	AN0121

7 Revision history

Table 500. Document revision history

Date	Version	Revision note
2021.11.26	2.0.0	Initial release
2022.06.15	2.0.1	Added descriptions of peripheral library functions
2022.11.15	2.0.2	Modified descriptions of I2C in “abbreviations of peripherals”
2023.07.18	2.0.3	Added CRC related registers and functions
2023.10.26	2.0.4	Added interrupt_flag_get function for each IP.