武汉芯源半导体有限公司
WUHAN XINYUAN SEMICONDUCTOR CO., LTD

# CW32F003 Reference Manual

## ARM® Cortex®-M0+ 32-bit MCU

Rev 1.0

www.whxy.com

# Statement

### Important notice

This reference manual is aimed at application developers and provides complete information on how to use the CW32F003 microcontroller memory and peripherals. This operation applies only to CW32F003 devices.

### Disclaimer

Wuhan Xinyuan Semiconductor Co., Ltd. reserves the right to make changes, corrections, enhancements, and modifications to the product and this document at any time without notice. Buyers should obtain the latest product information on the official website.

### Copyright statement



### Technical support

If you have any comments or suggestions in the process of using, please feel free to contact us.

Websit : *www.whxy.com*

Address : 5 WuDaYuan 3rd Road, Donghu High-tech Development Zone Wuhan Hubei, China

Postcode : 430070

# Contents

# 1 Documentation conventions

## 1.1 Typographical conventions

The typographical conventions used in this document are:

| | |
|---|---|
| *Italic* | Indicates a special term or citation, or prompts a note. |
| Bold | Indicates a component or signal name, such as for title descriptions without chapter numbers, titles for diagrams, etc. |
| English uppercase | Used for terms that have a specific technical meaning and are included in the glossary, such as register or bit field names. |
| Color text | Indicates a link. Including: |

- URL external link, such as *www.whxy.com*
- Cross-references, internal chapters of this paper refer to each other
- Internal links, figures, tables or appendices, glossary entries

## 1.2 Register protocol

| Symbol example | Describe |
|---|---|
| 0x53CA | A numeric string prefixed with '0x' represents a hexadecimal number, and the value is composed of 0~9 or A~F. |
| REGNAME[n] | A specific bit of REGNAME, REGNAME can be a register or register field, for example, CR1[2] refers to bit 2 of the CR1 register (field). |
| REGNAME[m:n] | A specific bit of REGNAME, REGNAME can be a register or a register field, for example, CR1[7:5] refers to the 7th to 5th bits of the CR1 register (field). |
| RW | Read and write, software can read and write these bit fields. |
| RO | Read-only, software can only read these bit fields. |
| WO | Write-only, software can only write to this bit field, and invalid data will be returned when reading this bit field. |
| RW0 | Software can read and write this bit field, only 0 can be written to this bit field, and writing 1 has no effect on this bit field. |
| RW1 | Software can read and write this bit field, only 1 can be written to this bit field, and writing 0 has no effect on this bit field. |
| R0W1 | When the software reads this bit field as 0, only 1 can be written to this bit field, and writing 0 has no effect on this bit field. |
| R1W0 | When the software reads this bit field as 1, only 0 can be written to this bit field, and writing 1 has no effect on this bit field. |
| RFU | Reserved bit field, please keep the default value. |
| Word | The data length of a word is 32 bits. |
| Half Word | The data length of a half word is 16 bits. |
| Byte | The data length of one byte is 8bit. |

# 2 System and memory overview

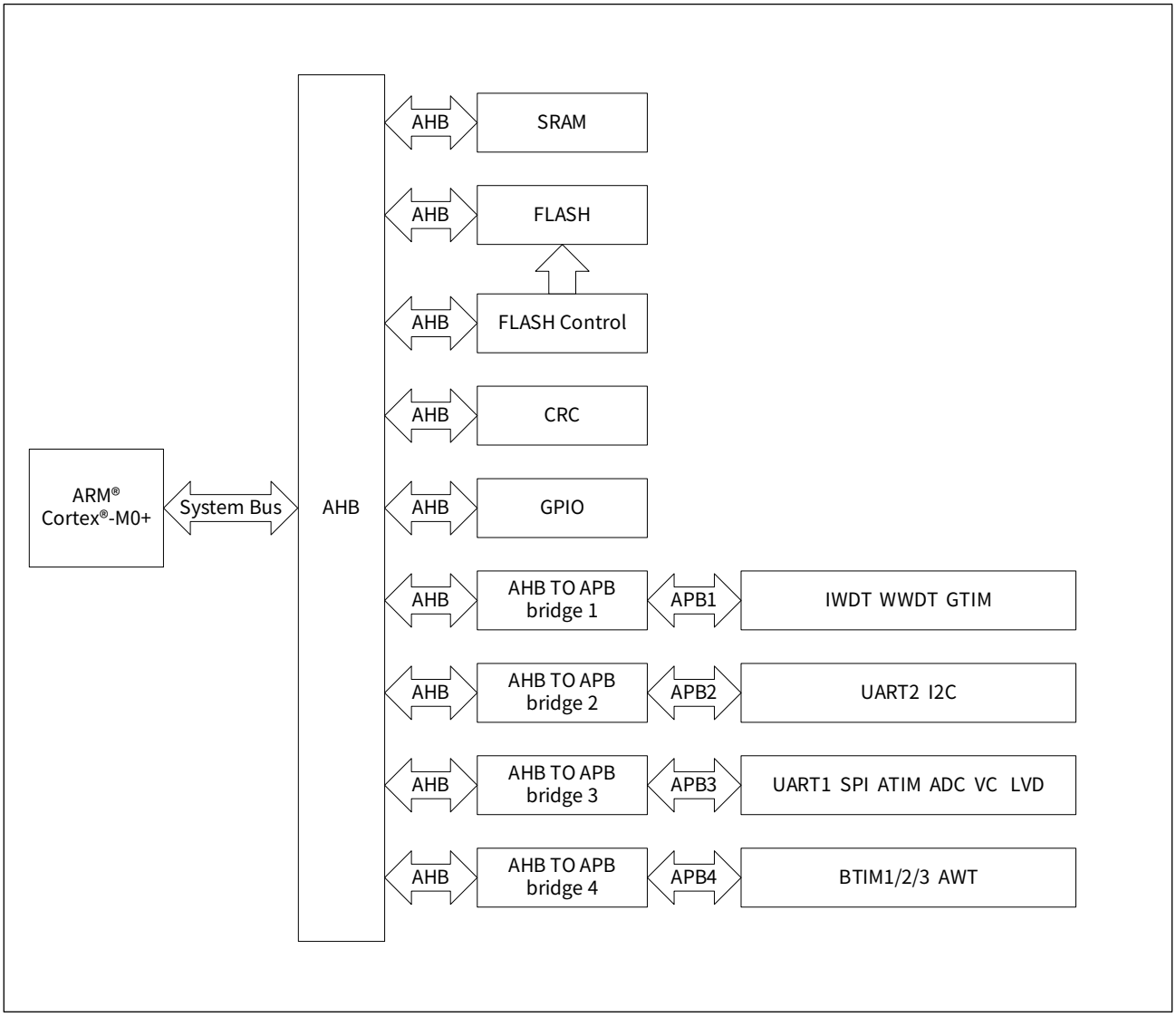
## 2.1 System architecture

The consists of CW32F003 microcontroller system contains:

- 1 master device:
  - ARM® Cortex®-M0+ core
- Multiple slave device:
  - On-chip SRAM
  - On-chip FLASH
  - FLASH controller
  - CRC redundancy calculation unit
  - GPIO port
  - AHB to APB1 conversion bridge and all devices on the APB1 bus
  - AHB to APB2 conversion bridge and all devices on the APB2 bus
  - AHB to APB3 conversion bridge and all devices on the APB3 bus
  - AHB to APB4 conversion bridge and all devices on the APB4 bus

The master and slave devices are connected through AHB bus matrix, and the system architecture is shown as follows:

Figure 2-1 System architecture




- System bus

  Realize the connection of peripheral bus and AHB bus of M0+ microprocessor.

- AHB To APB Bridge 1/2/3/4

  Provides a fully synchronized connection from the AHB bus to the APB1/APB2/APB3/APB4 bus, that is, the bridge between the AHB and APB buses.

## 2.2 Memory organization

### 2.2.1 Overview

The CW32F003 core is a 32-bit ARM® Cortex®-M0+ microprocessor with a maximum addressing space of 4GB. The built-in program memory, data memory, peripherals and port registers of the chip are uniformly addressed in the same 4GB linear address space.

Bytes in memory are organized in little-endian format. The lowest byte data of a word storage space is the least significant bit of the word, and the highest byte data is the most significant bit. E.g:

Store 0x1122 3344 in the memory space at address 0x2000 0000, the actual result is:

0x20000000 bytes store 0x44,

0x20000001 bytes store 0x33,

0x20000002 bytes store 0x22,

0x20000003 bytes store 0x11.

The system address allocation is shown in the figure below, RES is the reserved area.

Figure 2-2 System address assignment

### 2.2.2 Memory map and register boundary addresses

The detailed starting address space allocation of on-chip memory and peripherals is shown in the following table:

Table 2-1 Memory and Peripheral Address Allocation

| Device or Bus | Boundary address | Size | Peripherals |
|---|---|---|---|
| Main FLASH memory | 0x0000 0000 - 0x0000 4FFF | 20KB | Main FLASH |
| OTP memory | 0x0010 0770 - 0x0010 0785 | 22B | OTP |
| Boot program memory | 0x0010 0000 - 0x0010 09FF | 2.5KB | BootLoader |
| SRAM memory | 0x2000 0000 - 0x2000 0BFF | 3KB | SRAM |
| APB1 Peripheral | 0x4000 0400 - 0x4000 07FF | 1KB | GTIM |
| | 0x4000 2C00 - 0x4000 2FFF | 1KB | WWDT |
| | 0x4000 3000 - 0x4000 33FF | 1KB | IWDT |
| APB2 Peripheral | 0x4000 4400 - 0x4000 47FF | 1KB | UART2 |
| | 0x4000 5400 - 0x4000 57FF | 1KB | I2C |
| APB3 Peripheral | 0x4001 0000 - 0x4001 03FF | 1KB | SYSCTRL |
| | 0x4001 2400 - 0x4001 27FF | 1KB | ADC |
| | 0x4001 2800 - 0x4001 2BFF | 1KB | VC/LVD |
| | 0x4001 2C00 - 0x4001 2FFF | 1KB | ATIM |
| | 0x4001 3000 - 0x4001 33FF | 1KB | SPI |
| | 0x4001 3800 - 0x4001 3BFF | 1KB | UART1 |
| APB4 Peripheral | 0x4001 4800 - 0x4001 4BFF | 1KB | BTIM1/2/3 |
| | 0x4001 4C00 - 0x4001 4FFF | 1KB | AWT |
| AHB Peripheral | 0x4002 2000 - 0x4002 23FF | 1KB | FLASH CTRL |
| | 0x4002 3000 - 0x4002 33FF | 1KB | CRC |
| | 0x4800 0000 - 0x4800 03FF | 1KB | GPIOA |
| | 0x4800 0400 - 0x4800 07FF | 1KB | GPIOB |
| | 0x4800 0800 - 0x4800 0BFF | 1KB | GPIOC |
| M0+ Peripheral | 0xE000 0000 - 0xE00F FFFF | 1MB | M0+ Core Peripheral |

## 2.3 On-chip SRAM memory

CW32F003 integrates 3KB on-chip SRAM, and the starting address is 0x2000 0000. SRAM supports access in 3 widths of byte (8bit), half word (16bit) or full word (32bit), and can be accessed by the CPU at the maximum system clock frequency with zero latency.

- Parity check

  The SRAM supports the parity check function, which is enabled by default after the chip is powered on, and cannot be configured by the user.

  The data bus of SRAM is 36 bits, including 32 bits of data bits and 4 bits of parity bits (each 8 bits of data bits is equipped with 1 bit of parity bits), which is used to increase the robustness of the memory.

  The parity bit is calculated and stored when the CPU writes to the SRAM, and is automatically checked when the CPU reads the SRAM. After the system detects that the parity check fails, it will generate the corresponding interrupt flag.

For a detailed introduction to SRAM, see chapter *6 RAM Memory*.

## 2.4 On-chip FLASH memory

The on-chip FLASH memory consists of two physical areas: the main FLASH memory and the boot program memory.

- Main FLASH memory, a total of 20KB, the address space is 0x00000000-0x00004FFF. This area is mainly used to store application code and user data, which is programmable by the user.
- Boot program memory, a total of 2.5KB, the address space is 0x00100000-0x001009FF. This area is mainly used to store the BootLoader startup program, which has been programmed when the chip leaves the factory and cannot be changed by the user.

The FLASH controller implements various operations on FLASH (erase, write, read).

FLASH supports access with 3 widths of byte (8bit), half word (16bit) or full word (32bit), and the highest frequency of access is 24MHz. If the MCLK clock frequency of the system configuration is higher than 24MHz, a reasonable response waiting time must be configured through the WAIT bit field of the FLASH control register FLASH_CR2 to ensure that the FLASH is correctly accessed.

For a detailed introduction to FLASH, see chapter *7 FLASH Memory*.

## 2.5 One-time programmable OTP memory

There is 22B memory space inside the chip as the OTP area, and the address space is 0x0010 0770 - 0x0010 0785. This memory area can only be written to by ISP programming instructions, and can only be written once, and then can only be read, not erased and written. This area is mainly used to store unchangeable information, such as user-defined product UID, algorithm key, etc., which are written when the device leaves the factory.

## 2.6 System boot configuration

The device supports the following 2 different boot modes:

- Boot from main FLASH memory, run user program
- Start from the boot program memory, and fix the BootLoader program of the running chip. At this time, the user can use the ISP communication protocol to perform FLASH programming through the UART1 interface (PA02/PA05).

The steps for CW32F003 to enter ISP mode are as follows:

Step 1: Put the chip in the RESET state;

Step 2: Provide a 50kHz square wave to the chip's UART1_RXD (SWDIO);

Step 3: Release the RESET state of the chip and delay 5ms;

Step 4: The chip enters ISP mode.

In ISP mode, the user CODE code can be written, and the MCU needs to be reset after the download is completed to execute the user application program.

After the system is started, the CPU obtains the address of the top of the stack from the address 0x00000000 of the memory, and starts executing code from the address indicated by 0x00000004 of the memory.

The BootLoader program is located in the boot program memory area and is programmed by the equipment provider at production time. Users can use the ISP communication protocol for FALSH programming through UART1 (pins are PA02/PA05).

## 2.7 Precautions

CW32F003 needs to pay attention to the following matters when using:

- A few registers such as FLASH, SRAM and GPIOx_ODR, CRC_DR support 8bit/16bit/32bit access mode, and other peripherals only support 32bit access mode. 8bit/16bit access to an address that does not support 8bit/16bit access mode will generate a HardFault hardware error exception.
- The RES areas in the system address allocation in *Figure 2-2 System address assignment* are reserved address spaces without corresponding physical devices. If these reserved address spaces are accessed, a HardFault hardware error exception will occur.
- After the chip is reset, all peripheral clocks except SYSTICK and SRAM are turned off. Before using these peripherals, the user must pass the peripheral clock enable control register (AHB peripheral clock enable control register SYSCTRL_AHBEN, APB peripheral clock Enable control register SYSCTRL_APBEN1/2) to open the configuration clock and working clock of the corresponding peripheral. Failure to turn on the peripheral configuration clock will result in an access failure: writing data fails, and no HardFault hardware error exception will be generated, and the result of reading data is not available. If the operating clock of the peripheral is not turned on, the main function of the peripheral is invalid.

# 3 Power Control (PWR) and Power Consumption

## 3.1 Overview

CW32F003 needs a set of power supply when working: VDD, VSS (VDDA is internally connected to VDD).

CW32F003 embeds a 1.5V low dropout LDO voltage regulator to supply power to the digital power domain inside the chip.

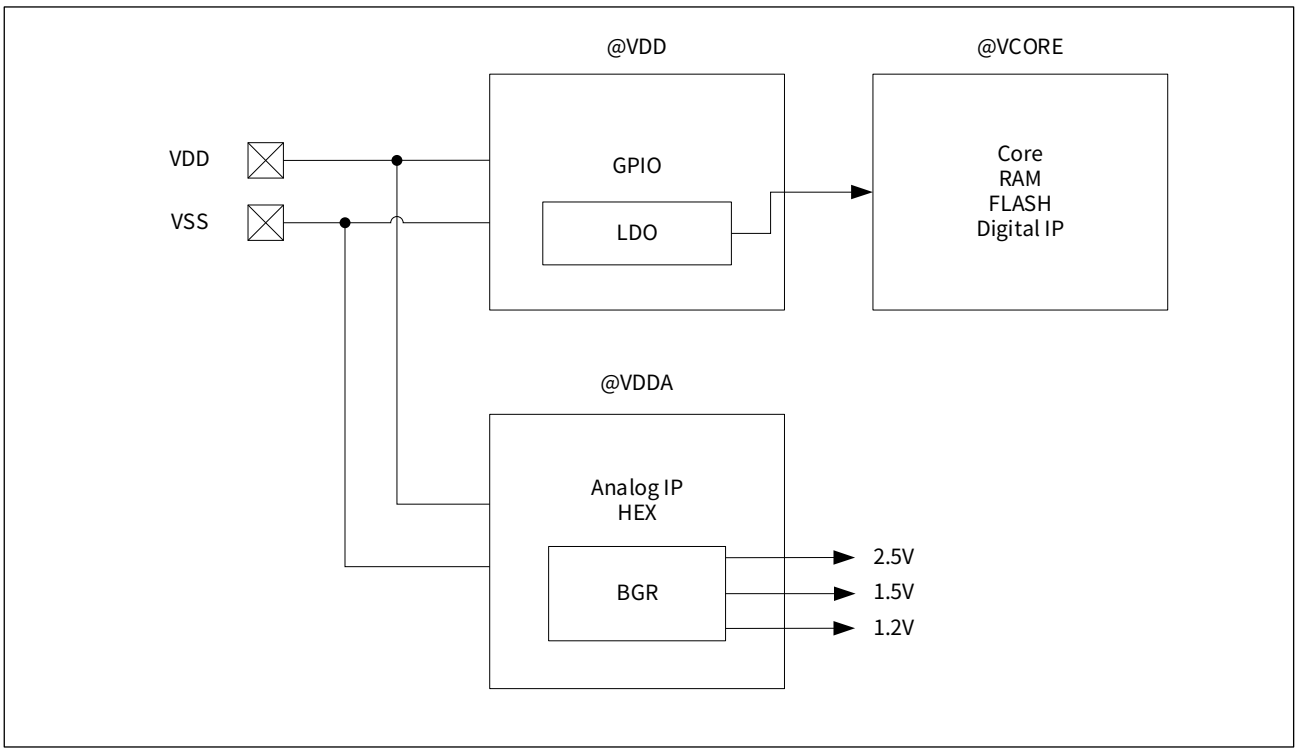
The chip has a built-in reference voltage generator (BGR) circuit, which can provide reference voltage for other analog modules.

*Caution:*

*The difference between the analog supply voltage and the reference voltage must be higher than 0.3V. Please refer to the relevant section of the datasheet for detailed parameters of the voltage reference.*

The division and allocation of power domains within the chip are shown in the following figure:

Figure 3-1 System Power Distribution



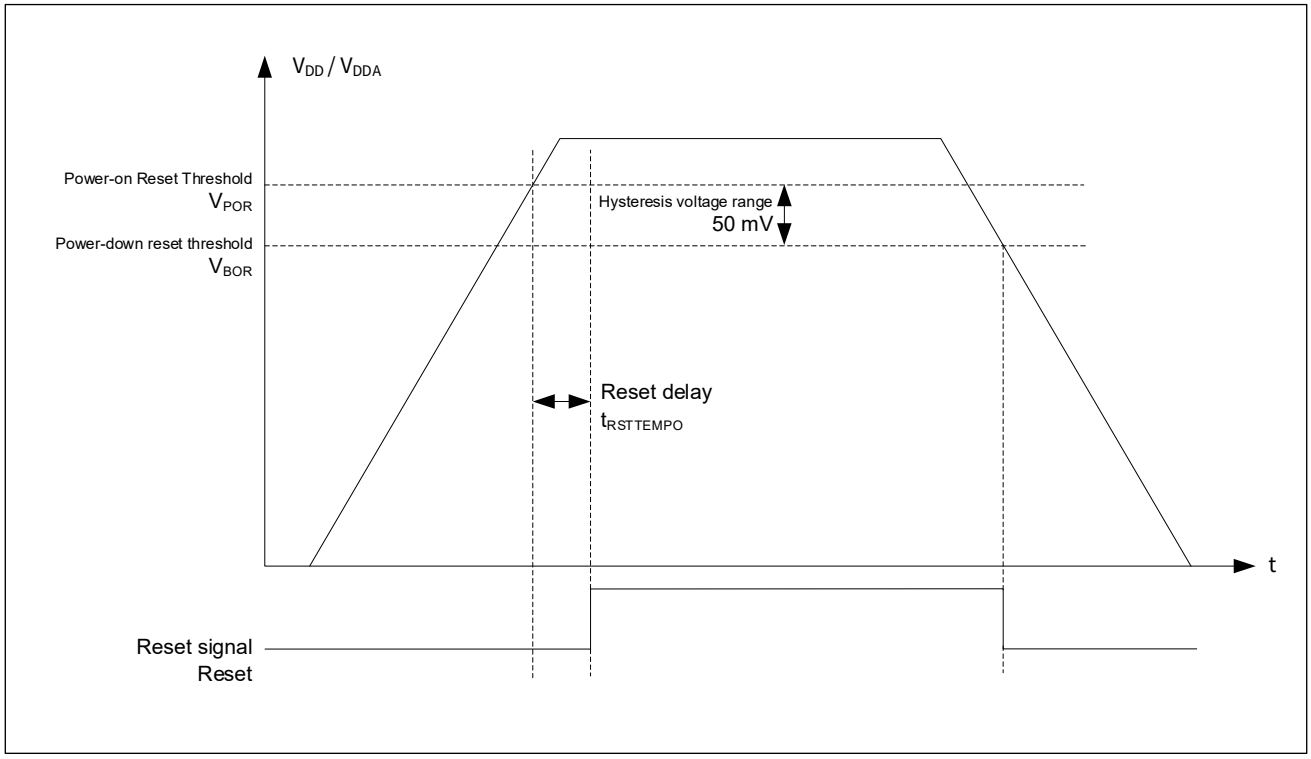
## 3.2 Power supply supervisor

### Power on reset (POR)/ Brown-out reset (BOR)

CW32F003 integrates power-on reset (POR) and Brown-out reset (BOR) power monitoring circuits, and is always in working state after power-on. POR/BOR monitors both the VDD power supply voltages. When the monitored power supply voltage is lower than the reset threshold ($V_{BOR}$), the system will enter the reset state. Users do not need to add an external hardware reset circuit.

The reset signal waveforms of the power-on and power-down phases are shown in the following figure:

Figure 3-2 Power on reset/power down reset waveform




For more details on the power on / power down reset threshold, refer to the electrical characteristics section in the datasheet.

## 3.3 Operating mode

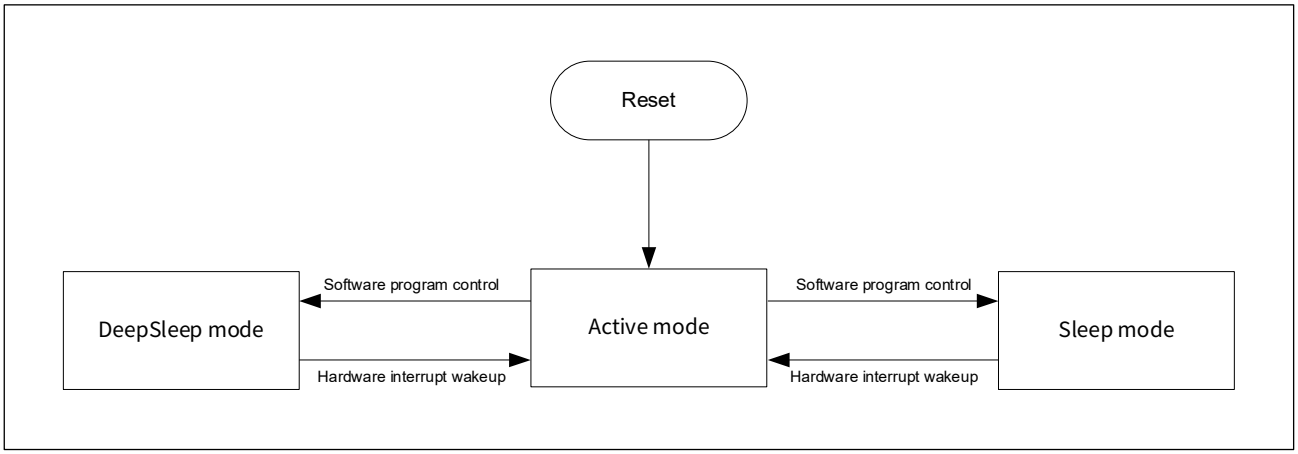
CW32F003 supports three operating modes, and the integrated power management module of the built-in power supply automatically completes the unified management. The three working modes are:

- Active mode
- Sleep mode
- DeepSleep mode

After the power is turned on, the system automatically enters the active mode. Users can enter two low-power operating states of sleep or deep sleep through software programs; In the low-power operating state, the wake-up mechanism can be triggered by a hardware interrupt to return the system to the active mode.

The conversion mechanism of the three operating modes is shown in the following figure:

Figure 3-3 The conversion mechanism of the operating modes




In the active mode, the CPU runs normally, and all module users can use it normally. In Sleep mode, the CPU stops running, all peripherals are unaffected, and all I/O pins remain in the same state. In DeepSleep mode, the CPU stops running, the high-speed clock (HSIOSC, HEX) is automatically turned off, and the low-speed clock (LSI, RC10K, RC150K) remains in the original state.

The CPU and clock states in different operating modes are shown in the following table:

Table 3-1 Operating mode and CPU and Clock States

| Operating mode | CPU state | High-speed clock | Low-speed clock |
|---|---|---|---|
| Active mode | Run | ON/OFF | ON/OFF |
| Sleep mode | Stop | ON/OFF | ON/OFF |
| DeepSleep mode | Stop | OFF | ON |

### 3.3.1 Entering Sleep mode or DeepSleep mode

Using the M0+ core's ARM waits for interrupt-specific instruction, WFI (Wait for Interrupt), in conjunction with the SLEEPONEXIT and SLEEPDEEP bit fields of the M0+ core's system control register (SCR, System Control Register), Sleep mode or DeepSleep mode can be entered or exited immediately (interrupt service routine).

- Entering immediately

  Executing the WFI instruction, the MCU will immediately enter Sleep mode (when SLEEPDEEP is 0) or DeepSleep mode (when SLEEPDEEP is 1)

- Entering on exit

  Setting the SLEEPONEXIT bit to 1, after exiting the lowest priority interrupt service routine, the MCU will enter Sleep mode (when SLEEPDEEP is 0) or DeepSleep mode (when SLEEPDEEP is 1), needn't execute the WFI instruction.

Table 3-2 Entering Sleep mode or DeepSleep mode

| Entry method | SLEEPONEXIT bit | Condition | SLEEPDEEP bit | Enter mode |
|---|---|---|---|---|
| Entering immediately | - | Executing the WFI instruction | 0 | Sleep mode |
| | | | 1 | DeepSleep mode |
| Entering on exit | 1 | After exiting the lowest priority interrupt service routine | 0 | Sleep mode |
| | | | 1 | DeepSleep mode |

*Caution 1:*

*In DeepSleep mode, the system will automatically turn off the high-speed clock. If users need to keep some peripherals running in DeepSleep mode, they must start the corresponding low-speed clock and set the peripheral clock to this low-speed clock before entering DeepSleep mode.*

*Caution 2:*

*Before entering DeepSleep mode, the user must configure the HCLK clock frequency to be less than or equal to 4MHz, otherwise the core may be damaged.*

*Caution 3:*

*If VCx is enabled, you must wait for the VCx_SR.READY flag to be set 1 before entering DeepSleep mode, otherwise you cannot enter DeepSleep mode.*

*Caution 4:*

*Before entering DeepSleep mode, if the FLASH is performing an erase/write operation, you must wait for the FLASH_CR1.BUSY flag bit to clear to 0, and make sure FLASH_CR1.MODE is 0 at the same time.*

### 3.3.2 Exiting Sleep mode or DeepSleep mode

In Sleep mode or DeepSleep mode, the interrupt can be used to wake up the CPU and return to active mode. However, it is worth noting that if the user executes the WFI command in the interrupt service routine to enter Sleep (including DeepSleep), an interrupt with a higher priority than this interrupt is required to wake up the CPU, so we strongly recommend that the user is ready to enter sleep, all interrupt service routines should be processed and all interrupt requests and interrupt flags should be cleared.

In different operating modes, the types of interrupts that the CPU can respond to are shown in the following table:

Table 3-3 Operating Modes and Interrupt Sources

| Interrupt Number | Interrupt Sources | Active mode | Sleep mode | DeepSleep mode |
|---|---|---|---|---|
| 0 | IWDT | Y | Y | Y |
| | WWDT | Y | Y | N |
| 1 | LVD | Y | Y | Y |
| 3 | FLASH | Y | Y | N |
| | RAM | Y | Y | N |
| 4 | RCC | Y | Y | Y |
| 5 | GPIOA | Y | Y | Y |
| 6 | GPIOB | Y | Y | Y |
| 7 | GPIOC | Y | Y | Y |
| 12 | ADC | Y | Y | N |
| 13 | ATIM | Y | Y | N |
| 14 | VC1 | Y | Y | Y |
| 15 | VC2 | Y | Y | Y |
| 16 | GTIM | Y | Y | N |
| 20 | BTIM1 | Y | Y | N |
| 21 | BTIM2 | Y | Y | N |
| 22 | BTIM3 | Y | Y | N |
| 23 | I2C | Y | Y | N |
| 25 | SPI | Y | Y | N |
| 27 | UART1 | Y | Y | Y |
| 28 | UART2 | Y | Y | Y |
| 30 | AWT | Y | Y | Y |

Exiting Sleep mode using an interrupt, the user must enable the enable bit for this interrupt before entering Sleep (including DeepSleep).

After the interrupt wakes up and exits Sleep mode, the CPU will immediately enter the interrupt service routine for this interrupt. If the user does not set this interrupt service routine, and when entering sleep immediately: the CPU will continue to execute the next statement of the WFI instruction that entered sleep; when entering sleep when exiting: continue to execute the next statement of the last entered interrupt service routine. In general, based on system reliability considerations, it is strongly recommended that the user set the service routine for this interrupt, and clear the interrupt request and interrupt flag in the interrupt service routine.

When the interrupt wakes up to exit the DeepSleep mode, the CPU operation state is the same as the exit from the Sleep mode.

In DeepSleep mode, the system will automatically turn off the high-speed clock. When exiting DeepSleep, CW32F003 adds an additional system clock option for users. Users can choose to continue to use the clock used when entering DeepSleep, or select HSI as the system clock. Configure the WAKEUPCLK bit field of the system control register SYSCTRL_CR2 to 1, then the internal high-speed clock HSI is automatically used as the system clock to accelerate the system wake up after the interrupt wakes up and exits the DeepSleep mode.

### 3.3.3 Operating Modes and Reset Sources

Even in Sleep mode or DeepSleep mode, the CPU can respond to some reset sources. The hardware reset or software reset that the CPU can respond to in different working modes is shown in the following table:

Table 3-4 Operating Modes and Reset Sources

| Reset Sources | Active mode | Sleep mode | DeepSleep mode |
|---|---|---|---|
| Power-on reset/Brown-out reset (POR/BOR) | Y | Y | Y |
| Pin input reset (NRST) | Y | Y | Y |
| LVD reset | Y | Y | Y |
| IWDT reset | Y | Y | Y |
| WWDT reset | Y | Y | N |
| Kernel LOCKUP fault reset | Y | N | N |
| Kernel SYSRESETREQ reset | Y | N | N |

## 3.4 Low-power consumption applied

In Sleep mode, the CPU stops running, and all peripherals keep running, including ARM®Cortex®-M0+ core peripherals, such as NVIC, SysTick and other peripherals. Sleep mode consumes less power than active mode.

In DeepSleep mode, the CPU stops running, the high-speed clock is turned off, the low-speed clock remains unchanged, some peripherals can be configured to continue to run, and the NVIC interrupt processing still works. DeepSleep mode consumes much less power than Sleep mode.

Users can reduce system running power consumption by:

**Decrease system clock frequency**

- Use low-frequency high-speed clock HSI, HEX or low-speed clock LSI
- Reduce the frequency of SYSCLK, HCLK, PCLK by programming the prescaler registers
  - Set the SYSCLK bit field of the SYSCTRL_CR0 register to select the appropriate clock source
  - Set the HCLKPRS bit field of the SYSCTRL_CR0 register to reduce the HCLK frequency
  - Set the PCLKPRS bit field of the SYSCTRL_CR0 register to reduce the PCLK frequency

**Turn off clocks and peripherals which not in use during sleep**

- AHB bus clock HCLK and APB bus clock PCLK, can be turned off as needed
- Turn off clocks to peripherals not related to wakeup
  - AHB peripheral clock enable control register, SYSCTRL_AHBEN
  - APB peripheral clock enable control register 1, SYSCTRL_AHBEN1
  - APB peripheral clock enable control register 2, SYSCTRL_AHBEN2

## 3.5 Cortex®-M0+ kernel system control register (SCB->SCR)

Address: 0xE000 ED10 Reset value: 0x0000 0000

| Bit filed | Name | Permission | Function description |
|---|---|---|---|
| 31:5 | RFU | - | Reserved bits, please keep the default value |
| 4 | SEVONPEND | RW | When set to 1, each new suspension of an interrupt generates an event, which can be used to wake up the processor if WFE sleep is used. |
| 3 | RFU | - | Reserved bits, please keep the default value |
| 2 | SLEEPDEEP | RW | When set to 1, enter DeepSleep mode when WFI or SLEEPONEXIT is 1 and all interrupt service routines are exited;<br>When set to 0, enter Sleep mode when WFI or SLEEPONEXIT is 1 and all interrupt service routines are exited |
| 1 | SLEEPONEXIT | RW | When set to 1, the processor automatically enters Sleep mode (or DeepSleep mode) when all interrupt service routines are exited;<br>When set to 0, the hibernation function on exit is disabled |
| 0 | RFU | - | Reserved bits, please keep the default value |

# 4 Reset and clock control (RCC)

## 4.1 System reset

CW32F003 supports the following 6 system resets:

- Power on reset (POR)/ Brown-out reset (BOR)
- Pin input reset (NRST)
- IWDT/WWDT reset
- LVD reset
- Kernel SYSRESETREQ reset
- Kernel LOCKUP reset

The reset range of each system reset is shown in the following table:

Table 4-1 Reset method and range

| Reset method | Reset range |
|---|---|
| Power on reset (POR)/ Brown-out reset (BOR) | Entire MCU |
| Pin input reset (NRST) | Entire MCU |
| IWDT/WWDT reset | M0+ core/peripheral (Except RAM controller) |
| LVD reset | M0+ core/peripheral (Except LVD) |
| Kernel SYSRESETREQ reset | M0+ core (Except SWD debug logic) /<br>Peripheral (Except RAM controller/LVD) |
| Kernel LOCKUP fault reset | M0+ core/peripheral (Except RAM controller/LVD) |

After the system reset occurs, the CPU runs again, most of the registers are reset to default values, and the program starts to execute from the reset interrupt entry address of the interrupt vector table.

The user can query the reset source of this system reset through the system reset flag register SYSCTRL_RESETFLAG. The reset flag is set by hardware and cleared by software. It is recommended that users clear the relevant flag bits of this register to 0 after reading the flag bits to avoid confusion after the next reset.

### 4.1.1 Power on reset/Brown-out reset (POR/BOR)

The CW32F003 integrates special POR and BOR circuits to monitor the power supply voltage, and keeps the chip in the reset state when the power supply voltage is lower than the safe range to prevent the chip from malfunctioning during power-on/power-off. In order to ensure the stable operation of the system, the user must keep the power supply voltage within a safe range.

For the specific functions of POR and BOR, please refer to chapter *3 Power Control (PWR) and Power Consumption*.

### 4.1.2 Pin input reset (NRST)

CW32F003 has a special reset input pin, inputting a low-level signal of a certain width will cause the system to reset. A dedicated anti-shake circuit is designed inside the chip, and low-level pulse signals shorter than 20μs will be shielded.

The reset input pin has a built-in pull-up resistor. If the user needs to connect an external RC circuit, the influence of the internal pull-up resistor must be considered. For the circuit characteristics of the built-in pull-up resistor, please refer to the relevant section of the datasheet.

### 4.1.3 IWDT/WWDT reset

The CW32F003 integrates an independent watchdog (IWDT) and a window watchdog (WWDT). When the IWDT or WWDT meets the reset condition, a reset signal will be generated to cause a system reset.

For detailed functions of IWDT and WWDT reset, please refer to chapters *14 Independent watchdog timer (IWDT)* and *15 Window watchdog timer (WWDT)*.

### 4.1.4 LVD low voltage detection reset

The CW32F003 integrates a low voltage detector (LVD), which can continuously compare the selected monitoring voltage with the set LVD threshold voltage. When the comparison result meets the trigger condition, an LVD reset signal will be generated to cause a system reset.

Compared with the POR/BOR reset method, the LVD low voltage detection reset function is more powerful. The reset threshold voltage, monitoring voltage source, pulse filter width and hysteresis time can be set through the relevant registers.

See chapter *22 Low voltage detector (LVD)* for LVD details.

### 4.1.5 Kernel SYSRESETREQ reset

The core SYSRESETREQ reset is a software reset, which is implemented by setting the SYSRESETREQ bit field of the Application Interrupt and Reset Control Status Register (AIRCR) of the ARM® Cortex®-M0+. Setting this bit to 1 by the application program will generate a core SYSRESETREQ reset, thereby implementing a software reset.

### 4.1.6 Kernel LOCKUP fault reset

When the CPU encounters a serious exception (such as the read instruction is invalid, the bit width does not match the target address when accessing FLASH), it will stop the PC pointer at the current address and lock it, and generate the kernel LOCKUP fault reset signal.

After the chip is powered on, the LOCKUP reset function is disabled by default, and the user needs to manually enable it. By setting the LOCKUP bit field of the system control register SYSCTRL_CR2 to 1, the LOCKUP reset function can be enabled.

## 4.2 Peripheral reset

Users can use software to reset various peripherals inside CW32F003 individually. Peripheral reset can restore the registers, state machines and various control logic of each peripheral to the default state after power-on reset. The user can perform the independent reset operation of each peripheral module by setting the three registers: SYSCTRL_AHBRST, SYSCTRL_APBRST1, and SYSCTRL_APBRST2.

For power-on reset/brown-off reset (POR/BOR), the internal peripheral modules are already in the default state after reset, and the module can be initialized and configured directly, and there is no need to independently reset each module.

For other types of system resets, some peripherals may retain the working state before the reset. If the user needs to restore the peripherals to the default state of power-on, they should reset the peripherals through the corresponding reset registers before using them.

## 4.3 Clock and Control

### 4.3.1 Overview

The CW32F003 has a built-in multi-channel clock generation circuit, which generates various clocks of different frequencies through the frequency divider and multiplexer for use by the CPU and various peripherals. The schematic diagram of the system clock tree structure is shown in the *Figure 4-1 System internal clock tree*.

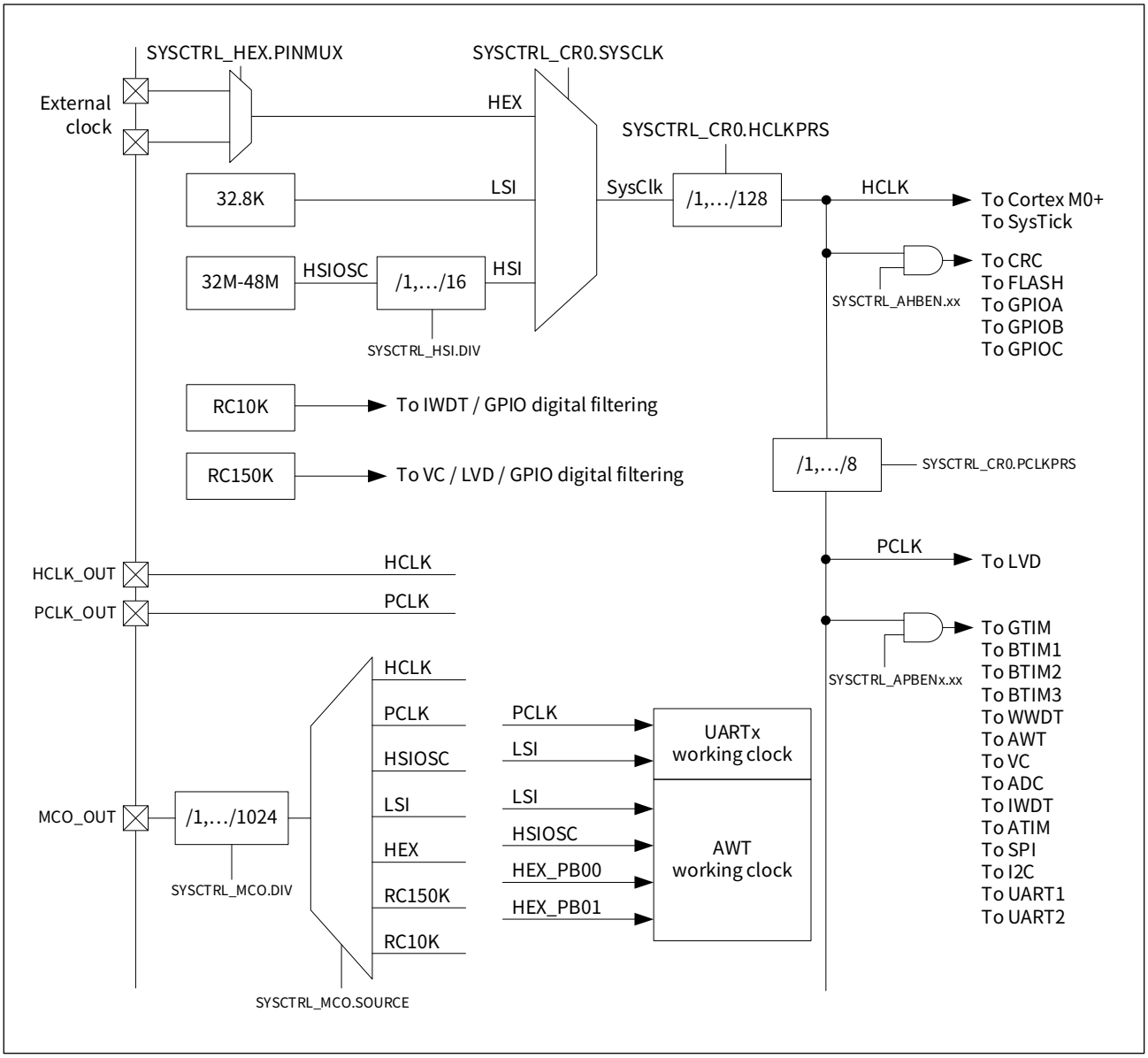
The internal clock SysClk of the system is divided to provide the advanced high-performance bus clock HCLK for the CPU core, and the HCLK clock is divided to provide the advanced peripheral clock PCLK for digital and analog peripherals.

System clock SYSCLK has 3 clock sources:

- The HSI clock is generated by dividing the internal high-speed RC oscillator clock (HSIOSC)
- Low-speed internal RC oscillator clock (LSI)
- External pin input clock (HEX)

The system internal clock tree is shown in the following figure:

Figure 4-1 System internal clock tree



The high-speed internal RC oscillator clock HSIOSC is divided by the frequency divider to generate the HSI clock. The frequency division coefficient is set through the DIV bit field of the built-in high-frequency clock control register SYSCTRL_HSI, and the effective frequency division coefficients are 1, 2, 4, 6, 8, 10, 12, 14, and 16.

The system clock SysClk can be selected from 3 clock sources: HSI, LSI, HEX, which can be selected by the SYSCLK bit field of the system control register SYSCTRL_CR0.

Advanced high-performance bus clock HCLK is generated by the frequency division of the system clock SysClk, and is used as the configuration clock and working clock of the M0+ core, SysTick, FLASH, CRC, GPIO and other modules. The frequency division factor is set by the HCLKPRS bit field of the system control register SYSCTRL_CR0, and the effective frequency division factor is $2^n$ (n = 0~7).

Advanced peripheral clock PCLK, which is generated by frequency division of HCLK, is used as the configuration clock and working clock of peripherals such as timer, SPI, and I2C. The frequency division factor is set by the PCLKPRS bit field of the system control register SYSCTRL_CR0, and the effective frequency division factor is $2^n$ (n = 0~3).

In addition to the 3 clock sources used as the system clock, the CW32F003 also has 2 dedicated low-speed clock sources:

- Internal RC10K clock

  Can be used as IWDT module count clock and filter clock for GPIO port interrupt input signal.

- Internal RC150K clock

  Can be used as the filter clock of the digital filter module of LVD and VC, and the filter clock of the GPIO port interrupt input signal.

### 4.3.2 System Clock and Operating Mode

CW32F003 supports three operating modes: active mode, Sleep mode and DeepSleep mode.

In the active mode, the CPU runs normally, and all module users can use it normally.

In the Sleep mode, the CPU stops running, and the clock oscillators and peripherals remain unchanged.

In DeepSleep mode, the CPU stops running, and the high-speed clock HEX and HSIOSC oscillator are automatically turned off to save power; The oscillators of the low-speed clocks LSI, RC10K, RC150K remain unchanged; Whether the system clock SysClk and the HCLK and PCLK clocks are valid depends on the clock source status of the SysClk system clock.

After waking up from DeepSleep mode, if the WAKEUPCLK bit field of the system control register SYSCTRL_CR2 is configured as 1, the system will automatically use the HSI as the clock source of the system clock. If SYSCTRL_CR2.WAKEUPCLK is 0, the system will wait for the system clock source used before the system enters deep sleep to stabilize before starting to run; If the clock source of the system clock is HEX, the clock recovery speed and system response speed will be slower. Since the HSI starts up quickly and can quickly respond to user needs, it is recommended to switch the clock source of the system clock to the HSI clock or configure SYSCTRL_CR2.WAKEUPCLK to 1 before entering DeepSleep mode.

### 4.3.3 HEX clock

Set the PINEN bit field of the external input clock control register SYSCTRL_HEX to 1 to enable the external clock input function. The HEX clock is input from the external PB00 or PB01 pin, which is configured through the PINMUX bit field of the SYSCTRL_HEX register, and the corresponding clock input pin needs to be configured as a digital input function.

The external input clock signal can be square wave, sine wave or triangular wave, the duty cycle must be between 40%~60%, and the frequency is between 4~32MHz.

The stable state of the HEX clock can be determined through the STABLE flag bit of the external input clock control register SYSCTRL_HEX. If the STABLE flag is 1, the HEX clock is stable, and if it is 0, the HEX clock is not stable yet.

*Caution:*

*All parameters of the HEX clock should be set before startup, and modification of relevant parameters is prohibited after startup.*

### 4.3.4 HSIOSC clock

The HSIOSC clock is generated by the internal RC oscillator, no external circuit is required, the startup speed is faster. The HSIOSC clock frequency is fixed at 48MHz.

The frequency of the RC oscillator output clock is affected by the chip processing process, operating voltage, ambient temperature and other factors. CW32F003 provides the HSIOSC clock frequency calibration function. Users can calibrate the HSIOSC clock by setting the TRIM bit field value of the built-in high-frequency clock control register SYSCTRL_HSI frequency, see section *4.4.2 Clock calibration* for details.

The HSIOSC internal high-speed RC oscillator is turned on by default after the chip is powered on. Users can turn it off by setting the HSIEN bit field of SYSCTRL_CR1 in the system control register to 0. If the user stops and restarts the HSIOSC oscillator, the STABLE flag of the built-in high-frequency clock control register SYSCTRL_HSI can be used to determine whether the HSI clock is stable. If the STABLE flag is 1, it means that the HSIOSC clock is stable, and if it is 0, it means that the HSIOSC clock is not stable yet.

*Caution:*

*All parameters of the HSIOSC oscillator should be set before starting, and it is forbidden to modify the relevant parameters after starting.*

The HSIOSC clock outputs the HSI clock after frequency division. The frequency division coefficient is set by the DIV bit field of the built-in high-frequency clock control register SYSCTRL_HSI. The effective frequency division coefficients are 1, 2, 4, 6, 8, 10, 12, 14, 16, The default value after power-on is 6, so the default frequency of the HSI clock is 8MHz.

### 4.3.5 LSI clock

The LSI clock is generated by an internal low-speed RC oscillator, no external circuits are required, and the default frequency is 32.8kHz.

The frequency of the RC oscillator output clock is affected by the chip processing process, operating voltage, ambient temperature and other factors. CW32F003 provides the LSI clock frequency calibration function. Users can calibrate the LSI clock frequency by setting the value of the TRIM bit field of the built-in low-frequency clock control register SYSCTRL_LSI. For details, please refer to *4.4.2 Clock calibration*.

The LSI internal low-speed RC oscillator is disabled by default, and is enabled by setting the LSIEN bit field of the system control register SYSCTRL_CR1 to 1. After the LSI oscillator is started, if the internal clock monitoring module of the chip detects a certain number of LSI clock signals, the LSI clock is considered to be stable. The number of detection clocks can be set by the WAITCYCLE bit field of the built-in low-frequency oscillator control register SYSCTRL_LSI, as shown in the following table:

Table 4-2 LSI Stability Detection Clock Signal Number

| SYSCTRL_LSI.WAITCYCLE | Number of LSI detection clocks |
|---|---|
| 00 | 6 (Defaults) |
| 01 | 18 |
| 10 | 66 |
| 11 | 258 |

The STABLE flag of the built-in low-frequency clock control register SYSCTRL_LSI can be used to determine whether the LSI clock is stable. If the STABLE flag is 1, it means that the LSI clock is stable, and if it is 0, it means that the LSI clock is not stable.

*Caution:*

*All parameters of the LSI oscillator should be set before starting, and it is forbidden to modify the relevant parameters after starting.*

### 4.3.6 SysClk system clock

System clock SysClk can choose from 3 clock sources, including HEX, HSI and LSI.

After the system power-on reset is completed, HSI is selected as the clock source of SysClk by default, and the default value of the clock frequency is 8MHz.

The user can select the system clock source through the SYSCLK bit field of the system control register SYSCTRL_CR0. After selecting a clock as the system clock source, the user cannot stop the clock by setting the enable bit SYSCTRL_CR1.xxxEN of the clock circuit to 0.

### 4.3.7 On-chip peripheral clock control

On-chip peripherals generally need the configuration clock and the working clock. The configuration clock is used to respond to the read and write operations of the peripheral registers by the CPU. The working clock is used for the function realization of each peripheral (such as the transmission clock of the UART, the count clock of the timer, etc.).

The configuration clock and working clock of the peripheral must be turned on before using the peripheral, otherwise the peripheral cannot work. By setting the corresponding bits of the AHB peripheral clock enable control register SYSCTRL_AHBEN and the APB peripheral clock enable control register SYSCTRL_APBEN1 and SYSCTRL_APBEN2 to 1, the configuration clock and working clock of the corresponding peripherals are turned on.

Peripherals such as UART, AWT, and FLASH only open the configuration clock, and the working clock is configured through the clock source selection register of each module.

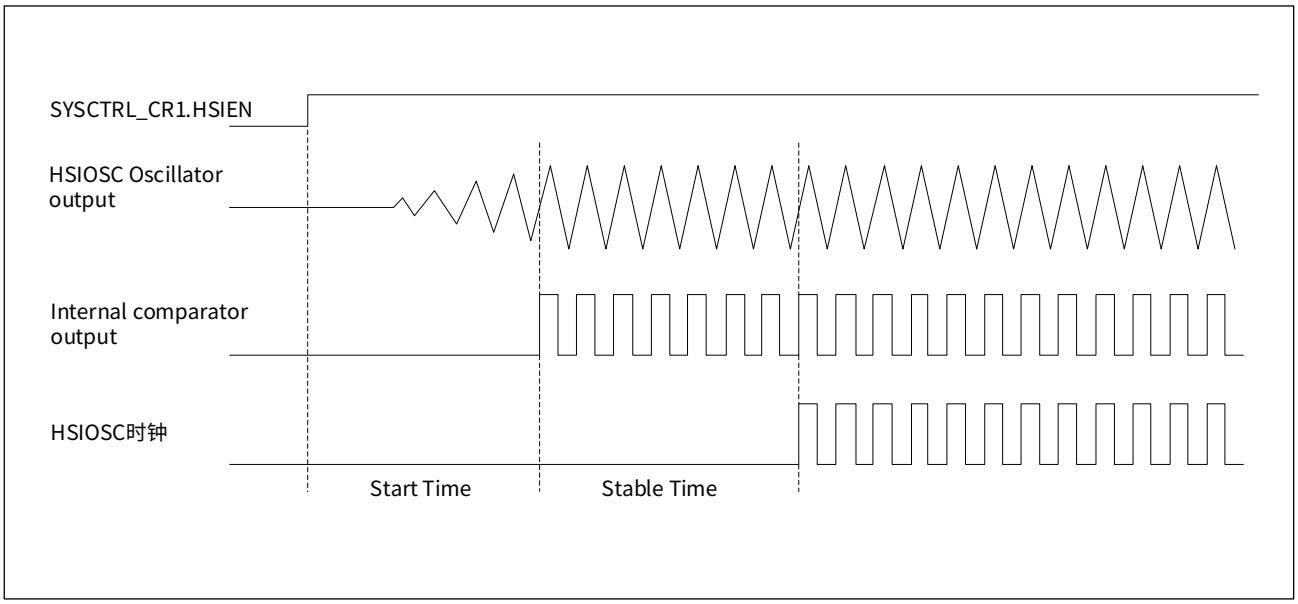
When the peripherals do not need to be used, the peripherals can be disabled by turning off the configuration clocks and operating clocks of the peripherals, which can effectively reduce the power consumption of the chip.

## 4.4 Clock Start, Calibration, and Status Detection

### 4.4.1 Clock start

The clock source startup process of CW32F003 is similar. Take HSIOSC as an example to illustrate the stabilization process after clock startup, as shown in the following figure:

Figure 4-2 HSIOSC clock startup process




When SYSCTRL_CR1.HSIEN is set to 1, the HSIOSC clock oscillation circuit starts to work, but the output clock signal amplitude is very small at this time. After the start-up time stage, the amplitude and duty cycle of the output clock signal can meet the needs of the internal sampling circuit and enter the stabilization time stage. In the stable time stage, the internal clock monitoring circuit of the chip counts the clock signals output by the HSIOSC. When the count value reaches the set number, the HSIOSC clock signal is considered to be stable, and the HSIOSC clock stability flag bit SYSCTRL_HSI.STABLE is set to 1.

The clock start-up process of other clock oscillators is similar, but note that the number of detection clocks in the stable phase, the detection time and the number of detection clocks for start-up failure detection are different for each clock oscillator. For details, see the description of each clock oscillator.

### 4.4.2 Clock calibration

Clock calibration is mainly for HSIOSC clock and LSI clock. Clock frequency calibration is achieved by adjusting the TRIM value of the oscillator.

#### HSIOSC clock calibration

The safe working range of HSIOSC is 32~48MHz. If it exceeds the safe range, the chip may be abnormal. When the chip leaves the factory, the calibration parameters of the 48MHz frequency point have been pre-adjusted and stored in FLASH. The application program only needs to read and write the calibration value in FLASH to SYSCTRL_HSI.TRIM to obtain a precise 48MHz clock. The storage address of the 48MHz frequency calibration value is: 0x0010 07B8 – 0x0010 07B9. If other frequency clocks are needed, users need to adjust the value of SYSCTRL_HSI.TRIM by themselves.

#### LSI clock calibration

The LSI output clock frequency range is 32.8kHz±10%. If the LSI output clock frequency is adjusted outside this range, the clock may be abnormal. When the chip leaves the factory, the calibration parameters of the 32.8kHz frequency point are pre-adjusted and stored in the FLASH. The application program only needs to read the calibration value in the FLASH and write it into SYSCTRL_LSI.TRIM to obtain an accurate 32.8kHz clock. 32.8kHz frequency calibration value storage address: 0x0010 07B8 - 0x0010 07B9. If other frequency clocks are needed, users need to adjust the value of SYSCTRL_LSI.TRIM by themselves.

### 4.4.3 Clock state detection

The 3 clock sources, HSIOSC, LSI, and HEX, all support the clock stability detection function. Users can determine the clock status through the stability flag of the corresponding clock source. The clock stable flag is cleared by hardware when the clock source is turned off, and set 1 by hardware after the clock source is enabled and stable.

Note that the clock stable flag is only for the clock start-up process. During the clock stable operation process, the detection of clock operation failure will not affect the clock stable flag.

Taking the HSIOSC clock source as an example, the HSIOSC clock stability flag and clock stability interrupt flag are described as follows:

- SYSCTRL_HSI.STABLE

  Cleared by hardware when HSIOSC is turned off, and set by hardware when it detects that the HSIOSC clock is stable.

- SYSCTRL_ISR.HSISTABLE

  It belongs to the same signal as SYSCTRL_HSI.STABLE. Although this signal is located in the interrupt flag register SYSCTRL_ISR, it will not cause an interrupt request after being set to 1 by hardware.

- SYSCTRL_ISR.HSIRDY

  When the hardware detects that the clock changes from an unstable state to a stable state (that is, the SYSCTRL_HSI.STABLE flag changes from 0 to 1), it is set to 1, which is the clock stability interrupt flag. Users can clear this flag by setting SYSCTRL_ICR.HSIRDY to 0.

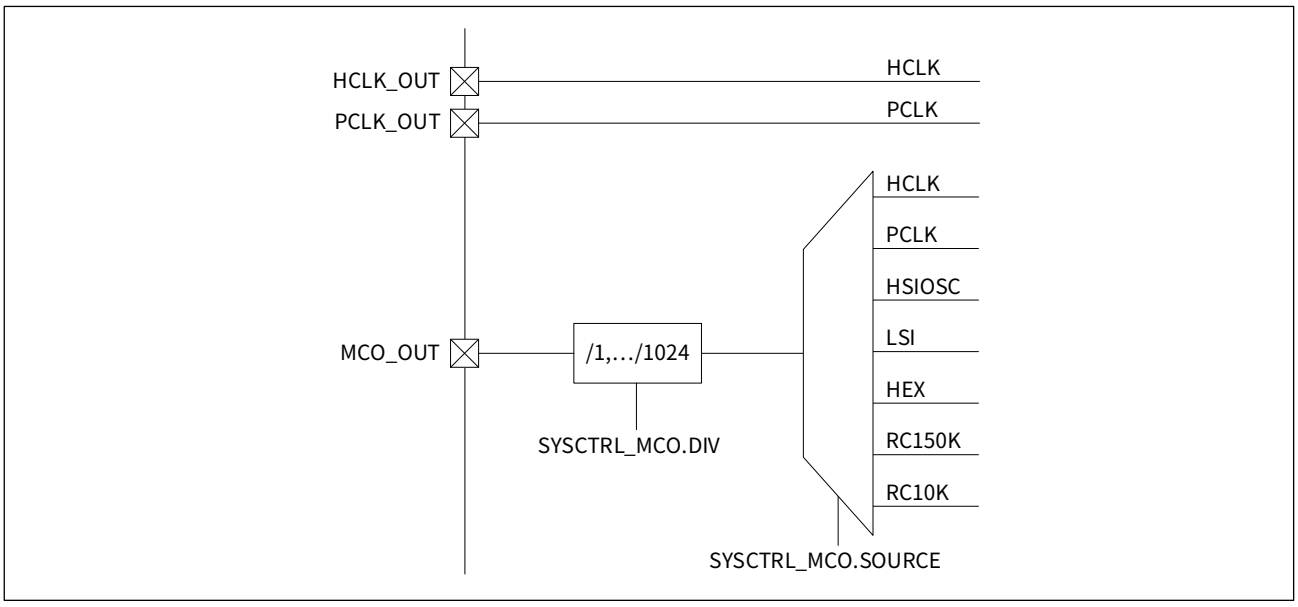
### 4.4.4 Clock verification and output

CW32F003 supports outputting various internal clock signals to external pins. Users can use this function to measure the current system's CPU operating frequency, system bus frequency, peripheral operating frequency, etc.

- HCLK_OUT pin
  Output HCLK clock signal
- PCLK_OUT pin
  Output PCLK clock signal
- MCO_OUT pin
  Output HCLK/PCLK/HSIOSC/LSI/HEX/RC150K/RC10K clock signal, before the clock is output to the MCO_OUT pin, it can be divided by a prescaler (effective frequency division factors are 1, 2, 8, 64, 128, 256, 512, 1024), so that the low-bandwidth instrument can accurately measure the signal.

The clock output block diagram is shown in the following figure:

Figure 4-3 Clock output



## 4.5 SysClk system clock switch

The system clock SysClk can select 5 clock sources, including HEX, HSI, and LSI. By setting the SYSCLK bit field of the system control register SYSCTRL_CR0, it can be switched between different clock sources.

The main application scenarios include:

- Automatic clock failover

  When it is detected that the current system clock source fails, it can quickly and safely switch to other available clock sources to improve product reliability.

- Power management

  Switch the clock source of SysClk to low-speed clock in standby state to reduce system power consumption, and switch the clock source of SysClk to high-speed clock in normal use state to quickly respond to user needs.

The rules for safe switching of clock sources are as follows:

- Any two of the three clock sources of HEX, HSI and LSI can be switched between each other

The system clock switching operation must be performed in accordance with the seven clock switching procedures described below, otherwise an exception may occur.

*Caution:*

*When the system clock is switched, the FLASH control register FLASH_CR2.WAIT needs to be configured synchronously to read the waiting period parameters: if the system clock frequency is not greater than 24MHz, the FLASH control register FLASH_CR2.WAIT should be set to 0; if the system clock frequency is greater than 24MHz, the FLASH control register FLASH_CR2.WAIT should be set is 1; if the system clock frequency is greater than 48MHz, the FLASH control register FLASH_CR2. WAIT should be set to 2.*

### 4.5.1 Standard Clock Switching Process

The standard clock switching operation flow is as follows:

Step 1: If the new clock source is HEX, you need to configure the corresponding external pin as a digital input function and enable the input function of the port;

Step 2: Configure the oscillation parameters of the new clock source, such as drive capability, oscillation amplitude, stabilization period, etc.;

Step 3: Set the new clock source oscillator enable bit to 1;

Step 4: According to the higher frequency of the current clock source and the new clock source, configure the FLASH_CR2.WAIT read wait period parameter according to the requirements of the chapter *7 FLASH Memory*;

Step 5: Wait for the new clock source to output a stable frequency (the STABLE signal of the new clock source becomes 1);

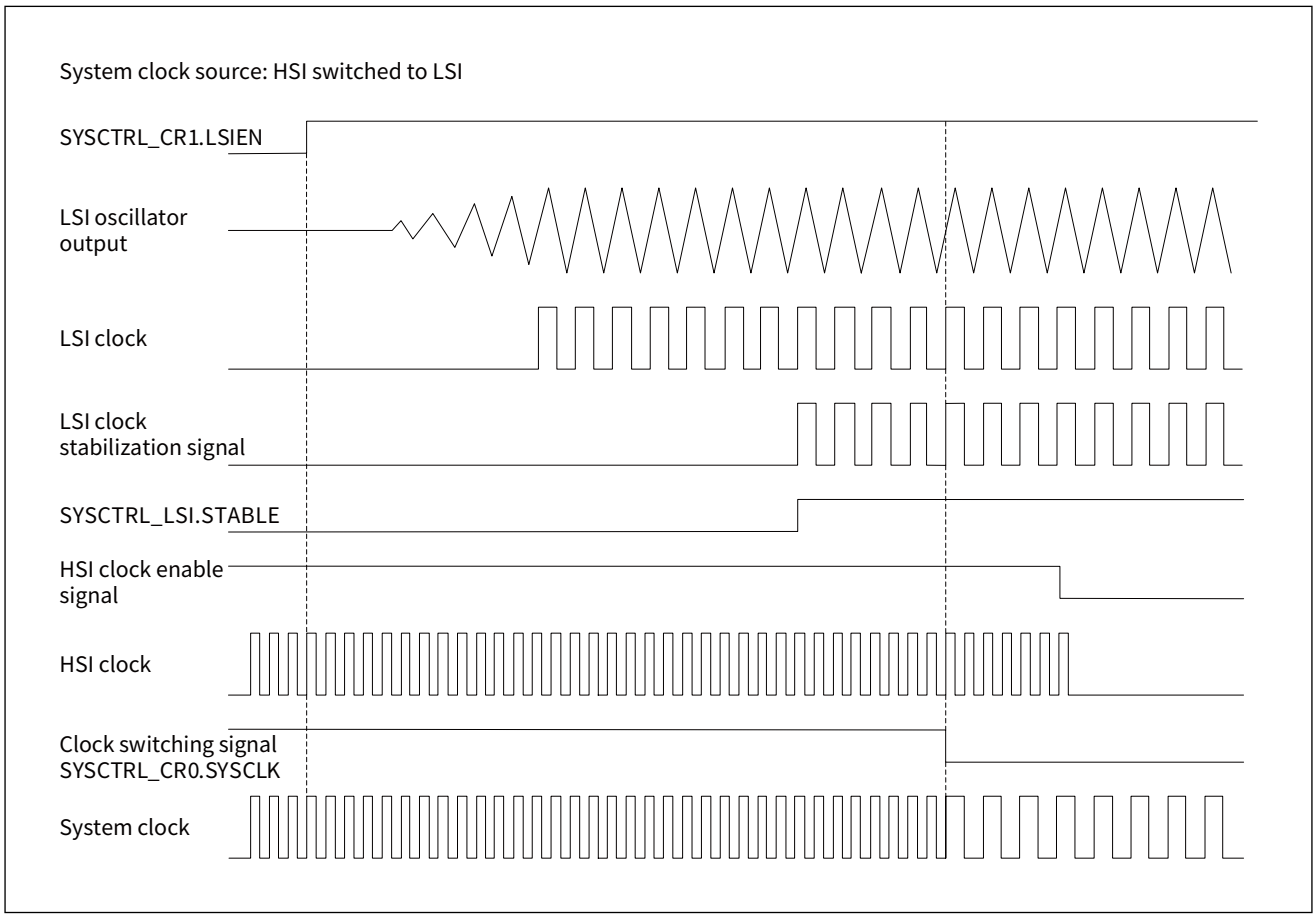
Step 6: Configure the system control register SYSCTRL_CR0.SYSCLK, and select the clock source of the system clock SysClk as the new clock source;

Step 7: According to the frequency of the new clock source, configure the FLASH_CR2.WAIT read waiting period parameter according to the requirements of the chapter *7 FLASH Memory*;

Step 8: Turn off clock sources that are no longer in use.

Taking the system clock switching from HSI to LSI as an example, the switching process of the old and new clocks is shown in the following figure:

Figure 4-4 HSI clock switched to LSI clock



### 4.5.2 HSI clock switching process between different frequencies

Switching between different frequencies of the HSI clock does not require starting a new clock source, but only needs to change the frequency division ratio of the HSI clock prescaler. The switching speed is fast, and there is no need to wait for the clock stabilization time before and after switching. The switching operation process is as follows:

Step 1: According to the higher frequency of the current clock source and the new clock source, configure the FLASH_CR2.WAIT read wait period parameter according to the requirements of the chapter *7 FLASH Memory*;

Step 2: Read the value of the built-in high-frequency clock control register SYSCTRL_HSI register as Value;

Step 3: According to the required target frequency, set the built-in high-frequency clock control register SYSCTRL_HSI.DIV to the corresponding frequency division coefficient (note that the SYSCTRL_HSI.TRIM field should remain unchanged);

Step 4: According to the frequency of the new clock source, configure the FLASH_CR2.WAIT read wait period parameter as required in the chapter *7 FLASH Memory*.

### 4.5.3 Example of switching from other clocks to HEX

The switching operation process is as follows:

Step 1: Set SYSCTRL_HEX.PINEN to 1, allowing the clock signal on the external pin to be input to the HEX circuit;

Step 2: Configure SYSCTRL_HEX.PINMUX, and configure the corresponding pin as a digital input function;

Step 3: Set the system control register SYSCTRL_CR1. HEXEN is 1, enabling the external input clock HEX;

*Caution:*

*The SYSCTRL_CR1 register has the KEY protection feature. The upper 16-bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 4: According to the higher frequency of the current clock and HEX, configure the FLASH_CR2.WAIT read waiting period parameter according to the requirements of the 7 FLASH memory chapter;

Step 5: Cyclic query and wait for the external input clock control register SYSCTRL_HEX.STABLE stable flag to change 1;

Step 6: Set the system control register SYSCTRL_CR0.SYSCLK to 1, and switch the SysClk clock source to HEX;

*Caution:*

*The SYSCTRL_CR0 register has the KEY protection feature. The upper 16-bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 7: According to the clock frequency of HEX, configure the FLASH_CR2.WAIT read waiting period parameter according to the requirements of th chapter *7 FLASH Memory*;

Step 8: Set the system control register SYSCTRL_CR1.xxxEN to 0, and turn off the original clock.

### 4.5.4 Example of switching from other clocks to LSI

The switching operation process is as follows:

Step 1: Configure the built-in low-frequency clock control register SYSCTRL_LSI.TRIM and the built-in low-frequency clock control register SYSCTRL_LSI.WAITCYCLE to appropriate values;

Step 2: Set the system control register SYSCTRL_CR1.LSIEN to 1 to enable the LSIRC oscillator circuit;

*Caution:*

*The SYSCTRL_CR1 register has the KEY protection feature. The upper 16-bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 3: Cyclic query and wait for the built-in low-frequency clock control register SYSCTRL_LSI.STABLE stable flag to become 1, that is, wait for the LSI to output a stable clock;

Step 4: Set the system control register SYSCTRL_CR0.SYSCLK to 3, and switch the SysClk clock source to LSI;

*Caution:*

*The SYSCTRL_CR0 register has the KEY protection feature. The upper 16-bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 5: Configure the FLASH_CR2.WAIT read wait period parameter to 0;

Step 6: Set the system control register SYSCTRL_CR1.xxxEN to 0 to turn off the original clock.

### 4.5.5 Example of switching from other clocks to HSI

The switching operation process is as follows:

Step 1: Configure the built-in high-frequency clock control registers SYSCTRL_HSI.TRIM and SYSCTRL_HSI.DIV to appropriate values;

Step 2: Set the system control register SYSCTRL_CR1.HSIEN to 1 to enable the HSIOSC clock oscillation circuit;

*Caution:*

*The SYSCTRL_CR1 register has the KEY protection feature. The upper 16-bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 3: Cyclic query and wait for the built-in high-frequency clock control register SYSCTRL_HSI.STABLE stable flag to become 1, that is, wait for the HSIOSC to output a stable clock;

Step 4: According to the higher frequency of the current clock and HSI, configure the FLASH_CR2.WAIT read waiting period parameter according to the requirements of the chapter *7 FLASH Memory*;

Step 5: Set the system control register SYSCTRL_CR0.SYSCLK to 0, and switch the SysClk clock source to HSI;

*Caution:*

*The SYSCTRL_CR0 register has the KEY protection feature. The upper 16-bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 6: According to the clock frequency of HSI, configure the FLASH_CR2.WAIT read waiting period parameter according to the requirements of the chapter *7 FLASH Memory*;

Step 7: Set the system control register SYSCTRL_CR1.xxxEN to 0 to turn off the original clock.

## 4.6 List of registers

SYSCTRL base address: SYSCTRL_BASE = 0x4001 0000

Table 4-3 List of SYSCTRL registers

| Register name | Register address | Register description |
|---|---|---|
| SYSCTRL_CR0 | SYSCTRL_BASE + 0x00 | System Control Register 0 |
| SYSCTRL_CR1 | SYSCTRL_BASE + 0x04 | System Control Register 1 |
| SYSCTRL_CR2 | SYSCTRL_BASE + 0x08 | System Control Register 2 |
| SYSCTRL_IER | SYSCTRL_BASE + 0x0C | System Interrupt Enable Control Register |
| SYSCTRL_ISR | SYSCTRL_BASE + 0x10 | System Interrupt Flag Register |
| SYSCTRL_ICR | SYSCTRL_BASE + 0x14 | System Interrupt Flag Clear Register |
| SYSCTRL_HSI | SYSCTRL_BASE + 0x18 | Internal high frequency clock control register |
| SYSCTRL_HEX | SYSCTRL_BASE + 0x1C | External Input Clock Control Register |
| SYSCTRL_LSI | SYSCTRL_BASE + 0x20 | Internal low frequency clock control register |
| SYSCTRL_DEBUG | SYSCTRL_BASE + 0x2C | Debug Status Timer Control Register |
| SYSCTRL_AHBEN | SYSCTRL_BASE + 0x30 | AHB Peripheral Clock Enable Control Register |
| SYSCTRL_APBEN2 | SYSCTRL_BASE + 0x34 | APB Peripheral Clock Enable Control Register 2 |
| SYSCTRL_APBEN1 | SYSCTRL_BASE + 0x38 | APB Peripheral Clock Enable Control Register 1 |
| SYSCTRL_AHBRST | SYSCTRL_BASE + 0x40 | AHB Peripheral Reset Control Register |
| SYSCTRL_APBRST2 | SYSCTRL_BASE + 0x44 | APB Peripheral Reset Control Register 2 |
| SYSCTRL_APBRST1 | SYSCTRL_BASE + 0x48 | APB Peripheral Reset Control Register 1 |
| SYSCTRL_RESETFLAG | SYSCTRL_BASE + 0x4C | System Reset Flag Register |
| SYSCTRL_GTIMCAP | SYSCTRL_BASE + 0x50 | General-Purpose Timer Input Capture Source Configuration Register |
| SYSCTRL_ATIMETR | SYSCTRL_BASE + 0x60 | Advanced Timer ETR Source Configuration Register |
| SYSCTRL_GTIMETR | SYSCTRL_BASE + 0x64 | General Purpose Timer ETR Source Configuration Register |
| SYSCTRL_TIMITR | SYSCTRL_BASE + 0x6C | Timer ITR Source Configuration Register |
| SYSCTRL_MCO | SYSCTRL_BASE + 0x70 | System Clock Output Control Register |

## 4.7 Register description

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 4.7.1 SYSCTRL_CR0 System Control Register 0

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | KEY | WO | Only when KEY is 0x5A5A, the write operation of this register is valid |
| 15:8 | RFU | - | Reserved bits, please keep the default value |
| 7:5 | HCLKPRS | RW | Configure the clock source of HCLK<br>000: Set the clock source of HCLK to SysClk<br>001: Set the clock source of HCLK to SysClk / 2<br>010: Set the clock source of HCLK to SysClk / 4<br>011: Set the clock source of HCLK to SysClk / 8<br>100: Set the clock source of HCLK to SysClk / 16<br>101: Set the clock source of HCLK to SysClk / 32<br>110: Set the clock source of HCLK to SysClk / 64<br>111: Set the clock source of HCLK to SysClk / 128 |
| 4:3 | PCLKPRS | RW | Configure the clock source of PCLK<br>00: Set the clock source of PCLK to HCLK<br>01: Set the clock source of PCLK to HCLK / 2<br>10: Set the clock source of PCLK to HCLK / 4<br>11: Set the clock source of PCLK to HCLK / 8 |
| 2:0 | SYSCLK | RW | Configure the clock source of SysClk<br>000: Set the clock source of SysClk to HSI<br>001: Set the clock source of SysClk to HEX<br>011: Set the clock source of SysClk to LSI |

### 4.7.2 SYSCTRL_CR1 System Control Register 1

Address offset: 0x04 Reset value: 0x0000 0001

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | KEY | WO | Only when KEY is 0x5A5A, the write operation of this register is valid |
| 15:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | LSIEN | RW | Internal low-speed clock LSI enable control<br>0: Disable<br>1: Enable<br>Caution: When the system enters DeepSleep, this low-speed clock will not automatically shut down. |
| 2 | RFU | - | Reserved bits, please keep the default value |
| 1 | HEXEN | RW | External input clock HEX enable control<br>0: Disable<br>1: Enable<br>Caution: When the system enters DeepSleep, this high-speed clock will be automatically turned off. |
| 0 | HSIEN | RW | Internal high-speed clock HSIOSC enable control<br>0: Disable<br>1: Enable<br>Caution: When the system enters DeepSleep, this high-speed clock will be automatically turned off. |

### 4.7.3 SYSCTRL_CR2 SYSCTRL_CR2 System Control Register 2

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | KEY | WO | Only when KEY is 0x5A5A, the write operation of this register is valid |
| 15:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | WAKEUPCLK | RW | When DeepSleep wakes up, the source configuration of the system clock<br>0: Keep the original system clock source<br>1: Switch the system clock source to HSI 8MHz, keep the original system clock source enabled |
| 2 | LOCKUP | RW | Cortex-M0+ LockUp Functional configuration<br>0: Disable<br>1: Enable<br>Caution: When this function is enabled, the CPU will reset the MCU when it reads an invalid instruction. |
| 1 | SWDIO | RW | SWD Port function configuration<br>0: PA02, PA05 are used as SWD ports, GPIO function is not available<br>1: PA02, PA05 are used as GPIO ports, SWD function is not available |
| 0 | RSTIO | RW | NRST Port function configuration<br>0: PC05 is used as NRST port, GPIO function is not available.<br>1: PC05 is used as GPIO port, NRST function is not available.<br>Caution: Any write to this register will cause this bit to be locked. After locking, this bit cannot be modified. Only POR power-on can unlock this bit. |

### 4.7.4 SYSCTRL_HSI Internal high frequency clock control register

Address offset: 0x18 Reset value: 0x---- ----

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15 | STABLE | RO | HSIOSC Clock Stable Status Bits<br>0: HSIOSC clock has not stabilized<br>1: HSIOSC clock has stabilized |
| 14:11 | DIV | RW | HSI clock and HSIOSC clock division factor configuration<br>0101: HSI = HSIOSC / 6 (Default)<br>0110: HSI = HSIOSC / 1<br>1000: HSI = HSIOSC / 2<br>1001: HSI = HSIOSC / 4<br>1011: HSI = HSIOSC / 8<br>1100: HSI = HSIOSC / 10<br>1101: HSI = HSIOSC / 12<br>1110: HSI = HSIOSC / 14<br>1111: HSI = HSIOSC / 16 |
| 10:0 | TRIM | RW | Clock frequency adjustment, changing the value of this register bit can adjust the oscillation frequency of the HSIOSC. Every time the TRIM value increases by 1, the oscillation frequency of the HSIOSC increases by about 0.2%, and the total adjustment range is 32~48MHz. The calibration value of 48MHz has been saved in FLASH, and the accurate frequency can be obtained by reading the calibration value in FLASH and writing it into SYSCTRL_HSI.TRIM.<br>48.00MHz Calibration value address: 0x0010 07B8 – 0x0010 07B9 |

### 4.7.5 SYSCTRL_LSI Internal low frequency clock control register

Address offset: 0x20 Reset value: 0x---- ----

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15 | STABLE | RO | LSI Clock Stable Status Bit<br>0: LSI clock has not stabilized<br>1: LSI clock has stabilized |
| 14:12 | RFU | - | Reserved bits, please keep the default value |
| 11:10 | WAITCYCLE | RW | Internal low-speed clock LSI stabilization time selection<br>00: 6 cycles<br>01: 18 cycles<br>10: 66 cycles<br>11: 258 cycles |
| 9:0 | TRIM | RW | Internal low-speed clock frequency adjustment, 1 set of frequency calibration values are saved in FLASH.<br>Read the calibration value in FLASH and write it to SYSCTRL_LSI.TRIM to get the exact frequency.<br>32.8K Calibration value address: 0x0010 07BA - 0x0010 07BB |

### 4.7.6 SYSCTRL_HEX External Input Clock Control Register

Address offset: 0x1C Reset value: 0x0007 FF22

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:20 | RFU | - | Reserved bits, please keep the default value |
| 19 | STABLE | RO | External input clock HEX stable status bit<br>0: HEX clock has not stabilized<br>1: HEX clock has stabilized |
| 18:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | PINMUX | RW | External clock input pin source configuration<br>0: External input clock comes from PB00 pin<br>1: External input clock comes from PB01 pin |
| 6 | PINEN | RW | External clock input pin enable control<br>0: The clock signal on the disable pin is input to the HEX circuit<br>1: Allows the clock signal on the pin to be input to the HEX circuit |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 4.7.7 SYSCTRL_IER System Interrupt Enable Control Register

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | KEY | WO | Only when KEY is 0x5A5A, the write operation of this register is valid |
| 15:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | LSIRDY | RW | LSI Stable Interrupt Enable Control<br>0: Disable LSI stable interrupt<br>1: Enable LSI stable interrupt |
| 2 | RFU | - | Reserved bits, please keep the default value |
| 1 | HEXRDY | RW | HEX Stable Interrupt Enable Control<br>0: Disable HEX stable interrupt<br>1: Enable HEX stable interrupt |
| 0 | HSIRDY | RW | HSIOSC Stable Interrupt Enable Control<br>0: Disable HSIOSC stable interrupt<br>1: Enable HSIOSC stable interrupt |

### 4.7.8 SYSCTRL_ISR System Interrupt Flag Register

Address offset: 0x10 Reset value: 0x0000 0801

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:15 | RFU | - | Reserved bits, please keep the default value |
| 14 | LSISTABLE | RO | LSI clock stable status bit; hardware set to clear<br>0: LSI clock has not stabilized<br>1: LSI clock has stabilized |
| 13 | RFU | - | Reserved bits, please keep the default value |
| 12 | HEXSTABLE | RO | HEX clock stable status bit; hardware set to clear<br>0: HEX clock has not stabilized<br>1: HEX clock has stabilized |
| 11 | HSISTABLE | RO | HSIOSC clock stable status bit; hardware set to clear<br>0: HSIOSC clock has not stabilized<br>1: HSIOSC clock has stabilized |
| 10:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | LSIRDY | RO | LSI Clock Stable Flag<br>0: LSI clock has not stabilized<br>1: LSI clock has stabilized |
| 2 | RFU | - | Reserved bits, please keep the default value |
| 1 | HEXRDY | RO | HEX Clock Stable Flag<br>0: HEX clock has not stabilized<br>1: HEX clock has stabilized |
| 0 | HSIRDY | RO | HSIOSC Clock Stable Flag<br>0: HSIOSC clock has not stabilized<br>1: HSIOSC clock has stabilized |

### 4.7.9 SYSCTRL_ICR System Interrupt Flag Clear Register

Address offset: 0x14 Reset value: 0x0000 000B

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | LSIRDY | R1W0 | LSI Clock stable flag clear control<br>W0: Clear the corresponding flag in the ISR register<br>W1: No function |
| 2 | RFU | - | Reserved bits, please keep the default value |
| 1 | HEXRDY | R1W0 | HEX Clock stable flag clear control<br>W0: Clear the corresponding flag in the ISR register<br>W1: No function |
| 0 | HSIRDY | R1W0 | HSIOSC Clock stable flag clear control<br>W0: Clear the corresponding flag in the ISR register<br>W1: No function |

### 4.7.10 SYSCTRL_AHBEN AHB Peripheral Clock Enable Control Register

Address offset: 0x30 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | GPIOC | RW | GPIOC Port configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 5 | GPIOB | RW | GPIOB Port configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 4 | GPIOA | RW | GPIOA Port configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 3 | RFU | - | Reserved bits, please keep the default value |
| 2 | CRC | RW | CRC Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 1 | FLASH | RW | FLASH Module configuration clock enable control<br>0: Disable<br>1: Enable |
| 0 | RFU | - | Reserved bits, please keep the default value |

### 4.7.11 SYSCTRL_APBEN1 APB Peripheral Clock Enable Control Register 1

Address offset: 0x38 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:12 | RFU | - | Reserved bits, please keep the default value |
| 11 | I2C | RW | I2C Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 10:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | UART2 | RW | UART2 Module configuration clock enable control<br>0: Disable<br>1: Enable |
| 6 | RFU | - | Reserved bits, please keep the default value |
| 5 | IWDT | RW | IWDT Module configuration clock enable control<br>0: Disable<br>1: Enable |
| 4 | WWDT | RW | WWDT Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 3:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | GTIM | RW | GTIM Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 0 | RFU | - | Reserved bits, please keep the default value |

### 4.7.12 SYSCTRL_APBEN2 APB Peripheral Clock Enable Control Register 2

Address offset: 0x34 Reset value: 0x0000 8000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:14 | RFU | - | Reserved bits, please keep the default value |
| 13 | AWT | RW | AWT Module configuration clock enable control<br>0: Disable<br>1: Enable |
| 12 | BTIM | RW | BTIM1-3 Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 11:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | UART1 | RW | UART1 Module configuration clock enable control<br>0: Disable<br>1: Enable |
| 8 | SPI | RW | SPI Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 7 | ATIM | RW | ATIM Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 6:5 | RFU | - | Reserved bits, please keep the default value |
| 4 | VC | RW | VC Module configuration clock enable control<br>0: Disable<br>1: Enable |
| 3 | RFU | RW | Reserved bits, please keep the default value |
| 2 | ADC | RW | ADC Module configuration clock and working clock enable control<br>0: Disable<br>1: Enable |
| 1:0 | RFU | - | Reserved bits, please keep the default value |

### 4.7.13 SYSCTRL_AHBRST AHB Peripheral Reset Control Register

Address offset: 0x40 Reset value: 0x0000 0076

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | GPIOC | RW | GPIOC Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 5 | GPIOB | RW | GPIOB Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 4 | GPIOA | RW | GPIOA Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 3 | RFU | - | Reserved bits, please keep the default value |
| 2 | CRC | RW | CRC Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 1 | FLASH | RW | FLASH Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 0 | RFU | - | Reserved bits, please keep the default value |

### 4.7.14 SYSCTRL_APBRST1 APB Peripheral Reset Control Register 1

Address offset: 0x48 Reset value: 0x0000 08B3

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:12 | RFU | - | Reserved bits, please keep the default value |
| 11 | I2C | RW | I2C Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 10:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | UART2 | RW | UART2 Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 6 | RFU | - | Reserved bits, please keep the default value |
| 5 | IWDT | RW | IWDT Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 4 | WWDT | RW | WWDT Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 3:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | GTIM | RW | GTIM Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 0 | RFU | - | Reserved bits, please keep the default value |

### 4.7.15 SYSCTRL_APBRST2 APB Peripheral Reset Control Register 2

Address offset: 0x44 Reset value: 0x0000 339C

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:14 | RFU | - | Reserved bits, please keep the default value |
| 13 | AWT | RW | AWT Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 12 | BTIM | RW | BTIM1-3 Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 11:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | UART1 | RW | UART1 Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 8 | SPI | RW | SPI Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 7 | ATIM | RW | ATIM Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 6:5 | RFU | - | Reserved bits, please keep the default value |
| 4 | VC | RW | VC Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 3 | RFU | - | Reserved bits, please keep the default value |
| 2 | ADC | RW | ADC Module reset control<br>0: Module is in reset state<br>1: Module works fine |
| 1:0 | RFU | - | Reserved bits, please keep the default value |

### 4.7.16 SYSCTRL_RESETFLAG System Reset Flag Register

Address offset: 0x4C Reset value: 0x---- ----

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | SYSRESETREQ | RW0 | Cortex-M0+ CPU SYSRESETREQ soft reset flag<br>0: No CPU SYSRESETREQ soft reset occurred<br>1: CPU SYSRESETREQ soft reset has occurred<br>Write 0 to clear, write 1 to invalid |
| 8 | LOCKUP | RW0 | Cortex-M0+ CPU Lockup reset flag<br>0: No Lockup reset occurred<br>1: Lockup reset has occurred<br>Write 0 to clear, write 1 to invalid |
| 7 | RFU | - | Reserved bits, please keep the default value |
| 6 | RSTB | RW0 | NRST pin reset flag<br>0: No Pin reset occurred<br>1: Pin reset has occurred<br>Write 0 to clear, write 1 to invalid |
| 5 | WWDT | RW0 | WWDT reset flag, requires software initialization and clearing<br>0: No WWDT reset occurred<br>1: WWDT reset has occurred<br>Write 0 to clear, write 1 to invalid |
| 4 | IWDT | RW0 | IWDT reset flag<br>0: No IWDT reset occurred<br>1: IWDT reset has occurred<br>Write 0 to clear, write 1 to invalid |
| 3 | LVD | RW0 | LVD reset flag<br>0: No LVD reset occurred<br>1: LVD reset has occurred<br>Write 0 to clear, write 1 to invalid |
| 2:1 | RFU | - | Reserved bits, please keep the default value |
| 0 | POR | RW0 | POR/BOR reset flag<br>0: No POR/BOR reset occurred<br>1: POR/BOR reset has occurred<br>Write 0 to clear, write 1 to invalid |

### 4.7.17 SYSCTRL_DEBUG Debug Status Timer Control Register

Address offset: 0x2C Reset value: 0x0000 06E3F

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:11 | RFU | - | Reserved bits, please keep the default value |
| 10 | WWDT | RW | In the debugging state, WWDT count function configuration<br>0: Normal count<br>1: Pause count |
| 9 | IWDT | RW | In the debugging state, IWDT count function configuration<br>0: Normal count<br>1: Pause count |
| 8:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | AWT | RW | In the debugging state, AWT count function configuration<br>0: Normal count<br>1: Pause count |
| 5 | BTIM123 | RW | In the debugging state, BTIM1 / BTIM2 / BTIM3 count function configuration<br>0: Normal count<br>1: Pause count |
| 4:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | GTIM | RW | In the debugging state, GTIM count function configuration<br>0: Normal count<br>1: Pause count |
| 0 | ATIM | RW | In the debugging state, ATIM count function configuration<br>0: Normal count<br>1: Pause count |

### 4.7.18 SYSCTRL_GTIMCAP General-Purpose Timer Input Capture Source Configuration Register

Address offset: 0x50 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:15 | RFU | - | Reserved bits, please keep the default value |
| 14:12 | CH4 | RW | CH4 input capture source configuration of GTIM<br>See GTIMCAP [2:0] |
| 11 | RFU | - | Reserved bits, please keep the default value |
| 10:8 | CH3 | RW | CH3 input capture source configuration of GTIM<br>See GTIMCAP [2:0] |
| 7 | RFU | - | Reserved bits, please keep the default value |
| 6:4 | CH2 | RW | CH2 input capture source configuration of GTIM<br>See GTIMCAP [2:0] |
| 3 | RFU | - | Reserved bits, please keep the default value |
| 2:0 | CH1 | RW | CH1 input capture source configuration of GTIM<br>000: Configured by GPIOx_AFRL<br>001: RXD signal of UART1<br>010: RXD signal of UART2<br>100: Compare output signal of VC1<br>101: Compare output signal of VC2<br>110: LVD output signal |

### 4.7.19 SYSCTRL_ATIMETR Advanced Timer ETR Source Configuration

Address offset: 0x60 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:3 | RFU | - | Reserved bits, please keep the default value |
| 2:0 | ATIMETR | RW | ETR source configuration for ATIM<br>See GTIMETR [2:0] |

### 4.7.20 SYSCTRL_GTIMETR General Purpose Timer ETR Source Configuration Register

Address offset: 0x64 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:3 | RFU | - | Reserved bits, please keep the default value |
| 2:0 | GTIMETR | RW | ETR source configuration for GTIM<br>000: Configured by GPIOx_AFRL<br>001: RXD signal of UART1<br>010: RXD signal of UART2<br>100: Compare output signal of VC1<br>101: Compare output signal of VC2<br>110: LVD output signal |

### 4.7.21 SYSCTRL_TIMITR Timer ITR Source Configuration Register

Address offset: 0x6C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:24 | RFU | - | Reserved bits, please keep the default value |
| 23:21 | BTIM3ITR | RW | ITR source configuration for BTIM3<br>See TIMITR [2:0] |
| 20:18 | BTIM2ITR | RW | ITR source configuration for BTIM2<br>See TIMITR [2:0] |
| 17:15 | BTIM1ITR | RW | ITR source configuration for BTIM1<br>See TIMITR [2:0] |
| 14:6 | RFU | - | Reserved bits, please keep the default value |
| 5:3 | GTIMITR | RW | ITR source configuration for GTIM<br>See TIMITR [2:0] |
| 2:0 | ATIMITR | RW | ITR source configuration for ATIM<br>000: The overflow signal of BTIM1<br>001: The overflow signal of BTIM2<br>010: The overflow signal of BTIM3<br>011: The overflow signal of GTIM<br>111: Master mode output signal of ATIM |

*Caution:*

*The ITR source of the TIM cannot be configured as its own overflow signal.*

### 4.7.22 SYSCTRL_MCO System Clock Output Control Register

Address offset: 0x70 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6:4 | DIV | RW | MCO output frequency division control<br>000:1 Frequency Division<br>001:2 Frequency Division<br>010:8 Frequency Division<br>011:64 Frequency Division<br>100:128 Frequency Division<br>101:256 Frequency Division<br>110:512 Frequency Division<br>111:1024 Frequency Division |
| 3:0 | SOURCE | RW | MCO output signal source configuration<br>0000: Zero output<br>0001: HCLK<br>0010: PCLK<br>0011: HSIOSC<br>0100: LSI<br>0101: HEX<br>1000: RC150K<br>1001: RC10K |

# 5 Interrupt

## 5.1 Overview

Nested Vectored Interrupt Controller (NVIC) for the ARM® Cortex®-M0+ core to manage interrupts and exceptions. The NVIC is tightly coupled to the processor core, enabling low-latency exception and interrupt handling.

The processor supports up to 32 interrupt request (IRQ) inputs and supports multiple internal exceptions.

This chapter only introduces the 32 external interrupt requests (IRQ0 ~ IRQ31) of the processor. Please refer to the "ARM® Cortex®-M0+Technical Reference Manual" and "ARM® v6-M Architecture Reference Manual" for details of the processor's internal exceptions.

## 5.2 Main features

- 16 inner exceptions
- 32 maskable external interrupts
- 4 programmable priorities
- Low-latency exception and interrupt handling
- Support for interrupt nesting
- Interrupt vector table remapping

## 5.3 Interrupt priority

The external interrupt can be set to 4 levels of priority, the highest priority is "0", the lowest priority is "3", and the default value is "0".

If a higher priority interrupt occurs while the processor is executing an interrupt handler, the interrupt is preempted. If an interrupt occurs with the same or lower priority as the interrupt being processed, the interrupt will not be preempted, but the state of the new interrupt will change to pending. If multiple pending interrupts have the same priority, the pending interrupt with the smaller interrupt number takes precedence. For example, if both IRQ [0] and IRQ [1] are pending, and both have the same priority, then IRQ [0] is processed first.

## 5.4 Interrupt vector table

When the ARM® Cortex®-M0+ responds to an interrupt, the processor automatically fetches the start address of the Interrupt Service Routine (ISR) from the interrupt vector table in the memory. The interrupt vector table includes the initial value of the main stack pointer (MSP), the entry address of the service routine for internal exceptions and external interrupts. Each interrupt vector occupies 1 word (4 bytes), and the storage address of the interrupt vector is the vector number multiplied by 4, as shown in the following table:

Table 5-1 Interrupt vector table

| Vector number | External interrupt number (IRQ#) | Priority | Interrupt source | Introduction | Address |
|---|---|---|---|---|---|
| 0 | - | - | - | MSP initial value | 0x0000 0000 |
| 1 | - | -3 | Reset | Reset vector | 0x0000 0004 |
| 2 | - | -2 | NMI | Non-maskable interrupt [2] | 0x0000 0008 |
| 3 | - | -1 | HardFault | Hardware error exception (fault) | 0x0000 000C |
| 4-10 | - | - | - | Reserved | 0x0000 0010~002B |
| 11 | - | Configurable | SVCall | Hypervisors invoked via SWI instructions | 0x0000 002C |
| 12-13 | - | - | - | Reserved | 0x0000 0030~0037 |
| 14 | - | Configurable | PendSV | Suspendable requests for system services | 0x0000 0038 |
| 15 | - | Configurable | SysTick | System tick timer | 0x0000 003C |
| 16 | 0 | Configurable | WDT[1] | Windowed watchdog and independent watchdog interrupts | 0x0000 0040 |
| 17 | 1 | Configurable | LVD | LVD global interrupt | 0x0000 0044 |
| 18 | 2 | - | - | Reserved | 0x0000 0048 |
| 19 | 3 | Configurable | FLASHRAM[1] | FLASH/RAM global interrupt | 0x0000 004C |
| 20 | 4 | Configurable | RCC | RCC global interrupt [3] | 0x0000 0050 |
| 21 | 5 | Configurable | GPIOA | GPIOA global interrupt | 0x0000 0054 |
| 22 | 6 | Configurable | GPIOB | GPIOB global interrupt | 0x0000 0058 |
| 23 | 7 | Configurable | GPIOC | GPIOC global interrupt | 0x0000 005C |
| 24 | 8 | - | - | Reserved | 0x0000 0060 |
| 25 | 9 | - | - | Reserved | 0x0000 0064 |
| 26 | 10 | - | - | Reserved | 0x0000 0068 |
| 27 | 11 | - | - | Reserved | 0x0000 006C |

| Vector number | External interrupt number (IRQ#) | Priority | Interrupt source | Introduction | Address |
|---|---|---|---|---|---|
| 28 | 12 | Configurable | ADC | ADC global interrupt | 0x0000 0070 |
| 29 | 13 | Configurable | ATIM | ATIM global interrupt | 0x0000 0074 |
| 30 | 14 | Configurable | VC1 | VC1 global interrupt | 0x0000 0078 |
| 31 | 15 | Configurable | VC2 | VC2 global interrupt | 0x0000 007C |
| 32 | 16 | Configurable | GTIM | GTIM global interrupt | 0x0000 0080 |
| 33 | 17 | - | - | Reserved | 0x0000 0084 |
| 34 | 18 | - | - | Reserved | 0x0000 0088 |
| 35 | 19 | - | - | Reserved | 0x0000 008C |
| 36 | 20 | Configurable | BTIM1 | BTIM1 global interrupt | 0x0000 0090 |
| 37 | 21 | Configurable | BTIM2 | BTIM2 global interrupt | 0x0000 0094 |
| 38 | 22 | Configurable | BTIM3 | BTIM3 global interrupt | 0x0000 0098 |
| 39 | 23 | Configurable | I2C | I2C global interrupt | 0x0000 009C |
| 40 | 24 | - | - | Reserved | 0x0000 00A0 |
| 41 | 25 | Configurable | SPI | SPI global interrupt | 0x0000 00A4 |
| 42 | 26 | - | - | Reserved | 0x0000 00A8 |
| 43 | 27 | Configurable | UART1 | UART1 global interrupt | 0x0000 00AC |
| 44 | 28 | Configurable | UART2 | UART2 global interrupt | 0x0000 00B0 |
| 45 | 29 | - | - | Reserved | 0x0000 00B4 |
| 46 | 30 | Configurable | AWT | AWT global interrupt | 0x0000 00B8 |
| 47 | 31 | - | - | Reserved | 0x0000 00BC |

*Caution 1:*

*Since the interrupts of some peripherals multiplex an IRQ interrupt source, the user should first check the interrupt flag bit in the interrupt service routine to determine the peripheral that generates the interrupt.*

*Caution 2:*

*NMI is not used in CW32F003.*

*Caution 3:*

*LSI, HSIOSC and HEX clock signal stability correspond to RCC global interrupt.*

## 5.5 Interrupt related registers

### 5.5.1 NVIC interrupt enable and disable enable

The ARM® Cortex-M0+ processor supports up to 32 external interrupt sources, corresponding to the 32 enable bits of the interrupt enable setting register NIVC_ISER and the 32 disable bits of the interrupt enable clear register NVIC_ICER. Setting the enable bit to 1 enables interrupts; Set the disable bit to 1, disable interrupts.

The above NVIC interrupt enable is only for the processor NVIC, whether the interrupt of the peripheral is enabled is also controlled by the interrupt control register of the corresponding peripheral.

### 5.5.2 NVIC interrupt pending and clear pending

When an interrupt occurs, if the system is processing an interrupt of the same priority or higher, the system will not process the interrupt immediately, but will set the interrupt status to pending and save it in the interrupt pending status register; The pending state will remain valid until the processor enters this interrupt processing, unless the pending state is manually cleared. When the processor begins to enter the interrupt processing, the hardware will automatically clear the corresponding interrupt pending state.

The user can set the status of this interrupt to the pending state by setting the corresponding bit of the interrupt pending setting register NVIC_ISPR. If the system is not processing an interrupt with the same priority or higher priority, the interrupt will be immediately responded and deal with.

The user can set the state of this interrupt to the pending clear state by setting the corresponding bit of the interrupt pending clear register NVIC_ICPR.

### 5.5.3 NVIC interrupt priority

The interrupt priority control registers NVIC_IPR0~NVIC_IPR7 are used to set the interrupt priority of IRQ0~IRQ31. Each interrupt source uses 8 bits. In CW32F003, only the upper two bits are used, and up to 4 interrupt priorities can be set.

*Caution:*

*The setting of the interrupt priority register of ARM® Cortex-M0+ should be before the interrupt is enabled. The user cannot change the interrupt priority after the interrupt is enabled, which will lead to unpredictable results.*

### 5.5.4 NVIC interrupt mask

In some special occasions, all interrupts need to be disabled, which can be realized by using the interrupt mask register PRIMASK. Only the lowest 1 bit of PRIMASK is valid. If this bit is set to 1, all external interrupts and exceptions except NMI and hardware error exceptions are disabled; after clearing to 0, it is allowed to respond to interrupts and exceptions. This bit defaults to 0 after reset.

ARM® Cortex-M0+ has dedicated ARM instructions for modifying PRIMASK register, CPSIE i and CPSID i, please refer to "ARM® v6-M Architecture Reference Manual" for details.

Assembly instruction example reference:

```
CPSIE i        ; Clear PRIMASK (Enable interrupt)
CPSID i        ; Set up PRIMASK (Disable interrupts)
```

C-language (Calling the CMSIS device driver library) Example reference:

```
void __enable_irq(void);        //Clear PRIMASK
void __disable_irq(void);       //Set up PRIMASK
```

### 5.5.5 Peripheral interrupt enable

Peripheral modules generally have their own interrupt enable registers. When using interrupts, you must first enable the peripheral interrupt enable, and refer to *Table 5-1 Interrupt vector table* to enable the NVIC interrupt for the interrupt source. For the interrupt enable of specific peripherals, please refer to the description of the relevant peripheral modules.

## 5.6 List of registers

NVIC base address: NVIC_BASE = 0xE000 E000

Table 5-2 List of NVIC registers

| Register name | Register address | Register description |
|---|---|---|
| NVIC_ISER | NVIC_BASE + 0x100 | IRQ0~IRQ31 Interrupt Enable Setting Register |
| NVIC_ICER | NVIC_BASE + 0x180 | IRQ0~IRQ31 Interrupt Enable Clear Register |
| NVIC_ISPR | NVIC_BASE + 0x200 | IRQ0~IRQ31 Interrupt Pending Setup Register |
| NVIC_ICPR | NVIC_BASE + 0x280 | IRQ0~IRQ31 Interrupt Pending Clear Register |
| NVIC_IPR0 | NVIC_BASE + 0x400 | IRQ0~IRQ3 Interrupt Priority Control Register 0 |
| NVIC_IPR1 | NVIC_BASE + 0x404 | IRQ4~IRQ7 Interrupt Priority Control Register 1 |
| NVIC_IPR2 | NVIC_BASE + 0x408 | IRQ8~IRQ11 Interrupt Priority Control Register 2 |
| NVIC_IPR3 | NVIC_BASE + 0x40C | IRQ12~IRQ15 Interrupt Priority Control Register 3 |
| NVIC_IPR4 | NVIC_BASE + 0x410 | IRQ16~IRQ19 Interrupt Priority Control Register 4 |
| NVIC_IPR5 | NVIC_BASE + 0x414 | IRQ20~IRQ23 Interrupt Priority Control Register 5 |
| NVIC_IPR6 | NVIC_BASE + 0x418 | IRQ24~IRQ27 Interrupt Priority Control Register 6 |
| NVIC_IPR7 | NVIC_BASE + 0x41C | IRQ28~IRQ31 Interrupt Priority Control Register 7 |

## 5.7 Register description

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 5.7.1 NVIC_ISER Interrupt Enable Setting Register

Address offset: 0x100 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:0 | SETIRQ | RW | Set to enable external interrupt IRQ0~IRQ31; write "1" to set, write "0" is invalid.<br>[0]: IRQ0<br>[1]: IRQ1<br>[2]: IRQ2<br>……<br>[31]: IRQ31<br>The read value indicates the current interrupt enable state |

### 5.7.2 NVIC_ICER Interrupt Enable Clear Register

Address offset: 0x180 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:0 | CLRIRQ | RW | Set to disable external interrupt IRQ0~IRQ31; write "1" to set, write "0" is invalid.<br>[0]: IRQ0<br>[1]: IRQ1<br>[2]: IRQ2<br>……<br>[31]: IRQ31<br>The read value indicates the current interrupt enable state |

### 5.7.3 NVIC_ISPR Interrupt Pending Setup Register

Address offset: 0x200 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:0 | SETPEND | RW | Set the pending state of external interrupt IRQ0~IRQ31; write "1" to set, write "0" is invalid.<br>[0]: IRQ0<br>[1]: IRQ1<br>[2]: IRQ2<br>……<br>[31]: IRQ31<br>The read value indicates the current interrupt pending state |

### 5.7.4 NVIC_ICPR Interrupt Pending Clear Register

Address offset: 0x280 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:0 | CLRPEND | RW | Clear the pending state of external interrupt IRQ0~IRQ31; write "1" to clear, write "0" is invalid.<br>[0]: IRQ0<br>[1]: IRQ1<br>[2]: IRQ2<br>……<br>[31]: IRQ31<br>The read value indicates the current interrupt pending state |

### 5.7.5 NVIC_IPR0 Interrupt Priority Control Register 0

Address offset: 0x400 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ3 | RW | The priority of interrupt IRQ3, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ2 | RW | The priority of interrupt IRQ2, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ1 | RW | The priority of interrupt IRQ1, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ0 | RW | The priority of interrupt IRQ0, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 5.7.6 NVIC_IPR1 Interrupt Priority Control Register 1

Address offset: 0x404 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ7 | RW | The priority of interrupt IRQ7, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ6 | RW | The priority of interrupt IRQ6, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ5 | RW | The priority of interrupt IRQ5, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ4 | RW | The priority of interrupt IRQ4, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 5.7.7 NVIC_IPR2 Interrupt Priority Control Register 2

Address offset: 0x408 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ11 | RW | The priority of interrupt IRQ11, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ10 | RW | The priority of interrupt IRQ10, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ9 | RW | The priority of interrupt IRQ9, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ8 | RW | The priority of interrupt IRQ8, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 5.7.8 NVIC_IPR3 Interrupt Priority Control Register 3

Address offset: 0x40C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ15 | RW | The priority of interrupt IRQ15, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ14 | RW | The priority of interrupt IRQ14, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ13 | RW | The priority of interrupt IRQ13, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ12 | RW | The priority of interrupt IRQ12, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 5.7.9 NVIC_IPR4 Interrupt Priority Control Register 4

Address offset: 0x410 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ19 | RW | The priority of interrupt IRQ19, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ18 | RW | The priority of interrupt IRQ18, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ17 | RW | The priority of interrupt IRQ17, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ16 | RW | The priority of interrupt IRQ16, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 5.7.10 NVIC_IPR5 Interrupt Priority Control Register 5

Address offset: 0x414 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ23 | RW | The priority of interrupt IRQ23, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ22 | RW | The priority of interrupt IRQ22, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ21 | RW | The priority of interrupt IRQ21, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ20 | RW | The priority of interrupt IRQ20, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 5.7.11 NVIC_IPR6 Interrupt Priority Control Register 6

Address offset: 0x418 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ27 | RW | The priority of interrupt IRQ27, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ26 | RW | The priority of interrupt IRQ26, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ25 | RW | The priority of interrupt IRQ25, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ24 | RW | The priority of interrupt IRQ24, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

### 5.7.12 NVIC_IPR7 Interrupt Priority Control Register 7

Address offset: 0x41C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | IPRIRQ31 | RW | The priority of interrupt IRQ31, 00 has the highest priority and 11 has the lowest priority |
| 29:24 | RFU | - | Reserved bits, please keep the default value |
| 23:22 | IPRIRQ30 | RW | The priority of interrupt IRQ30, 00 has the highest priority and 11 has the lowest priority |
| 21:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | IPRIRQ29 | RW | The priority of interrupt IRQ29, 00 has the highest priority and 11 has the lowest priority |
| 13:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | IPRIRQ28 | RW | The priority of interrupt IRQ28, 00 has the highest priority and 11 has the lowest priority |
| 5:0 | RFU | - | Reserved bits, please keep the default value |

# 6 RAM Memory

## 6.1 Overview

CW32F003 integrates 3KB embedded RAM for users to use, which is used to store various data in the process of program execution. The starting address of the RAM is 0x2000 0000, and the data is stored in the little-endian mode in the RAM, that is, the lowest byte address space stores the least significant byte data of the data.

## 6.2 Main features

- Support access in 3 widths of byte (8bit), half word (16bit) or full word (32bit)
- Zero latency, can be accessed by the CPU at the maximum system clock frequency
- Support parity check function

## 6.3 RAM Memory operation

The RAM memory operations that the user can perform include: read operations and write operations.

The read and write operations of RAM support three-bit widths of 8bit, 16bit and 32bit. The user program can complete the read and write by directly accessing the absolute address, but it should be noted that the data bit width of read and write must be aligned with the corresponding address boundary, otherwise read and write the operation is invalid and will result in a HardFault hardware error exception.

### 6.3.1 Read operation

The read operation supports 3 different bit widths, which can be read by direct access to the absolute address, but it should be noted that the read data bit width must be aligned with the corresponding address boundary.

Code example:

8bit read:

```
tempdata = * ( ( uint8_t * ) 0x2000 0001 ) ;
```

16bit read:

```
tempdata = * ( ( uint16_t * ) 0x2000 0002 ) ;
```

32bit read:

```
tempdata = * ( ( uint32_t * ) 0x2000 0004 ) .
```

### 6.3.2 Write operation

The write operation supports 3 different bit widths, and data can be written by directly accessing the absolute address, but it should be noted that the written data bit width must be aligned with the corresponding address boundary.

Code example:

8bit write:

```
* ( ( uint8_t * ) 0x2000 0001 ) = 0x12;
```

16bit write:

```
* ( ( uint16_t * ) 0x2000 0002 ) = 0x1234;
```

32bit write:

```
* ( ( uint32_t * ) 0x2000 0004 ) = 0x1234 5678.
```

## 6.4 Parity check function

CW32F003 supports the parity check function of RAM. After power-on, the parity check function is enabled by default and cannot be configured by the user.

Each byte of RAM data is actually stored in a 9bit physical space, including 8bit data bits and 1bit parity bits.

When the CPU writes to the RAM, the parity check unit of the RAM calculates the check digit and writes it into the corresponding check digit space. When reading RAM data, the data is read together with the check digit. The CPU performs parity check on each byte of the data, and compares the calculated check digit with the read check digit. If they are consistent, it means that the data in the RAM is correct. If it is inconsistent, the parity error flag RAM_ISR.PARITY is set. If the RAM parity error interrupt enables control bit RAM_IER.PARITY is set to 1, the CPU will respond to the interrupt service. The user program can set RAM_ICR.PARITY to 0 to clear the parity error flag.

The user can obtain the RAM address where the parity error occurs by reading the parity error address register RAM_ADDR.

## 6.5 List of registers

RAM base address: RAM_BASE = 0x4002 2400

Table 6-1 List of RAM registers

| Register name | Register address | Register description |
|---|---|---|
| RAM_IER | RAM_BASE + 0x00 | Interrupt Enable Control Register |
| RAM_ADDR | RAM_BASE + 0x04 | Parity Error Address Register |
| RAM_ISR | RAM_BASE + 0x08 | Interrupt Flag Register |
| RAM_ICR | RAM_BASE + 0x0C | Interrupt Flag Clear Register |

## 6.6 Register description

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 6.6.1 RAM_ADDR Parity Error Address Register

Address offset: 0x04 Reset value: 0x2000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:0 | ADDR | RO | The address of the RAM where the parity error is located |

### 6.6.2 RAM_IER Interrupt Enable Control Register

Address offset: 0x00 Reset value: 0x0000 0001

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | PARITY | RW | RAM parity error interrupt enable control<br>0: Disable<br>1: Enable |
| 0 | EN | RO | RAM parity enable flag<br>0: Disable<br>1: Enable |

### 6.6.3 RAM_ISR Interrupt Flag Register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:1 | RFU | - | Reserved bits, please keep the default value |
| 0 | PARITY | RO | Parity error flag<br>0: No parity error occurred<br>1: A parity error has occurred |

### 6.6.4 RAM_ICR Interrupt Flag Clear Register

Address offset: 0x0C Reset value: 0x0000 0001

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:1 | RFU | - | Reserved bits, please keep the default value |
| 0 | PARITY | R1W0 | Parity Error Flag Clear Control<br>W0: Clear parity error flag<br>W1: No function |

# 7 FLASH Memory

## 7.1 Overview

CW32F003 integrates 20KB embedded FLASH for users to use, which can be used to store application programs and user data. The chip supports read, erase and write operations of FLASH memory, and supports erase and write protection and read protection; the chip has a built-in high-voltage BOOST circuit required for FLASH programming, and no additional programming voltage is required.

## 7.2 Main features

- Support access in 3 widths of byte (8bit), half word (16bit) or full word (32bit)
- Supports read, erase and write operations
- Support erase and write protection, read protection
- Support low-power mode

## 7.3 FLASH Memory organization

CW32F003 integrates 20KB user-accessible FLASH memory, which is managed by paging of 512 bytes per page, with a total of 40 pages. Users can perform full-page erasing and byte-by-byte programming operations on FLASH.

In addition to the user-accessible 20KB FLASH memory, the CW32F003 also integrates a 2.5KB startup program memory, which solidified with BootLoader and other codes when it leaves the factory, and the user cannot access this memory.

## 7.4 FLASH memory read wait cycle configuration

The internal FLASH memory of CW32F003 supports the operation clock with the fastest frequency of 24MHz. When the configured HCLK frequency is greater than 24MHz, the number of inserted waiting HCLK cycles needs to be configured through the WAIT bit field of the FLASH control register FLASH_CR2. The configuration rules are shown in the following table:

Table 7-1 FLASH read wait cycle configuration

| FLASH_CR2.WAIT | The number of HCLK cycles waiting for the instruction fetch cycle | HCLK Frequency |
|---|---|---|
| 000 | 0 | HCLK ⩽ 24MHz |
| 001 | 1 | HCLK ⩽ 48MHz |
| 010 | 2 | HCLK ⩽ 72MHz |

*Caution:*

*When configuring the FLASH read wait period, the FLASH configuration clock must be enabled first.*

## 7.5 FLASH Memory operation

User-executable FLASH memory operations include: read operations, erase operations, and write (programming) operations.

The read and write operations of FLASH support 3 widths of 8bit, 16bit and 32bit. The user program can complete the read and write by directly accessing the absolute address, but it should be noted that the data bit width of read and write must be aligned with the corresponding address boundary, otherwise it will cause HardFault hardware error exception.

### 7.5.1 Page Erase

The minimum unit of the page erase operation of FLASH is 1 page, that is, 512 bytes. After the page erase operation is completed, the data content of all address spaces of the page is 0xFF.

If the page erase operation is performed on the unlocked FLASH page, the operation will fail, and the FLASH_ISR.PAGELOCK flag will be set by hardware. If the FLASH_IER.PAGELOCK is set to 1, the CPU will execute the corresponding interrupt service routine. User can clear the FLASH_ISR.PAGELOCK interrupt flag by setting FLASH_ICR.PAGELOCK to 0.

If the program runs in FLASH, and the page erase operation is performed on the storage space of the page where the PC (program pointer) is located, the operation will fail, and the FLASH_ISR.PC flag will be set by hardware. If FLASH_IER.PC is set to 1, then The CPU will execute the corresponding interrupt service routine. The user can clear the FLASH_ISR.PC interrupt flag by setting FLASH_ICR.PC to 0.

FLASH page erase operation steps are as follows:

Step 1: Set the FLASH operation mode to page erase mode, and write 0x02 to the working mode bit field MODE of the FLASH control register FLASH_CR1;

*Caution:*

*The FLASH_CR1 register has the KEY protection feature, the high 16bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 2: Unlock the erasure protection of the page to be erased, see *Table 7-2 FLASH_PAGELOCK Erase lock protection*, write 1 to the lock bit field LOCKx corresponding to the erase/write lock register FLASH_PAGELOCK;

*Caution:*

*The FLASH_PAGELOCK register has the KEY protection feature, the high 16bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 3: Write arbitrary data to any address in the page to be erased, and trigger an erase operation of the page.

*Caution:*

*The write data bit width needs to be aligned with the corresponding address boundary.*

Code example:

```
*( ( uint8_t * ) 0x0000 0001 ) = 0x55;
```

Step 4: Wait for the erase be completed, and query the FLASH busy flag bit FLASH_CR1.BUSY to become 0;

Step 5: To erase other pages, repeat steps 2 to 4;

Step 6: After the erase operation is completed, if you want to restore the lock protection of FLASH, it is necessary to write 0 to the lock bit field LOCKx corresponding to the erasing lock register FLASH_PAGELOCK to activate the lock protection function to protect the FLASH content from being erased and rewritten by mistake;

Step 7: Set the FLASH operation mode to read mode, and send the FLASH The working mode bit field MODE of the control register FLASH_CR1 is written to 0x00.

### 7.5.2 Write operation

Based on the characteristics of embedded FLASH, the write operation can only rewrite the bit data in the FLASH memory from '1' to '0', but cannot be rewritten from '0' to '1'. Therefore, before writing data, the page where the corresponding address is located must be erased.

The following three principles must be followed for the FLASH write operation:

- Cannot write to the address whose data bit content is '0'
- Addresses within the locked area cannot be written to
- Cannot write to the address of the page where the PC (Program Pointer) is located

When the CW32F003 performs a write operation, it will check whether the contents of the storage space are all '1'. FLASH has three write operation bit widths of 8bit, 16bit and 32bit, and the number of bytes to be checked is 1Byte, 2Byte and 4Byte respectively.

If the internal storage space is not all '1', the write operation will fail, and the FLASH_ISR.PROG flag will be set by hardware. If FLASH_IER.PROG is set to 1, the CPU will execute the corresponding interrupt service routine. User can clear the FLASH_ISR.PROG interrupt flag by setting FLASH_ICR.PROG to 0.

If the storage space of the unlocked FLASH page is written, the operation will fail, and the FLASH_ISR. PAGELOCK flag will be set by hardware. If FLASH_IER.PAGELOCK is set to 1, the CPU will execute the corresponding interrupt service routine. User can clear the FLASH_ISR.PAGELOCK interrupt flag by setting FLASH_ICR.PAGELOCK to 0.

If the program runs in FLASH, and the write operation is performed on the storage space of the page where the PC (program pointer) is located, the operation will fail, and the FLASH_ISR.PC flag will be set by hardware. If FLASH_IER.PC is set to 1, the CPU will Execute the corresponding interrupt service routine. The user can clear the FLASH_ISR.PC interrupt flag by setting FLASH_ICR.PC to 0.

The FLASH write operation steps are as follows:

Step 1: Set the FLASH operation mode to write (programming) mode, and write 0x01 to the working mode bit field MODE of the FLASH control register FLASH_CR1;

*Caution:*

*The FLASH_CR1 register has the KEY protection feature, the high 16bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 2: Unlock the erasure protection of the page to be written, see *Table 7-2 FLASH_PAGELOCK Erase lock protection*, and write 1 to the lock bit field LOCKx corresponding to the erasure lock register FLASH_PAGELOCK;

*Caution:*

*The FLASH_PAGELOCK register has the KEY protection feature, the high 16bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

Step 3: Write the data to be written into the target address and trigger the FLASH write operation. Note that the bit width of the written data needs to be aligned with the corresponding address boundary;

8bit width code example:

```
*( ( uint8_t * ) 0x0000 0001 ) = 0x55;
```

16bit width code example:

```
*( ( uint16_t * ) 0x0000 0002 ) = 0x1234;
```

32bit width code example:

```
*( ( uint32_t * ) 0x0000 0004 ) = 0x1234 5678;
```

Step 4: Wait for the programming to be completed, query and wait for the FLASH busy flag bit FLASH_CR1. BUSY to become 0;

Step 5: If you need to write more data to the address range of the currently unlocked area, repeat steps 3 to 4; if the write address exceeds the address range of the current unlocked area, repeat steps 2 to 4;

Step 6: After the write operation is completed, if you need to restore the lock protection of the FLASH, write 0 to the lock bit field LOCKx corresponding to the erase/write lock register FLASH_PAGELOCK to activate the lock protection function to protect the FLASH content from being erased by mistake;

Step 7: Set the FLASH operation mode to read mode, and write 0x00 to the working mode bit field MODE of the FLASH control register FLASH_CR1.

### 7.5.3 Read operation

CW32F003 supports 3 different bit widths for the FLASH read operation, which can be read by direct access to the absolute address. Note that the read data bit width must be aligned with the corresponding address boundary.

8bit width code example:

```
tempdata = *( ( uint8_t * ) 0x0000 0001 ) ;
```

16bit width code example:

```
tempdata = *( ( uint16_t * ) 0x0000 0002 ) ;
```

32bit width code example:

```
tempdata = *( ( uint32_t * ) 0x0000 0004 ) ;
```

## 7.6 FLASH Memory protection

FLASH memory has erase and write protection and read protection functions.

The erasure protection includes lock page erasure protection and PC address page erasure protection. Pages in the protected state cannot be erased and written, which can avoid accidental rewriting of FLASH content.

The read protection takes the entire FLASH as the protection object, and does not support single-page protection, which can prevent the user code from being illegally read.

### 7.6.1 Erase and write protection

CW32F003 realizes the erasing and writing locking of FLASH pages by setting the erasing and writing lock register FLASH_PAGELOCK. Pages in the locked state cannot be page erased or written.

The internal FLASH memory of CW32F003 is divided into 40 pages, and each 4 pages corresponds to a LOCKx lock bit of the FLASH_PAGELOCK register. The LOCKx field has a total of 10 bits, which can realize the lock protection of all 40 pages of FLASH memory. The correspondence between the fields of the erasing lock register FLASH_PAGELOCK and the FLASH lock page is shown in the following table:

Table 7-2 FLASH_PAGELOCK Erase lock protection

| Bit | Bit field name | Lock page |
|---|---|---|
| 9 | LOCK9 | Page36 – Page39 |
| 8 | LOCK8 | Page32 – Page35 |
| 7 | LOCK7 | Page28 – Page31 |
| 6 | LOCK6 | Page24 – Page27 |
| 5 | LOCK5 | Page20 – Page23 |
| 4 | LOCK4 | Page16 – Page19 |
| 3 | LOCK3 | Page12 – Page15 |
| 2 | LOCK2 | Page8 – Page11 |
| 1 | LOCK1 | Page4 – Page7 |
| 0 | LOCK0 | Page0 – Page3 |

When performing page erasing and writing operations on FLASH, you must first unlock the corresponding page by setting the corresponding FLASH_PAGELOCK.LOCKx bit field to 1. An example of the unlock operation is as follows:

Lock Page0-Page3 code example:

```
CW_FLASH->PAGELOCK = 0x5A5A 0000 | ( CW_FLASH->PAGELOCK & 0x0000 FFFE );
```

Unlock Page32-Page35 code example:

```
CW_FLASH->PAGELOCK = 0x5A5A 0000 | ( CW_FLASH->PAGELOCK | 0x0000 0100 );
```

*Caution:*

*The FLASH_PAGELOCK register has the KEY protection feature. The upper 16bit data of the written data must be 0x5A5A, otherwise it cannot be written.*

If the page erase or write operation is performed directly on the unlocked FLASH page, the operation will fail and an interrupt flag will be generated. Please refer to section *7.5.1 Page Erase* and section *7.5.2 Write operation*.

### 7.6.2 Erase and write PC page protection

The FLASH memory of CW32F003 supports the function of erasing and writing PC page protection.

When the user program runs FLASH, if the current program pointer PC is just within the range of the FLASH address page to be erased, the erase operation will fail, and the FLASH_ISR.PC flag will be set by hardware. If FLASH_IER.PC is set to 1, the CPU will execute the corresponding interrupt service routine. The user can clear the FLASH_ISR.PC interrupt flag by setting FLASH_ICR.PC to 0.

### 7.6.3 Read protection

The CW32F003 supports the FLASH read protection function. After the read protection is set, the FLASH cannot be read by ISP or SWD. Read protection only supports whole-chip FLASH protection, and does not support page-by-page protection.

The read protection is divided into 4 protection levels. The current protection level can be obtained by reading the security bit field SECURITY of the FLASH control register FLASH_CR1. The security bit field is a read-only attribute and cannot be modified. The read protection level can be set by the ISP command, please refer to the ISP programming documentation.

The CPU fetches instructions from FLASH and the program's read operation to FLASH is not affected by the FLASH read protection function.

The read protection function of FLASH is shown in the following table:

Table 7-3 FLASH read protection

| Protection level | FLASH_CR1.SECURITY | Function description |
|---|---|---|
| Level0 | 00 | Read protection is not set.<br>The FLASH can be read by SWD or ISP. |
| Level1 | 01 | FLASH content cannot be read by SWD or ISP.<br>The protection level can be reduced to Level 0 through ISP or SWD. After the downgrade, the content in the FLASH is all 0xFF, that is, it is in the empty chip state of the whole chip erasing. |
| Level2 | 10 | FLASH content cannot be read by SWD or ISP.<br>The protection level can only be lowered by ISP, but after the downgrade, the content in the FLASH is all 0xFF, that is, it is in an empty chip state of whole chip erasing. |
| Level3 | 11 | FLASH content cannot be read by SWD or ISP.<br>Both ISP and SWD downgrade functions are disabled.<br>At this protection level, the chip can only be programmed once. |

*Caution:*

*The FLASH read protection level can only be set up to 48 times.*

### 7.6.4 FLASH memory programming

CW32F003 internal integrated FLASH memory supports FLASH erasing and programming through SWD or ISP. The SWD method is to use the debugging tool to erase and program the FLASH through the SWD interface. The ISP method is to erase and program the FLASH through the BootLoader code pre-installed before the chip leaves the factory. ISP provides fast and efficient in-circuit erasing and programming methods, please refer to ISP programming documentation.

## 7.7 Precautions

In order to operate FLASH correctly and improve the access efficiency and service life of FLASH, users need to pay attention to the following matters when programming applications:

- Address Alignment Requirements

  The address boundary is aligned, that is, the address when accessing FLASH with a 16-bit width must be an even address, and when using a 32-bit width, the address must be an address that is a multiple of 4.

  Code example for correct address alignment:

  8bit read:

  ```
  tempdata = *( ( uint8_t * ) 0x0000 0001 ) ;
  ```

  16bit read:

  ```
  tempdata = *( ( uint16_t * ) 0x0000 0002 ) ;
  ```

  32bit read:

  ```
  tempdata = *( ( uint32_t * ) 0x0000 0004 ) ;
  ```

  Code example with wrong address alignment:

  16bit read:

  ```
  tempdata = *( ( uint16_t * ) 0x0000 0001 ) ;
  ```

  32bit read:

  ```
  tempdata = *( ( uint32_t * ) 0x0000 0003 ) ;
  ```

- Operation complete flag query

  When the CPU fetches instructions from FLASH and runs, if the page erase/write operation to FLASH is performed, the CPU will automatically stop the next instruction access, and the hardware will automatically wait for the erase and write operation to complete (the FLASH_CR1.BUSY status bit becomes 0), so the user program does not need to loop through the query operation completion flag to determine whether the operation is completed.

  When the CPU fetches instructions from RAM and runs, if a page erase/write operation to FLASH is performed, the CPU will access the next instruction while performing the operation. In order to ensure the correct execution of the next instruction of the program, The user program must query the FLASH_CR1. BUSY flag cyclically after the FLASH erase and write operations, until the FLASH_CR1.BUSY flag becomes 0, and then the subsequent tasks can be performed.

  Before entering DeepSleep mode, if the FLASH is performing an erase/write operation, you must wait for the FLASH_CR1.BUSY flag bit to clear to 0, and make sure FLASH_CR1.MODE is 0 at the same time.

- Service life

  Based on the characteristics of embedded FLASH, the number of operations and storage time of FLASH are limited. Users should try to avoid frequent erasing and writing operations on the FLASH memory of a certain page or address in the application program to ensure reliable data storage. Please refer to the datasheet for specific life data.

- Data storage mode

  The application of CW32F003 stipulates that data is stored in FLASH in little-endian mode, that is, the least significant byte data of the data is stored in the lowest byte address space.

- Low Power Features

  FLASH has the characteristics of low power consumption. By setting FLASH_CR1.STANDBY to 1, FLASH can automatically enter a low power consumption state after the system enters DeepSleep mode, ensuring that the product has lower power consumption in low power consumption mode.

## 7.8 List of registers

FLASH base address: FLASH_BASE = 0x4002 2000

Table 7-4 List of FLASH registers

| Register name | Register address | Register description |
|---|---|---|
| FLASH_CR1 | FLASH_BASE + 0x00 | Control register 1 |
| FLASH_CR2 | FLASH_BASE + 0x04 | Control register 2 |
| FLASH_PAGELOCK | FLASH_BASE + 0x08 | Erase lock register |
| FLASH_IER | FLASH_BASE + 0x20 | Interrupt Enable Register |
| FLASH_ISR | FLASH_BASE + 0x24 | Interrupt Flag Register |
| FLASH_ICR | FLASH_BASE + 0x28 | Interrupt Flag Clear Register |

## 7.9 Register description

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 7.9.1 FLASH_CR1 Control register 1

Address offset: 0x00 Reset value: 0x0000 0010

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | KEY | WO | Only when KEY is 0x5A5A, the write operation of this register is valid |
| 15:8 | RFU | - | Reserved bits, please keep the default value |
| 7:6 | SECURITY | RO | Current protection level<br>00: Level0, ISP can read and write, SWD can read and write<br>01: Level1, ISP downgradable, SWD downgradable; Data cannot be read<br>10: Level2, ISP downgradable, SWD has no function; Data cannot be read<br>11: Level3, ISP has no function, SWD has no function; Data cannot be read |
| 5 | BUSY | RO | Erase status flag<br>0: Erase operation completed<br>1: Erase operation is not completed |
| 4 | STANDBY | RW | Low power enable control<br>0: When the system enters DeepSleep mode, FLASH does not enter low power consumption mode<br>1: When the system enters DeepSleep mode, FLASH enters low power consumption mode<br>Caution: It is recommended to configure this value to 1, otherwise it will cause the power consumption of DeepSleep mode to be too large |
| 3:2 | RFU | - | Reserved bits, please keep the default value |
| 1:0 | MODE | RW | Operation Mode Configuration<br>00: Read mode, Read<br>01: Write mode, Program<br>10: Page erase mode, PageErase<br>11: Chip erase mode, ChipErase |

### 7.9.2 FLASH_CR2 Control register 2

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | KEY | WO | Only when KEY is 0x5A5A, the write operation of this register is valid |
| 15:3 | RFU | - | Reserved bits, please keep the default value |
| 2:0 | WAIT | RW | FLASH instruction fetches cycle configuration<br>000: 1 HCLK cycle for HCLK <= 24MHz<br>001: 2 HCLK cycle for 24MHz < HCLK <= 48MHz<br>010: 3 HCLK cycle for 48MHz < HCLK <= 72MHz |

### 7.9.3 FLASH_PAGELOCK Erase lock register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | KEY | WO | Only when KEY is 0x5A5A, the write operation of this register is valid |
| 15:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | LOCK9 | RW | Page36 – Page39 erase lock configuration<br>0: locked, non-rewritable<br>1: open, rewritable |
| 8 | LOCK8 | RW | Page32 – Page35 erase lock configuration<br>0: locked, non-rewritable<br>1: open, rewritable |
| … | … | … | … |
| 1 | LOCK1 | RW | Page4 – Page7 erase lock configuration<br>0: locked, non-rewritable<br>1: open, rewritable |
| 0 | LOCK0 | RW | Page0 – Page3 erase lock configuration<br>0: locked, non-rewritable<br>1: open, rewritable |

### 7.9.4 FLASH_IER Interrupt Enable Register

Address offset: 0x20 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:5 | RFU | - | Reserved bits, please keep the default value |
| 4 | PROG | RW | Programming Error Interrupt Enable Control<br>0: Disable<br>1: Enable |
| 3:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | PAGELOCK | RW | Erase and write PAGELOCK locked page interrupt enable control<br>0: Disable<br>1: Enable |
| 0 | PC | RW | Erase and write the page where the PC is located, interrupt enable control<br>0: Disable<br>1: Enable |

### 7.9.5 FLASH_ISR Interrupt Flag Register

Address offset: 0x24 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:5 | RFU | - | Reserved bits, please keep the default value |
| 4 | PROG | RO | Programming error flag<br>0: The content of each byte of the current address to be written is all 0xFF<br>1: The content of each byte of the current address to be written is not all 0xFF |
| 3:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | PAGELOCK | RO | Erase PAGELOCK locked page interrupt flag<br>0: The current erase address is outside the page locked by PAGELOCK<br>1: The current erase address is within the page locked by PAGELOCK |
| 0 | PC | RO | Erase and write the interrupt flag of the page where the PC pointer is located<br>0: The current erase address is outside the page where the PC pointer is located<br>1: The current erase address is within the page where the PC pointer is located |

### 7.9.6 FLASH_ICR Interrupt Flag Clear Register

Address offset: 0x28 Reset value: 0x0000 000F

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:5 | RFU | - | Reserved bits, please keep the default value |
| 4 | PROG | WO | Programming error flag clear<br>W0: Clear programming error flag<br>W1: No function |
| 3:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | PAGELOCK | R1W0 | Erase and write PAGELOCK locked page interrupt flag clear<br>W0: Clear the page interrupt flag locked by erasing PAGELOCK<br>W1: No function |
| 0 | PC | R1W0 | Erase and write the page where the PC pointer is located to clear the interrupt flag<br>W0: Clear and erase the interrupt flag of the page where the PC pointer is located<br>W1: No function |

# 8 General-purpose input/output (GPIO)

## 8.1 Overview

The GPIO controller realizes the connection between various digital and analog circuits inside the chip and the physical pins.

GPIO can be configured as digital input and output and analog functions, supports peripheral function multiplexing, supports 4 interrupt sources of high level, low level, rising edge and falling edge, can wake up the MCU return to active mode through external interrupt in DeepSleep mode.

## 8.2 Main features

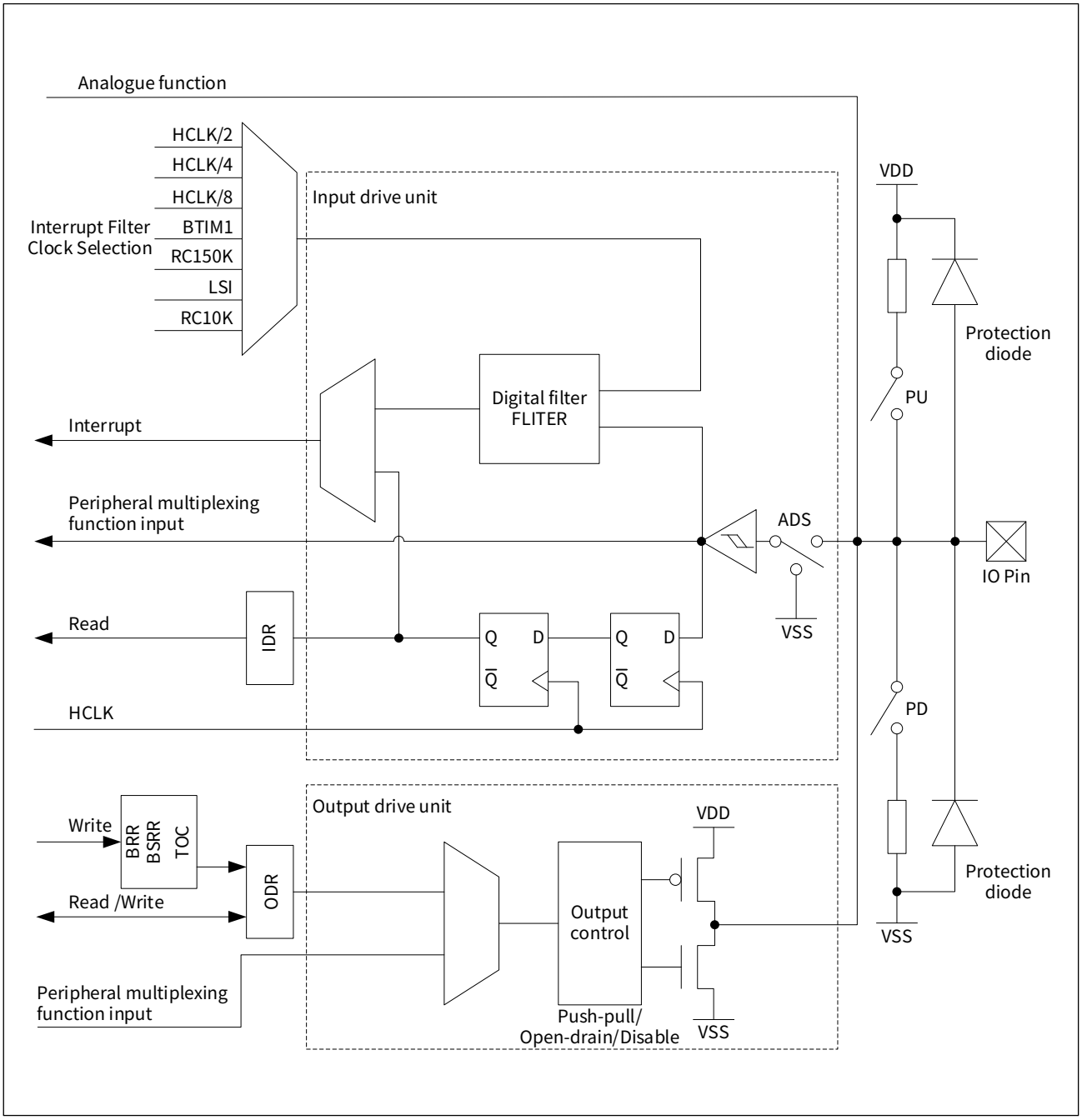
- All registers are read and written through the AHB bus interface
- With digital input and output and analog function
- Digital input and output support common GPIO and function multiplexing
- The analog function can be used as the input signal of ADC, VC, LVD
- Support a variety of internal clock signal output
- Digital input supports internal pull-up, pull-down and high-impedance modes
- Digital outputs support push-pull and open-drain modes
- The digital output supports atomic bit operations of bit setting, bit clearing, and bit flipping
- Interrupt function supports high level, low level, rising edge, falling edge trigger mode
- Interrupts with digital filtering, selectable from 7 clock sources
- Supports wake-up of MCU via external interrupt in DeepSleep mode

## 8.3 Function description

### 8.3.1 Functional block diagram

The functional block diagram of the GPIO controller is shown in the following figure:

Figure 8-1 GPIO functional block diagram



### 8.3.2 Digital output

Clear the analog-digital configuration register GPIOx_ANALOG[y] (x is the GPIO port number, x=A, B, C; y is the pin number, y=0~7; the same below), and configure the corresponding GPIO port as Digital function; clear the input and output direction register GPIOx_DIR[y] to configure the GPIO port as output mode. Digital output signal sources can be

- Output data register GPIOx_ODR
- On-chip digital peripherals

Configure the output mode through the output mode register GPIOx_OPENDRAIN, which can select push-pull output or open-drain output.

### 8.3.3 Digital input

Clear the analog-digital configuration register GPIOx_ANALOG[y] to configure the GPIO port as a digital function; set the input and output direction register GPIOx_DIR[y] to configure the corresponding GPIO port as input mode. Digital input signals are configurable:

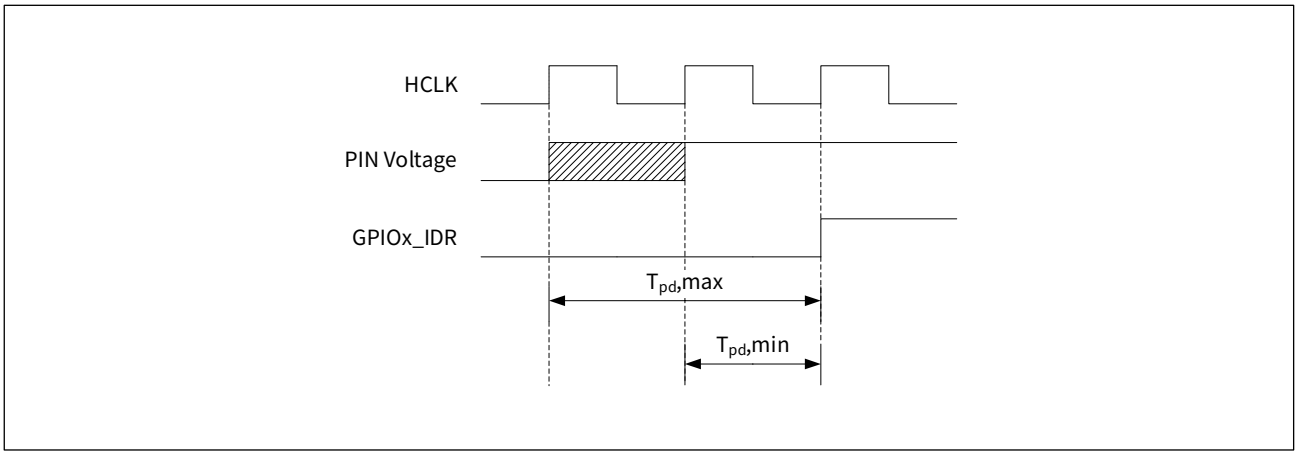
- Reach the input data register GPIOx_IDR
- Reach on-chip digital peripherals
- Trigger interrupt

In this mode, the digital input signal is directed to the internal digital input circuit through the ADS switch.

After the level state is confirmed by the Schmitt trigger, it can be directly sent to the input of the digital peripheral pointed to by the on-chip multiplexing function, or through an HCLK-based synchronizer, present on the input data register GPIOx_IDR [y].

The bits of the GPIOx_IDR register and the previous latch form a synchronizer, which can avoid the signal instability caused by the pin level jumping during the system clock change, but it will cause a certain read delay. The synchronization timing of the read port pins is shown in the following figure:

Figure 8-2 Read Port Pin Synchronization Timing




In the clock cycle after the rising edge of the system clock, the pin level signal will be latched in the internal register, as shown in the shaded part in the figure, after the next rising edge of the system clock, the stable pin level signal is read, and then a on the rising edge of the system clock, the data is latched into the GPIOx_IDR register. The signal delay $T_{pd}$ is 1~2 system clocks.

The built-in hardware filter circuit can also be enabled if this input signal is considered for triggering an interrupt. The filter circuit is implemented based on a double D flip-flop synchronizer, which has seven clock sources, some of which are specific to the low-power mode. For example, a key debounce operation without software intervention can be easily implemented. For specific clock source options and edge/level trigger options, see section *8.3.6 Interrupt Function*.

In this mode, the internal pull-up and pull-down functions can be individually selected to be turned on or off through the pull-up resistor register GPIOx_PUR and the pull-down resistor register GPIOx_PDR.

### 8.3.4 Analogue function

For a GPIO port configured with analog function, you can open the GPIO analog signal channel by setting the analog digital configuration register GPIOx_ANALOG[y] to 1. When the GPIO analog signal channel is turned on, the digital function of the port is turned off, the internal pull-up and pull-down are disconnected, the internal digital input signal is shorted to VSS through the ADS switch, and the internal digital output function is disabled.

### 8.3.5 Multiplexing function

The multiplexing function of the input and output ports can be realized through the multiplexing function registers (GPIOx_AFRH and GPIOx_AFRL). Each 3-bit bit field in the multiplexing function register corresponds to the multiplexing function selection of a GPIO port, and up to 8 input and output signal targets can logically be selected. The specific definition of GPIO multiplexing function is shown in the following table:

Table 8-1 GPIO Multiplexing function settings

| GPIOx_AFRL[4×y] | Multiplexing function |
|---|---|
| 000 | GPIO |
| 001 | AF1 |
| 010 | AF2 |
| 011 | AF3 |
| 100 | AF4 |
| 101 | AF5 |
| 110 | AF6 |
| 111 | AF7 |

*Caution:*

*y are pin numbers: y=0~7 .*

The specific peripheral functions corresponding to each AF are shown in the following table:

Table 8-2 GPIO multiplexing function assignment table

| Pin name | Multiplexing function | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | AF0 | AF1 | AF2 | AF3 | AF4 | AF5 | AF6 | AF7 |
| PA00 | GPIO | UART1_RXD | UART2_RTS | SPI_SCK | ATIM_CH3A | GTIM_CH4 | BTIM1_TOGN | VC1_OUT |
| PA01 | GPIO | UART2_TXD | VC2_OUT | SPI_MOSI | ATIM_CH3B | GTIM_CH1 | BTIM2_TOGP | MCO_OUT |
| PA02 | GPIO | UART1_RXD | UART2_TXD | I2C_SDA | GTIM_ETR | GTIM_CH3 | VC2_OUT | AWT_ETR |
| PA03 | GPIO | UART2_TXD | UART1_RXD | PCLK_OUT | ATIM_BK | GTIM_ETR | BTIM2_TOGP | LVD_OUT |
| PA04 | GPIO | UART1_RXD | IR_OUT | SPI_MISO | ATIM_CH3B | GTIM_CH2 | BTIM2_TOGN | GTIM_ETR |
| PA05 | GPIO | UART1_TXD | UART2_RXD | I2C_SCL | | GTIM_CH4 | BTIM_ETR | MCO_OUT |
| PA06 | GPIO | UART1_CTS | UART2_TXD | I2C_SDA | ATIM_CH2B | GTIM_CH3 | BTIM3_TOGP | LVD_OUT |
| PA07 | GPIO | UART1_RTS | UART2_RXD | VC1_OUT | ATIM_CH1B | GTIM_CH4 | BTIM3_TOGN | ATIM_BK |
| PB00 | GPIO | UART1_RXD | I2C_SDA | SPI_CS | ATIM_CH1B | GTIM_CH1 | GTIM_TOGP | AWT_ETR |
| PB01 | GPIO | UART1_TXD | LVD_OUT | I2C_SCL | ATIM_BK | GTIM_CH2 | GTIM_TOGN | AWT_ETR |
| PB02 | GPIO | UART1_TXD | UART2_CTS | SPI_CS | ATIM_CH2B | GTIM_CH3 | BTIM1_TOGP | MCO_OUT |
| PB03 | GPIO | UART2_RXD | I2C_SDA | PCLK_OUT | ATIM_CH2A | GTIM_CH2 | BTIM3_TOGP | IR_OUT |
| PB04 | GPIO | UART2_TXD | I2C_SCL | GTIM_ETR | ATIM_ETR | GTIM_CH1 | BTIM3_TOGN | ATIM_BK |
| PB05 | GPIO | UART1_RXD | I2C_SDA | BTIM_ETR | ATIM_CH1B | GTIM_TOGN | BTIM2_TOGN | ATIM_BK |
| PB06 | GPIO | UART1_TXD | I2C_SCL | SPI_CS | ATIM_CH1A | GTIM_TOGP | BTIM2_TOGP | HCLK_OUT |
| PB07 | GPIO | UART2_RXD | UART1_TXD | SPI_SCK | | GTIM_CH1 | BTIM2_TOGN | BTIM_ETR |
| PC00 | GPIO | UART2_RXD | UART1_TXD | SPI_SCK | ATIM_CH1A | GTIM_CH2 | BTIM1_TOGP | HCLK_OUT |
| PC01 | GPIO | UART2_TXD | GTIM_ETR | SPI_MISO | ATIM_CH2A | GTIM_CH3 | BTIM1_TOGN | VC1_OUT |
| PC02 | GPIO | UART2_RXD | IR_OUT | SPI_MOSI | ATIM_CH3A | GTIM_CH4 | HCLK_OUT | AWT_ETR |
| PC03 | GPIO | UART1_TXD | SPI_CS | SPI_MISO | ATIM_CH3B | GTIM_CH3 | GTIM_TOGP | ATIM_BK |
| PC04 | GPIO | UART1_RXD | IR_OUT | SPI_MOSI | ATIM_CH2B | GTIM_CH4 | GTIM_TOGN | |

### 8.3.6 Interrupt Function

When each GPIO is set to digital input mode, it can be used as an external interrupt signal source, and the interrupt signal source can be set to four types: high level, low level, rising edge, and falling edge. Interrupt triggering methods can be used in combination, but share the same interrupt flag bit.

After the interrupt is triggered, the corresponding bit of the interrupt flag register GPIOx_ISR will be set by hardware, and the program can determine which port generated the interrupt by querying GPIOx_ISR. Through the interrupt flag clear register GPIOx_ICR[y], the corresponding interrupt flag bit can be cleared.

The internal interrupt digital filter can digitally filter the input signal on the pin, providing 7 filter clock options, as shown in the following table:

Table 8-3 Digital Filter Clock Selection

| GPIOx_FILTER.FLTCLK | Digital Filter Clock Frequency |
|---|---|
| 000 | HCLK / 2 |
| 001 | HCLK / 4 |
| 010 | HCLK / 8 |
| 011 | BTIM1 overflow |
| 100 | RC150K (About 150kHz) |
| 101 | LSI (About 32.8kHz) |
| 110 | RC10K (About 10kHz) |
| 111 | Unfiltered clock (disable setting) |

Due to the wide range of selected filter clock cycles, users can easily implement flexible input interrupt debounce functions. Changes in input levels that are not maintained for more than one full filter clock cycle will not be communicated through the hardware filter to the internal interrupt trigger circuitry. Changes in input levels that persist for more than two complete filter clock cycles must pass through the hardware filter.

For the edge trigger type, considering the sensitivity to the time of the trigger edge, it is recommended to disable the hardware filter function in the interrupt digital filter configuration register GPIOx_FILTER[y], because the hardware filter can improve signal stability while also inserting certain delay.

When CW32F003 works in Sleep mode or DeepSleep mode, the external interrupt function of GPIO can still be used. When an external interrupt is generated, the chip can be woken up from Sleep mode or DeepSleep mode to Active mode.

*Caution:*

*The same group of GPIOx.PINy shares a hardware filter clock source selection register, so the same group of GPIOs can only filter input signal jitter with the same filter clock.*

### 8.3.7 Other functions

#### Atomic bit operation

The GPIO controller supports bit-setting, bit-clearing and bit-flipping functions.

Writing 1 to the GPIO bit clear register GPIOx_BSRR[y] or the bit clear register GPIOx_BRR[y] will directly change the state of the corresponding bit of the output data register GPIOx_ODR, thereby indirectly affecting the final output level without affecting the Status of other bits in this register.

Writing 1 to the GPIO bit toggle register GPIOx_TOG[y] will toggle the level state of the output port.

#### Port reset state

After power-on or reset, SWCLK (PA05) and SWDIO (PA02) default to digital pull-up. Other ports are analog high-resistance input by default, and pull-up or pull-down are not turned on by default.

## 8.4 Programming example

When configuring the GPIO port, you must first set SYSCTRL_AHBEN.GPIOx to 1, enable the corresponding GPIO configuration clock and working clock.

### 8.4.1 Digital output programming example

Step 1: Set GPIOx_ANALOG.PINy to 0 to configure the port as a digital function;

Step 2: Set GPIOx_DIR.PINy to 0, configure the port as output;

Step 3: Configure the GPIOx_OPENDRAIN register to set the port output mode;

Step 4: Configure the GPIOx_ODR register to set the port output level.

### 8.4.2 Digital input programming example

Step 1: Set GPIOx_ANALOG.PINy to 0, configure the port as a digital function;

Step 2: Set GPIOx_DIR.PINy to 1, configure the port as input;

Step 3: Configure the GPIOx_PUR register and select whether to enable the internal pull-up resistor;

Step 4: Configure GPIOx_PDR register, select whether to enable the internal pull-down resistor;

Step 5: Read the GPIOx_IDR register and read the port input level.

### 8.4.3 Analog functional programming example

Step 1: Set GPIOx_ANALOG.PINy to 1 to configure the port for analog function.

### 8.4.4 Multiplexed function programming example

Step 1: Configure the port as digital output or digital input according to application requirements. For specific register configuration steps, please refer to section *8.4.1 Digital output programming example* and section *8.4.2 Digital input programming example*;

Step 2: Configure the GPIOx_AFRH or GPIOx_AFRL register to set the port multiplexing function, see *Table 8-2 GPIO multiplexing function assignment table*.

### 8.4.5 Interrupt function programming example

Step 1: Configure the port as a digital input. For the specific register configuration steps, see section *8.4.2 Digital input programming example*;

Step 2: Configure GPIOx_FILTER.FLTCLK, and select the port interrupt filter clock;

Step 3: Set GPIOx_FILTER.PINy to 1 to enable the corresponding port filter Clock;

Step 4: Configure the NVIC controller, see chapter *5 Interrupt*;

Step 5: Configure the GPIOx_RISEIE, GPIOx_FALLIE, GPIOx_HIGHIE, GPIOx_LOWIE registers according to the application requirements, and select the GPIO interrupt trigger mode;

Step 6: The port interrupt input signal triggers the GPIO interrupt, execute the interrupt service function.

## 8.5 List of registers

GPIOA base address: GPIOA_BASE = 0x4800 0000

GPIOB base address: GPIOB_BASE = 0x4800 0400

GPIOC base address: GPIOC_BASE = 0x4800 0800

Table 8-4 List of GPIO registers

| Register name | Register address | Register description |
|---|---|---|
| GPIOx_DIR | GPIOx_BASE + 0x00 | GPIOx I/O direction register |
| GPIOx_OPENDRAIN | GPIOx_BASE + 0x04 | GPIOx output mode register |
| GPIOx_PDR | GPIOx_BASE + 0x0C | GPIOx pull-down Resistor Register |
| GPIOx_PUR | GPIOx_BASE + 0x10 | GPIOx pull-up Resistor Register |
| GPIOx_AFRL | GPIOx_BASE + 0x18 | GPIOx Multiplexing function register low segment |
| GPIOx_ANALOG | GPIOx_BASE + 0x1C | GPIOx Analog Digital Configuration Register |
| GPIOx_RISEIE | GPIOx_BASE + 0x24 | GPIOx Rising edge interrupt enable register |
| GPIOx_FALLIE | GPIOx_BASE + 0x28 | GPIOx Falling edge interrupt enable register |
| GPIOx_HIGHIE | GPIOx_BASE + 0x2C | GPIOx High level interrupt enable register |
| GPIOx_LOWIE | GPIOx_BASE + 0x30 | GPIOx Low level interrupt enable register |
| GPIOx_ISR | GPIOx_BASE + 0x34 | GPIOx Interrupt flag register |
| GPIOx_ICR | GPIOx_BASE + 0x38 | GPIOx Interrupt flag clear register |
| GPIOx_FILTER | GPIOx_BASE + 0x40 | GPIOx Interrupt digital filter configuration register |
| GPIOx_IDR | GPIOx_BASE + 0x50 | GPIOx input data register |
| GPIOx_ODR | GPIOx_BASE + 0x54 | GPIOx output data register |
| GPIOx_BRR | GPIOx_BASE + 0x58 | GPIOx bit clear register |
| GPIOx_BSRR | GPIOx_BASE + 0x5C | GPIOx bit-setting clear register |
| GPIOx_TOG | GPIOx_BASE + 0x60 | GPIOx bit flip register |

## 8.6 Register description

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 8.6.1 GPIOx_DIR GPIO I/O direction register ( x =A, B, C )

Address offset: 0x00 Reset value: 0x0000 00FF(GPIOA)
0x0000 00FF(GPIOB)
0x0000 003F(GPIOC)

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port input and output direction control<br>0: Configure the port as output<br>1: Configure the port as input |

### 8.6.2 GPIOx_OPENDRAIN GPIO output mode register ( x =A, B, C )

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port output mode control bit<br>0: Push-pull output<br>1: Open-drain output |

### 8.6.3 GPIOx_PDR GPIO Pull-down resistor register (x =A, B, C )

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port pull-down resistor enable control<br>0: Disable pull-down resistors<br>1: Enable pull-down resistors<br>Caution: When GPIOx_PDR.PINy and GPIOx_PUR.PINy are set to 1 at the same time, the port state is to enable the pull-up resistor. |

### 8.6.4 GPIOx_PUR GPIO Pull-up resistor register ( x =A, B, C )

Address offset: 0x10 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port pull-up resistor enable control<br>0: Disable pull-up resistors<br>1: Enable pull-up resistors<br>Caution: When GPIOx_PDR.PINy and GPIOx_PUR.PINy are set to 1 at the same time, the port state is to enable the pull-up resistor. |

### 8.6.5 GPIOx_AFRL GPIO Multiplexing function register low segment (x =A, B, C )

Address offset: 0x18 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:28<br>26:24<br>22:20<br>18:16<br>14:12<br>10:8<br>6:4<br>2:0 | AFRy<br>y = 0 ~ 7 | RW | Port digital multiplexing function control<br>000: GPIO<br>001: AF1<br>010: AF2<br>011: AF3<br>100: AF4<br>101: AF5<br>110: AF6<br>111: AF7 |

### 8.6.6 GPIOx_ANALOG GPIO Analog Digital Configuration Register ( x =A, B, C )

Address offset: 0x1C Reset value: 0x0000 00DB(GPIOA)

0x0000 00FF(GPIOB)

0x0000 003F(GPIOC)

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port analog/digital function configuration<br>0: Configure the port as a digital function<br>1: Configure the port as an analog function |

### 8.6.7 GPIOx_RISEIE GPIO Rising edge interrupt enable register ( x =A, B, C )

Address offset: 0x24 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port rising edge interrupt enable control<br>0: Disable the rising edge interrupt of the corresponding port<br>1: Enable the rising edge interrupt of the corresponding port |

### 8.6.8 GPIOx_FALLIE GPIO Falling edge interrupt enable register ( x =A, B, C )

Address offset: 0x28 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port falling edge interrupt enable control<br>0: Disable the falling edge interrupt of the corresponding port<br>1: Enable the falling edge interrupt of the corresponding port |

### 8.6.9 GPIOx_HIGHIE GPIO High level interrupt enable register ( x =A, B, C )

Address offset: 0x2C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port high level interrupt enable control<br>0: Disable the high-level interrupt of the corresponding port<br>1: Enable the high-level interrupt of the corresponding port |

### 8.6.10 GPIOx_LOWIE GPIO Low level interrupt enable register ( x =A, B, C )

Address offset: 0x30 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port low level interrupt enable control<br>0: Disable the low-level interrupt of the corresponding port<br>1: Enable the low-level interrupt of the corresponding port |

### 8.6.11 GPIOx_ISR GPIO Interrupt flag register ( x =A, B, C )

Address offset: 0x34 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RO | Port interrupt status flag<br>0: Enabled interrupt not detected<br>1: Enabled interrupt has been detected |

### 8.6.12 GPIOx_ICR GPIO Interrupt flag clear register ( x =A, B, C )

Address offset: 0x38 Reset value: 0x0000 00FF(GPIOA)
0x0000 00FF(GPIOB)
0x0000 003F(GPIOC)

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | R1W0 | Port interrupt status flag clear<br>W0: Clear the corresponding interrupt flag bit<br>W1: No function |

### 8.6.13 GPIOx_FILTER GPIO Interrupt digital filter configuration register( x =A, B, C )

Address offset: 0x40 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:19 | RFU | - | Reserved bits, please keep the default value |
| 18:16 | FLTCLK | RW | Port interrupt filter clock selection<br>000: HCLK / 2<br>001: HCLK / 4<br>010: HCLK / 8<br>011: BTIM1 overflow<br>100: RC150K (About 150KHz)<br>101: LSI (About 32.8KHz)<br>110: RC10K (About 10KHz)<br>111: Unfiltered clock (disable setting) |
| 15:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Port filter clock enable control<br>0: Disable the corresponding port to filter the clock<br>1: Enable the corresponding port to filter the clock<br>Caution: Filter out pulses with a width less than the FLTCLK clock width |

### 8.6.14 GPIOx_IDR GPIO input data register ( x =A, B, C )

Address offset: 0x50 Reset value: 0x0000 0024(GPIOA)

0x0000 0000(GPIOB)

0x0000 0020(GPIOC)

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RO | Read port input level status<br>0: The port is low level<br>1: The port is high level |

### 8.6.15 GPIOx_ODR GPIO output data register ( x =A, B, C )

Address offset: 0x54 Reset value: 0x---- ----

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | RW | Set the port output level<br>0: Set port output low level<br>1: Set port output high level |

### 8.6.16 GPIOx_BRR GPIO port bit set clear register ( x =A, B, C )

Address offset: 0x58 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | BRRy<br>y=0 ~ 7 | R0W1 | Port bit clear control<br>0: Does not affect the corresponding bits of the GPIOx_ODR register<br>1: Set the corresponding bit of the GPIOx_ODR register to 0 |

### 8.6.17 GPIOx_BSRR GPIO port bit-setting clear register ( x =A, B, C )

Address offset: 0x5C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:23 | RFU | - | Reserved bits, please keep the default value |
| 22:16 | BRRy<br>y=0 ~ 7 | R0W1 | Port bit-setting clear control<br>0: Does not affect the corresponding bits of the GPIOx_ODR register<br>1: Set the corresponding bit of the GPIOx_ODR register to 0 |
| 15:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | BSSy<br>y=0 ~ 7 | R0W1 | Port bit-setting control<br>0: Does not affect the corresponding bits of the GPIOx_ODR register<br>1: Set the corresponding bit of the GPIOx_ODR register to 1<br>Caution: When BRRy and BSSy are both set 1, BSSy has higher priority |

### 8.6.18 GPIOx_TOG GPIO Port Bit Flip Register( x =A, B, C )

Address offset: 0x60 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | PINy<br>y=0 ~ 7 | R0W1 | Port Bit Flip Control<br>0: Does not affect the corresponding bits of the GPIOx_ODR register<br>1: Invert the corresponding bit of the GPIOx_ODR register |

# 9 Cyclic redundancy check (CRC)

## 9.1 Overview

Cyclic Redundancy Check (CRC) is mainly used to verify the correctness and integrity of data transmission or data storage. The CW32F003 integrates a CRC calculation unit, which supports the use of multiple CRC algorithms to perform CRC calculation on the input data.

## 9.2 Main features

- 8bit input data bit width
- CRC-16 polynomial 2: $x^{16} + x^{12} + x^5 + 1$
- Four kinds of commonly used algorithms
  Combination based on polynomial, initial value, result XOR value, input/output inversion

## 9.3 Functional description

The CRC unit obtains the CRC calculation result by performing a 'division' operation on the input data (or the inversion of the input data) and the selected polynomial value, and then inverting or not inverting the obtained remainder, as well as XOR processing.

Before the CRC unit is used, it is necessary to set SYSCTRL_AHBEN.CRC to 1, and turn on the configuration clock and working clock of the CRC unit, which are generally set during system initialization.

### 9.3.1 Algorithmic modes

The CRC unit of CW32F003 supports multiple algorithm modes. Different CRC algorithms have different parameters such as polynomial, initial value, input data inversion, output data inversion, and result XOR value, as shown in the following table:

Table 9-1 CRC algorithm modes

| Algorithm name | Polynomial value | Initial value | Input inversion | Output inversion | Result XOR value |
|---|---|---|---|---|---|
| CRC16_CCITT | 0x1021 | 0x0000 | True | True | 0x0000 |
| CRC16_CCITT_False | 0x1021 | 0xFFFF | False | False | 0x0000 |
| CRC16_X25 | 0x1021 | 0xFFFF | True | True | 0xFFFF |
| CRC16_XMODEM | 0x1021 | 0x0000 | False | False | 0x0000 |

The meaning of each parameter is as follows:

- Polynomial value

  The polynomial is the description of the code group, such as CRC-16 polynomial: $x^{16} + x^{12} + x^5 + 1$, the corresponding code group is 1 0001 0000 0010 0001. Because the highest bit of the polynomial code group is fixed to 1, and the position of the highest bit is known, the code group after removing the highest bit 1 is generally called a polynomial value. For example, the value of CRC-16 polynomial is 0001 0000 0010 0001, that is 0x1021.

- Initial value

  The initial value of the CRC register before calculating the CRC check value.

- Input data inversion

  That is, before the calculation starts, the high and low order bits of the data that need to calculate the CRC check value are reversed, such as the data bit 1011, which is 1101 after the inversion.

- Output data inversion

  After the CRC calculation is completed, before the XOR value is performed with the result, the high and low order bits of the calculated value are reversed. If the calculation result is 1011, it will be 1101 after inversion.

- Result XOR value

  After the CRC calculation is completed, the obtained CRC calculation value is XORed with the result XOR value to obtain the final CRC check value.

### 9.3.2 Input data bit width

The CRC calculation unit supports 8bit input data bit width.

For example, the user needs to calculate the CRC check value of the group of data 0x00, 0x11, 0x22 , 0x33, 0x44, 0x55, 0x66, 0x77, the writing sequence is: 0x00, 0x11, 0x22 , 0x33, 0x44, 0x55, 0x66, 0x77

Code example:

```
CW_CRC->DR = 0x00;
CW_CRC->DR = 0x11;
CW_CRC->DR = 0x22;
CW_CRC->DR = 0x33;
CW_CRC->DR = 0x44;
CW_CRC->DR = 0x55;
CW_CRC->DR = 0x66;
CW_CRC->DR = 0x77;
```

## 9.4 Programming examples

### 9.4.1 CRC16_CCITT algorithm mode

Step 1: Set CRC_CR.MODE to 0x04, select the CRC16_CCITT algorithm mode, and the initial value of the CRC register is automatically configured by hardware to 0x0000;

Step 2: Write the raw data to be encoded into the data register CRC_DR in turn, and the input data bit width can be selected from 8bit. See the code example in section *9.3.2 Input data bit width*;

Step 3: Read CRC_RESULT[15:0] to get the CRC check value.

Code example:

```
tempdata = CW_CRC -> RESULT;
```

## 9.5 List of registers

CRC base address: CRC_BASE =0x4002 3000

Table 9-2 List of CRC registers

| Register name | Register address | Register description |
|---|---|---|
| CRC_CR | CRC_BASE + 0x00 | Control register |
| CRC_DR | CRC_BASE + 0x08 | Data register |
| CRC_RESULT | CRC_BASE + 0x0C | Result register |

## 9.6 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 9.6.1 CRC_CR control register

Address offset: 0x00 Reset value: 0x0000 0004

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:4 | RFU | - | Reserved bits, please keep the default value |
| 3:0 | MODE | RW | CRC algorithm mode configuration<br>0100: CRC16_CCITT<br>0101: CRC16_CCITT_False<br>0110: CRC16_X25<br>0111: CRC16_XMODEM |

### 9.6.2 CRC_DR data register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | DR | RW | Data insertion register |

### 9.6.3 CRC_RESULT result register

Address offset: 0x0C Reset value: 0x0000 FFFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | RESULT16 | RO | CRC16 calculation result |

# 10 Automatic wake-up timer (AWT)

## 10.1 Overview

CW32F003 integrates one automatic wake-up timer (AWT), AWT contains a 16bit down counter and is driven by a programmable prescaler. AWT can choose five kinds of counting clock sources, can work in timing mode or counting mode. When the counter clock source is LSI, the AWT can keep running in DeepSleep mode, and the underflow interrupt can wake up the MCU back to Active mode.

## 10.2 Main features

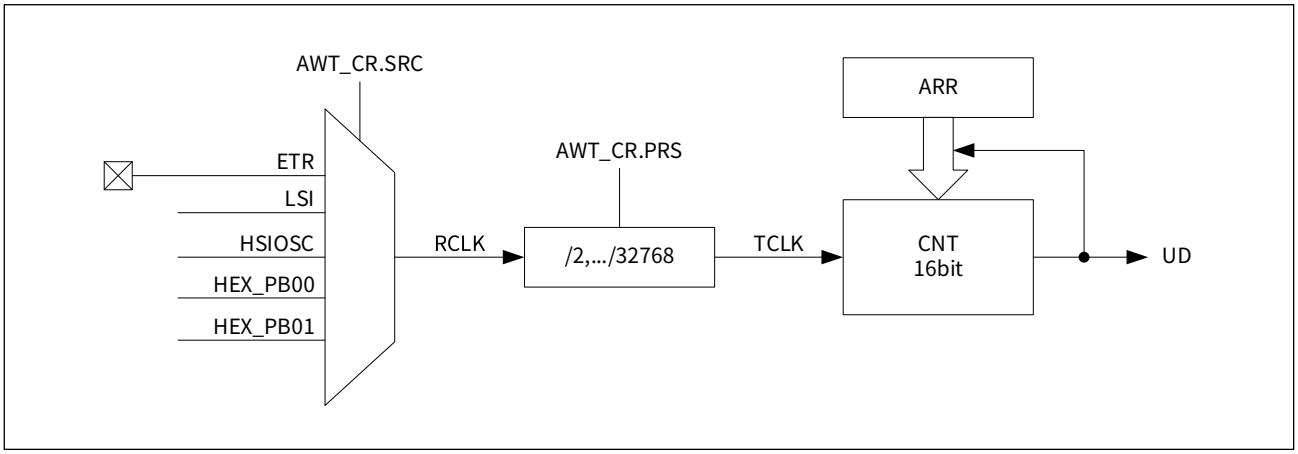
- 16bit down counter
- 5 kinds of working clock source: HSIOSC/LSI/HEX_PB00/HEX_PB01/ETR
- Programmable prescaler: 2~32768 frequency division
- Supports to keep running in DeepSleep mode, interrupt wake-up MCU
- When the counting clock source is LSI, the wake-up period is 60μs~65536s

## 10.3 Functional description

### 10.3.1 Functional block diagram

The functional block diagram of AWT is shown in the following figure:

Figure 10-1 AWT functional block diagram




**Clock source**

The counter clock of AWT can be selected from 5 clock sources RCLK: HSIOSC, LSI and the ETR signal input from the external AWT_ETR pin, the external input clock signal HEX_PB00 or HEX_PB01, which is selected by the SRC bit field of the control register AWT_CR. The clock source is divided by the prescaler as the counter clock of the AWT.

*Caution:*

*The clock source selection must be completed before starting the timer, and modification is prohibited during the running process of the timer, otherwise the AWT will work abnormally.*

**Prescaler**

AWT has a built-in 4-bit prescaler, and the counter clock source RCLK is divided by the prescaler to drive the counter to count. The specific frequency division coefficient is selected through the PRS bit field of the control register AWT_CR. The corresponding relationship between the setting value and the frequency division coefficient is shown in the following table:

Table 10-1 AWT counting clock prescaler coefficient

| PRS | Frequency division coefficient | PRS | Frequency division coefficient | PRS | Frequency division coefficient | PRS | Frequency division coefficient |
|---|---|---|---|---|---|---|---|
| 0x00 | Reserve, not configurable | 0x04 | 16 | 0x08 | 256 | 0x0C | 4096 |
| 0x01 | 2 | 0x05 | 32 | 0x09 | 512 | 0x0D | 8192 |
| 0x02 | 4 | 0x06 | 64 | 0x0A | 1024 | 0x0E | 16384 |
| 0x03 | 8 | 0x07 | 128 | 0x0B | 2048 | 0x0F | 32768 |

*Caution:*

*The setting of the prescaler coefficient must be completed before starting the timer, and it is forbidden to modify it during the running process of the timer, otherwise the AWT will work abnormally.*

### Counting unit

The core component of the counting unit is a 16bit down counter CNT and a 16bit automatic reload register ARR.

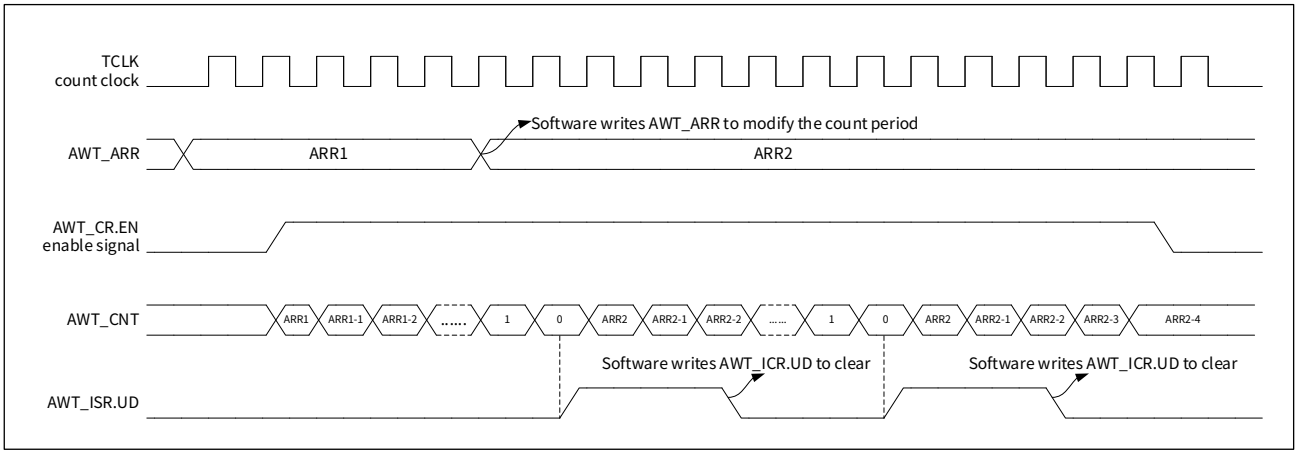
The reload register ARR is used to set the timing or counting period, and the number of timing or counting periods is ARR+1. The counter CNT is used to count down, underflow after decrementing to 0, and start counting again from the reload value ARR.

Set AWT_CR.EN to 1 to enable AWT, and the counter CNT starts to count down from the reload value ARR under the drive of the counting clock TCLK. When it decrements to 0, an underflow occurs, and the underflow interrupt flag bit AWT_ISR.UD will be set by hardware , the counter CNT reloads the value of the AWT_ARR register and restarts the countdown of the next cycle.

The user can set the value of the reload value register AWT_ARR at any time, but it does not affect the current count value of CNT, and the newly set value will take effect in the next timing or counting cycle.

The AWT count timing diagram is shown in the following figure:

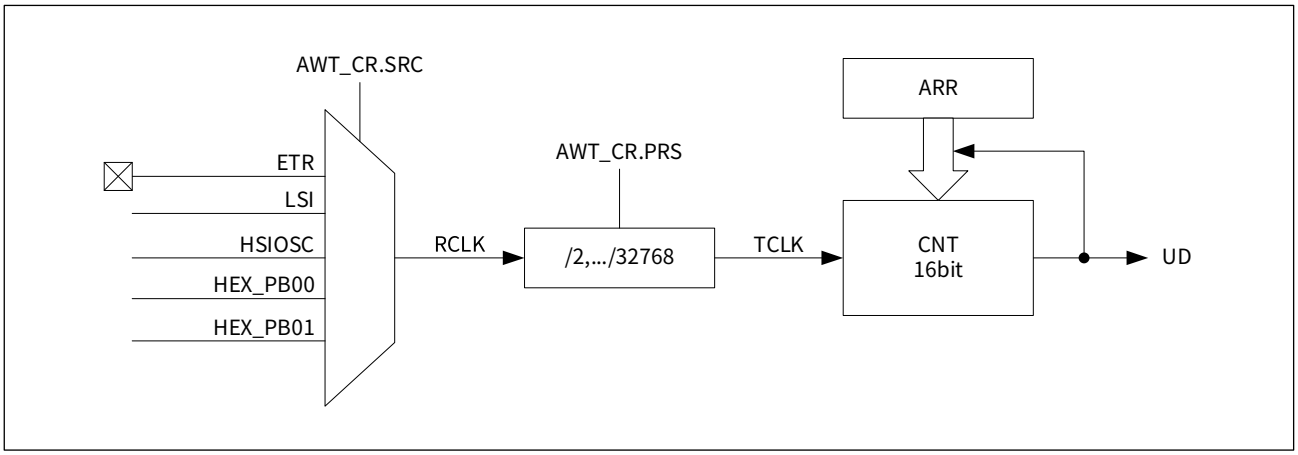
Figure 10-2 AWT counting and interrupt flags



### 10.3.2 Timing function

The AWT timing function is generally used to delay or generate time base signals at fixed intervals. HSIOSC/LSI/HEX_PB00/HEX_PB01/ETR can be used as its clock source. When selecting ETR as its clock source, it is necessary to ensure that the signal input by ETR is a fixed period signal. The functional block diagram of AWT timing mode is shown in the following figure:

Figure 10-3 Timing function block diagram




Timing time T calculation formula:

$$T = ( 2^{PRS} / RCLK ) \times ( ARR + 1 )$$

Among them, RCLK is the counter clock source, PRS is the prescaler coefficient, and ARR is the reload value.

Example:

When the counter clock source RCLK is LSI (the clock frequency is 32800Hz), the timing is required to be 200ms.

If the prescaler coefficient PRS is set to 0x01, calculate

$$T = 200\ ms = ( 2^{1} / 32800 ) \times ( ARR + 1 )$$

that

ARR=3279

That is, it is necessary to set the reload value ARR to 0xCCF.

Set the timer enable control bit AWT_CR.EN to 1, and the reload value ARR is loaded into the counter CNT. After that, every time a TCLK clock comes, the CNT count value is decremented by 1. When the count value is decremented to 0, an underflow occurs, and the underflow interrupt flag AWT_ISR.UD is set by hardware. If the interrupt is enabled (set AWT_IER.UD to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, AWT_ICR.UD should be set to 0 to clear this flag to avoid repeatedly entering the interrupt service routine.

The timing function configuration process is as follows (when the selected clock source is HSIOSC/LSI/HEX_PB00/HEX_PB01, steps 1 and 2 do not need to be performed):

Step 1: Set the relevant bit of the peripheral clock enable control register SYSCTRL_AHBEN to 1, and enable the configuration clock and working clock of the GPIO port corresponding to AWT_ETR;

Step 2: Set the relevant bits of the GPIO multiplexed function registers GPIOx_AFRL, and configure the corresponding pin to be the AWT_ETR function of the AWT timer;

Step 3: Set the peripheral clock enable control register SYSCTRL_APBEN2.AWT to 1, and turn on the configuration clock of the AWT module;

Step 4: Configure the control register AWT_CR.SRC and select the AWT count clock source;

Step 5: Configure the control register AWT_CR.PRS and select the frequency division coefficient of the AWT count clock;

Step 6: Set the interrupt enable register AWT_IER.UD to 1 to enable the counter CNT underflow event to trigger an interrupt request;

Step 7: Configure the reload value register AWT_ARR, and set the timing period. For the specific configuration, please refer to the relevant description in this section;

Step 8: Set the control register AWT_CR.EN to 1, and start the timer to start timing;

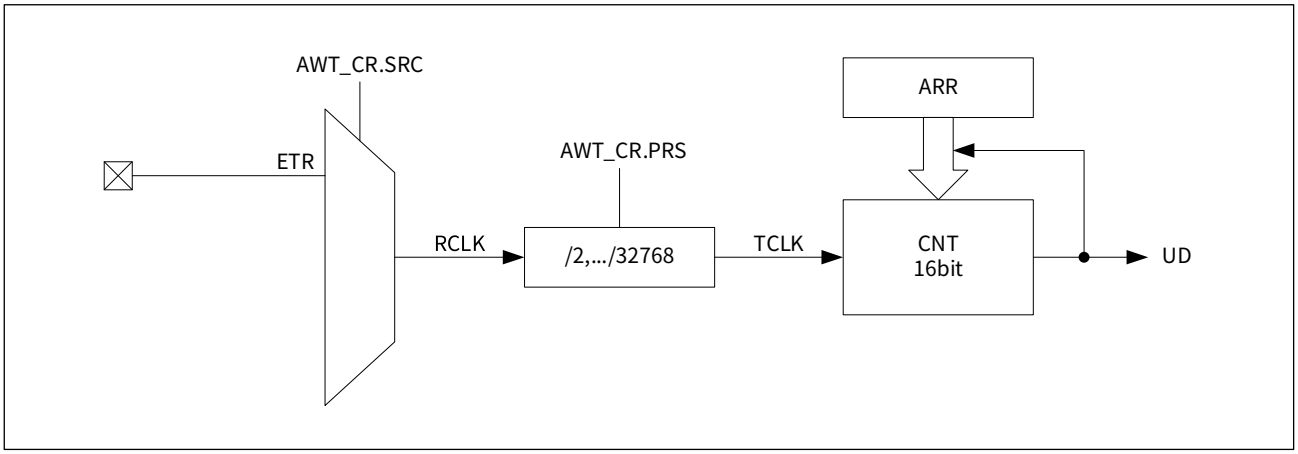
Step 9: The counter CNT overflows, enters the interrupt service function, and sets AWT_ICR.UD to 0 to clear the interrupt flag;

Step 10: If you do not want to continue timing, set the control register AWT_CR.EN to 0 to disable the timer timing function.

### 10.3.3 Count function

The AWT count function is used to measure the number of times an event occurs. In the counting mode, ETR is selected as the counting clock source, and the user inputs the external signal that needs to be counted through the AWT_ETR pin (for specific pins, please refer to the pin definition in the datasheet). The functional block diagram of AWT counting mode is shown in the following figure:

Figure 10-4 Count function block diagram




The counting clock source is selected by AWT_CR.SRC, the prescaler coefficient is selected by AWT_CR.PRS, and the counting period is configured by AWT_ARR.ARR, and the counting period is AWT_ARR+1.

Set the timer enable control bit AWT_CR.EN to 1, and the reload value ARR is loaded into the counter CNT. After that, every time a TCLK clock comes, the CNT count value is decremented by 1. When the count value is decremented to 0, an underflow occurs, and the underflow interrupt flag AWT_ISR.UD is set by hardware. If the interrupt is enabled (set AWT_IER.UD to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, AWT_ICR.UD should be set to 0 to clear this flag to avoid repeatedly entering the interrupt service routine.

The configuration process of count function is as follows:

Step 1: Set the relevant bit of the peripheral clock enable control register SYSCTRL_AHBEN to 1, and enable the configuration clock and working clock of the GPIO port corresponding to AWT_ETR;

Step 2: Set the relevant bits of the GPIO multiplexed function registers GPIOx_AFRL, and configure the corresponding pin to be the AWT_ETR function of the AWT timer;

Step 3: Set the peripheral clock enable control register SYSCTRL_APBEN2.AWT to 1, and turn on the configuration clock of the AWT module;

Step 4: Set the control register AWT_CR.SRC to 0x04, and select the AWT count clock source as ETR;

Step 5: Configure the control register AWT_CR.PRS and select the frequency division coefficient of the AWT count clock;

Step 6: Set the interrupt enable register AWT_IER.UD to 1 to enable the counter CNT underflow event to trigger an interrupt request;

Step 7: According to the number N to be counted, set the value of the reload value register AWT_ARR to N-1;

Step 8: Set the control register AWT_CR.EN to 1, and start the counter to start counting;

Step 9: The counter CNT overflows, enters the interrupt service function, and sets AWT_ICR.UD to 0 to clear the interrupt flag;

Step 10: If you do not want to continue counting, set the AWT control register AWT_CR.EN to 0 to disable the counter counting function.

## 10.4 Low power mode

When the counter clock source is LSI, the AWT can keep running in DeepSleep mode, and the underflow interrupt can wake up the MCU back to Active mode.

When selecting LSI (32.8KHz) as the counter clock source, set the prescaler coefficient AWT_CR.PRS to 0x01 and the reload value AWT_ARR to 0, then the shortest timing period is 60μs; Set the prescaler coefficient AWT_CR.PRS to 0x0F, and the reload value AWT_ARR to 0xFFFF, so the longest timing period is 65536s.

Therefore, in low-power timing mode, the wake-up period can be configured from 60μs ~ 65536s.

## 10.5 AWT interrupts

When the value of the 16-bit counter CNT decrements to 0, an underflow occurs, and the underflow interrupt flag bit AWT_ISR.UD will be set by hardware. If the interrupt is enabled (the interrupt enable bit AWT_IER.UD is set to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, the interrupt flag clear register AWT_ICR.UD should be set to 0 to clear the flag bit to avoid repeated entry into the interrupt service program.

When LSI is selected as the AWT working clock source, the AWT can continue to work in DeepSleep mode, and the AWT interrupt can wake up the MCU to Active mode.

## 10.6 Debugging support

AWT supports stop or continue counting in debug mode, which is set by the AWT bit field of the debug status timer control register SYSCTRL_DEBUG.

Set SYSCTRL_DEBUG.AWT to 1, then suspend the AWT counter counting in the debug state;

Set SYSCTRL_DEBUG.AWT to 0, then the AWT counter continues to count in the debug state.

In the debug state, the program stops running. If the counting of the AWT counter is not stopped, the AWT_ISR.UD interrupt flag will be generated soon, which will cause the program to jump to the interrupt service routine and affect the debugging of other functions. It is recommended to enable the AWT counter to stop counting function in debug mode.

## 10.7 List of registers

AWT base address: AWT_BASE = 0x4001 4C00

Table 10-2 List of AWT registers

| Register name | Register address | Register description |
|---|---|---|
| AWT_CR | AWT_BASE + 0x00 | Control register |
| AWT_ARR | AWT_BASE + 0x04 | Reload value register |
| AWT_CNT | AWT_BASE + 0x08 | Count value register |
| AWT_IER | AWT_BASE + 0x10 | Interrupt enable register |
| AWT_ISR | AWT_BASE + 0x14 | Interrupt flag register |
| AWT_ICR | AWT_BASE + 0x1C | Interrupt flag clear register |

## 10.8 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 10.8.1 AWT_CR control register

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:11 | RFU | - | Reserved bits, please keep the default value |
| 10:8 | SRC | RW | Counting clock source selection<br>000: HSIOSC<br>001: LSI<br>010: HEX_PB00<br>011: HEX_PB01<br>100: ETR<br>Caution: When the clock source is LSI/ HEX_PB00/ HEX_PB01, the timer can work in DeepSleep mode. |
| 7:4 | PRS | RW | Counting clock prescaler configuration<br>0000: reserved, not configurable<br>0001: 2 frequency division<br>0010: 4 frequency division<br>0011: 8 frequency division<br>0100: 16frequency division<br>0101: 32 frequency division<br>0110: 64 frequency division<br>0111: 128 frequency division<br>1000: 256 frequency division<br>1001: 512 frequency division<br>1010: 1024 frequency division<br>1011: 2048 frequency division<br>1100: 4096 frequency division<br>1101: 8192 frequency division<br>1110: 16384 frequency division<br>1111: 32768 frequency division |
| 3 | RFU | - | Reserved bits, please keep the default value |
| 2:1 | MD | RW | Please write 11 |
| 0 | EN | RW | Timer enable control<br>0: Disabled<br>1: Enabled<br>Caution: The EN bit field can be set to 1 only after the configuration of AWT_CR[10:1] is completed. |

### 10.8.2 AWT_ARR reload value register

Address offset: 0x04 Reset value: 0x0000 FFFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | ARR | RW | Counter reload value |

### 10.8.3 AWT_CNT count value register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CNT | RO | Counter count value |

### 10.8.4 AWT_IER interrupt enable register

Address offset: 0x10 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | UD | RW | Counter underflow interrupt enable configuration<br>0: Disabled<br>1: Enabled |
| 2:0 | RFU | - | Reserved bits, please keep the default value |

### 10.8.5 AWT_ISR interrupt flag register

Address offset: 0x14 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | UD | RO | Counter underflow interrupt flag<br>0: The timer has not overflowed<br>1: The timer has overflowed |
| 2:0 | RFU | - | Reserved bits, please keep the default value |

### 10.8.6 AWT_ICR interrupt flag clear register

Address offset: 0x1C Reset value: 0x0000 003F

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:4 | RFU | - | Reserved bits, please keep the default value |
| 3 | UD | R1W0 | Counter underflow interrupt flag clear<br>W0: Clear overflow flag<br>W1: No function |
| 2:0 | RFU | - | Reserved bits, please keep the default value |

# 11 Basic timer (BTIM)

## 11.1 Overview

CW32F003 integrates three basic timers (BTIM), each BTIM is completely independent and has the same function, each contains a 16-bit automatic reload counter and is driven by a programmable prescaler. BTIM supports four working modes: timer mode, counter mode, trigger start mode and gate control mode, and supports overflow event trigger interrupt request. Thanks to the fine processing design of the trigger signal, the BTIM can automatically perform the filtering operation of the trigger signal by the hardware, and can also cause the trigger event to generate interrupts.

## 11.2 Main features

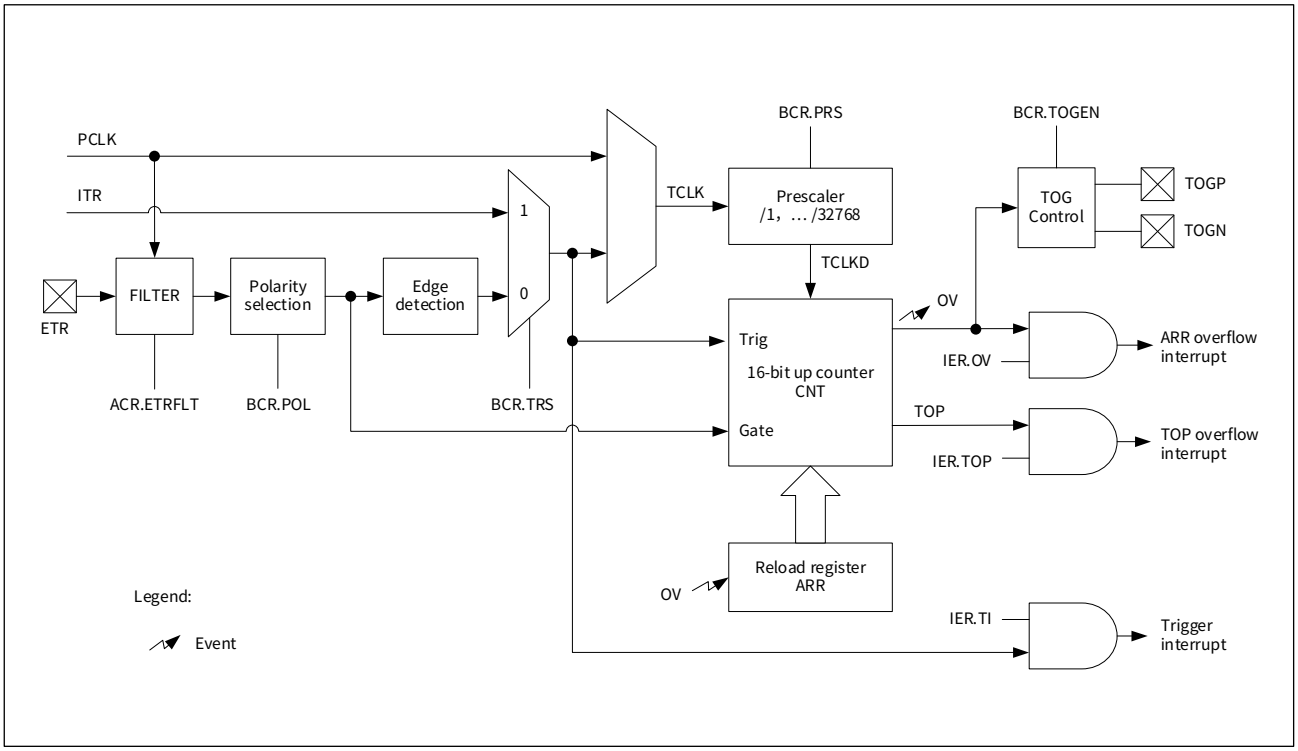
- 16-bit auto-reload up counter
- Programmable prescaler supports 1,2,4,8... 32768 frequency division
- Supports single counting mode and continuous counting mode
- Counting function for internal ITR or external ETR signal
- Internal ITR or external ETR signal triggers to start counting
- Gated function controlled by external ETR input signal
- Flexible external ETR signal filter processing
- Overflow event triggers digital output level transition
- Internal cascade ITR and external interconnect ETR
- Counter overflow triggers interrupt
- Internal ITR or external ETR signal triggers interrupt

## 11.3 Functional description

### 11.3.1 Functional block diagram

The functional block diagram of BTIM is shown in the following figure:

Figure 11-1 BTIM functional block diagram



#### 11.3.1.1 Filter unit

BTIM supports filtering the signal input from external BTIM_ETR pin. The filter samples the ETR signal at a certain sampling frequency. When N consecutive samples of the same level are obtained, the signal is valid, otherwise the signal is invalid, so as to filter out high-frequency clutter signals.

The sampling clock of the filter unit is PCLK or a frequency division of PCLK. The frequency division ratio of PCLK and the number of sampling points N can be selected through the ETRFLT bit field of the advanced control register BTIMx_ACR.

*Caution:*

*Since the hardware samples the signal input from the external BTIM_ETR pin with the PCLK clock or the frequency division of the PCLK clock, the input signal whose frequency is higher than half of the actual sampling frequency may be missed.*

The following example illustrates the relationship between the configuration of BTIMx_ACR.ETRFLT and the cutoff frequency.

Example:

When the PCLK clock frequency is 24MHz, set BTIMx_ACR.ETRFLT to 0x05 (ie $F_{sample}$ = PCLK / 4 and N = 6), then

$$F_{sample} = 6MHz, N = 6$$

Continuous sampling to 6 valid levels, the required time is:

$$6 \times T_{sample} = 1\mu s$$

Therefore, input signals with frequencies above 0.5MHz will be filtered out.

#### 11.3.1.2 Polarity selection unit

When the BTIM is configured in gated mode, the polarity selection unit is used to select whether the signal input by the external BTIM_ETR pin is active high or active low, which is selected by the POL bit field of the basic control register BTIMx_BCR.

When setting BTIMx_BCR.POL to 0, the ETR signal is valid when it is high; when setting BTIMx_BCR.POL to 1, it is valid when the ETR signal is low level.

#### 11.3.1.3 Edge detection unit

When the BTIM is configured in the trigger start mode, the edge detection unit is used to select the edge trigger timing of the signal input by the external BTIM_ETR pin. Specifically, it is selected through the POL bit field of the basic control register BTIMx_BCR.

When BTIMx_BCR.POL is set to 0, the ETR signal is valid when the rising edge is set; when BTIMx_BCR.POL is set to 1, the ETR signal is valid when the falling edge is set.

#### 11.3.1.4 Prescaler

The prescaler divides the frequency of the counter clock source TCLK by a factor of 2, and the maximum frequency division coefficient is 32768. The prescaler coefficient is configured by the PRS bit field of the basic control register BTIMx_BCR. The frequency division coefficient configuration is shown in the following table:

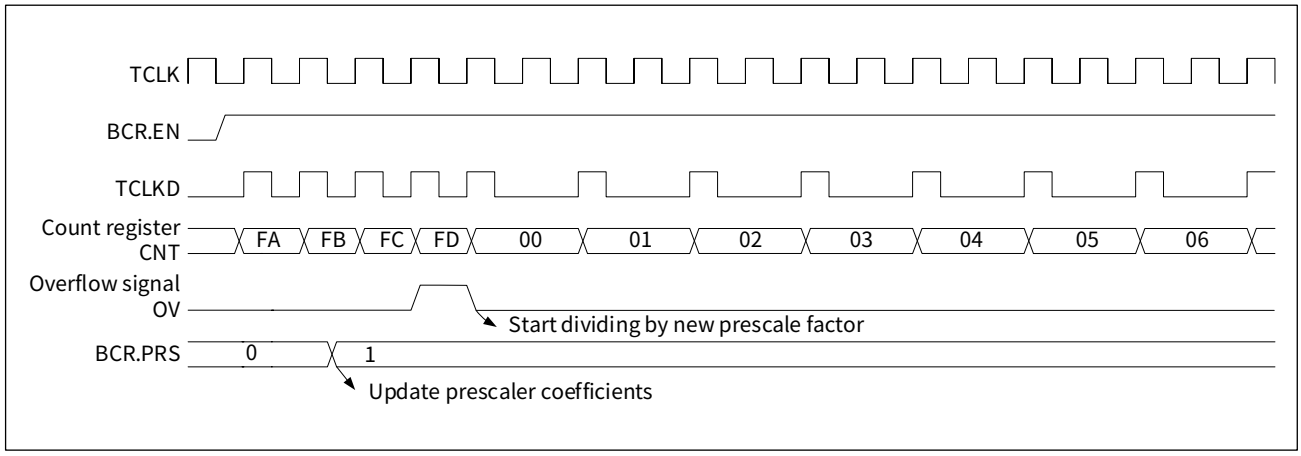
Table 11-1 Prescaler frequency division coefficient configuration table

| PRS | Frequency division ratio | PRS | Frequency division ratio | PRS | Frequency division ratio | PRS | Frequency division ratio |
|---|---|---|---|---|---|---|---|
| 0x00 | 1 | 0x04 | 16 | 0x08 | 256 | 0x0C | 4096 |
| 0x01 | 2 | 0x05 | 32 | 0x09 | 512 | 0x0D | 8192 |
| 0x02 | 4 | 0x06 | 64 | 0x0A | 1024 | 0x0E | 16384 |
| 0x03 | 8 | 0x07 | 128 | 0x0B | 2048 | 0x0F | 32768 |

It is allowed to modify BTIMx_BCR.PRS during the timer operation, but the new prescale coefficient will not take effect immediately. When the timer overflows or the running control bit BTIMx_BCR.EN changes from 0 to 1, the new prescale coefficient will not take effect immediately.

The example of changing the prescaler coefficient during operation is shown in the following figure:

Figure 11-2 Timing diagram of the counter with the prescaler coefficient changing from 1 to 2 (ARR=0xFD)



#### 11.3.1.5 Counting unit

The core component of the counting unit is a 16-bit up counter CNT and a 16-bit automatic reload register ARR.

It is allowed to modify the reload register ARR while the timer is running, and the value of ARR will take effect immediately. When the value of the counter CNT is less than the value of ARR, after modifying the current ARR value to be less than the current CNT value, the ARR overflow signal OV will not be generated, and restart counting from 0x0000.

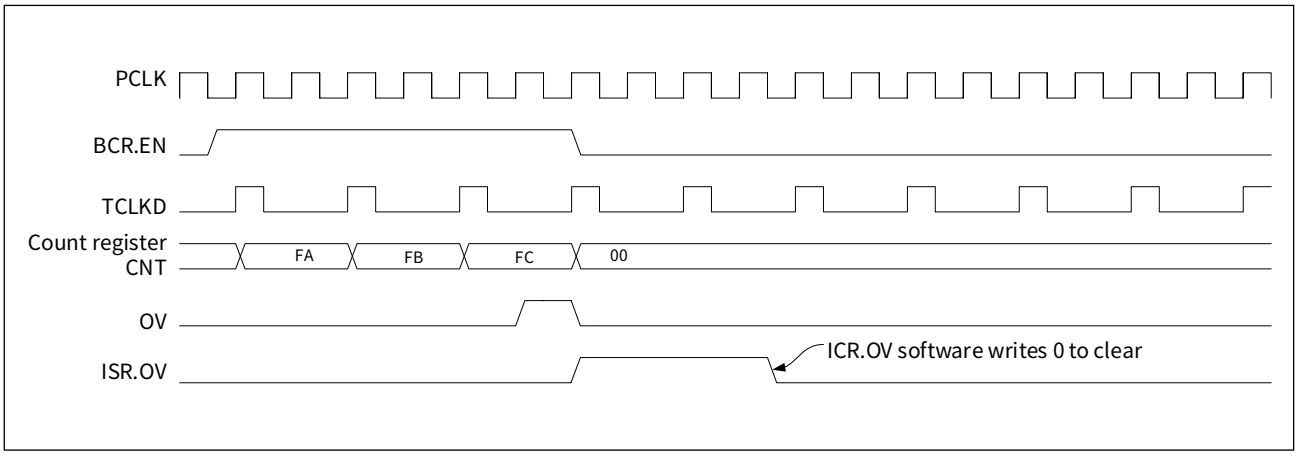
The counter can work in single counting or continuous counting mode, which is selected by the ONESHOT bit field of the basic control register BTIMx_BCR.

**Single counting mode**

Set BTIMx_BCR.ONESHOT to 1 to make the timer work in one-shot counting mode. Set BTIMx_BCR.EN to 1 to enable BTIMx, and the counter CNT counts up under the drive of the count clock TCLKD. When the count value reaches the reload value ARR, the overflow signal OV is generated (the overflow signal OV is kept for one PCLK cycle and then automatically cleared), when the OV signal is cleared, the counter ARR overflow flag bit BTIMx_ISR.OV is set by hardware, and the counter stops counting at the same time, BTIMx_BCR.EN is automatically reset by hardware.

The following figure is an example of single counting mode:

Figure 11-3 Single counting mode(PRS=0x01, ARR=0xFC)

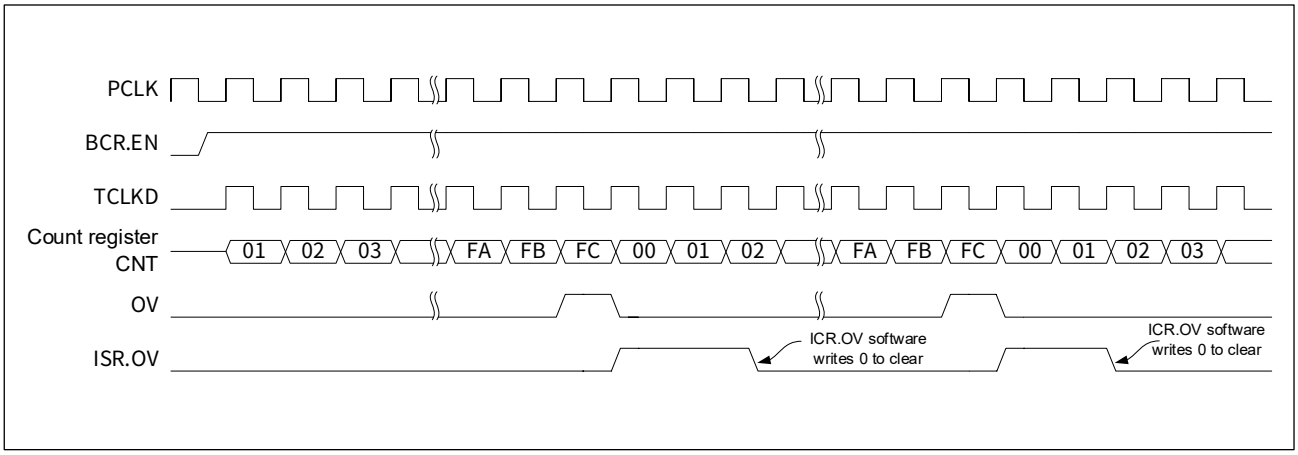


### Continuous counting mode

Set BTIMx_BCR.ONESHOT to 0 to make the timer work in continuous counting mode. Set BTIMx_BCR.EN to 1 to enable BTIMx, and the counter CNT counts up under the drive of the count clock TCLKD. When the count value reaches the reload value ARR, an overflow signal OV is generated (the overflow signal OV remains for one PCLK cycle, and then automatically cleared). When the count value changes from ARR to 0, the counter ARR overflow flag bit BTIMx_ISR.OV is set by hardware, and the counter starts to count up in the next period.

The following figure is an example of continuous counting mode:

Figure 11-4 Continuous counting mode (PRS=0x00, ARR=0xFC)



#### 11.3.1.6 Flip output unit

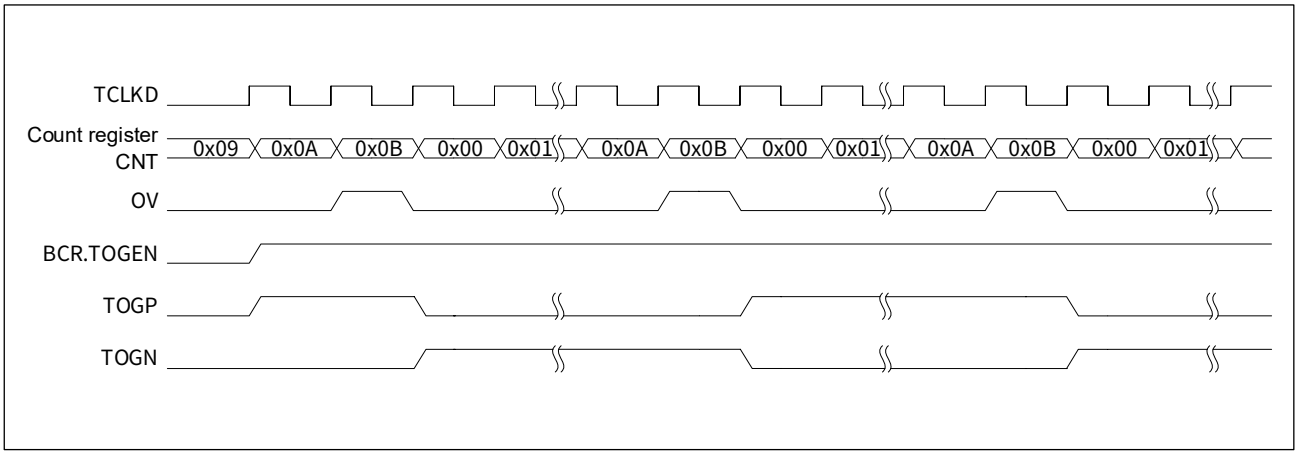
The flip output unit can control the external BTIMx_TOGP and BTIMx_TOPN pins to output the flip signal through the ARR overflow signal OV.

When BTIMx_BCR.TOGEN is set to 0, both BTIMx_TOGP and BTIMx_TOPN pins output low level.

When BTIMx_BCR.TOGEN is set to 1, the BTIMx_TOGP and BTIMx_TOPN pins output signals with opposite levels (BTIMx_TOGP default level is high); when the counter ARR overflows (BTIMx_ISR.OV is 1), the output levels of BTIMx_TOGP and BTIMx_TOPN pins will flip.

The following figure shows the schematic diagram of BTIMx_TOGP and BTIMx_TOPN pins level flip output in continuous counting mode:

Figure 11-5 Schematic diagram of level flip output (continuous counting mode, ARR=0x0B)




The TOGP/TOGN pins supported by BTIM are shown in the following table, and the corresponding GPIOs need to be multiplexed during application.

Table 11-2 BTIM toggle output pins

| TOGP/TOGN | Pin | AFR |
|---|---|---|
| BTIM1_TOGP | PB02/PC00 | 0x06 |
| BTIM1_TOGN | PA00/PC01 | 0x06 |
| BTIM2_TOGP | PA01/PA03/PB06 | 0x06 |
| BTIM2_TOGN | PA04/PB05/PB07 | 0x06 |
| BTIM3_TOGP | PA06/PB03 | 0x06 |
| BTIM3_TOGN | PA07/PB04 | 0x06 |

### 11.3.2 Operating modes

BTIM supports 4 operating modes: timer mode, counter mode, trigger start mode and gated mode. Configured through the MODE bit field of the basic control register BTImx_BCR.

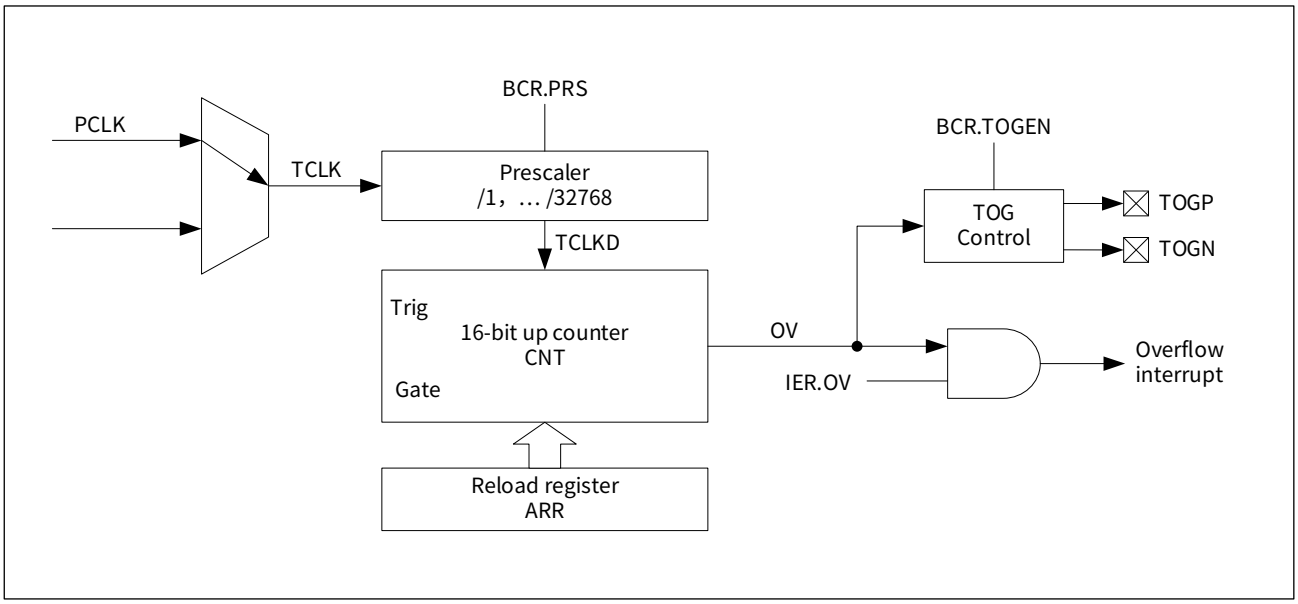
Table 11-3 BTIM operating modes

| Basic timer | MODE bit field value | Operating mode | Description |
|---|---|---|---|
| BTIMx_BCR (x=1,2,3) | 00 | Timer mode | The clock source is PCLK |
| | 01 | Counter mode | The counting source is TRS signal |
| | 10 | Trigger start mode | The clock source is PCLK, and the TRS signal triggers the counter to start |
| | 11 | Gated mode | The clock source is PCLK, and the ETR input signal is used as the gate control signal |

#### 11.3.2.1 Timer mode

The timer mode is mainly used to generate a time base signal with a fixed time interval. In this mode, the counter clock source is the internal system clock PCLK, which is divided by the prescaler to obtain the count clock TCLKD to drive the counter CNT to count. The functional block diagram of timer mode is shown in the following figure:

Figure 11-6 Timer mode block diagram




Set BTIMx_BCR.MODE to 0x00 to make BTIMx work in timer mode. Set BTIMx_BCR.EN to 1 to enable BTIMx, and the counter CNT will count up under the drive of the count clock TCLKD. When the count value reaches the reload value ARR, an overflow occurs, and the counter ARR overflow flag bit BTIMx_ISR.OV is set by hardware. If the interrupt is enabled (set the interrupt enable register BTIMx_IER.OV to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, the interrupt flag clear register BTIMx_ICR.OV should be set to 0 to clear this flag.

In practical applications, the frequency of the system clock PCLK is known. By setting the prescaler coefficient PRS reasonably, the clock with a fixed duration can be counted, and the setting of the reload value ARR and the use of the counter ARR overflow interrupt flag bit, you can accurately obtain a specific time period, so as to achieve the purpose of timing.

Timing time T calculation formula:

$$T = ( 2^{PRS} / PCLK ) \times ( ARR + 1 )$$

Where, PCLK is the counter clock source, PRS is the prescaler coefficient, and ARR is the reload value.

Example:

When the frequency of the counter clock source PCLK is 24MHz, the timing is required to be 100ms.

If the prescaler coefficient PRS is set to 0x08, calculate:

$$T = 100ms = ( 2^{8} / 24MHz ) \times ( ARR + 1 )$$

then

ARR = 9374 ( 0x249E )

That is, you need to set the reload value ARR to 0x249E.

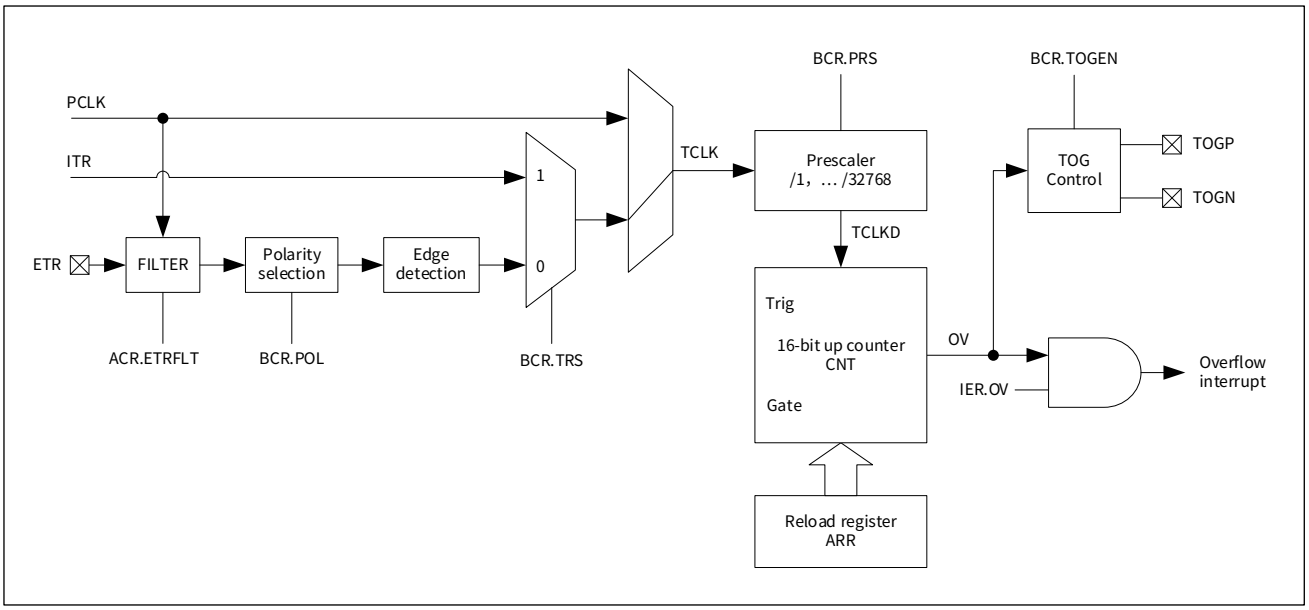
### 11.3.2.2 Counter mode

The counter mode is mainly used to measure the number of occurrences of an event. It can choose to count the signal input by the internal ITR or external BTIM_ETR pin, which is selected by the TRS bit field of the basic control register BTIMx_BCR.

When BTIMx_BCR.TRS is set to 1, the counting source is the internal ITR signal (refer to *Table 11-4 ITR source configuration*); when BTIMx_BCR.TRS is set to 0, the counting source is the signal input from the external BTIM_ETR pin, and this counting is selected When the source is used, the filter can be configured through BTIMx_ACR.ETRFLT, and the rising or falling edge of the ERT signal can be selected through BTIMx_BCR.POL to count.

The functional block diagram of counter mode is shown in the following figure:

Figure 11-7 Counter mode block diagram




Set BTIMx_BCR.MODE to 0x01 to make BTIMx work in counter mode. Set BTIMx_BCR.EN to 1 to enable BTIMx, and the counter CNT counts up the divided TCLKD signal. When the count value reaches the reload value ARR, an overflow occurs, and the counter ARR overflow flag bit BTIMx_ISR.OV is set by hardware. If the interrupt is enabled (set the interrupt enable register BTIMx_IER.OV to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, the interrupt flag clear register BTIMx_ICR.OV should be set to 0 to clear this flag.

For the specific register configuration process in counter mode, see section *11.5 Programming examples*.

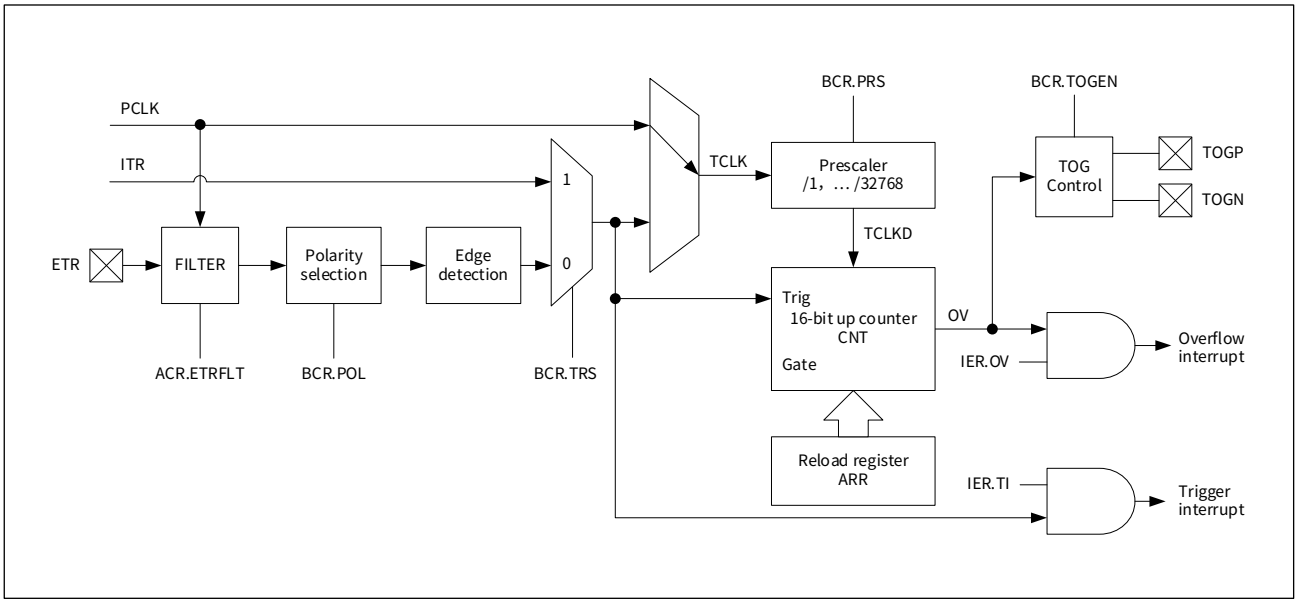
### 11.3.2.3 Trigger start mode

Set BTIMx_BCR.MODE to 0x02 to make BTIMx work in trigger start mode. In this mode, when BTImx_BCR.EN is set to 1 or the trigger signal is valid, the counter CNT will be started to count the TCLKD signal after the internal clock PCLK is divided by the prescaler.

The trigger signal can be selected from the internal ITR signal or the signal input from the external BTIM_ETR pin, which is selected by the TRS bit field of the basic control register BTIMx_BCR. When BTIMx_BCR.TRS is set to 1, the trigger source is the internal ITR signal (see *Table 11-4 ITR source configuration*); when BTIMx_BCR.TRS is set to 0, the trigger source is the external BTIM_ETR pin input signal, select When this trigger source is used, the filter can be configured through BTIMx_ACR.ETRFLT, and the ERT signal can be selected through BTIMx_BCR.POL to be valid on rising or falling edges.

The functional block diagram of trigger start mode is shown in the following figure:

Figure 11-8 Trigger start mode block diagram




When a valid trigger signal is detected, the following effects occur:

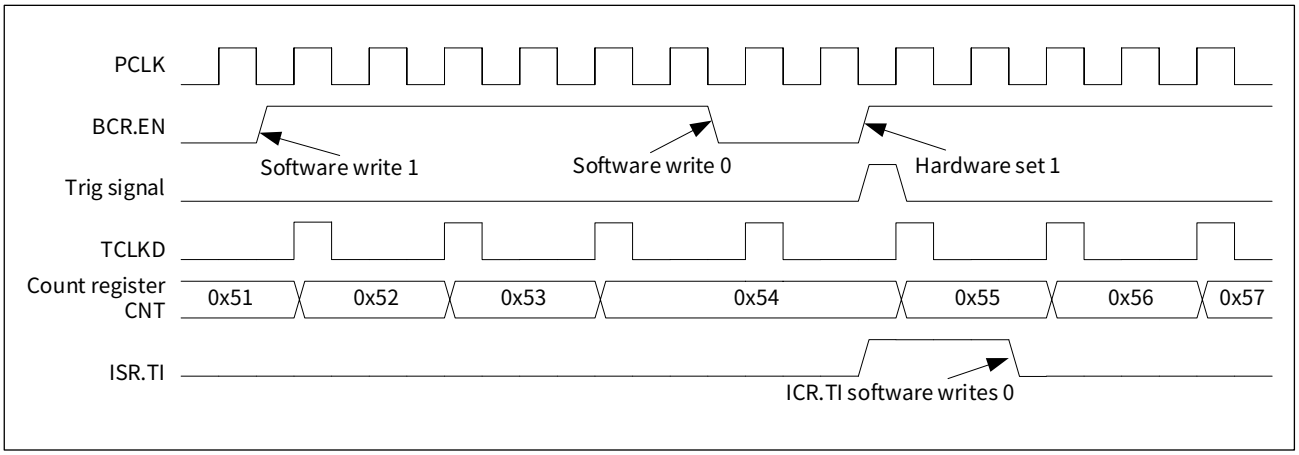
1. BTIMx_BCR.EN is set by hardware;
2. Trigger flag bit BTMx_ISR.TI is set by hardware;
3. The counter starts and starts counting.

After the counter starts, the counter starts to count up from the initial value. When the count value reaches the reload value ARR, an overflow occurs, and the counter ARR overflow flag bit BTIMx_ISR.OV is set by hardware. If the interrupt is enabled (set the interrupt enable register BTIMx_IER.OV to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, the interrupt flag clear register BTIMx_ICR.OV should be set to 0 to clear this flag.

After setting the run control bit BTIMx_BCR.EN to 0 at any time, the counter stops counting immediately. When the running control bit BTMx_BCR.EN is set to 1 again or the trigger signal is valid, the counter continues to count according to the previous setting.

The trigger start mode timing diagram is shown in the following figure:

Figure 11-9 Trigger start mode timing diagram (frequency division ratio is 2, PRS=0x01)



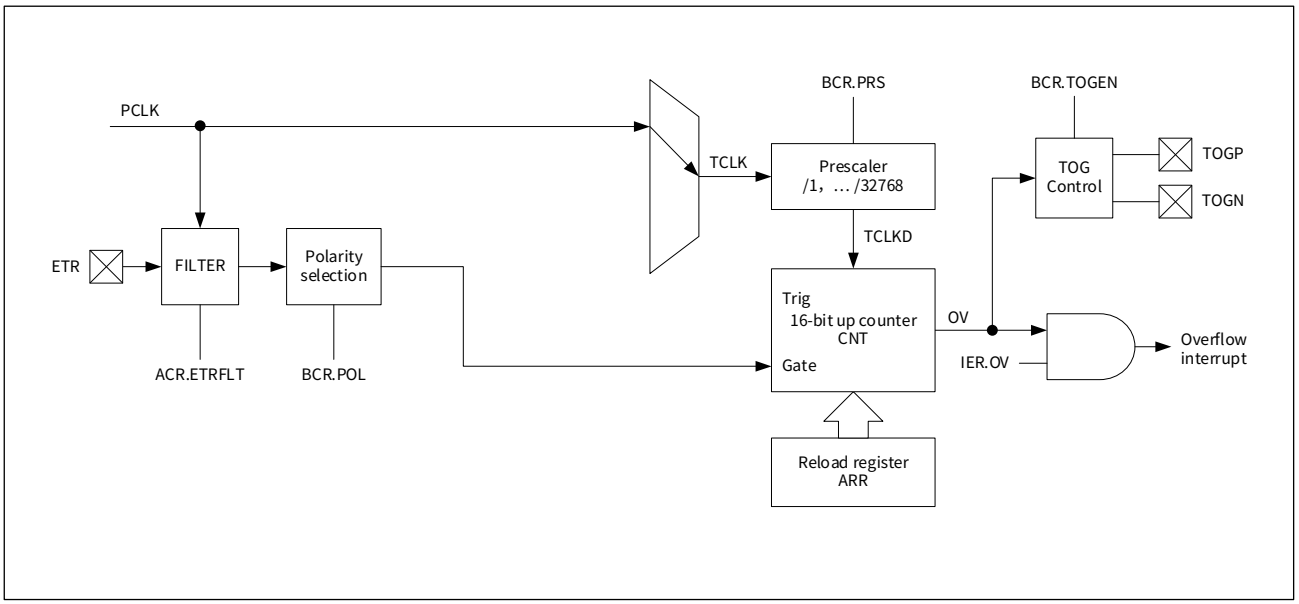

### 11.3.2.4 Gated mode

Set BTIMx_BCR.MODE to 0x03 to make BTIMx work in gated mode. In this mode, when BTMx_BCR.EN is set to 1 and the gate control signal is valid, the counter CNT will be started to count the TCLKD signal after the internal clock PCLK is divided by the prescaler.

The gate control signal is fixed as the signal input from the external BTIM_ETR pin. Set the POL bit field of the basic control register BTIMx_BCR to select whether the gate control signal is a high level or a low level as an active level. You can also use the ETRFLT bit of the advanced control register BTIMx_ACR. The domain configures whether to filter and the bandwidth of the filter.

The function block diagram of gated mode is shown in the following figure:

Figure 11-10 Gated mode block diagram

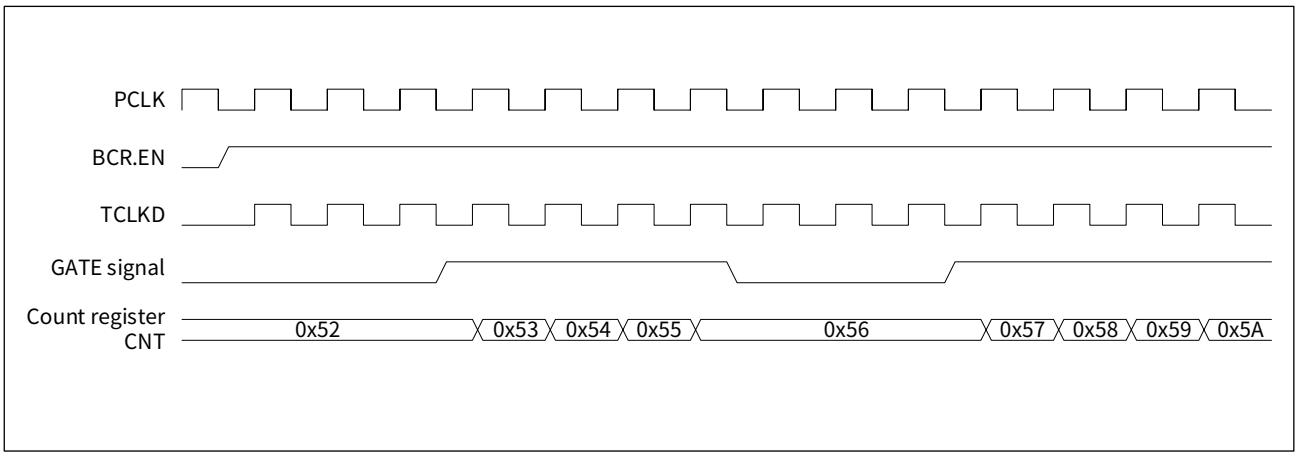


After the counter starts, the counter starts to count up from the initial value. When the count value reaches the reload value ARR, an overflow occurs, and the counter ARR overflow flag bit BTIMx_ISR.OV is set by hardware. If the interrupt is enabled (set the interrupt enable register BTIMx_IER.OV to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, the interrupt flag clear register BTIMx_ICR.OV should be set to 0 to clear this flag.

When the running control bit BTIMx_BCR.EN is set to 0 at any time or the gate control signal is invalid, the counter immediately stops counting. When the running control bit BTMx_BCR.EN is set to 1 again and the gate control signal is valid, the counter continues to count according to the previous setting.

The gated mode timing diagram is shown in the following figure:

Figure 11-11 Gated mode timing diagram (PRS=0x00, POL=0x00)

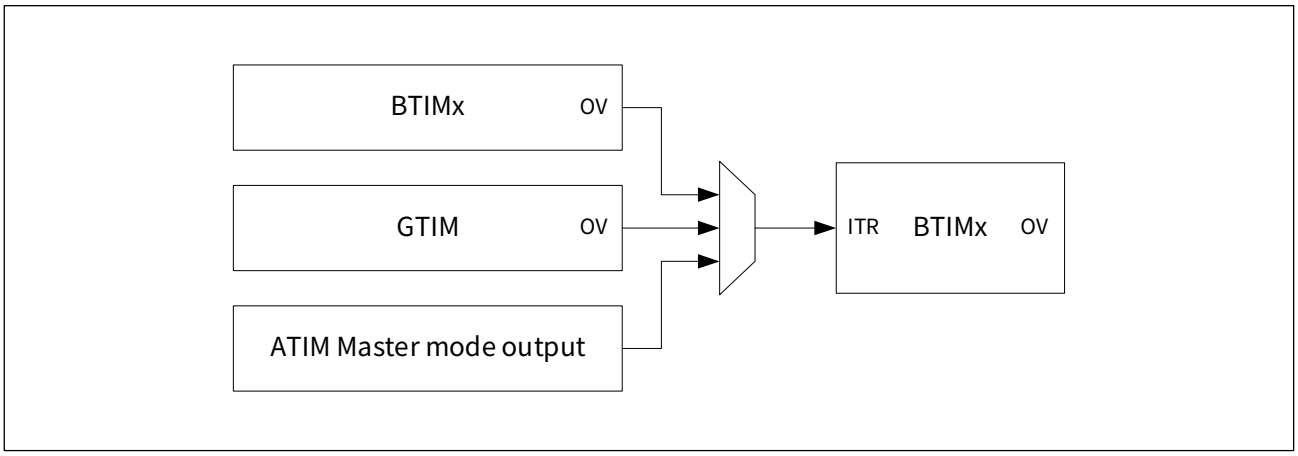


### 11.3.3 Internal cascade ITR

The timers can be cascaded to increase the bit width of the timer. For example, two 16-bit timers can be cascaded as a 32-bit timer, and the counting clock source PCLK can be 1~65535 by ITR cascade. Divide by any number in between.

The schematic diagram of BTIM two stage cascade is shown in the following figure:

Figure 11-12 Two stage cascade ITR




When used in cascade, the counting clock source of the first-level timer can be the signal input from the PLCK or external ETR pin, and the counting clock source of the subsequent-level timer is the ARR overflow signal OV of the previous-level timer or the advanced timer (ATIM) master mode output signal.

The ITR source of the timer can be selected through the timer ITR source configuration register SYSCTRL_TIMITR. The specific configuration is shown in the following table:

Table 11-4 ITR source configuration

| SYSCTRL_TIMITR bit field | Value | ITR source |
|---|---|---|
| BTIM3ITR<br>BTIM2ITR<br>BTIM1ITR<br>GTIMITR<br>ATIMITR | 000 | Overflow signal of BTIM1 |
| | 001 | Overflow signal of BTIM2 |
| | 010 | Overflow signal of BTIM3 |
| | 011 | Overflow signal of GTIM |
| | 111 | Master mode output signal of ATIM |

*Caution:*

*The timer's ITR source cannot select its own overflow signal OV.*

### 11.3.4 External interconnection ETR

BThe ETR signal source of BTIM can only be the external BTIM_ETR pin, and all BTIMs share one ETR signal, which is configured by the GPIO multiplexing function register GPIOx_AFRL, as shown in the following table:

Table 11-5 External BTIM_ETR pin multiplexing

| ETR | Pin | AFR |
|---|---|---|
| BTIM_ETR | PA05 | 0x06 |
| BTIM_ETR | PB05 | 0x03 |
| BTIM_ETR | PB07 | 0x07 |

## 11.4 Debugging support

BTIM supports to stop or continue counting in debug mode, which is set by the BTIM123 bit field of the debug status timer control register SYSCTRL_DEBUG.

Set SYSCTRL_DEBUG.BTIM123 to 1, then suspend the counter counting of BTIM1, BTIM2 and BTIM3 in the debug state;

Set SYSCTRL_DEBUG.BTIM123 to 0, then the counters of BTIM1, BTIM2 and BTIM3 continue to count in the debug state.

## 11.5 Programming examples

### 11.5.1 Timer mode programming example

Step 1: Set SYSCTRL_APBEN2.BTIM to 1, turn on the configuration clock and working clock of BTIM1, BTIM2 and BTIM3;

*Caution:*

*BTIM1, BTIM2 and BTIM3 share the SYSCTRL_APBEN2.BTIM bit.*

Step 2: Set BTIMx_BCR.MODE to 0x00 to make BTIMx work in timer mode;

Step 3: Configure BTMx_BCR.ONESHOT to select single or continuous counting mode. If it is configured as 0, it is continuous counting mode, and if it is configured as 1, it is single counting mode;

Step 4: Configure BTMx_BCR.PRS and select the prescaler ratio;

Step 5: Configure BTIMx_ARR, set BTIMx count overflow time;

Step 6: Set BTIMx_CNT to 0x0000 for counting;

Step 7: If you want the ARR overflow event to trigger an interrupt, set BTIMx_IER.OV to 1;

Step 8: Set BTIMx_BCR.EN to 1, start BTIMx;

Step 9: If an interrupt occurs, set BTIMx_ICR.OV to 0 to clear the interrupt flag.

### 11.5.2 Counter mode programming examples

#### 11.5.2.1 Count external ETR signals

To count the signals input by the external BTIM_ETR pin, the steps are as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, and enable the configuration clock and working clock of the GPIO port corresponding to the BTIM_ETR pin;

Step 2: Set the relevant bits of the GPIO multiplexing function registers (GPIOx_AFRH and GPIOx_AFRL), and configure the source of the ETR to the required hardware pins;

Step 3: Set the SYSCTRL_APBEN2.BTIM bit to 1, and turn on the configuration clock and working clock of BTIM1, BTIM2 and BTIM3;

*Caution:*

*BTIM1, BTIM2 and BTIM3 share the SYSCTRL_APBEN2.BTIM bit.*

Step 4: Configure BTIMx_BCR.MODE to 0x01 to make BTIMx work in counter mode;

Step 5: Configure BTIMx_BCR.ONESHOT to select single or continuous counting mode. If it is configured as 0, it is continuous counting mode, and if it is configured as 1, it is single counting mode;

Step 6: Configure BTIMx_BCR.TRS to be 0, and select the counting source as the signal input from the external BTIM_ETR pin;

Step 7: Configure BTIMx_ACR.ETRFLT, choose whether to filter or filter bandwidth;

Step 8: Configure BTImx_BCR.POL to select the ETR signal count edge. When configured as 0, the rising edge counts, and when configured as 1, the falling edge counts;

Step 9: Configure BTMx_BCR.PRS and select the prescaler ratio;

Step 10: Configure BTIMx_ARR, set BTIMx count overflow time;

Step 11: If you want the ARR overflow event to trigger an interrupt, set BTIMx_IER.OV to 1;

Step 12: Set BTIMx_BCR.EN to 1, start BTIMx;

Step 13: If an interrupt occurs, set BTIMx_ICR.OV to 0 to clear the interrupt flag.

#### 11.5.2.2 Count internal ITR signals

Step 1: Set the SYSCTRL_APBEN2.BTIM bit to 1, and turn on the configuration clock and working clock of BTIM1, BTIM2 and BTIM3;

*Caution:*

*BTIM1, BTIM2 and BTIM3 share the SYSCTRL_APBEN2.BTIM bit.*

Step 2: Configure BTIMx_BCR.MODE to 0x01, select BTIMx to work in counter mode;

Step 3: Configure BTMx_BCR.ONESHOT to select single or continuous counting mode. If it is configured as 0, it is continuous counting mode, and if it is configured as 1, it is single counting mode;

Step 4: Configure BTImx_BCR.TRS to 1, select the count signal from the internal ITR signal;

Step 5: Configure the SYSCTRL_TIMITR register and select the ITR source of the corresponding timer;

Step 6: Configure BTIMx_BCR.PRS and select the prescaler ratio;

Step 7: Configure BTIMx_ARR, select BTIMx count overflow time;

Step 8: If you want the ARR overflow event to trigger an interrupt, configure BTIMx_IER.OV to 1;

Step 9: Set BTIMx_BCR.EN to 1, start BTIMx;

Step 10: If an interrupt occurs, set BTIMx_ICR.OV to 0 to clear the interrupt flag.

### 11.5.3 Trigger start mode programming example

Step 1: Set the SYSCTRL_APBEN2.BTIM bit to 1, and turn on the configuration clock and working clock of BTIM1, BTIM2 and BTIM3;

*Caution:*

*BTIM1, BTIM2 and BTIM3 share the SYSCTRL_APBEN2.BTIM bit.*

Step 2: Configure BTIMx_BCR.TRS and select the trigger source. When configured as 0, the trigger signal comes from the external BTIM_ETR pin input signal; when configured as 1, the trigger signal comes from the internal cascaded ITR signal;

Step 3: If the trigger signal is input from the BTIM_ETR pin, BTIMx_ACR.ETRFLT can be configured to select whether to filter or filter bandwidth;

Step 4: If the trigger signal is input from the BTIM_ETR pin, configure BTIMx_BCR.POL and select the ETR signal to trigger the edge. If it is configured as 0, the rising edge counting starts, and if it is configured as 1, the falling edge counting starts;

Step 5: Configure BTMx_BCR.PRS and select the prescaler ratio;

Step 6: Configure BTMx_BCR.ONESHOT to select single or continuous counting mode. If it is configured as 0, it is continuous counting mode, and if it is configured as 1, it is single counting mode;

Step 7: If you want the trigger event to trigger an interrupt, set BTIMx_IER.IT to 1;

Step 8: If you want the ARR overflow event to cause an interrupt, set BTIMx_IER.OV to 1;

Step 9: Configure BTIMx_ARR, select BTIMx count overflow time;

Step 10: Configure BTIMx_BCR.MODE to 0x02, select BTIMx to work in trigger start mode;

Step 11: Determine whether to write 1 to BTIMx_BCR.EN as needed to immediately start BTIMx for counting;

Step 12: If an interrupt occurs, write 0 to the corresponding bit of the BTIMx_ICR register to clear the interrupt flag according to the event of the interrupt.

*Caution:*

*Set the polarity of the trigger signal first, and then set the timer mode, otherwise it will cause false triggering.*

### 11.5.4 Gated mode programming example

The signal input from the external BTIM_ETR pin is used as the gate control signal to control the count of the counter. The operation steps are as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, and enable the configuration clock and working clock of the GPIO port corresponding to the BTIM_ETR pin;

Step 2: Set the relevant bits of the GPIO multiplexing function registers (GPIOx_AFRH and GPIOx_AFRL), and configure the source of the ETR to the required hardware pins;

Step 3: Set the corresponding bit of the GPIOx_ANALOG register of the corresponding GPIO pin to 0, and configure it as a digital function;

Step 4: Set the corresponding bit of the GPIOx_DIR register of the corresponding GPIO pin to 1, and configure the port as an input;

Step 5: Set the SYSCTRL_APBEN2.BTIM bit to 1, and turn on the configuration clock and working clock of BTIM1, BTIM2 and BTIM3;

*Caution:*

*BTIM1, BTIM2 and BTIM3 share the SYSCTRL_APBEN2.BTIM bit.*

Step 6: Configure BTIMx_BCR.MODE to 0x03, select BTIMx to work in gated mode;

Step 7: Configure BTIMx_ACR.ETRFLT, choose whether to filter or filter bandwidth;

Step 8: Configure BTImx_BCR.POL to select ETR signal polarity. When configured as 0, the gate control mode is active at high level, and when configured as 1, the gated mode is active at low level;

Step 9: Configure BTMx_BCR.PRS and select the prescaler ratio;

Step 10: Configure BTIMx_BCR.ONESHOT to select single or continuous counting mode. When configured as 0, it is continuous counting mode, and when configured as 1, it is single counting mode;

Step 11: If you want the ARR overflow event to trigger an interrupt, configure BTIMx_IER.OV to 1;

Step 12: Configure the BTIMx_ARR register and select the BTIMx count overflow time;

Step 13: Set BTIMx_BCR.EN to 1, start BTIMx;

Step 14: If an interrupt occurs, set BTIMx_ICR.OV to 0 to clear the interrupt flag.

## 11.6 List of registers

BTIM1 base address: BTIM1_BASE = 0x4001 4800

BTIM2 base address: BTIM2_BASE = 0x4001 4900

BTIM3 base address: BTIM3_BASE = 0x4001 4A00

Table 11-6 List of BTIM registers

| Register name | Register address | Register description |
|---|---|---|
| BTIMx_ARR | BTIMx_BASE + 0x00 | Reload register |
| BTIMx_CNT | BTIMx_BASE + 0x04 | Count register |
| BTIMx_ACR | BTIMx_BASE + 0x0C | Advanced control register |
| BTIMx_BCR | BTIMx_BASE + 0x10 | Basic control register |
| BTIMx_IER | BTIMx_BASE + 0x14 | Interrupt enable register |
| BTIMx_ISR | BTIMx_BASE + 0x18 | Interrupt flag register |
| BTIMx_ICR | BTIMx_BASE + 0x1C | Interrupt flag clear register |

## 11.7 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 11.7.1 BTIMx_BCR basic control register

Address offset: 0x10 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:15 | RFU | - | Reserved bits, please keep the default value |
| 14:11 | PRSSTATUS | RO | The division coefficient currently being used by the prescaler<br>The value is updated from PRS when the timer overflows or when EN changes from 0 to 1 |
| 10:7 | PRS | RW | Prescaler division coefficient configuration<br>0: DIV1 4: DIV16 8: DIV256 12: DIV4096<br>1: DIV2 5: DIV32 9: DIV512 13: DIV8192<br>2: DIV4 6: DIV64 10: DIV1024 14: DIV16384<br>3: DIV8 7: DIV128 11: DIV2048 15: DIV32768 |
| 6 | TOGEN | RW | TOG pin output enable control<br>0: TOGP, TOGN output level is 0<br>1: TOGP, TOGN output signals with opposite levels |
| 5 | ONESHOT | RW | Single/continuous counting mode control<br>0: Continuous counting mode<br>1: Single counting mode |
| 4 | POL | RW | ETR signal polarity selection for external pin input<br>0: ETR is positive (the rising edge of trigger mode is valid, and the gated mode is valid at high level)<br>1: ETR is reversed (the falling edge of trigger mode is valid, and the gated mode is valid at low level) |
| 3 | TRS | RW | Trigger source selection<br>0: The signal input by the ETR pin<br>1: ITR, see section *11.3.3 Internal cascade ITR* |
| 2:1 | MODE | RW | Basic timing mode configuration<br>00: Timer mode, counting clock source is PCLK<br>01: Counter mode, counting clock source is TRS signal<br>10: 10: Trigger start mode, the counting clock source is PCLK, and the TRS signal triggers the counter to start<br>11: Gated mode, the counting clock source is PCLK, and the ETR pin input signal is used as gate control |
| 0 | EN | RW | Timer running control<br>0: Timer stopped<br>1: Timer running |

### 11.7.2 BTIMx_ACR advanced control register

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6:4 | ETRFLT | RW | The ETR pin input signal filtering configuration; the sampling clock is PCLK or PCLK frequency division, the number of sampling points is N, and N consecutive levels are valid.<br>000: no filter<br>001: $F_{sample}$ = PCLK, N=2<br>010: $F_{sample}$ = PCLK, N=4<br>011: $F_{sample}$ = PCLK, N=6<br>100: $F_{sample}$ = PCLK/4, N=4<br>101: $F_{sample}$ = PCLK/4, N=6<br>110: $F_{sample}$ = PCLK/16, N=4<br>111: $F_{sample}$ = PCLK/16, N=6 |
| 3:0 | RFU | - | Reserved bits, please keep the default value |

### 11.7.3 BTIMx_ARR reload register

Address offset: 0x00 Reset value: 0x0000 FFFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | ARR | RW | Timer reloads value |

### 11.7.4 BTIMx_CNT count register

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CNT | RW | Timer count value |

### 11.7.5 BTIMx_IER interrupt enable register

Address offset: 0x14 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:3 | RFU | - | Reserved bits, please keep the default value |
| 2 | TOP | RW | Counter TOP overflow interrupt enable control<br>0: TOP overflow interrupt disabled<br>1: TOP overflow interrupt enabled |
| 1 | TI | RW | Trigger interrupt enable control<br>0: Trigger interrupt disabled<br>1: Trigger interrupt enabled |
| 0 | OV | RW | Counter ARR overflow interrupt enable control<br>0: ARR overflow interrupt disabled<br>1: ARR overflow interrupt enabled |

### 11.7.6 BTIMx_ISR interrupt flag register

Address offset: 0x18 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:3 | RFU | - | Reserved bits, please keep the default value |
| 2 | TOP | RO | Counter TOP overflow flag<br>0: The counter has not TOP overflowed<br>1: The counter has TOP overflowed |
| 1 | TI | RO | Trigger flag<br>0: No trigger event occurred<br>1: The trigger event has occurred |
| 0 | OV | RO | Counter ARR overflow flag<br>0: The counter has not ARR overflowed<br>1: The counter has ARR overflowed |

### 11.7.7 BTIMx_ICR interrupt flag clear register

Address offset: 0x1C Reset value: 0x0000 0007

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:3 | RFU | - | Reserved bits, please keep the default value |
| 2 | TOP | R1W0 | Counter TOP overflow flag cleared<br>W0: Counter TOP overflow flag cleared<br>W1: No function |
| 1 | TI | R1W0 | Trigger flag cleared<br>W0: Trigger flag cleared<br>W1: No function |
| 0 | OV | R1W0 | Counter ARR overflow flag cleared<br>W0: Counter ARR overflow flag cleared<br>W1: No function |

# 12 General-purpose timer (GTIM)

## 12.1 Overview

CW32F003 integrates one general-purpose timer (GTIM), GTIM contains a 16bit automatic reload counter and is driven by a programmable prescaler. GTIM supports 4 basic working modes: timer mode, counter mode, trigger start mode and gate control mode. Each group has 4 independent capture/compare channels, which can measure the pulse width of input signal (input capture) or generate output waveforms (output compare and PWM).

## 12.2 Main features

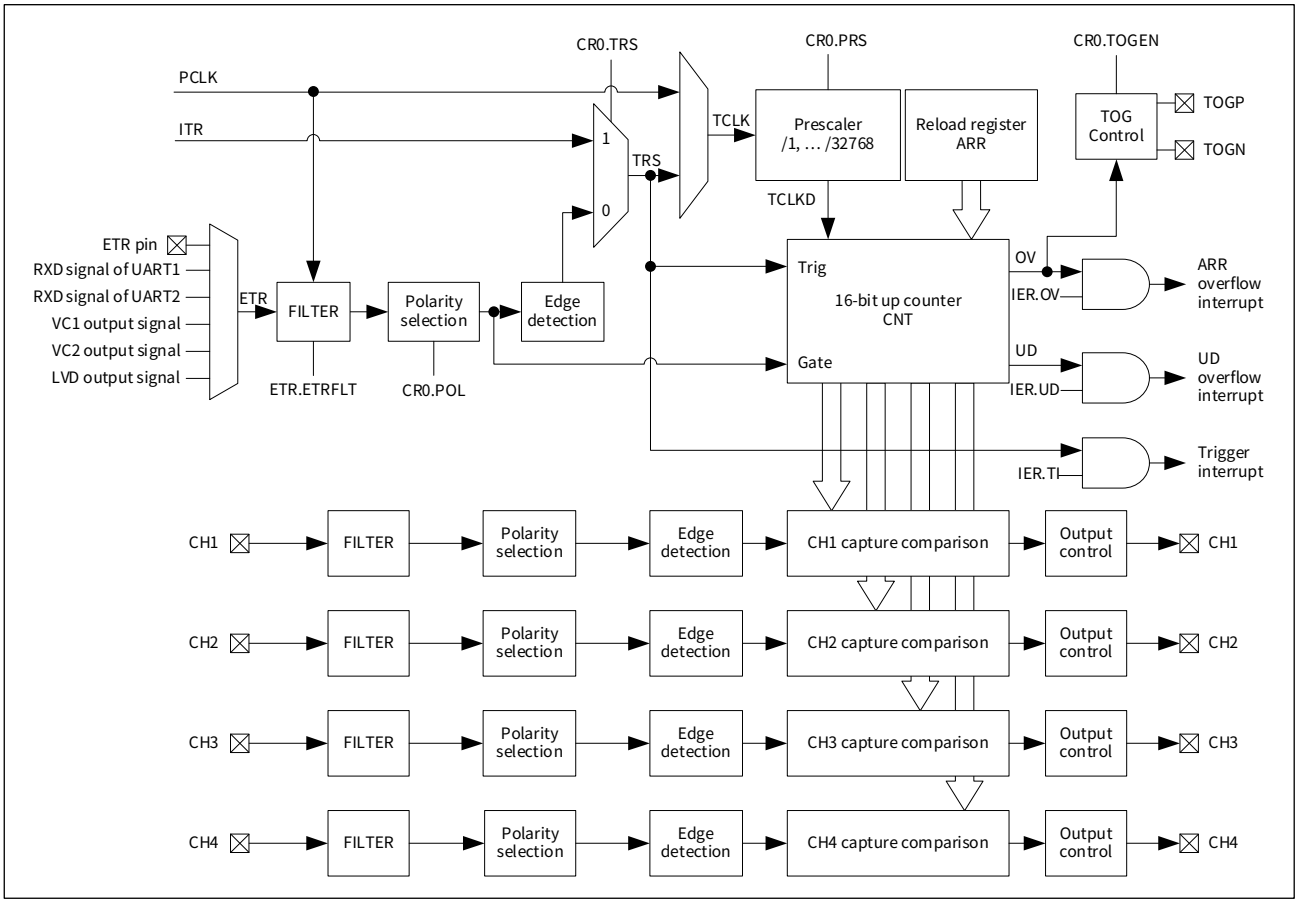
- 16-bit auto-reload counter
- Programmable prescaler supports 1,2,4,8,⋯,32768 frequency division
- Supports single counting mode and continuous counting mode
- Counting function for internal ITR or ETR input signal
- Internal ITR or ETR input signal triggers start counting
- 4 independent input capture and output compare channels
- Supports incremental (quadrature) encoders for positioning
- Overflow event triggers digital output level transition
- Internal cascade ITR and on-chip peripheral interconnect ETR
- Interrupt generated when various events occur
  - Counter count overflow
  - The counter is triggered to start
  - Input capture
  - Output compare

## 12.3 Functional description

### 12.3.1 Functional block diagram

The functional block diagram of GTIM is shown in the following figure:

Figure 12-1 GTIM functional block diagram



#### 12.3.1.1 Prescaler

The prescaler divides the frequency of the counter clock source TCLK by a factor of 2, and the maximum frequency division factor is 32768. The prescaler coefficient is configured by the PRS bit field of the control register GTIM_CR0, and the frequency division coefficient configuration is shown in the following table:

Table 12-1 Prescaler frequency division coefficient configuration table

| PRS | Frequency division ratio | PRS | Frequency division ratio | PRS | Frequency division ratio | PRS | Frequency division ratio |
|---|---|---|---|---|---|---|---|
| 0x00 | 1 | 0x04 | 16 | 0x08 | 256 | 0x0C | 4096 |
| 0x01 | 2 | 0x05 | 32 | 0x09 | 512 | 0x0D | 8192 |
| 0x02 | 4 | 0x06 | 64 | 0x0A | 1024 | 0x0E | 16384 |
| 0x03 | 8 | 0x07 | 128 | 0x0B | 2048 | 0x0F | 32768 |

It is allowed to modify GTIM_CR0.PRS during the timer operation, but the new prescale factor will not take effect immediately. When the timer overflows or the running control bit GTIM_CR0.EN changes from 0 to 1, the new prescale factor effective.

### 12.3.1.2 Counting unit

The core component of the counting unit is a 16-bit counter CNT and a 16-bit automatic reload register ARR.

It is allowed to modify the reload register ARR while the timer is running, and the value of ARR will take effect immediately. The correct counting range of the counter CNT is between 0 and the reload value ARR. Therefore, when setting the reload register ARR, ensure that the value of CNT is less than the set value of the ARR register.

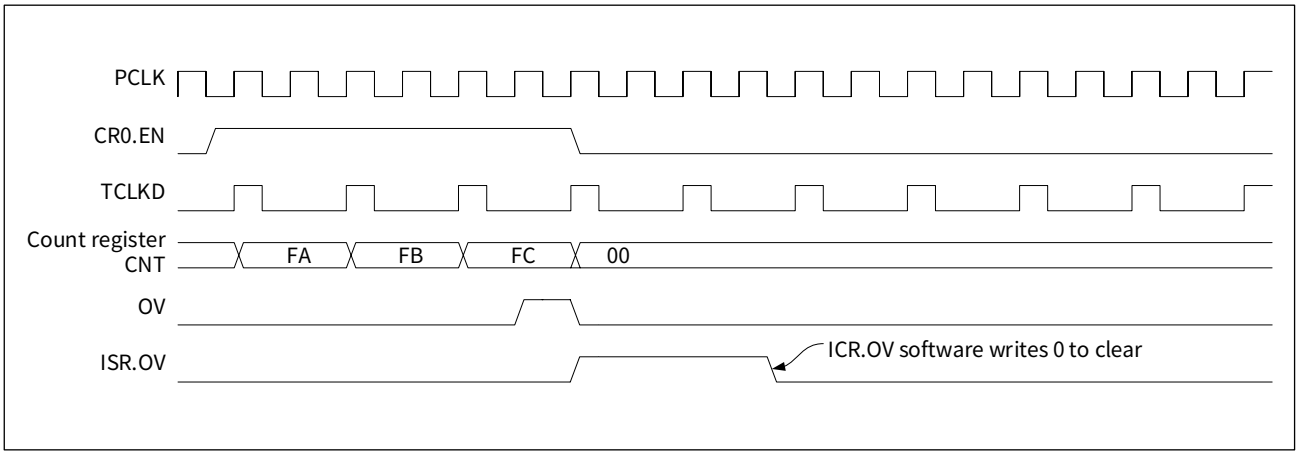
The counter can work in single count or continuous count mode, which is selected by the ONESHOT bit field of the control register GTIM_CR0.

#### Single counting mode

Set GTIM_CR0.ONESHOT to 1 to make the timer work in single counting mode. Set GTIM_CR0.EN to 1 to enable GTIM, and the counter CNT counts up under the drive of the TCLKD clock. When the count value reaches the reload value ARR, an overflow signal OV is generated (the overflow signal OV remains for one PCLK cycle, and then automatically cleared), and the counter stops counting, and GTIM_CR0.EN is automatically reset by hardware.

The following figure is an example of single counting mode:

Figure 12-2 Single counting mode (PRS=0x01, ARR=0xFC)

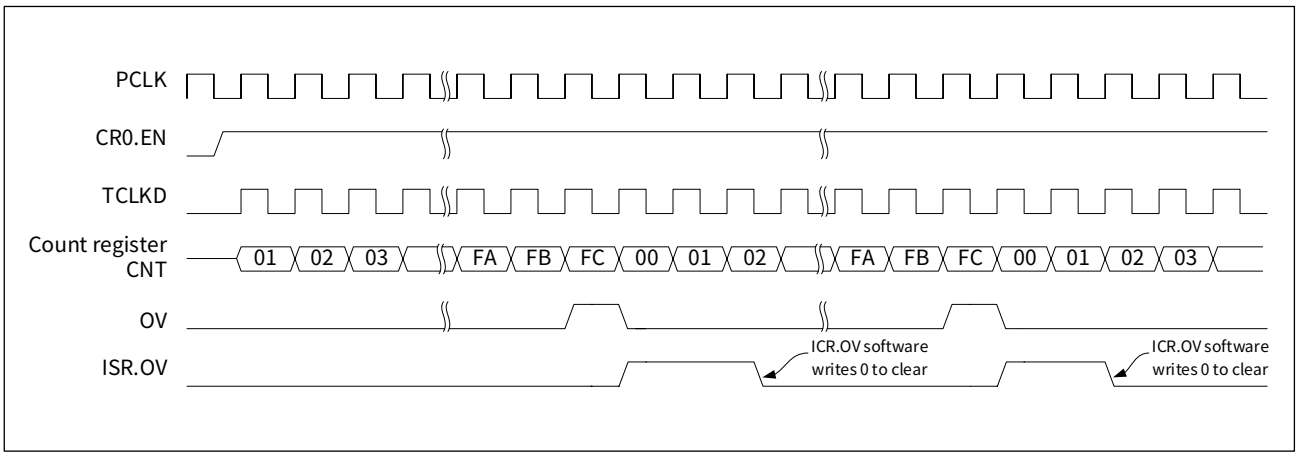


### Continuous counting mode

Set GTIM_CR0.ONESHOT to 0 to make the timer work in continuous counting mode. Set GTIM_CR0.EN to 1 to enable GTIM, and the counter CNT counts up under the drive of the TCLKD clock. When the count value reaches the reload value ARR, an overflow signal OV is generated (the overflow signal OV remains for one PCLK cycle, and then automatically cleared). When the count value changes from ARR to 0, the counter ARR overflow flag bit GTIM_ISR.OV is set by hardware, the counter starts to count up in the next cycle, and stops counting until GTIM_CR0.EN is set to 0.

The following figure is an example of continuous counting mode:

Figure 12-3 Continuous counting mode (PRS=0x00, ARR=0xFC)




### Encoding counting mode

When the register bit GTIM_CR0.ENCMODE is not 0x00, the counter works in the encoding counting mode. At this time, the counting direction of the counter is specified by the hardware, which can count up or count down. The counter CNT is automatically incremented or decremented by 1 according to GTIM_ISR. DIR. Whenever counting up to ARR, the counter will generate an overflow signal OV, and automatically start counting up from 0 again; whenever counting down to 0, the counter will generate an underflow signal UD, and start counting down from 0xFFFF.

#### 12.3.1.3 Input control unit

The ETR input signal and the four compare input signals CH1~CH4 all have their own input control units. The input control unit consists of filters, polarity selection and edge detection.

**Filter**

The filter adopts a digital filtering method to sample the input signal at a certain frequency. When N consecutive samples of the same level are obtained, the signal is valid, otherwise the signal is invalid, so as to filter out high-frequency clutter signals.

*Caution:*

*Since the hardware samples the input signal with the PCLK clock or the frequency division of the PCLK clock, the input signal whose frequency is higher than half of the actual sampling frequency may be missed.*

The filter of the ETR input signal is configured by the external trigger control register GTIM_ETR.ETRFLT to configure the sampling clock frequency and the number of sampling points. The filter of the comparison input CHy is configured by the control register GTIM_CR1.CHyFLT to configure the sampling clock frequency and the number of sampling points (y=1, 2, 3, 4).

**Polarity selection**

Polarity selection is used to select whether to invert the input signal. The ETR input signal is selected to be inverted through the control register GTIM_CR0.POL and applied to the gated mode. The input signal of the compare input CHy is selected by the control register GTIM_CR1.CHyPOL whether to invert or not.

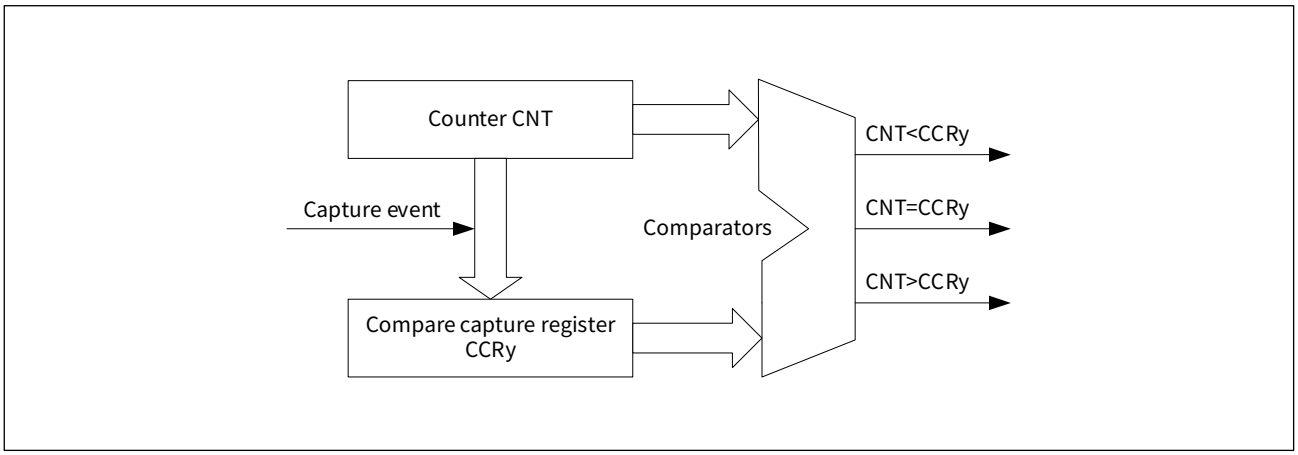
**Edge detection**

Edge detection is used to detect the transition of the input signal and whether the direction of the transition is rising or falling.

#### 12.3.1.4 Capture compare channel

The capture compare channel consists of a compare capture register and a comparator, as shown in the following figure:

Figure 12-4 Schematic diagram of the capture compare channel




When the input is captured, once the corresponding channel has a capture event, the current value of the counter CNT is immediately transferred to the compare capture register CCRy to save, if the corresponding capture compare interrupt is enabled, an interrupt request will be generated.

When the output is compared, the value of the counter CNT is always compared with the value of the compare capture register CCRy. When the compare result is the set requirement, a signal can be generated to control the output to change. An interrupt request will be generated if the corresponding capture compare interrupt is enabled.

#### 12.3.1.5 Output control unit

The output control unit is used to control the waveform of the output port when comparing and matching. The compare output mode of the corresponding channel is configured through the CCyM bit field of the compare capture control register GTIM_CMMR. For the specific configuration and application, see section *12.3.4 Output compare function*.

#### 12.3.1.6 Flip output unit

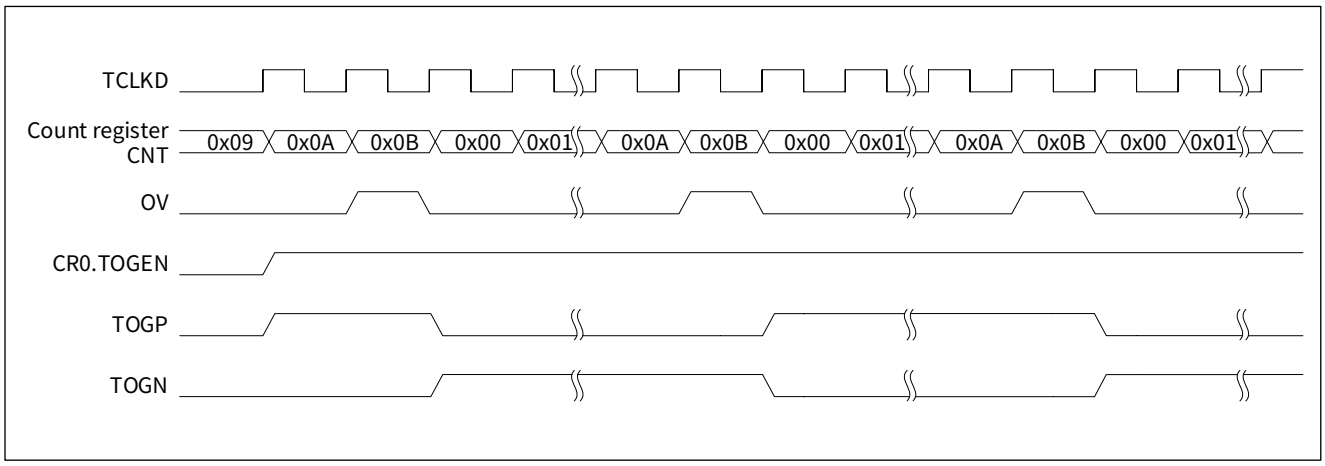
The flip output unit can control the external GTIM_TOGP and GTIM_TOPN pins to output the toggle signal through the ARR overflow signal OV.

When GTIM_CR0.TOGEN is set to 0, both GTIM_TOGP and GTIM_TOPN pins output low level.

When GTIM_CR0.TOGEN is set to 1, the GTIM_TOGP and GTIM_TOPN pins output signals with opposite levels (the default level of GTIM_TOGP is high); when the counter ARR overflows (GTIM_ISR.OV is 1), the GTIM_TOGP and GTIM_TOPN pins output The level will flip.

The following figure shows the schematic diagram of GTIM_TOGP and GTIM_TOPN pin level flip output in continuous counting mode:

Figure 12-5 Schematic diagram of level flip output (continuous counting mode, ARR=0x0B)




The TOGP/TOGN pins supported by GTIM are shown in the following table:

Table 12-2 GTIM flip output pins

| TOGP/TOGN | Pin | AFR |
|---|---|---|
| GTIM_TOGP | PB00/PC03 | 0x06 |
| GTIM_TOGP | PB06 | 0x05 |
| GTIM_TOGN | PB01/PC04 | 0x06 |
| GTIM_TOGN | PB05 | 0x05 |

### 12.3.2 Basic operating modes

GTIM supports 4 basic operating modes: timer mode, counter mode, trigger start mode and gated mode. Configured by the MODE bit field of the control register GTIM_CR0.

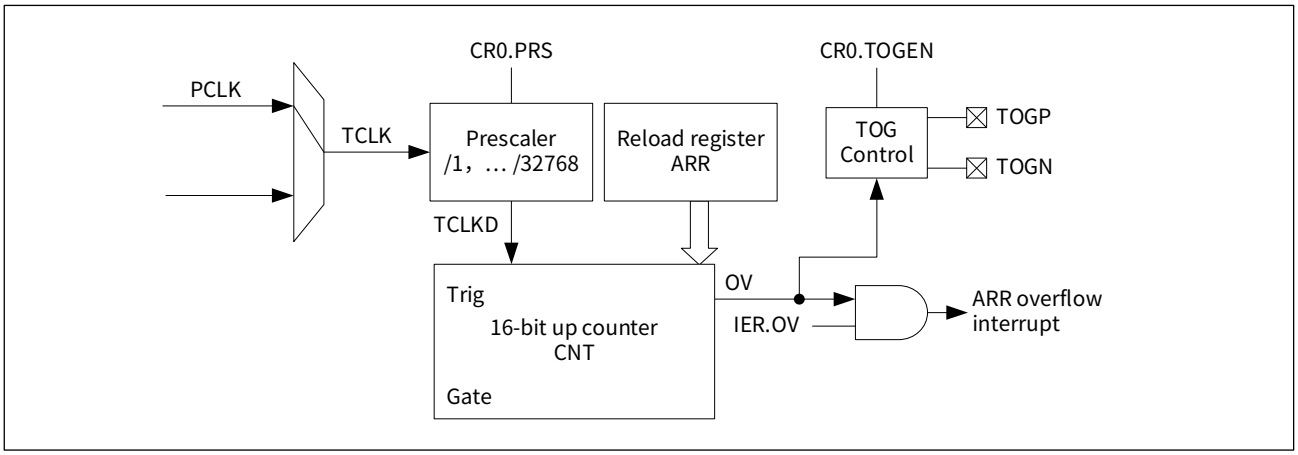
Table 12-3 GTIM basic operating modes

| General-purpose timer | MODE bit field value | Basic operating mode | Description |
|---|---|---|---|
| GTIM_CR0 | 00 | Timer mode | The clock source is PCLK |
| | 01 | Counter mode | The counting source is TRS signal |
| | 10 | Trigger start mode | The clock source is PCLK, and the TRS signal triggers the counter to start |
| | 11 | Gated mode | The clock source is PCLK, and the ETR input signal is used as the gate control signal |

#### 12.3.2.1 Timer mode

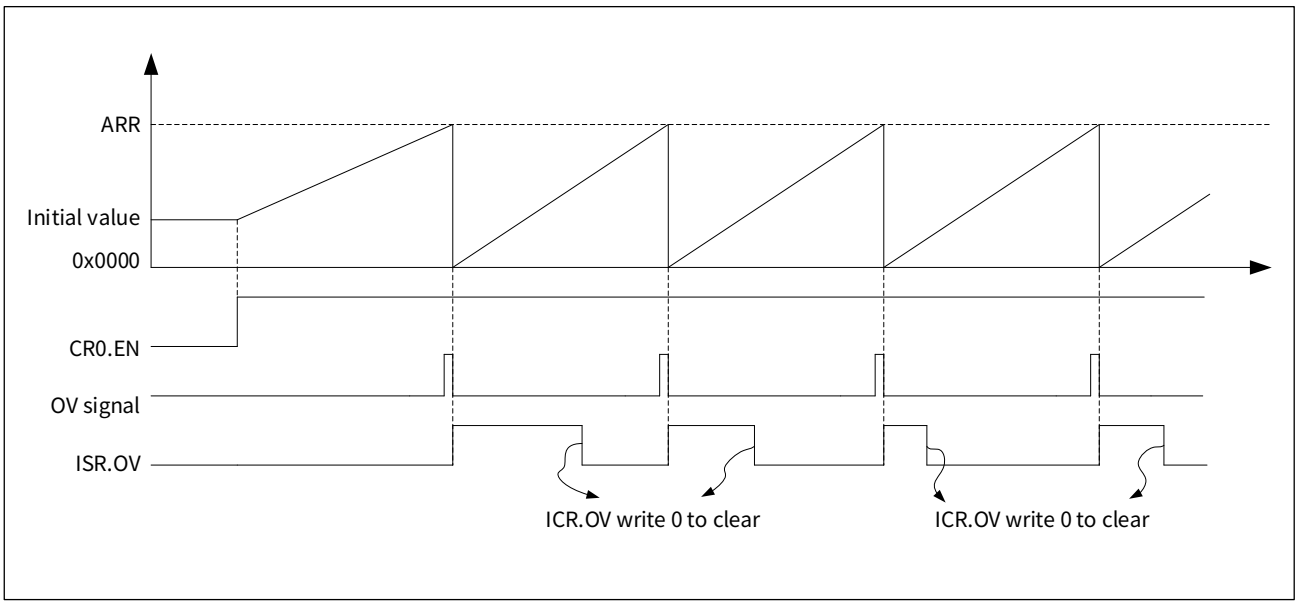
Set the MODE bit field of the control register GTIM_CR0 to 0x00 to make GTIM work in timer mode. In this mode, the counter clock source is the internal system clock PCLK, which is divided by the prescaler to obtain the count clock TCLKD to drive the counter CNT to count. The function block diagram of timer mode is shown in the following figure:

Figure 12-6 Timer mode block diagram



In practical applications, the frequency of the system clock PCLK is known. By setting the prescaler coefficient PRS reasonably, the clock with a fixed duration can be counted, in conjunction with the setting of the reload value ARR and the use of the counter ARR overflow interrupt flag bit, you can accurately obtain a specific duration, so as to achieve the purpose of timing. The following figure is the timing waveform of the timer in continuous counting mode:

Figure 12-7 Continuous counting, timer mode waveform




Timing time T calculation formula:

$$T = ( 2^{PRS} / PCLK ) \times ( ARR + 1 )$$

Where, PCLK is the counter clock source, PRS is the prescaler coefficient, and ARR is the reload value.

Example:

When the frequency of the counter clock source PCLK is 24MHz, the timing is required to be 10ms.

If the prescaler coefficient PRS is set to 0x08,

calculate:

$$T = 10ms = ( 2^{8} / 24MHz ) \times ( ARR + 1 )$$

then

$$ARR = 936.5$$

That is, the reload value ARR can be set to 937(0x3A9).

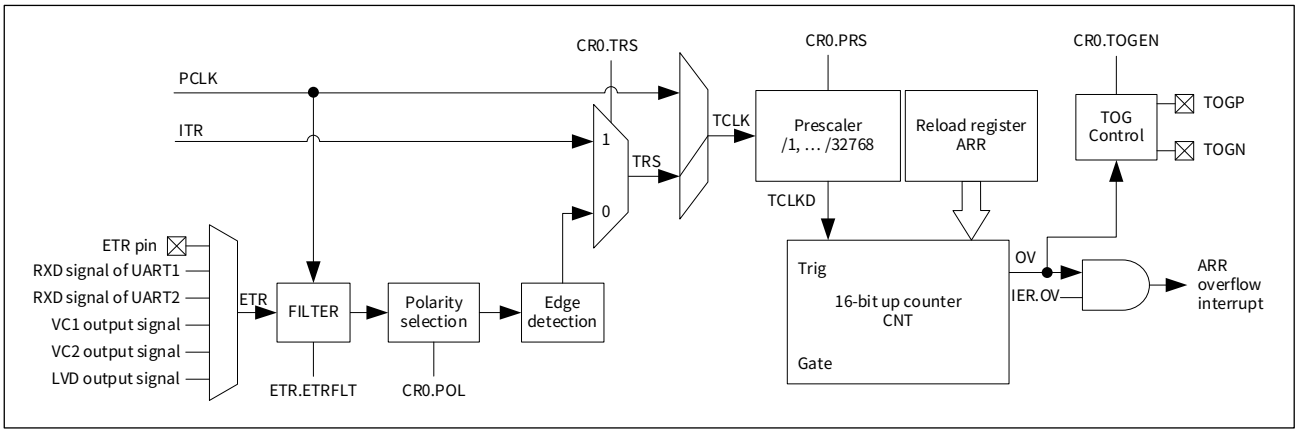
### 12.3.2.2 Counter mode

Set the MODE bit field of the control register GTIM_CR0 to 0x01 to make GTIM work in counter mode. In this mode, the counter clock source is the TRS signal, and the source of the TRS can be selected as the internal ITR signal or the ETR input signal through the TRS bit field of the control register GTIM_CR0.

When setting the TRS bit field of the control register GTIM_CR0 to 0, the source of the TRS signal is the ETR input signal. By setting the POL bit field of the control register GTIM_CR0, you can choose to count on the rising or falling edge of the ETR input signal; when setting the control register When the TRS bit field of the register GTIM_CR0 is 1, the source of the TRS signal is the internal ITR signal, and the ITR signal is the output signal of other timers (refer to *Table 12-8 ITR source configuration*).

The functional block diagram of counter mode is shown in the following figure:

Figure 12-8 Counter mode block diagram



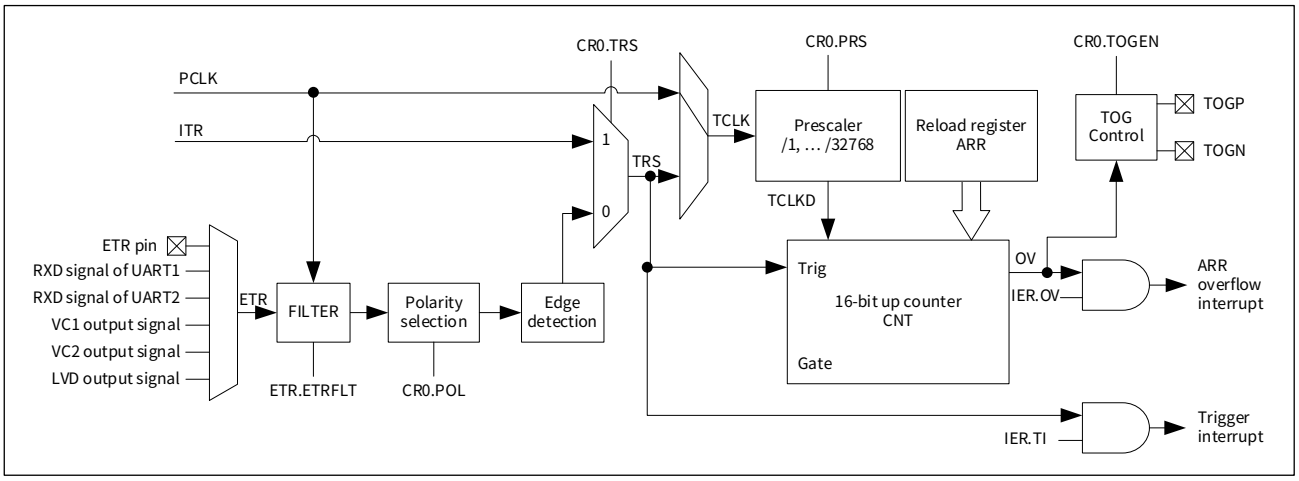

When the EN bit field of the control register GTIM_CR0 is set to 1, the counter accumulates the counted TCLKD signal after frequency division, overflows after counting to the ARR value, and starts counting from 0 again. For the specific register configuration process in counter mode, please refer to section *12.5 Programming examples*.

#### 12.3.2.3 Trigger start mode

Set the MODE bit field of the control register GTIM_CR0 to 0x02 to make GTIM work in trigger start mode. In this mode, the counting clock of the counter is the signal TCLKD after the system clock PCLK is divided by the prescaler. When GTIM_CR0.EN is set to 1 or the trigger signal is valid, the counter will start to count.

The source of the trigger signal TRS can be the internal ITR signal or the ETR input signal, which is selected by the TRS bit field of the control register GTIM_CR0. When the TRS bit field of the control register GTIM_CR0 is set to 0, the source of the TRS signal is the ETR input signal. By setting the POL bit field of the control register GTIM_CR0, the ETR input signal can be selected to be valid on the rising edge or falling edge; when setting the control register GTIM_CR0 When the TRS bit field is 1, the source of the TRS signal is the internal ITR signal, and the ITR signal is the output signal of other timers (refer to *Table 12-8 ITR source configuration*). The function block diagram of trigger start mode is shown in the following figure:

Figure 12-9 Trigger start mode block diagram

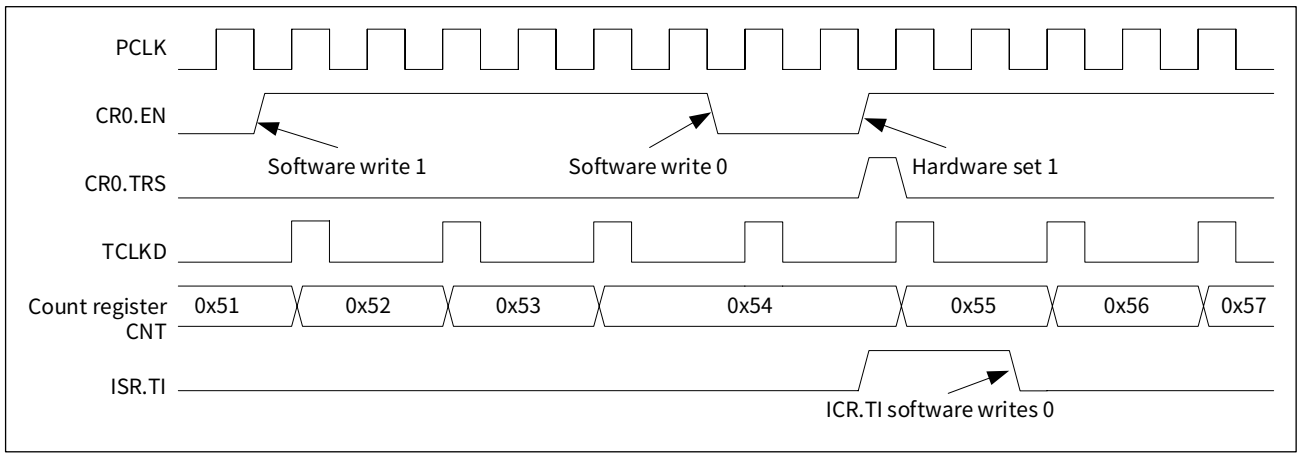



When a valid trigger signal is detected, the following effects occur:

1. GTIM_CR0.EN is set by hardware;
2. Trigger flag bit GTIM_ISR.TI is set by hardware;
3. The counter starts and starts counting.

After the counter is started, the counter starts to count up from the initial value. When the software sets GTIM_CR0.EN to 0, the counter suspends counting, and sets GTIM_CR0.EN to 1 again or when the trigger signal TRS is a valid transition, the counter resumes counting and generates a Trigger sign. The following figure shows an example of the trigger start mode timing diagram:

Figure 12-10 Timing diagram of trigger start mode (frequency division ratio is 2, PRS=0x01)



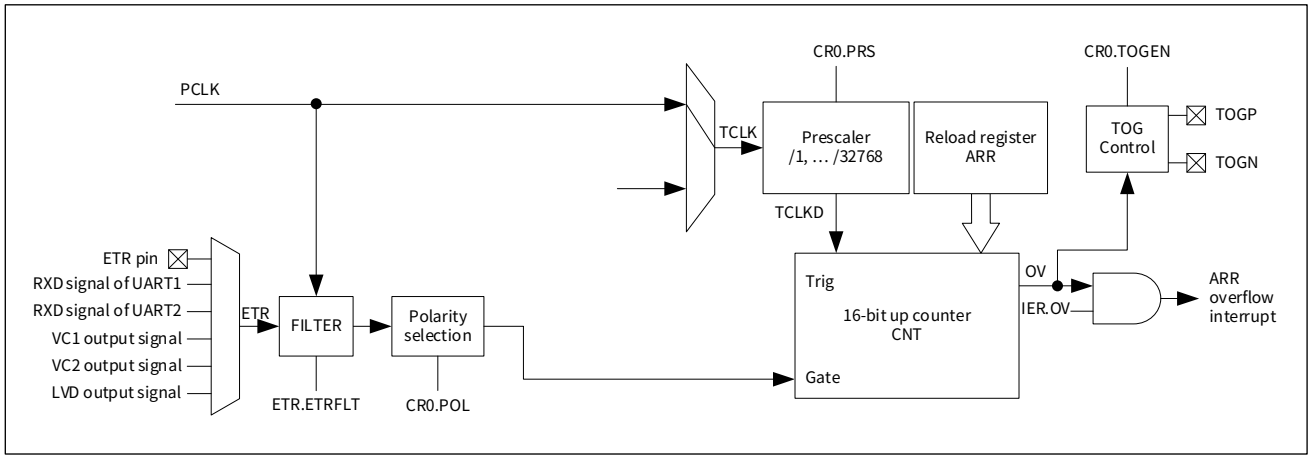
### 12.3.2.4 Gated mode

Set the MODE bit field of the control register GTIM_CR0 to 0x03 to make GTIM work in gated mode. In this mode, the counting clock of the counter is the signal TCLKD after the system clock PCLK is divided by the prescaler. When GTIM_CR0.EN is set to 1 and the gate control signal is valid, the counter will start to count.

The source of the gate control signal is the ETR input signal. The source of the ETR signal can be the external GTIM_ETR pin or other on-chip peripherals, which can be selected through the general-purpose timer ETR source configuration register SYSCTRL_GTIMETR (see section *12.3.7 On-chip peripheral interconnect ETR)*. By setting the POL bit field of the control register GTIM_CR0, it is possible to select whether the polarity of the ETR signal is reversed, so as to select whether the effective polarity of the gate control signal is a high level or a low level.

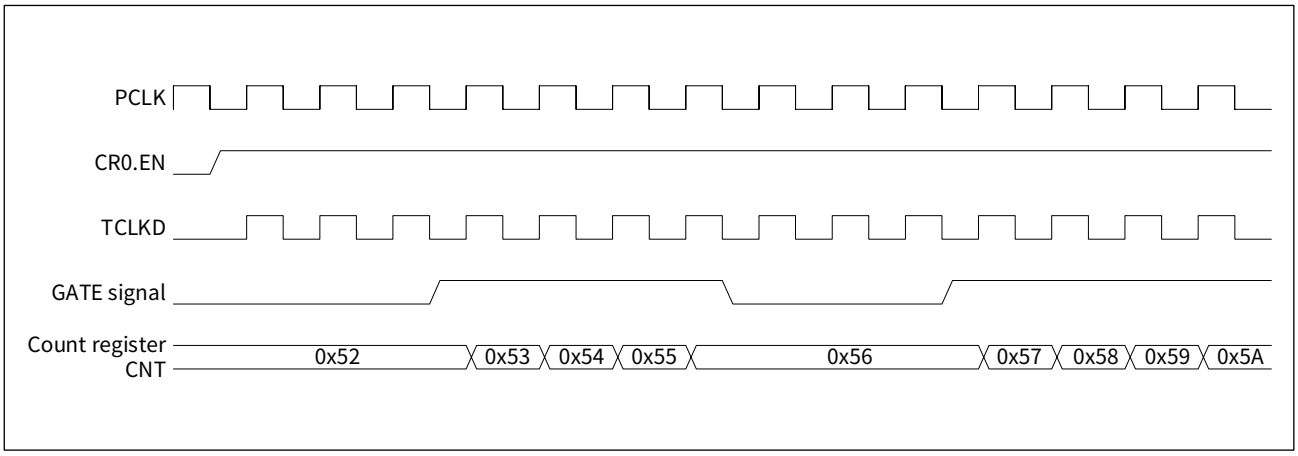
The function block diagram of gated mode is shown in the following figure:

Figure 12-11 Gated mode block diagram




The following figure is used to illustrate that in the gated mode, when the gated level is an invalid level, even if GTIM_CR0.EN is set to 1, the counter will not count, only when the gated level is a valid level, the counter will count.Once the gated level is invalid, the counting will be suspended immediately.

Figure 12-12 Timing diagram of gated mode
(frequency division ratio is 1, PRS=0x00, gated is active high, POL=0x00)

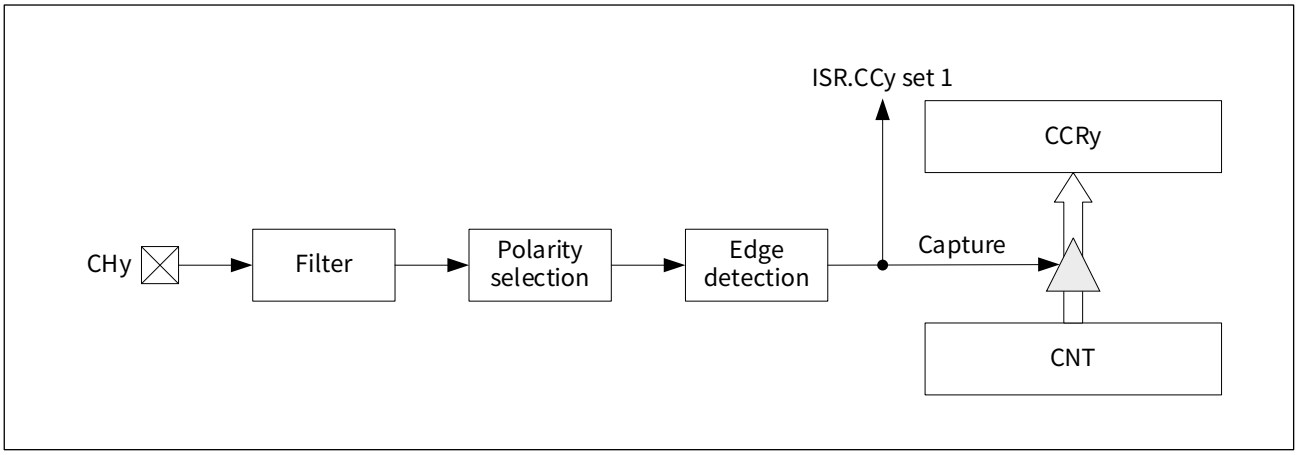


### 12.3.3 Input capture function

#### 12.3.3.1 Input capture

When the signal on the capture compare channel CHy transitions (rising edge or falling edge), the hardware automatically stores the value of the current count register GTIM_CNT in the compare capture register GTIM_CCRy of the corresponding channel to complete a capture. The input capture function can be used to measure pulse width or frequency. Its functional block diagram is shown in the following figure:

Figure 12-13 Input capture mode block diagram




The conditions for triggering capture on each channel are determined by the compare capture control register GTIM_CMMR, as shown in the following table:

Table 12-4 Input capture mode configuration

| General-purpose timer | Channel CHy | CCyM bit field value | Capture mode configuration |
|---|---|---|---|
| GTIM_CMMR | CCyM (y=1,2,3,4) | 0001 | CHy rising edge capture |
| | | 0010 | CHy falling edge capture |
| | | 0011 | The rising and falling edges of CHy are captured at the same time |

When a capture occurs, the channel CHy compare capture interrupt flag GTIM_ISR.CCy is set by hardware. If the interrupt is enabled (set the interrupt enable register GTIM_IER.CCy to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, the interrupt flag clear register GTIM_ICR.CCy should be set to 0 to clear this flag.

#### 12.3.3.2 Input capture sources

The input capture source of GTIM can be external GTIM_CHy pin or other on-chip peripherals, which can be configured through the general-purpose timer input capture source configuration register SYSCTRL_GTIMCAP.

When SYSCTRL_GTIMCAP.CHy is 0x00, the external input port to which the capture signal is input is configured by the GPIO multiplexing function registers GPIOx_AFRL.

When SYSCTRL_GTIMCAP.CHy is 0x01~0x06, the input capture signal comes from other on-chip peripherals, as shown in the following table:

Table 12-5 General-purpose timer input capture sources

| SYSCTRL_GTIMCAP bit field | Value | Input capture source of GTIM |
|---|---|---|
| CHy | 000 | Configured by GPIOx_AFRL |
| | 001 | RXD signal of UART1 |
| | 010 | RXD signal of UART2 |
| | 100 | Compare output signal of VC1 |
| | 101 | Compare output signal of VC2 |
| | 110 | LVD output signal |

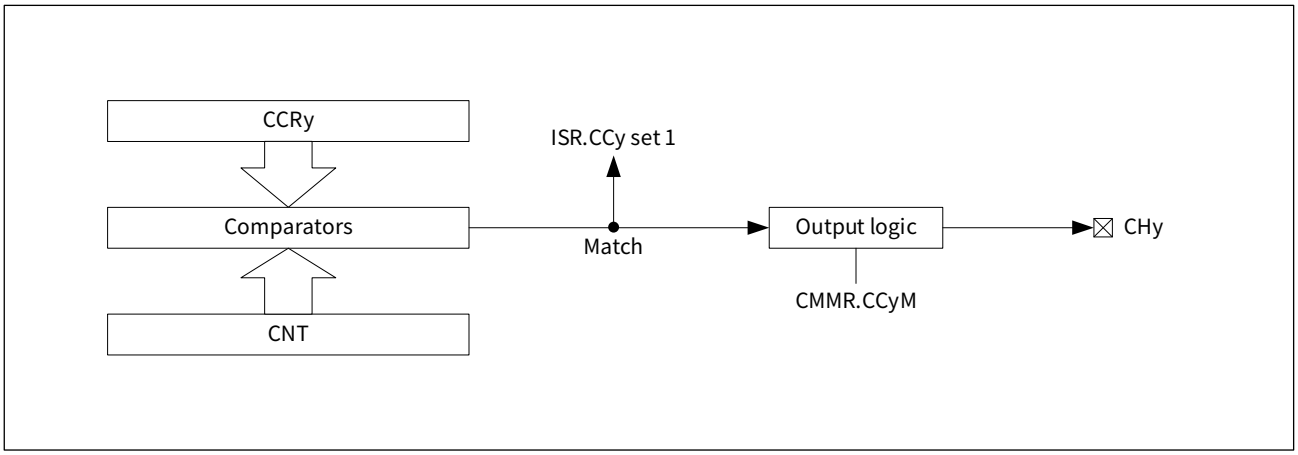
Under this configuration, the interconnection of external input can be realized inside the chip. For example, the RXD signal of the UART is used as the input capture source, and the automatic detection of the UART baud rate can be realized.

### 12.3.4 Output compare function

#### 12.3.4.1 Output compare

In the output compare mode, the value of the current count register GTIM_CNT is compared with the value of the compare capture register GTIM_CCRy of the corresponding channel CHy. When the two match, the compare capture channel CHy can output a high level or a low level and generate a compare interrupt at the same time. The output compare function can be used to control an output waveform or indicate that a given time has arrived. Its functional block diagram is shown in the following figure:

Figure 12-14 Output compare mode block diagram




The output action of the CHy pin depends on the CCyM bit field of the compare capture control register GTIM_CMMR, as shown in the following table:

Table 12-6 Output compare mode configuration

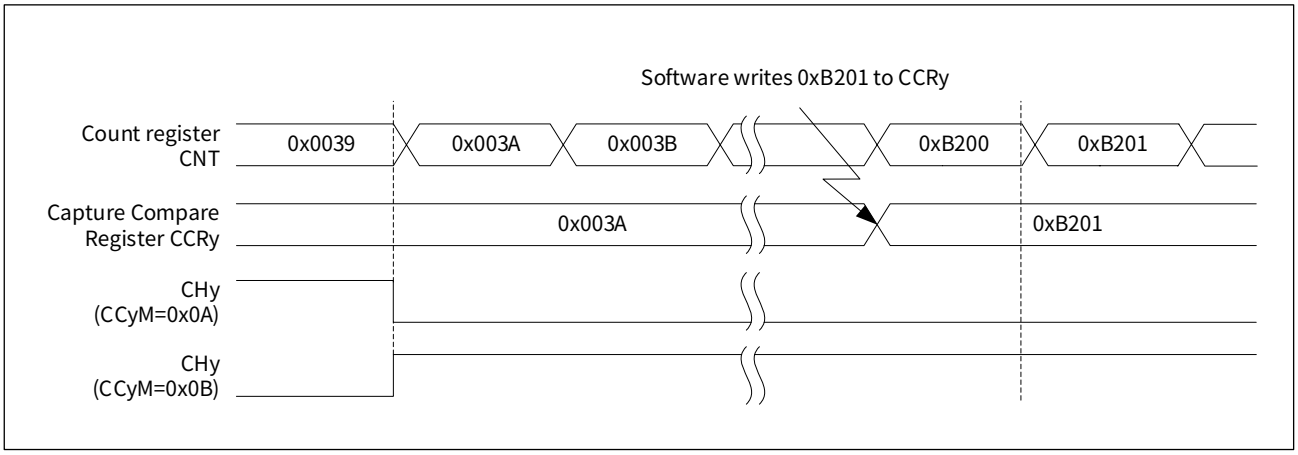
| General-purpose timer | Channel CHy | CCyM bit field value | Compare mode configuration |
|---|---|---|---|
| GTIM_CMMR | CCyM (y=1,2,3,4) | 1000 | Force CHy to output low level |
| | | 1001 | Force CHy to output high level |
| | | 1010 | CHy is set to 0 on compare match |
| | | 1011 | CHy is set to 1 on compare match |
| | | 1110 | PWM positive output (CNT >= CCR output high) |
| | | 1111 | PWM reverse output (CNT < CCR output high level) |

When the value of the CCyM bit of the GTIM_CMMR register is 0x0A, when a compare match occurs, the CHy pin outputs a low level, and the CCy bit of the GTIM_ISR register is set to 1.

When the value of the CCyM bit in the GTIM_CMMR register is 0x0B, when a compare match occurs, the CHy pin outputs a high level, and the CCy bit in the GTIM_ISR register is set to 1.

The following figure shows two compare output modes: set low level and set high level, where CCRy is preset to 0x003A.

Figure 12-15 Two compare output modes




### 12.3.4.2 Force output function

In forced output mode, the output compare signal can be forced to a high or low state directly by software, without depending on the compare result of the compare capture register GTIM_CCRy and the count register GTIM_CNT.

Set the corresponding CCyM bit field in the GTIM_CMMR register to 0x8 to force the CHy channel to output a low level. Set the corresponding CCyM bit field in the GTIM_CMMR register to 0x9 to force the CHy channel to output a high level.

In this mode, the comparison between the GTIM_CCRy register and the counter GTIM_CNT is still in progress, the corresponding flags are also modified, and corresponding interrupt is also generated.

#### 12.3.4.3 PWM output function

Pulse width modulation (PWM) mode can generate a signal whose frequency is determined by the reload register GTIM_ARR and the duty cycle is determined by the compare capture register GTIM_CCRy.

Writing 0xE or 0xF to the CCyM bits in the GTIM_CCMR register can independently set each CHy output channel to generate a PWM.

Set GTIM_CMMR.CCyM to 0xE, when GTIM_CNT>=GTIM_CCRy, the CHy channel outputs a high level, otherwise it outputs a low level.

*Caution:*

*If the compare value in GTIM_CCRy is greater than the value of the reload register GTIM_ARR, the CHy channel output remains low; if the compare value in GTIM_CCRy is 0, the CHy channel output remains high.*

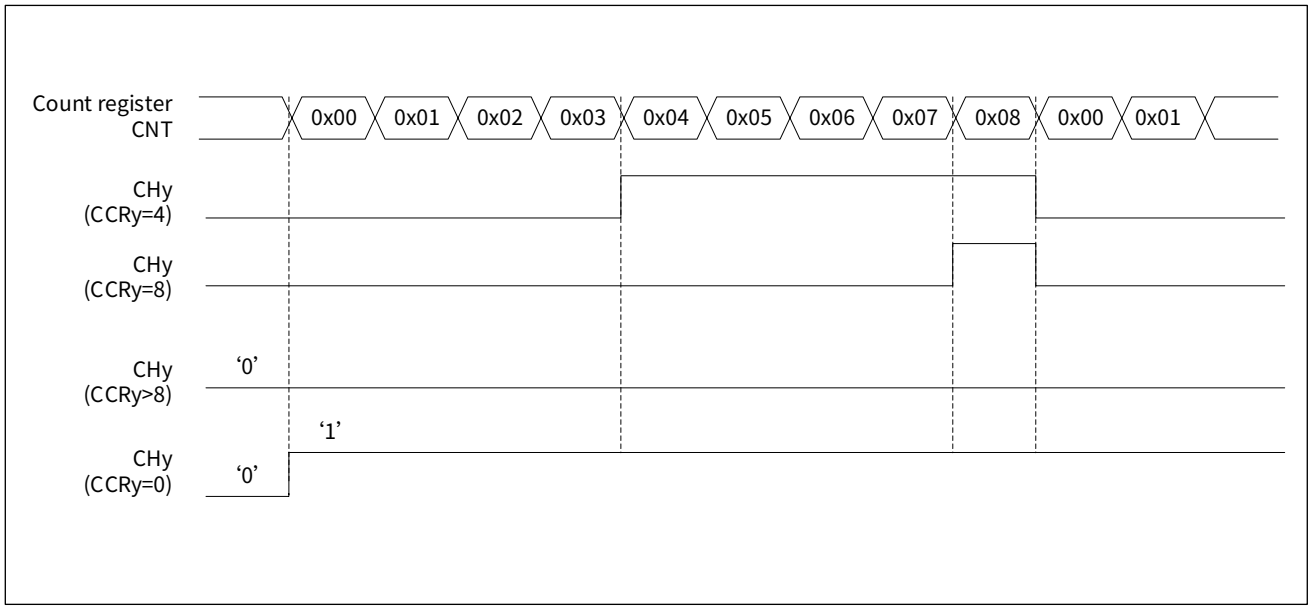
Set GTIM_CMMR.CCyM to 0xF, when GTIM_CNT<GTIM_CCRy, the CHy channel outputs a high level, otherwise it outputs a low level.

*Caution:*

*If the compare value in GTIM_CCRy is greater than the value of the reload register GTIM_ARR, the CHy channel output remains high; if the compare value in GTIM_CCRy is 0, the CHy channel output remains low.*

The following figure is an example of PWM waveform when GTIM_CMMR.CCyM is 0xE and GTIM_ARR is 0x08:

Figure 12-16 PWM waveform example



### 12.3.5 Encoding counting mode

The capture/compare channel CH1 and CH2 pins of GTIM can be connected with the encoder as the interface of the quadrature encoder. When the ENCMODE bit field of the control register GTIM_CR0 is set to a value other than 0x00, the GTIM works in the quadrature encoding mode, and the count clock of the counter is input by the CH1 and CH2 pins.

GTIM supports 3 encoding counting modes: set GTIM_CR0.ENCMODE to 0x01, the counter will only count on the edge of CH1; set GTIM_CR0.ENCMODE to 0x02, the counter will only count on the edge of CH2; set GTIM_CR0.ENCMODE to 0x03, the counter will count at the edges of ch1 and CH2 at the same time.

Through the CH1POL and CH2POL bit fields of the control register GTIM_CR1, you can select the polarity of the channel CH1 and CH2 input signals sent to the internal circuit. The input filters of channels CH1 and CH2 can be configured through the CH1FLT and CH2FLT bit fields of the control register GTIM_CR1.

Setting GTIM_CR0.EN to 1 starts the counter, which will be driven by each valid transition of the filtered and polarity controlled input signals of the channel CH1 and CH2 pins. According to the transition sequence of the two input signals, a count pulse and a direction signal are generated. The encoder's current counting direction flag GTIM_ISR.DIR is automatically set by hardware. When the counting direction is changed, the encoder counting direction change interrupt flag bit GTIM_ISR.DIRCHANGE will be set by hardware. If the interrupt is enabled (set GTIM_IER.DIRCHANGE to 1), the CPU will respond to the interrupt service routine. Before exiting the interrupt service routine, GTIM_ICR.DIRCHANGE should be set to 0 to clear this flag.

The relationship between counting direction and encoder signal is shown in the following table:

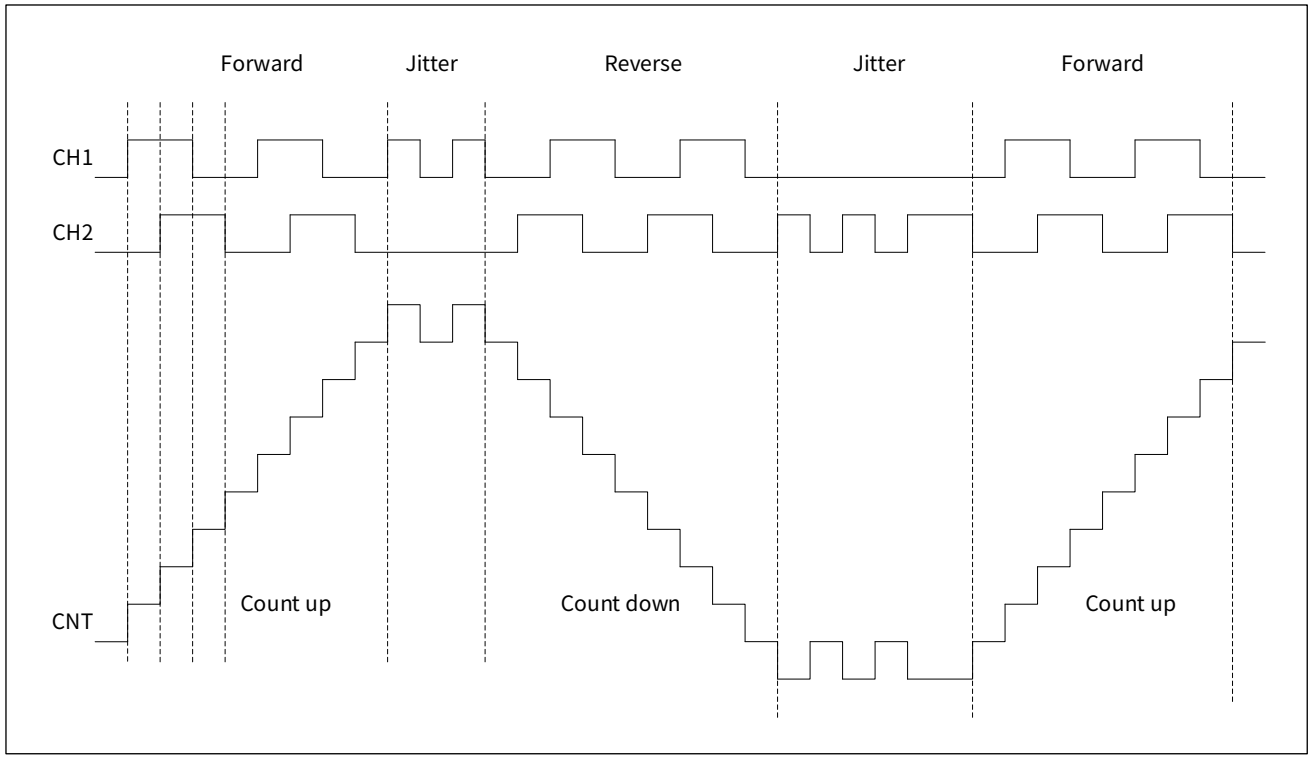
Table 12-7 Relationship between counting direction and encoder signal

| | Signal level | | CH1 | | CH2 | |
|---|---|---|---|---|---|---|
| | CH2 | CH1 | Rising | Falling | Rising | Falling |
| Mode 1 | High | - | Downcounting | Upcounting | No counting | No counting |
| | Low | - | Upcounting | Downcounting | No counting | No counting |
| Mode 2 | - | High | No counting | No counting | Upcounting | Downcounting |
| | - | Low | No counting | No counting | Downcounting | Upcounting |
| Mode 3 | High | High | Downcounting | Upcounting | Upcounting | Downcounting |
| | | Low | Upcounting | Downcounting | Downcounting | Upcounting |

The third signal output by the encoder represents the mechanical zero point, which can be connected to the CH3 channel. Through the ENCRELOAD bit field of the control register GTIM_CR0, you can choose to reload the value of the counter CNT on the rising or falling edge. You can also choose to reset the value of the counter CNT on the rising or falling edge through the ENCRESET bit field of the control register GTIM_CR0 (the reset operation makes the value of the CNT register is 0x0000).

The following figure is an example of the operation of the encoding counting mode, showing the generation and direction control of the count signal. It also shows how input jitter is suppressed when dual edge is selected; jitter can occur when the sensor is positioned close to a transition point.

Figure 12-17 Example of counter operation in encoder mode




### 12.3.6 Internal cascade ITR

The timer can be cascaded through the ITR signal, thereby expanding the counting range of the timer, and it can also divide the count clock source PCLK by any number between 1 and 65535.

The source of the cascaded input ITR of GTIM is BTIM, the overflow signal of GTIM and the main mode output signal of ATIM, which can be selected by the timer ITR source configuration register SYSCTRL_TIMITR. The specific configuration is shown in the following table:

Table 12-8 ITR source configuration

| SYSCTRL_TIMITR bit field | Value | ITR source |
|---|---|---|
| BTIM3ITR<br>BTIM2ITR<br>BTIM1ITR<br>GTIMITR<br>ATIMITR | 000 | Overflow signal of BTIM1 |
| | 001 | Overflow signal of BTIM2 |
| | 010 | Overflow signal of BTIM3 |
| | 011 | Overflow signal of GTIM |
| | 111 | Master mode output signal of ATIM |

*Caution:*

*The timer's ITR source cannot select its own overflow signal OV.*

### 12.3.7 On-chip peripheral interconnect ETR

The source of the ETR signal of GTIM can be the external GTIM_ETR pin or other on-chip peripherals, which can be selected through the general-purpose timer ETR source configuration register SYSCTRL_GTIMETR.

When SYSCTRL_GTIMETR.GTIMETR is 0x00, the external input port of the ETR signal is configured by the GPIO multiplexing function register GPIO_AFRL, as shown in the following table:

Table 12-9 External GTIM_ETR pin multiplexing

| ETR | Pin | AFR |
|---|---|---|
| GTIM_ETR | PA02 | 0x04 |
| | PA03 | 0x05 |
| | PA04 | 0x07 |
| | PB04 | 0x03 |
| | PC01 | 0x02 |

When SYSCTRL_GTIMETR.GTIMETR is 0x01~0x06, the ETR signal comes from other on-chip peripherals to realize the interconnection between on-chip peripherals, as shown in the following table:

Table 12-10 ETR sources for GTIM

| SYSCTRL_GTIMETR.GTIMETR | ETR source for GTIM |
|---|---|
| 000 | Configured by GPIOx_AFRL |
| 001 | RXD signal of UART1 |
| 010 | RXD signal of UART2 |
| 011 | RXD signal of UART3 |
| 100 | Compare output signal of VC1 |
| 101 | Compare output signal of VC2 |
| 110 | LVD output signal |

## 12.4 Debugging support

GTIM supports stop or continue counting in debug mode, which is set by the GTIM bit field of the debug status timer control register SYSCTRL_DEBUG.

Set SYSCTRL_DEBUG.GTIM to 1, then suspend the counter count of GTIM in the debug state;

Set SYSCTRL_DEBUG.GTIM to 0, then the counter of GTIM continues to count in the debug state.

## 12.5 Programming examples

### 12.5.1 Timer mode programming example

The following is a programming example of timer mode. In this programming example, set the frequency of the system clock PCLK to 24MHz. After starting the timer for 10ms (9.99ms), an ARR overflow interrupt request will be generated and the counting will stop.

The configuration process is as follows:

Step 1: Set the SYSCTRL_APBEN1.GTIM bit or the SYSCTRL_APBEN2.GTIM bit to 1, and turn on the configuration clock and working clock of GTIM;

Step 2: Set GTIM_CR0.MODE to 0x00 to make GTIM work in timer mode;

Step 3: Set GTIM_CR0.ONESHOT to 0x01 to make the counter work in single counting mode;

Step 4: Set GTIM_CR0.PRS to 0x08 and the frequency division ratio to 256, then one cycle of the count clock TCLKD is about 10.67μs;

Step 5: Set GTIM_ARR to 0x03A9 (937), then the ARR overflow period is about 10.008ms (ARR+1);

Step 6: Set GTIM_CNT to 0x0000 for counting;

Step 7: Set GTIM_IER.OV to 1 to allow ARR overflow interrupt;

Step 8: Set GTIM_CR0.EN to 1 and start GTIM;

Step 9: ARR overflow interrupt is generated, enter the interrupt service routine, set GTIM_ICR.OV to 0 to clear the interrupt flag.

### 12.5.2 Counter mode programming examples

#### 12.5.2.1 Count external ETR signals

The steps to measure the number of rising edges of the signal input from the external GTIM_ETR pin are as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.GTIM to 1, turn on the GPIO clock corresponding to the GTIM_ETR pin and the GTIM configuration clock and working clock;

Step 2: Configure the GPIO corresponding to the GTIM_ETR pin to the multiplexed input mode. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set GTIM_CR0.MODE to 0x01 to make GTIM work in counter mode;

Step 4: Set GTIM_CR0.TRS to 0x0, and select the counting source as the ETR input signal;

Step 5: Configure GTIM_ETR.ETRFLT, choose whether to filter or filter bandwidth;

Step 6: Set GTIM_CR0.POL to 0x0, and select the rising edge to count as valid;

Step 7: Set GTIM_CR0.PRS to 0x00. In this example, the input ETR signal is not divided;

Step 8: Set GTIM_ARR to 0xFFFF, and set the ARR overflow period to the maximum;

Step 9: Set GTIM_CNT to 0x0000 for counting;

Step 10: Set GTIM_CR0.EN to 0x1 to start GTIM.

#### 12.5.2.2 Count internal ITR signal

In the following example, BTIM1 is used as the frequency divider of GTIM to divide the PCLK clock by 13. The operation steps are as follows:

Step 1: Set the SYSCTRL_APBEN1.GTIM bit to 1, and turn on the configuration clock and working clock of GTIM;

Step 2: Set GTIM_CR0.MODE to 0x01 to make GTIM work in counter mode;

Step 3: Set GTIM_CR0.TRS to 0x1, and select the counting source as the internal ITR signal;

Step 4: Set SYSCTRL_TIMITR.GTIMITR to 0x00, select the ITR source of GTIM as the overflow signal of BTIM1;

Step 5: Set GTIM_CR0.PRS to 0x00, do not divide the ITR signal, and complete the frequency division through BTIM1;

Step 6: Set GTIM_ARR to 0xFFFF, and set the ARR overflow period to the maximum;

Step 7: Set the initial value of GTIM_CNT to 0x0000 for counting;

Step 8: Set GTIM_CR0.EN to 0x1 and start the timer;

Step 9: Set the SYSCTRL_APBEN2.BTIM bit to 1, and turn on the configuration clock and working clock of BTIM1;

Step 10: Set BTIM1_BCR.MODE to 0x00, select BTIM1 to work in timer mode, and use PCLK clock to count;

Step 11: Set BTIM1_BCR.PRS to 0x00, do not divide the frequency of the PCLK clock signal;

Step 12: Set BTIM1_BCR. ONESHOT to 0x00, BTIM1 is continuous counting mode;

Step 13: Set the BTIM1_ARR register to 0xC, so that BTIM1 will generate an overflow signal OV after counting (ARR+1) PCLK cycles, that is, 13 PCLK cycles. This OV signal is used as the input signal ITR of GTIM, causing 1 of GTIM count;

Step 14: Set the initial value of BTIM1_CNT to 0x0000 for counting;

Step 15: Set BTIM1_BCR.EN to 0x1 to start BTIM1.

### 12.5.3 Trigger start mode programming example

The counter is triggered to start counting by the signal input from the external GTIM_ETR pin. The operation steps are as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.GTIM to 1, turn on the GPIO clock corresponding to the GTIM_ETR pin and the GTIM configuration clock and working clock;

Step 2: Configure the GPIO corresponding to the GTIM_ETR pin to the multiplexed input mode. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set GTIM_CR0.TRS to 0x0, and select the trigger signal as the ETR input signal;

Step 4: Configure GTIM_ETR.ETRFLT, choose whether to filter or filter bandwidth;

Step 5: Configure GTIM_CR0.POL, and set whether the trigger signal is valid on the rising or falling edge;

Step 6: Set GTIM_CR0.PRS to 0x00. In this example, the PCLK clock is not divided;

Step 7: Set GTIM_ARR to 0xFFFF, and set the ARR overflow period to the maximum;

Step 8: Set the initial value of GTIM_CNT to 0x0000 for counting;

Step 9: Set GTIM_ICR.TI to 0x0, clear interrupt flag;

Step 10: If you need to enable the trigger interrupt request, set the register GTIM_IER.TI bit to 1;

Step 11: Set GTIM_CR0.MODE to 0x02 to make GTIM work in trigger mode. When a valid trigger signal is input, the counter will start to work.

*Caution:*

*Set the polarity of the trigger signal first, and then set the timer mode, otherwise it will cause false triggering.*

### 12.5.4 Gated mode programming example

The signal input from the external GTIM_ETR pin is used as the gate control signal to control the count of the counter. The operation steps are as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.GTIM to 1, turn on the GPIO clock corresponding to the GTIM_ETR pin and the GTIM configuration clock and working clock;

Step 2: Configure the GPIO corresponding to the GTIM_ETR pin to the multiplexed input mode. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set GTIM_CR0.MODE to 0x03 to make GTIM work in gated mode;

Step 4: Set GTIM_CR0.TRS to 0x0, and select the gate control signal as the ETR input signal;

Step 5: Configure GTIM_ETR.ETRFLT, choose whether to filter or filter bandwidth;

Step 6: Configure GTIM_CR0.POL, and set whether the gate control signal is active at high level or low level;

Step 7: Set GTIM_CR0.PRS to 0x00. In this example, the PCLK clock is not divided;

Step 8: Set GTIM_ARR to 0xFFFF, and set the ARR overflow period to the maximum;

Step 9: Set the initial value of GTIM_CNT to 0x0000 for counting;

Step 10: Set GTIM_CR0.EN to 0x01 to start the timer. When the gate control signal is valid, the counter will start to work.

### 12.5.5 Input capture programming example

In timer mode, the steps to measure the width of a pulse are as follows:

Step 1: Set the SYSCTRL_APBEN1.GTIM bit to 1, and turn on the configuration clock and working clock of GTIM;

Step 2: Set GTIM_CR0.MODE to 0x00 to make GTIM work in timer mode;

Step 3: Set GTIM_CR0.PRS and select the appropriate frequency division coefficient;

Step 4: Set GTIM_ARR to an appropriate value according to the characteristics of the signal. If GTIM_ARR is set smaller, the counter clock frequency is faster, and the input pulse width is longer, GTIM_ARR may overflow. If an overflow occurs, the overflow flag needs to be cleared in time, and the pulse width time is calculated according to the number of times the overflow flag is generated;

Step 5: Configure GTIM_CR1.CHyPOL and select the polarity of the signal sent to the internal circuit by channel CHy;

Step 6: Configure GTIM_CR1.CHyFLT, configure channel CHy input filter;

Step 7: Set GTIM_CMMR.CCyM to 0x01, select rising edge capture;

Step 8: Set GTIM_IER.CCy to 0x01, allowing capture;

Step 9: Set GTIM_CR0.EN to 0x1 and start the timer;

Step 10: Wait for the capture interrupt to be generated, and read the current value in GTIM_CCRy. Then set the corresponding bit of GTIM_ICR.CCy to 0x0, clear the interrupt flag;

Step 11: Set GTIM_CMMR.CCyM to 0x02 and select falling edge capture;

Step 12: Wait for the next capture interrupt to be generated, and read the current value in GTIM_CCRy. The difference between the GTIM_CCRy values read twice in this way is the length of the input pulse width.

### 12.5.6 Output compare programming example

In timer mode, the external output of low level through channel CH1 indicates that the timing time is up. The operation steps are as follows:

Step 1: Set the SYSCTRL_APBEN1.GTIM bit to 1, and turn on the configuration clock and working clock of GTIM;

Step 2: Set GTIM_CR0.MODE to 0x00 to make GTIM work in timer mode;

Step 3: Set GTIM_CR0.PRS and select the appropriate frequency division coefficient;

Step 4: Set GTIM_CMMR.CC1M to 0x0A, so that the CH1 channel of GTIM is set to 0 when it is compared and matched;

Step 5: Set GTIM_CNT to 0x0000, clear the counter;

Step 6: Set GTIM_CCR1, set timing time;

Step 7: Set GTIM_CR0.EN to 0x1 to start the timer. When the count value in GTIM_CNT is equal to the value of GTIM_CCR1, the output level on CH1 changes from high to low.

### 12.5.7 PWM output programming example

In timer mode, the operation steps to output a PWM waveform are as follows:

Step 1: Set the SYSCTRL_APBEN1.GTIM bit to 1, and turn on the configuration clock and working clock of GTIM;

Step 2: Set GTIM_CR0.MODE to 0x00 to make GTIM work in timer mode;

Step 3: Set GTIM_CR0.PRS and select the appropriate frequency division coefficient;

Step 4: According to the period requirement of the output PWM waveform, set GTIM_ARR to an appropriate value, that is, GTIM_ARR can change the period of PWM;

Step 5: According to the pulse width requirement of the output PWM waveform, set GTIM_CCRy to an appropriate value, that is, GTIM_CCRy can change the duty cycle of PWM;

Step 6: Set GTIM_CMMR.CCyM to 0x0E, select PWM forward output, that is GTIM_CNT>=GTIM_CCRy output high level;

Step 7: Set GTIM_IER.CCy to 0x01, enable compare interrupt;

Step 8: Set GTIM_CR0.EN to 0x1 and start the timer;

Step 9: Wait for the compare interrupt to be generated, modify the value of GTIM_CCRy as required to change the duty cycle of the PWM waveform, and then set the corresponding bit of GTIM_ICR.CCy to 0x0 to clear the interrupt flag.

### 12.5.8 Encoder mode programming example

The steps to configure GTIM to work in encoder mode 3 are as follows:

Step 1: Set the SYSCTRL_APBEN1.GTIM bit to 1, and turn on the configuration clock and working clock of GTIM;

Step 2: Set the register GTIM_CR0.ENCMODE to 0x03, so that GTIM works in encoding mode 3, and the counter counts on the edges of CH1 and CH2 at the same time;

Step 3: Set the register GTIM_ARR and select the appropriate value;

Step 4: Set the CH1POL and CH2POL bits in the GTIM_CR1 register to select the polarity of the CH1 and CH2 pin input signals sent to the internal circuit;

Step 5: Set the CH1FLT and CH2FLT bits in the GTIM_CR1 register to select the appropriate filter configuration;

Step 6: If you need to set the DIRCHANGE bit in the register GTIM_IER to enable, turn on the direction change interrupt;

Step 7: Set GTIM_CR0.EN to 0x1 to start the timer.

## 12.6 List of registers

GTIM base address: GTIM_BASE = 0x4000 0400

Table 12-11 List of GTIM registers

| Register name | Register address | Register description |
|---|---|---|
| GTIM_ARR | GTIM_BASE + 0x300 | Reload register |
| GTIM_CNT | GTIM_BASE + 0x304 | Count register |
| GTIM_CMMR | GTIM_BASE + 0x308 | Compare capture control register |
| GTIM_ETR | GTIM_BASE + 0x30C | External trigger control register |
| GTIM_CR0 | GTIM_BASE + 0x310 | Control register 0 |
| GTIM_IER | GTIM_BASE + 0x314 | Interrupt enable register |
| GTIM_ISR | GTIM_BASE + 0x318 | Interrupt flag register |
| GTIM_ICR | GTIM_BASE + 0x31C | Interrupt flag clear register |
| GTIM_CCR1 | GTIM_BASE + 0x320 | Compare capture register 1 |
| GTIM_CCR2 | GTIM_BASE + 0x324 | Compare capture register 2 |
| GTIM_CCR3 | GTIM_BASE + 0x328 | Compare capture register 3 |
| GTIM_CCR4 | GTIM_BASE + 0x32C | Compare capture register 4 |
| GTIM_CR1 | GTIM_BASE + 0x330 | Control register 1 |

## 12.7 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 12.7.1 GTIM_CR0 control register 0

Address offset: 0x310 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:21 | RFU | - | Reserved bits, please keep the default value |
| 20:19 | ENCRELOAD | RW | Encoding mode, count register external reload condition configuration<br>00: No function<br>01: CH3 rising edge reload count register CNT<br>10: CH3 falling edge reload count register CNT<br>Caution: The reload operation makes the value of the CNT register equal to the value of the ARR register |
| 18:17 | ENCRESET | RW | Encoding mode, count register external reset condition configuration<br>00: No function<br>01: CH3 rising edge reset count register CNT<br>10: CH3 falling edge reset count register CNT<br>Caution 1: The reset operation makes the value of the CNT register equal to 0x0000;<br>Caution 2: ENCRESET is valid when ENCRESET and ENCRELOAD are configured for the same edge. |
| 16:15 | ENCMODE | RW | Encoding counting mode enable configuration<br>00: The timer function is configured by the MODE bits<br>01: Encoding counting mode 1, CH1 signal change edge counting<br>10: Encoding counting mode 2, CH2 signal change edge counting<br>11: Encoding counting mode 3, CH1/CH2 signal change edge counting |
| 14:11 | PRSSTATUS | RO | The prescale factor currently being used by the divider circuit<br>The value will be updated from PRS when the timer overflows or when EN changes from 0 to 1 |
| 10:7 | PRS | RW | Prescaler division factor configuration<br>0: DIV1 4: DIV16 8: DIV256 12: DIV4096<br>1: DIV2 5: DIV32 9: DIV512 13: DIV8192<br>2: DIV4 6: DIV64 10: DIV1024 14: DIV16384<br>3: DIV8 7: DIV128 11: DIV2048 15: DIV32768 |
| 6 | TOGEN | RW | TOG pin output enable control<br>1: TOGP, TOGN output level is 0<br>1: TOGP, TOGN output signals with opposite levels |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 5 | ONESHOT | RW | Single/continuous counting mode control<br>0: Continuous counting mode<br>1: Single counting mode |
| 4 | POL | RW | ETR input signal polarity selection<br>0: ETR is positive (the rising edge of trigger mode is valid, and the gated mode is valid at high level)<br>1: ETR is reversed (the falling edge of trigger mode is valid, and the gated mode is valid at low level) |
| 3 | TRS | RW | Trigger source selection<br>0: ETR input signal<br>1: ITR, see section *12.3.6 Internal cascade ITR* |
| 2:1 | MODE | RW | Basic timing mode configuration<br>00: Timer mode, counting clock source is PCLK<br>01: Counter mode, counting clock source is TRS signal<br>10: Trigger start mode, the counting clock source is PCLK, and the TRS signal triggers the counter to start<br>11: Gated mode, counting clock source is PCLK, ETR input signal is used as gated |
| 0 | EN | RW | Timer running control<br>0: Timer stopped<br>1: Timer running |

### 12.7.2 GTIM_CR1 control register 1

Address offset: 0x330 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15 | CH4POL | RW | CH4 channel input signal polarity configuration<br>0: The in-phase signal of the CH4 pin is sent to the internal circuit<br>1: The inverted signal of the CH4 pin is sent to the internal circuit |
| 14:12 | CH4FLT | RW | CH4 input signal filter configuration<br>See CH1FLT for functional description |
| 11 | CH3POL | RW | CH3 channel input signal polarity configuration<br>0: The in-phase signal of the CH3 pin is sent to the internal circuit<br>1: The inverted signal of the CH3 pin is sent to the internal circuit |
| 10:8 | CH3FLT | RW | CH3 input signal filter configuration<br>See CH1FLT for functional description |
| 7 | CH2POL | RW | CH2 channel input signal polarity configuration<br>0: The in-phase signal of the CH2 pin is sent to the internal circuit<br>1: The inverted signal of the CH2 pin is sent to the internal circuit |
| 6:4 | CH2FLT | RW | CH2 input signal filter configuration<br>See CH1FLT for functional description |
| 3 | CH1POL | RW | CH1 channel input signal polarity configuration<br>0: The in-phase signal of the CH1 pin is sent to the internal circuit<br>1: The inverted signal of the CH1 pin is sent to the internal circuit |
| 2:0 | CH1FLT | RW | CH1 input signal filtering configuration; sampling clock is PCLK or PCLK frequency division, the number of sampling points is N<br>000: no filtering<br>001: $F_{sample}$ = PCLK, N=2<br>010: $F_{sample}$ = PCLK, N=4<br>011: $F_{sample}$ = PCLK, N=6<br>100: $F_{sample}$ = PCLK/4, N=4<br>101: $F_{sample}$ = PCLK/4, N=6<br>110: $F_{sample}$ = PCLK/8, N=4<br>111: $F_{sample}$ = PCLK/8, N=6 |

### 12.7.3 GTIM_ETR external trigger control register

Address offset: 0x30C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6:4 | ETRFLT | RW | ETR input signal filtering configuration; the sampling clock is PCLK or PCLK frequency division, and the number of sampling points is N<br>000: no filtering<br>001: $F_{sample}$ = PCLK, N=2<br>010: $F_{sample}$ = PCLK, N=4<br>011: $F_{sample}$ = PCLK, N=6<br>100: $F_{sample}$ = PCLK/4, N=4<br>101: $F_{sample}$ = PCLK/4, N=6<br>110: $F_{sample}$ = PCLK/8, N=4<br>111: $F_{sample}$ = PCLK/8, N=6 |
| 3:0 | RFU | - | Reserved bits, please keep the default value |

### 12.7.4 GTIM_CMMR compare capture control register

Address offset: 0x308 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:12 | CC4M | RW | CH4 capture compare mode configuration<br>See CC1M for functional description |
| 11:8 | CC3M | RW | CH3 capture compare mode configuration<br>See CC1M for functional description |
| 7:4 | CC2M | RW | CH2 capture compare mode configuration<br>See CC1M for functional description |
| 3:0 | CC1M | RW | CH1 capture compare mode configuration<br>0000: No function<br>0001: Capture on rising edge<br>0010: Capture on falling edge<br>0011: Capture the rising and falling edges at the same time<br>1000: Force output low level<br>1001: Force output high level<br>1010: Set to 0 on compare match<br>1011: Set to 1 on compare match<br>1110: PWM positive output (CNT >= CCR output high level)<br>1111: PWM reverse output (CNT < CCR output high level) |

### 12.7.5 GTIM_ARR reload register

Address offset: 0x300 Reset value: 0x0000 FFFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | ARR | RW | Timer reload value<br>Caution: In encoding counting mode, ARR should be set to 0xFFFF. |

### 12.7.6 GTIM_CNT count register

Address offset: 0x304 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CNT | RW | Timer count value |

### 12.7.7 GTIM_CCR1 compare capture register 1

Address offset: 0x320 Reset value: 0x0000 FFFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR | RW | CH1 channel compare/capture value |

### 12.7.8 GTIM_CCR2 compare capture register 2

Address offset: 0x324 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR | RW | CH2 channel compare/capture value |

### 12.7.9 GTIM_CCR3 compare capture register 3

Address offset: 0x328 Reset value: 0x0000 FFFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR | RW | CH3 channel compare/capture value |

### 12.7.10 GTIM_CCR4 compare capture register 4

Address offset: 0x32C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR | RW | CH4 channel compare/capture value |

### 12.7.11 GTIM_IER interrupt enable register

Address offset: 0x314 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | DIRCHANGE | RW | Encoder counting direction change interrupt control<br>0: Disabled<br>1: Enabled |
| 8:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | CC4 | RW | CH4 capture compare interrupt enable control<br>0: Disabled<br>1: Enabled |
| 5 | CC3 | RW | CH3 capture compare interrupt enable control<br>0: Disabled<br>1: Enabled |
| 4 | CC2 | RW | CH2 capture compare interrupt enable control<br>0: Disabled<br>1: Enabled |
| 3 | CC1 | RW | CH1 capture compare interrupt enable control<br>0: Disabled<br>1: Enabled |
| 2 | UD | RW | Underflow interrupt enable control<br>0: Underflow interrupt disabled<br>1: Underflow interrupt enabled<br>Caution: Only available in encoding counting mode |
| 1 | TI | RW | Trigger interrupt enable control<br>0: Trigger interrupt disabled<br>1: Trigger interrupt enabled |
| 0 | OV | RW | Counter ARR overflow interrupt enable control<br>0: ARR overflow interrupt disabled<br>1: ARR overflow interrupt enabled |

### 12.7.12 GTIM_ISR interrupt flag register

Address offset: 0x318 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:11 | RFU | - | Reserved bits, please keep the default value |
| 10 | DIR | RO | Encoder current counting direction flag<br>0: forward counting<br>1: reverse counting |
| 9 | DIRCHANGE | RO | Encoder counting direction change interrupt flag<br>0: The counting direction has not changed<br>1: The counting direction has been changed |
| 8:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | CC4 | RO | CH4 compare capture interrupt flag<br>0: No compare capture event occurred<br>1: The compare capture event has occurred |
| 5 | CC3 | RO | CH3 compare capture interrupt flag<br>0: No compare capture event occurred<br>1: The compare capture event has occurred |
| 4 | CC2 | RO | CH2 compare capture interrupt flag<br>0: No compare capture event occurred<br>1: The compare capture event has occurred |
| 3 | CC1 | RO | CH1 compare capture interrupt flag<br>0: No compare capture event occurred<br>1: The compare capture event has occurred |
| 2 | UD | RO | Counter underflow flag<br>0: The counter has not underflowed<br>1: The counter has underflowed<br>Caution: Only available in encoding counting mode |
| 1 | TI | RO | Trigger flag<br>0: No trigger event occurred<br>1: The trigger event has occurred |
| 0 | OV | RO | Counter ARR overflow flag<br>0: The counter has not ARR overflowed<br>1: The counter has ARR overflowed<br>Caution: This flag is generated when the counter value changes from ARR to 0 |

### 12.7.13 GTIM_ICR interrupt flag clear register

Address offset: 0x31C Reset value: 0x0000 03FF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | DIRCHANGE | R1W0 | Encoder counting direction change interrupt control<br>W0: The encoder counting direction change interrupt flag cleared<br>W1: No function |
| 8:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | CC4 | R1W0 | CH4 compare capture interrupt flag cleared<br>W0: Compare capture interrupt flag cleared<br>W1: No function |
| 5 | CC3 | R1W0 | CH3 compare capture interrupt flag cleared<br>W0: Compare capture interrupt flag cleared<br>W1: No function |
| 4 | CC2 | R1W0 | CH2 Compare capture interrupt flag cleared<br>W0: Compare capture interrupt flag cleared<br>W1: No function |
| 3 | CC1 | R1W0 | CH1 compare capture interrupt flag cleared<br>W0: Compare capture interrupt flag cleared<br>W1: No function |
| 2 | UD | R1W0 | Counter underflow flag cleared<br>W0: Counter underflow flag cleared<br>W1: No function<br>Caution: Only available in encoder counting mode |
| 1 | TI | R1W0 | Trigger flag cleared<br>W0: Trigger flag cleared<br>W1: No function |
| 0 | OV | R1W0 | Counter ARR overflow flag cleared<br>W0: Counter ARR overflow flag cleared<br>W1: No function |

# 13 Advanced-control timer (ATIM)

## 13.1 Overview

The Advanced-control Timer (ATIM) consists of a 16-bit auto-reload counter and 7 compare units, driven by a programmable prescaler. ATIM supports 6 independent capture/compare channels, which can realize 6 independent PWM outputs or 3 pairs of complementary PWM outputs or capture 6 inputs. Can be used for basic timing/counting, measuring pulse width and period of input signals, generating output waveforms (PWM, single pulse, complementary PWM with dead time inserted, etc.).

## 13.2 Main features

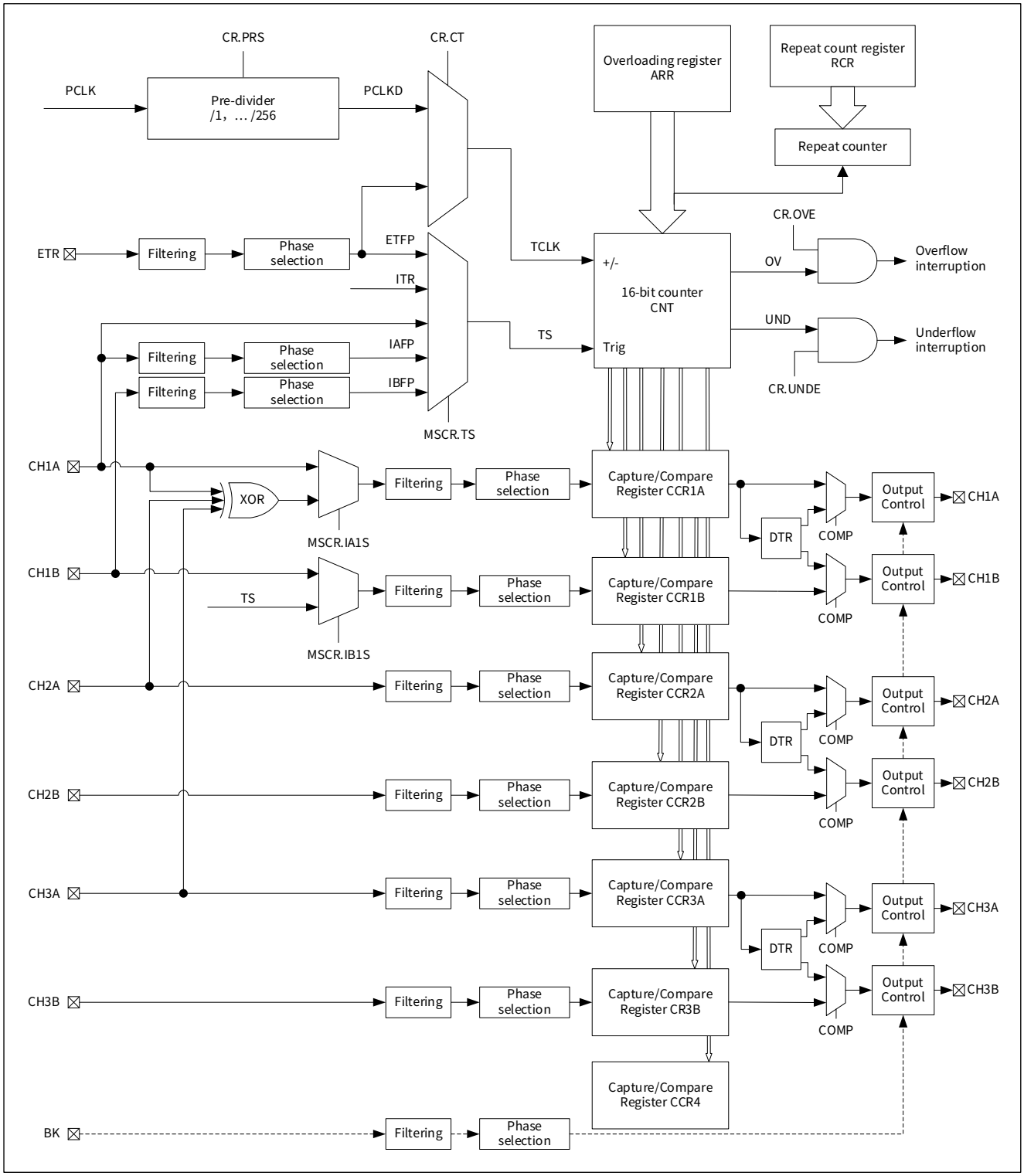
- 16-bit up, down, up/down autoload counter
- 6 independent channels for input capture and output compare
- PWM generation (Edge- and Center-aligned modes)
- Complementary PWM outputs with programmable dead-time
- Brake function
- One-pulse mode output
- Quadrature encoding counting function
- Supports for interrupt and events
  - Counter overflow/underflow
  - Capture and compare events
  - Capture data loss
  - Break event
  - Update event
  - Trigger event

## 13.3 Functional description

### 13.3.1 Functional block diagram

The functional block diagram of ATIM is shown in the following figure:

Figure 13-1 ATIM functional block diagram




Through the combination of 16-bit counter and 7 comparison unit, the functions of input capture and output comparison can be realized, which can be used to measure pulse width and period of input signal, and can output PWM waveform.

#### 13.3.1.1 Clock sources selection

The counting clock source of the counting unit can be selected from the internal system clock PCLK or the ETR input signal, which is specifically selected by the CT bit field of the control register ATIM_CR.

When ATIM_CR.CT is set to 0, the counting clock source is the internal system clock PCLK, and the internal clock PCLK can be divided by ATIM_CR.PRS.

When ATIM_CR.CT is set to 1, the counting clock source is the ETR input signal (for the specific source of the ETR signal, please refer to section *13.3.8 On-chip peripheral interconnect ETR*). The filter can be controlled by ATIM_FLTR.FLTET, and the ETR input phase can be selected by ATIM_FLTR.ETP.

#### 13.3.1.2 Update Event (UEV)

The update source of update event UEV (Update Event) is set through the URS bit field of the control register ATIM_CR, as shown in the following table:

Table 13-1 Update source settings

| ATIM_CR.URS | Update source |
|---|---|
| 0 | The counter overflow and underflow (and the number of repetition counts ATIM_RCR is 0);<br>UG is set by software; reset from slave mode |
| 1 | The counter overflow and underflow (and the number of repetition counts ATIM_RCR is 0) |

When an update event occurs (the internal UEV signal is held for one PCLK cycle), the ATIM performs the following actions:

1. The counter is reset:

   Edge-aligned mode: initialize to 0 when counting up, reload ATIM_ARR when counting down;

   Center Aligned Mode: change the counting direction.

2. If the cache register function is used, ATIM_ARR, ATIM_CHxCCRy will be updated to its cache register.
3. The prescaler counter is cleared, but does not affect the prescaler value in ATIM_CR.PRS.

   When an update event UEV occurs, the event update interrupt flag ATIM_ISR.UIF will be set by hardware. If the interrupt is enabled (set ATIM_CR.UIE to 1), an update interrupt request will be generated. Set ATIM_ICR.UIE to 0 to clear the flag bit.

*Caution:*

*If the repeatition counter is used (ATIM_RCR.RCR is not 0), the update event UEV will only be generated when the count reaches the repeat count (ATIM_RCR.RCR reaches 0).*

#### 13.3.1.3 Counting modes

The counter can be set to upcounting (edge-aligned mode), downcounting (edge-aligned mode), or up/down counting bidirectionally (center-aligned mode).

**Upcounting mode**

When the DIR bit of the control register ATIM_CR is set to 0 in edge-aligned mode, the counter works in the upcounting mode.

Setting ATIM_CR.EN to 1 enables ATIM, and the counter CNT starts to count up. When the count value reaches the reload value ARR, the overflow signal OV is generated (the overflow signal OV remains for one PCLK cycle, and then automatically cleared). At the next PCLK, the counter is initialized to 0, and the counter overflow interrupt flag bit ATIM_ISR.OVF is set by hardware. If the interrupt is enabled (set ATIM_CR.OVE to 1), an interrupt request will be generated, set ATIM_ICR.OVF to 0 to clear this flag bit.

If the repeatition counter is used, the update event (UEV) is generated when upcounting reaches the programmed number of repeat times(ATIM_RCR.RCR), else the update event is generated at each counter overflow.

The following figures show some examples of the counter behavior for different clock frequencies, where ATIM_ARR = 0x36.

Figure 13-2 Upcounting, internal clock divided by 1

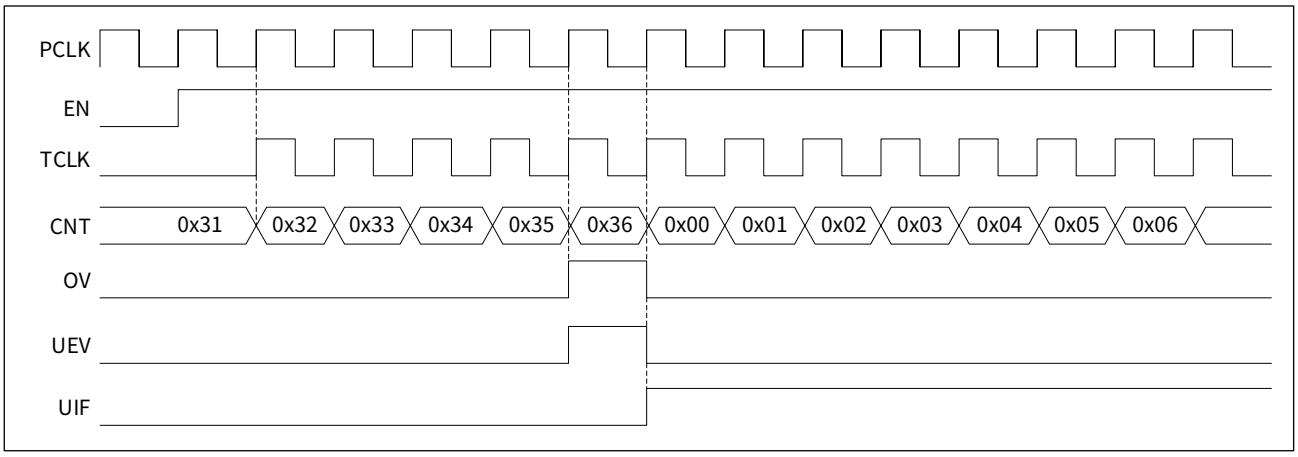



Figure 13-3 Upcounting, internal clock divided by 2

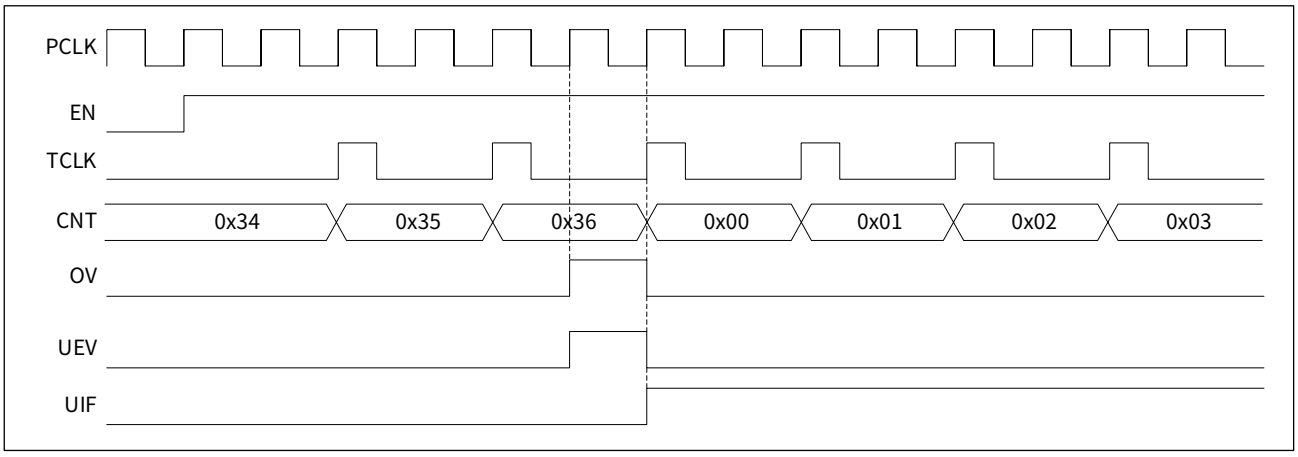


Figure 13-4 Upcounting, internal clock divided by 4

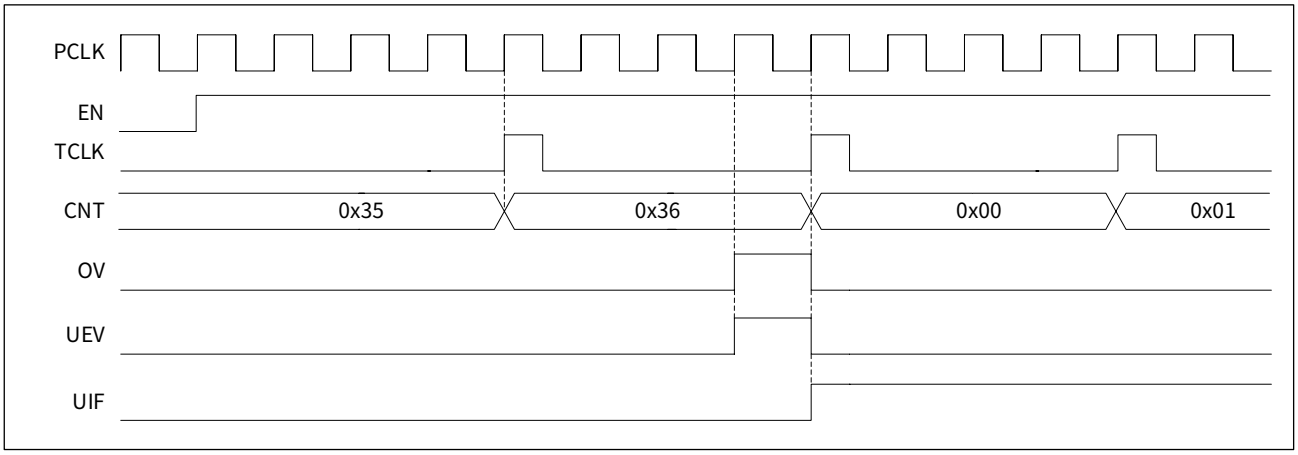



Figure 13-5 Upcounting, internal clock divided by N

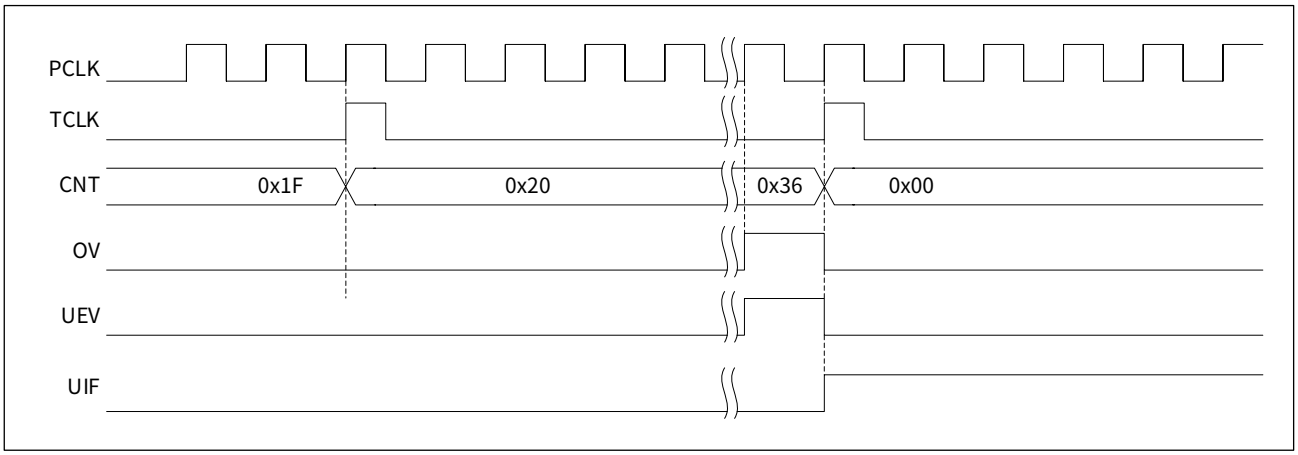



Figure 13-6 Upcounting, update event when reload buffer is disabled (ATIM_CR.BUFPEN=0x0)

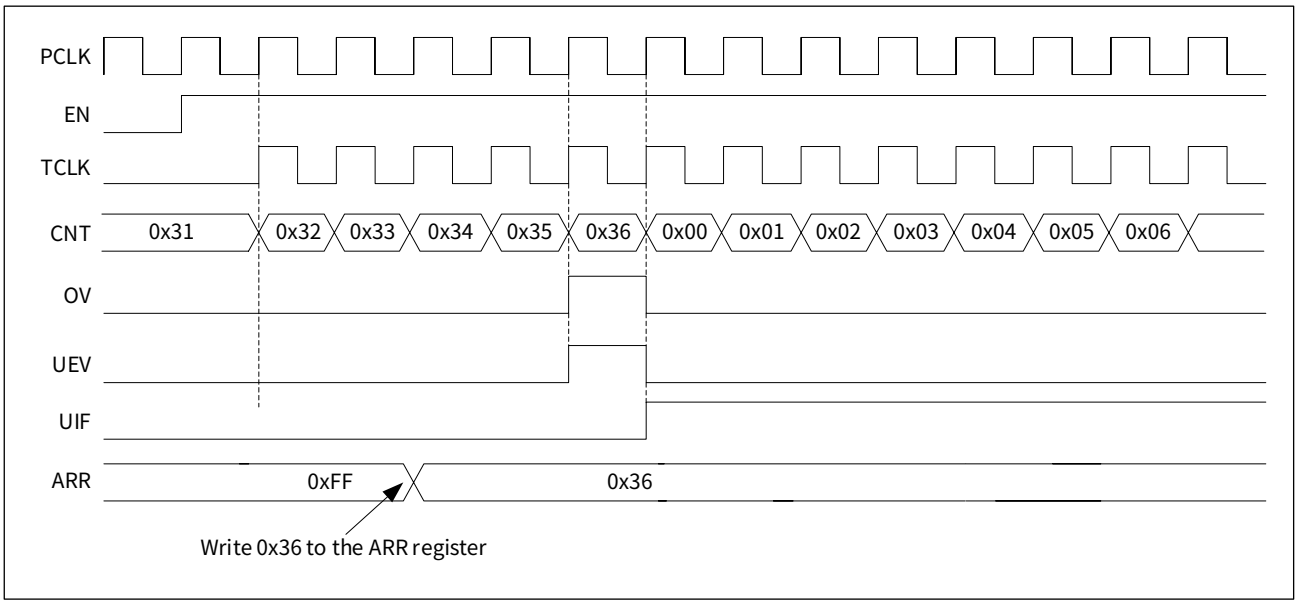
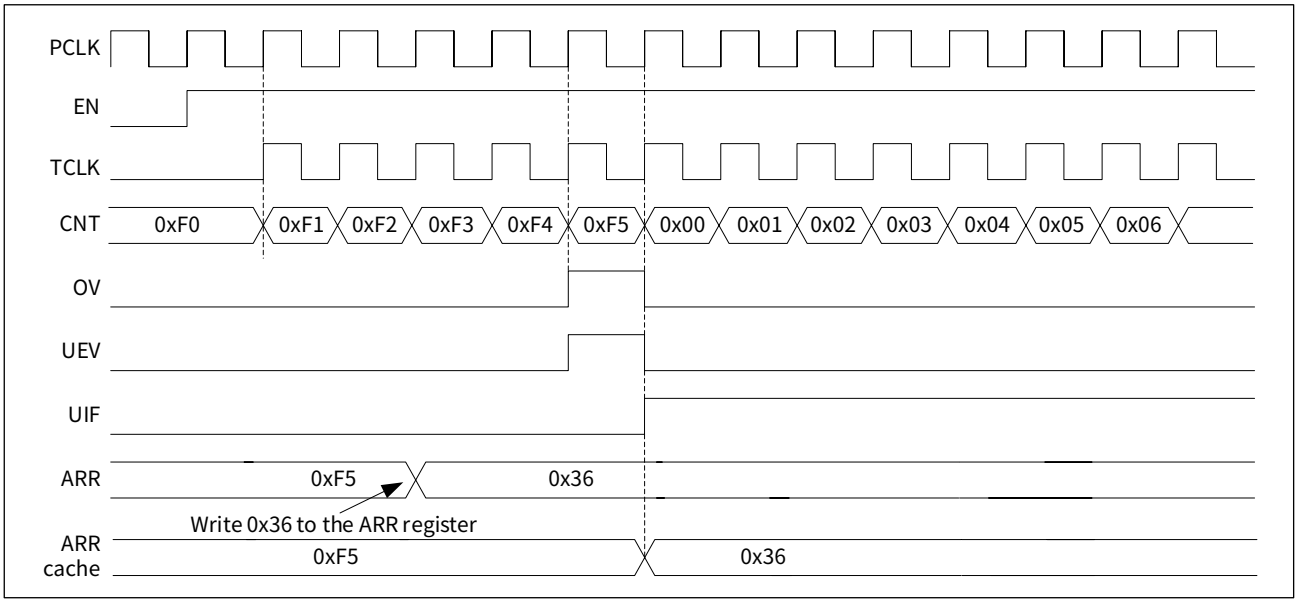


Figure 13-7 Upcounting, update event when reload buffer is enabled (ATIM_CR.BUFPEN=0x1)




## Downcounting mode

When the DIR bit of the control register ATIM_CR is set to 1 in edge-aligned mode, the counter works in the downcounting mode.

Set ATIM_CR.EN to 1 to start ATIM, the hardware automatically loads ARR to the counter CNT, and the counter starts to count down. When the count value reaches 0, the overflow signal UND is generated (the overflow signal UND remains for one PCLK cycle, and then automatically cleared). One PCLK after overflow, the counter is reloaded with the value of ARR, the counter underflow interrupt flag ATIM_ISR.UNDF is set by hardware, if interrupt is enabled (set ATIM_CR.UNDE to 1), an interrupt request will be generated, set ATIM_ICR.UNDF to 0 to clear this flag bit.

If the repeatition counter is used, the update event (UEV) is generated when upcounting reaches the programmed number of repeat times(ATIM_RCR.RCR), else the update event is generated at each counter downflow.

The following figures show some examples of the counter behavior for different clock frequencies, where ATIM_ARR = 0x36.

Figure 13-8 Downcounting, internal clock divided by 1

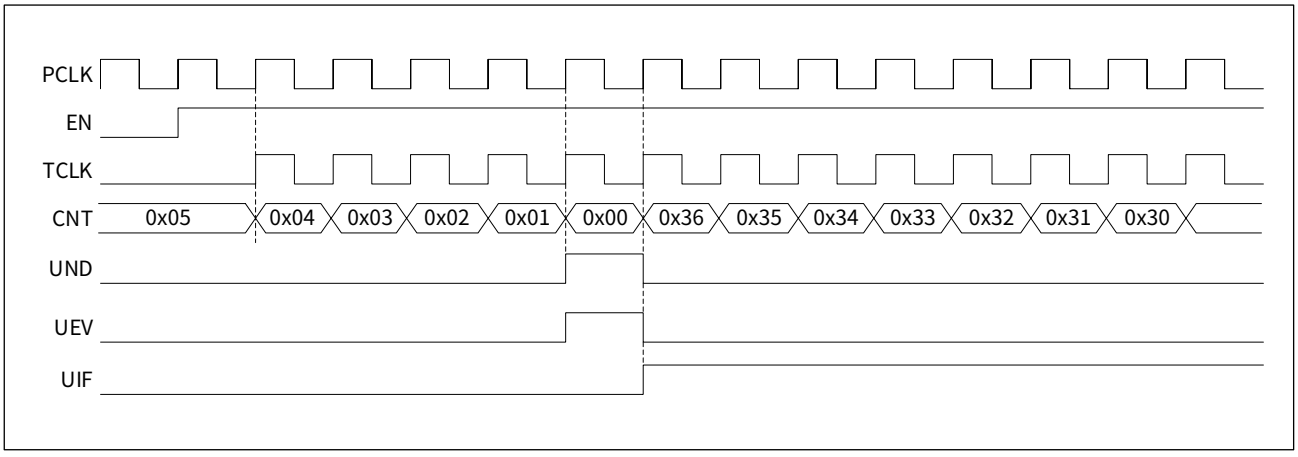


Figure 13-9 Downcounting, internal clock divided by 2

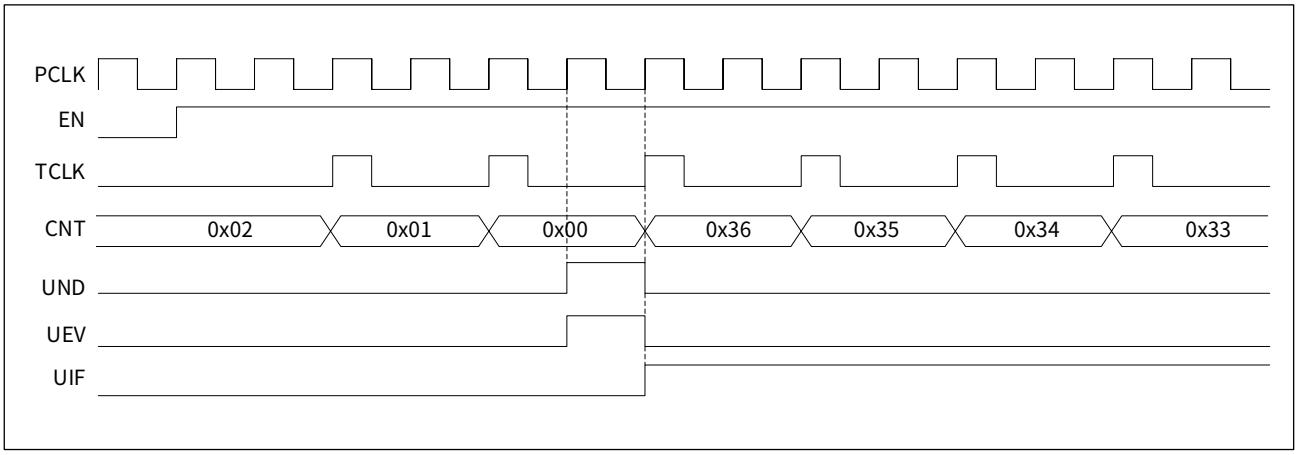



Figure 13-10 Downcounting, internal clock divided by 4

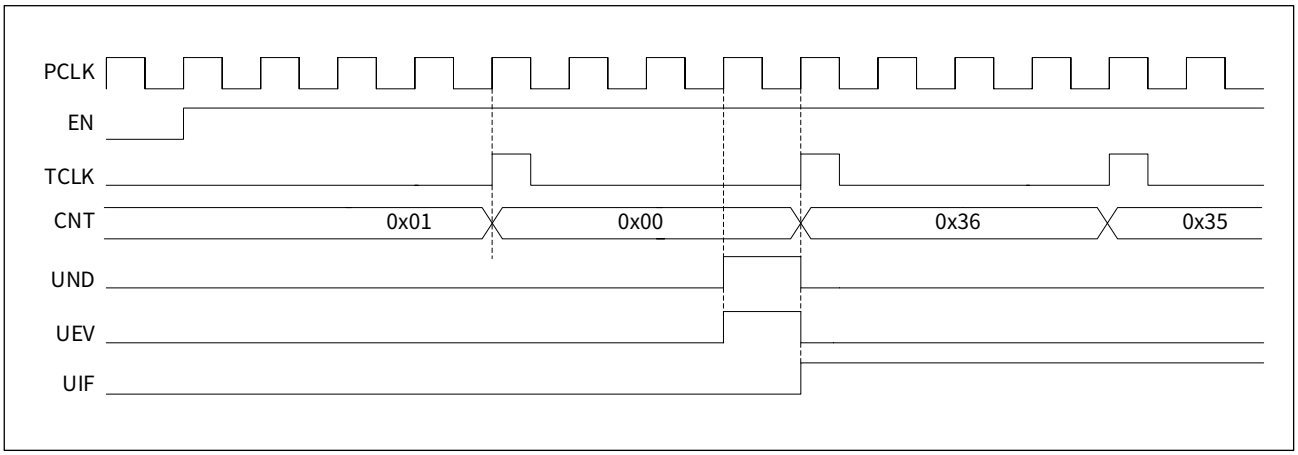



Figure 13-11 Downcounting, internal clock divided by N

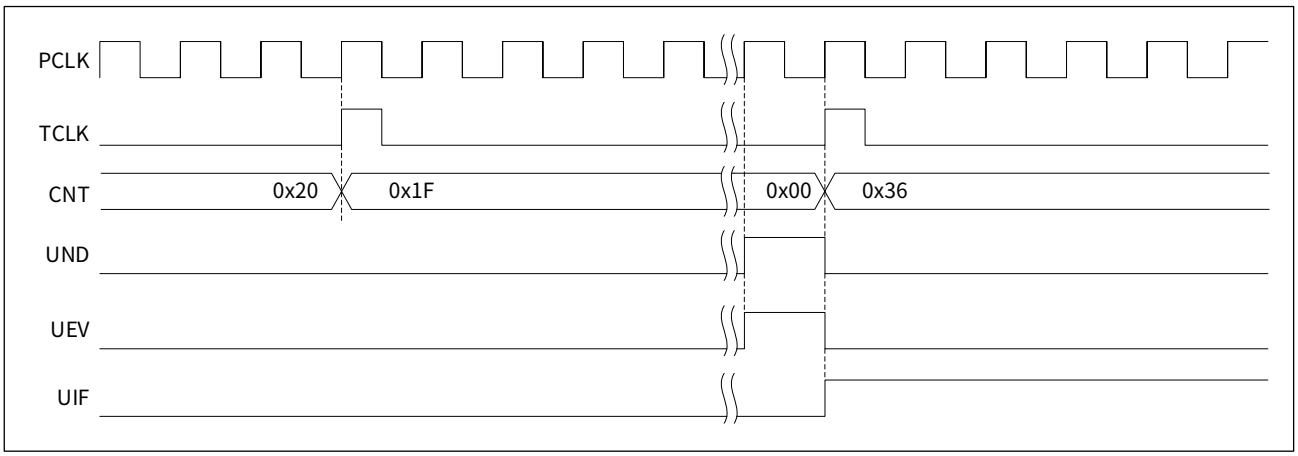
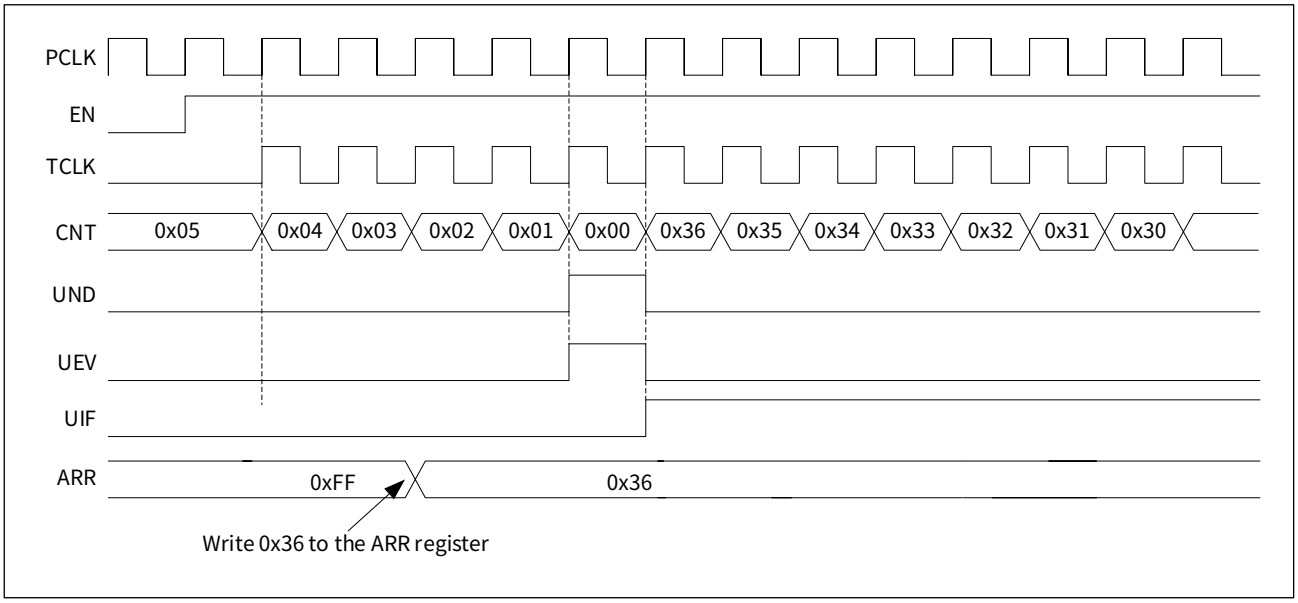


Figure 13-12 Downcounting, update event when reload buffer enabled is not used




### Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the reload value ARR, generates a counter overflow event, then counts down to 0 and generates a counter underflow event. Then it restarts counting up from 0.

In center-aligned mode, the DIR direction bit in the ATIM_CR register cannot be written but can be read. It is updated by hardware and gives the current direction of the counter. The DIR bit is automatically cleared to 0 when switching from other modes to center-aligned mode

If the repeatition counter is used, the update event (UEV) is generated when up/down counting reaches the programmed number of repeat times(ATIM_RCR.RCR), else the update event is generated at each counter overflow/downflow.

The following figures show some examples of the counter behavior for different clock frequencies:

Figure 13-13 Center-aligned mode, internal clock divided by 1, ARR=0x06

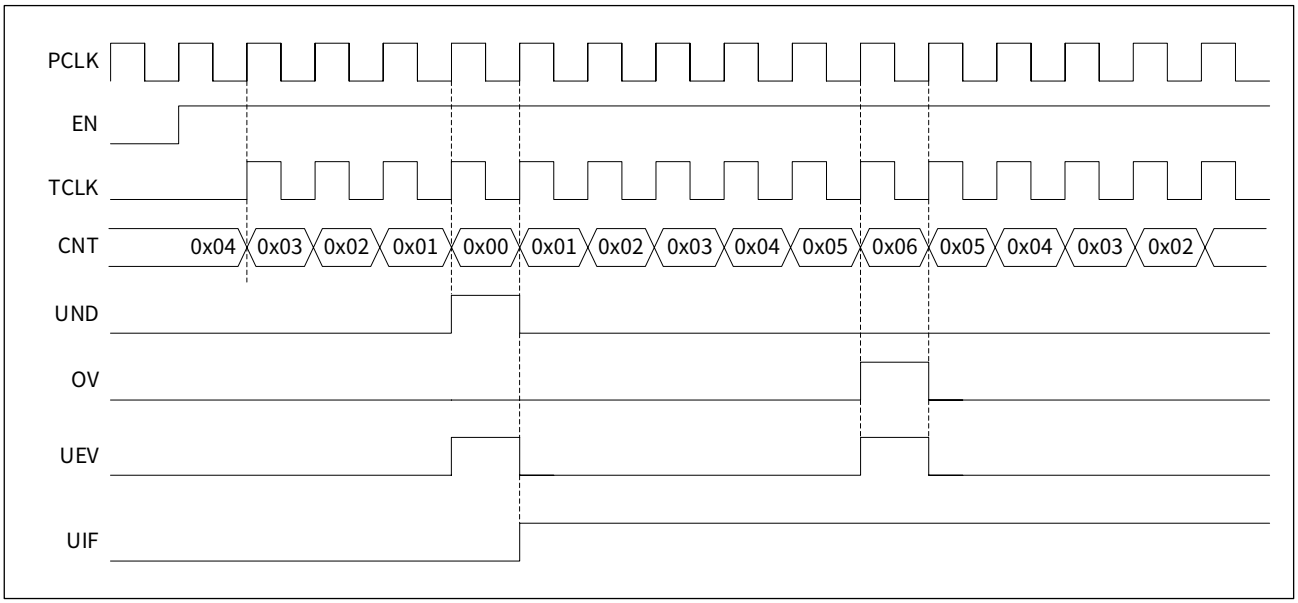


Figure 13-14 Center-aligned mode, internal clock divided by 2

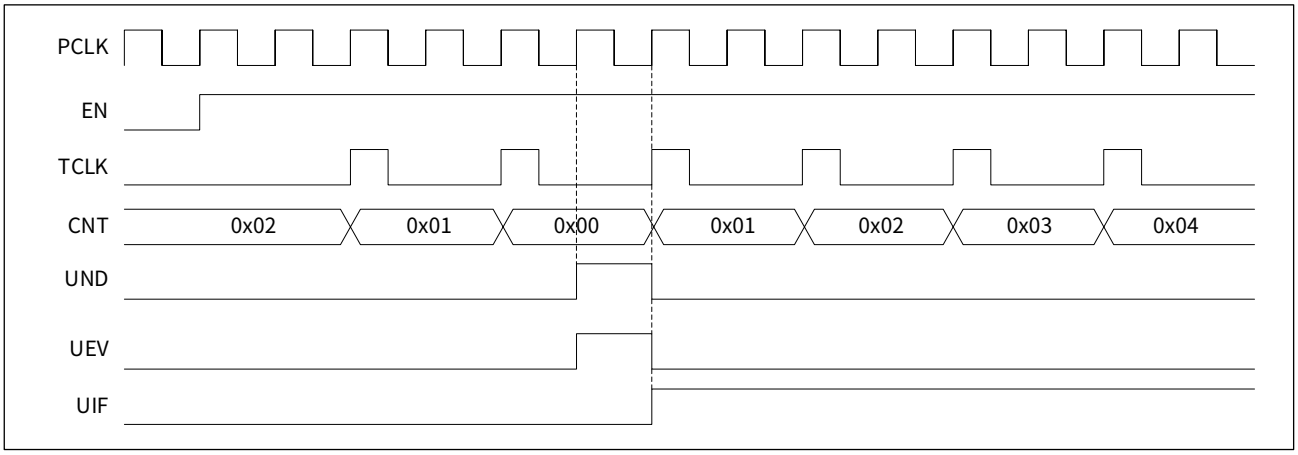



Figure 13-15 Center-aligned mode, internal clock divided by 4, ARR=0x36

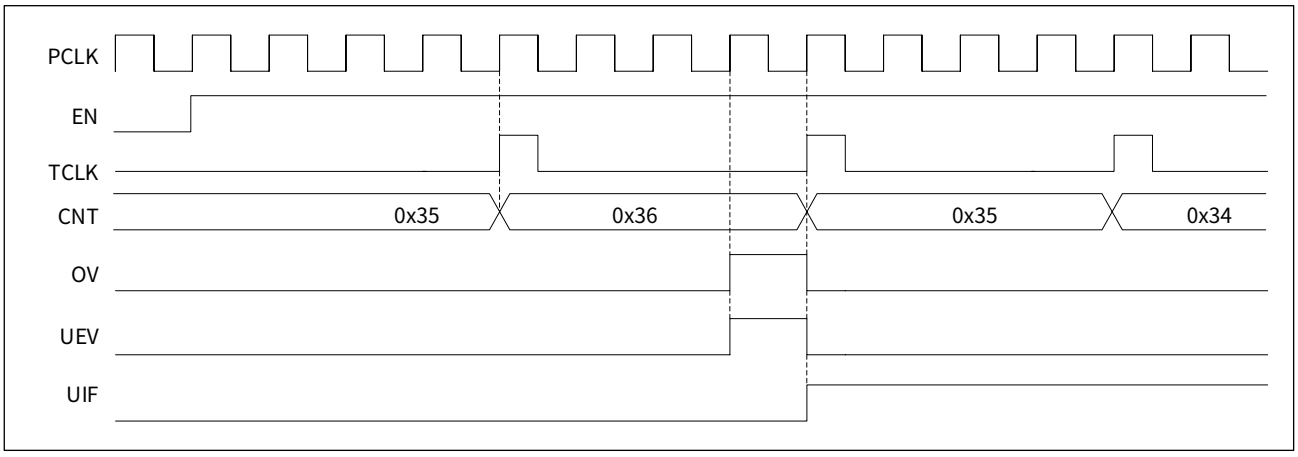



Figure 13-16 Center-aligned mode, internal clock divided by N

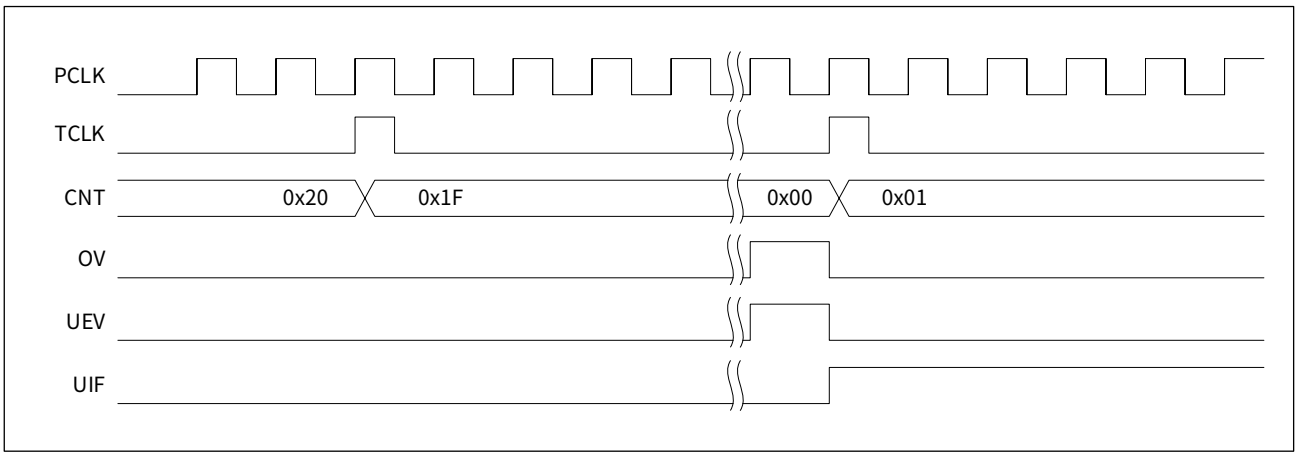


Figure 13-17 Center-aligned mode, update event when reload buffer is disabled (ATIM_CR.BUFPEN=0x0)

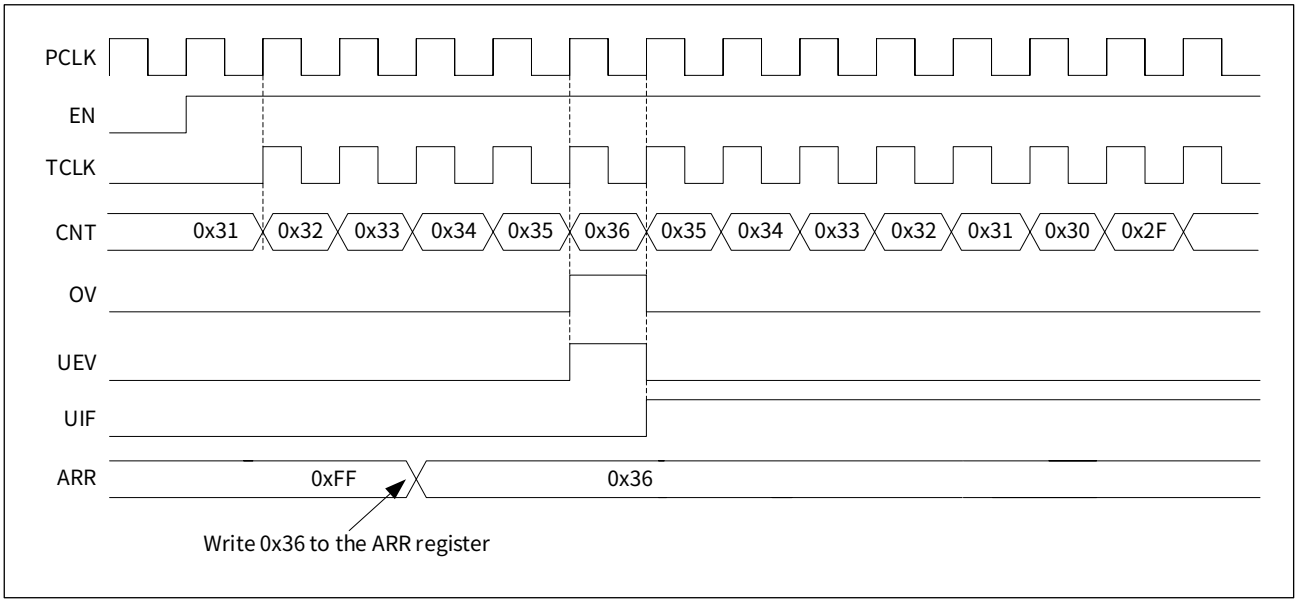
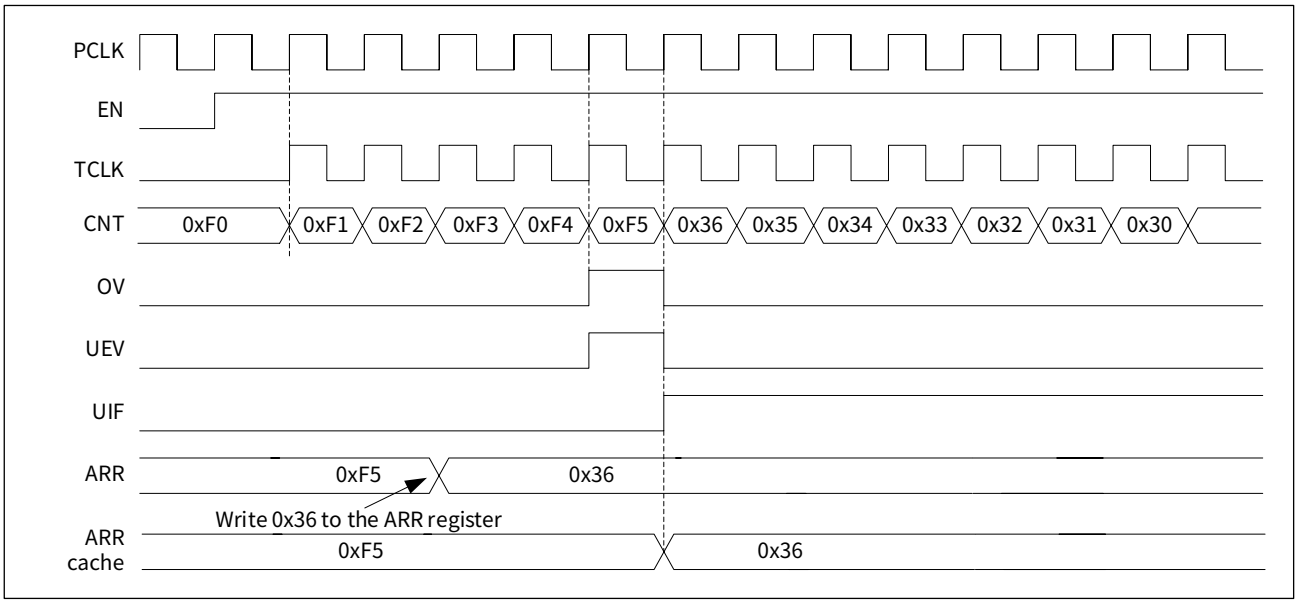



Figure 13-18 Center-aligned mode, update event when reload buffer is enabled (ATIM_CR.BUFPEN=0x1)




### 13.3.1.4 Reload register

The auto-reload register ATIM_ARR has a buffer function, which is started or closed by the BUFPEN bit field of the control register ATIM_CR.

When the counter is stopped or the buffer function is disabled, updating the reload register ARR will immediately update the buffer register. When the timer is running and the buffer function is valid, updating the reload register ARR will not update the buffer register immediately, and will only update the value of the reload register ARR to the buffer register when there is an update event UEV.

#### 13.3.1.5 Repetition counter

Starting ATIM will auto-load the RCR of the repeat count register ATIM_RCR to the repetition counter. When the main counter overflows, it will automatically decrease by one according to the UD and OV enable bits of the ATIM_RCR register, as shown in the following table:

Table 13-2 Repetition counter count configuration

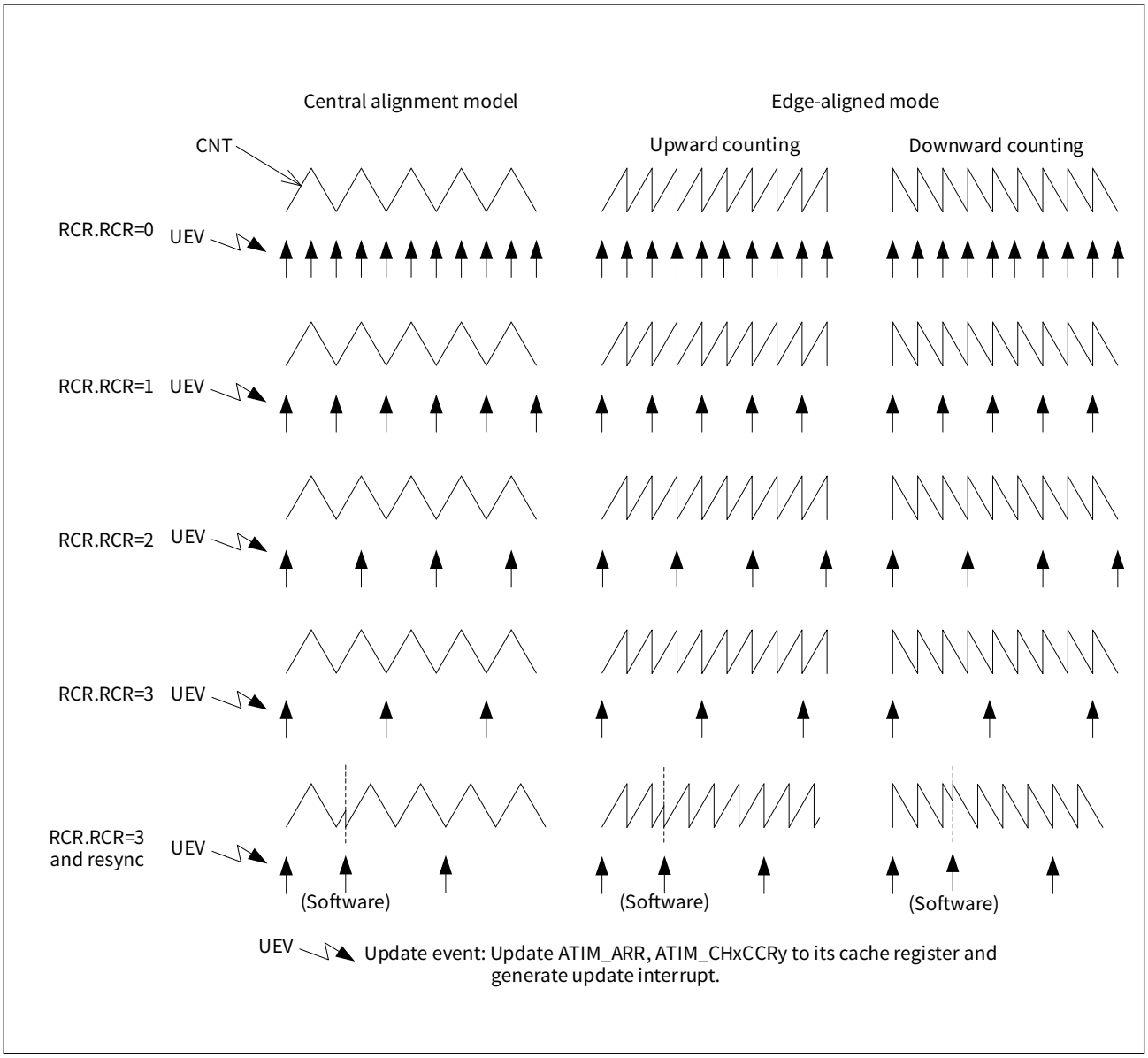
| Main counter overflow type | Working mode | ATIM_CR.DIR | ATIM_RCR.UD | ATIM_RCR.OV | Repetition counter action |
|---|---|---|---|---|---|
| Overflow | Edge-aligned mode | 0, upcounting | - | 0 | Minus one |
| | | | - | 1 | No count |
| | Center-aligned mode | - | - | 0 | Minus one |
| | | | - | 1 | No count |
| Underflow | Edge-aligned mode | 1, downcounting | 0 | - | Minus one |
| | | | 1 | - | No count |
| | Center-aligned mode | - | 0 | - | Minus one |
| | | | | - | No count |

When using the repeat counter function (the RCR of ATIM_RCR is not set to 0), the overflow of the main counter will generate the update event (UEV) only when the repeat counter is decremented to 0.

When UG is updated by software or reset from slave mode, the update event (UEV) will be generated immediately, regardless of the current value of the repetitiont counter, and the content of the RCR in the ATIM_RCR register will be loaded into the repetition counter immediately.

The following figure is example of different repetition count values, and the update event (UEV) is generated in different counting modes:

Figure 13-19 Repetition counting example



#### 13.3.1.6 Compare/capture channels

ATIM has 3 independent compare/capture channels CH1, CH2, CH3, each channel has A and B compare/capture registers and their buffer registers, as well as input signal processing units (digital filtering and phase detection), comparison units, and matching output unit.

Therefore, ATIM can realize 6-channel input capture, or 6-channel independent PWM output, or 3 pairs of complementary PWM output. Channel CH4 of ATIM is an internal channel of the chip, with no external pins, only one compare/capture register (ATIM_CH4CCR), which can only be used for comparison and cannot be used for capture.

The corresponding pins of the compare/capture channel supported by ATIM are shown in the following table, and the function multiplexing needs to be configured in the application.

Table 13-3 ATIM compare/capture channel pins

| ATIM | Pins | AFR |
|---|---|---|
| ATIM_CH1A | PB06、PC00 | 0x04 |
| ATIM_CH1B | PA07、PB00、PB05 | 0x04 |
| ATIM_CH2A | PB03、PC01 | 0x04 |
| ATIM_CH2B | PA06、PB02、PC04 | 0x04 |
| ATIM_CH3A | PA00、PC02 | 0x04 |
| ATIM_CH3B | PA01、PA04、PC03 | 0x04 |

### 13.3.2 Input capture mode

#### 13.3.2.1 Input capture

Input capture is supported in both edge-aligned and center-aligned modes. Set the CSy bit field of the channel x control register ATIM_CHxCR to 1, so that the channel CHxy (x=1~3, y=A, B) works in the input capture mode, and the input capture can be triggered by software or hardware.

Software trigger: When ATIM_CHxCR.CCGy is set to 1, the software triggers a capture immediately, the value of main count register ATIM_CNT is latched into the compare/capture register ATIM_CHxCCRy of the corresponding channel, and a capture is completed. After the capture is completed, the CCGy bit is automatically cleared by hardware .

Hardware trigger: When the valid edge on the channel CHxy is detected, the value of main count register ATIM_CNT is latched into the compare capture register ATIM_CHxCCRy of the corresponding channel to complete a capture. The valid edge that triggers the capture is selected by the BKSy bit field of the ATIM_CHxCR register, as shown in the following table:

Table 13-4 Input capture mode configuration

| Advanced Timer | ChannelCHxy | BKSy bit field value | Capture trigger condition |
|---|---|---|---|
| ATIM_CHxCR (x=1,2,3) | BKSy (y=A,B) | 00 | Prohibit |
| | | 01 | Capture on rising edge |
| | | 10 | Capture on falling edge |
| | | 11 | Capture on both rising and falling edges |

When a capture occurs, the capture interrupt flag bit ATIM_ISR.CxyF of the corresponding channel is set by hardware. If the interrupt is enabled (set ATIM_CHxCR.CIEy to 1), an interrupt request will be generated. If the ATIM_ISR.CxyF flag is already high when a capture event occurs, the recapture flag ATIM_ISR.CxyE will be set by hardware. Set ATIM_ICR.CxyF to 0 to clear the ATIM_ISR.CxyF flag, set ATIM_ICR.CxyE to 0 to clear the ATIM_ISR.CxyE flag.

*Caution:*

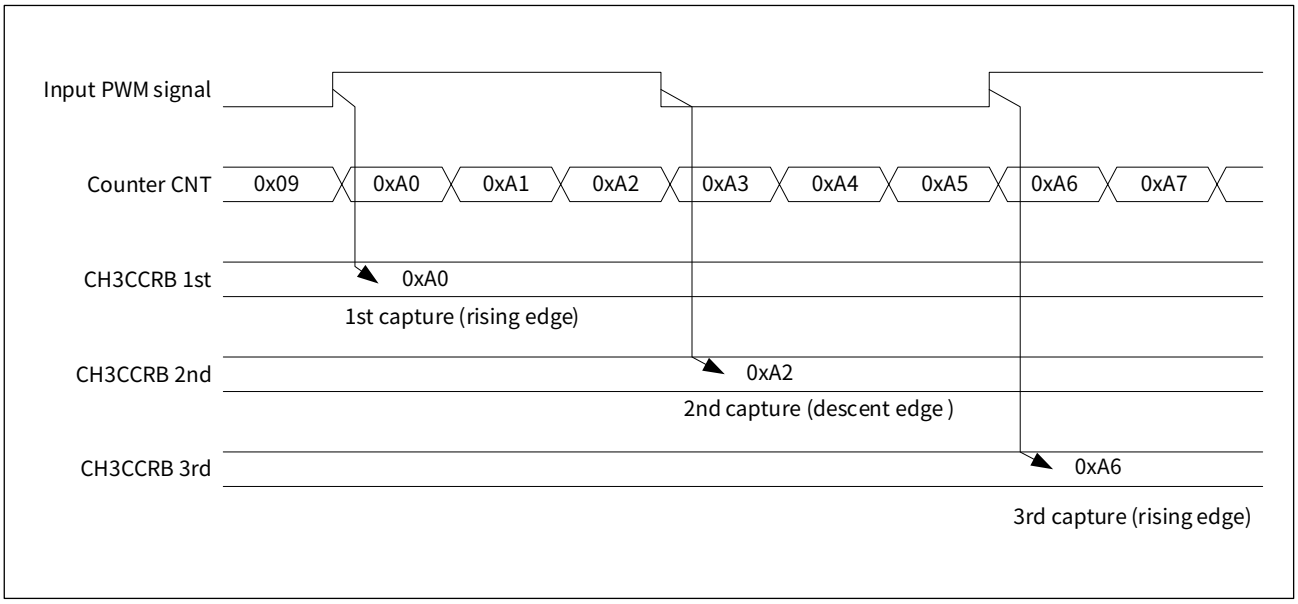
*Even if ATIM is not started and ATIM_CNT does not start counting, when the valid edge of channel CHxy is detected, the capture will still be triggered and the capture action will be executed.*

### 13.3.2.2 PWM input mode

PWM input mode is an application of input capture mode. Set the BKSy bit field of the channel x control register ATIM_CHxCR to 0x03, so that the timer captures both rising and falling edges of PWM input signal. After three captures, the pulse width and period of the PWM signal can be calculated.

The schematic diagram of PWM signal measurement of CH3B channel is shown in the following figure:

Figure 13-20 Schematic diagram of PWM measurement



#### 13.3.2.3 Input capture trigger source

The input capture trigger source of ATIM can choose the external CHxy pin input signal, or the internal trigger TS signal.

The input capture of channels CH2A, CH2B, CH3A, and CH3B can only select the corresponding external GPIO pin input. The channel filter setting can be performed through the OCMxyFLTxy bit field of the filter register ATIM_FLTR, and the phase setting can be performed through the ATIM_CHxCR.BKSy bit field.

CH1 of ATIM is an input channel with advanced functions. Input terminals CH1A and CH1B can be used as trigger capture and trigger start of main counter.

The input capture of CH1A is determined by the IA1S bit field of the master-slave mode control register ATIM_MSCR, and the input of CH1A or the exclusive OR input of CH1A, CH2A and CH3A can be selected.

Select CHIA input when ATIM_MSCR.IA1S is set to 0, the channel filter can be set by ATIM_FLTR.OCM1AFLT1A, and the phase can be set by ATIM_CH1CR.BKSA; when ATIM_MSCR.IA1S is set to 1, the XOR input of CH1A, CH2A, and CH3A can be selected. This feature is used for Connect the hall sensor.

The input capture of CH1B is determined by the IB1S bit field of the master-slave mode control register ATIM_MSCR, and the CH1B input or the internal trigger TS signal can be selected.

When ATIM_MSCR.IB1S is set to 0, the CH1B input is selected, the channel filter can be set by ATIM_FLTR.OCM1BFLT1B, and the phase can be set by ATIM_CH1CR.BKSB; when ATIM_MSCR.IB1S is set to 1, the internal trigger TS signal is selected, and the TS signal is configured by ATIM_MSCR.TS source, as shown in the following table:

Table 13-5 TS signal sources

| ATIM_MSCR.TS | TS signal |
|---|---|
| 000 | ETR filtered and phase-selected signal ETFP |
| 001 | Internal interconnect signal ITR |
| 101 | Edge signal of port CH1A |
| 110 | Filtered and phase-selected signal IAFP at port CH1A |
| 111 | Filtered and phase-selected signal IBFP at port CH1B |

### 13.3.3 Output compare function

#### 13.3.3.1 Output compare

Set the CSy bit field of the channel x control register ATIM_CHxCR to 0, so that the channel CHxy works in the output compare mode. In the output compare mode, compare the value of the count register ATIM_CNT with the value of the compare/capture register ATIM_CHxCCRy of the corresponding channel CHxy. When the two match, the compare/capture channel CHxy outputs a settable level state.

The output state of the compare channel is set by the OCMxyFLTxy bit field of the ATIM_FLTR register, as shown in the following table:

Table 13-6 Output compare mode configuration

| ATIM_FLTR.OCMxyFLTxy | Compare mode configuration |
|---|---|
| 000 | Force CHxy to output low level |
| 001 | Force CHxy to output high level |
| 010 | CHxy is set to 0 on compare match |
| 011 | CHxy is set to 1 on compare match |
| 100 | CHxy flips output on compare match |
| 101 | Output high level for one count period on compare match |
| 110 | PWM mode 1 |
| 111 | PWM mode 2 |

The output polarity control bit CCPxy in the ATIM_FLTR register sets the output polarity of CHxy.

When a compare match occurs, the hardware sets the channel compare/match interrupt flag ATIM_ISR.CxyF, and at the same time:

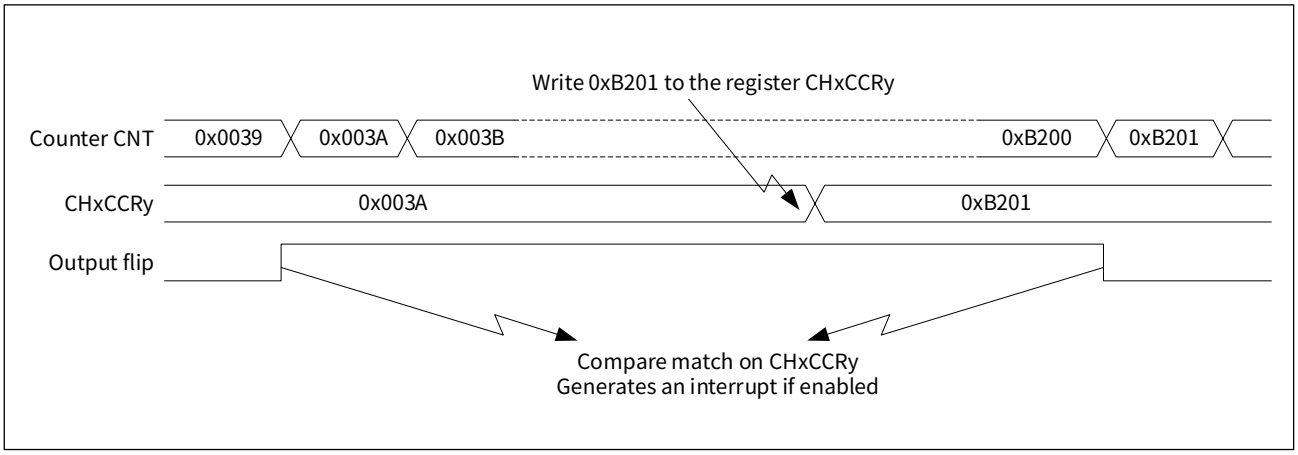
- If the interrupt is enabled, that is, ATIM_CHxCR.CIEy is 1, an interrupt request will be generated;
- If ADC triggering is enabled, that is, ATIM_TRIG.ADTE is 1, and ATIM_TRIG.CMxyE is 1, a compare match will trigger an ADC conversion request.

The compare/capture register ATIM_CHxCCRy has a cache function, and whether to use the buffer function is selected by the BUFEy bit of the ATIM_CHxCR register.

When ATIM_CHxCR.BUFEy is 0, the compare cache function of channel CHxy is disabled, and the ATIM_CHxCCRy register can be updated by software at any time, and the updated value takes effect immediately and affects the output waveform; when ATIM_CHxCR.BUFEy is 1, the compare cache of channel CHxy is enabled function, updating the ATIM_CHxCCRy register will not update the cache register immediately, only when the update event UEV occurs, the value of the ATIM_CHxCCRy register will be updated to the cache register.

The example of the compare/capture register cache function is shown in the following figure:

Figure 13-21 Compare/capture register cache function




### 13.3.3.2 Forced output

In forced output mode, the output state of the compare channel CHxy is directly forced to be 0 or 1 by the software, without depending on the comparison result. The OCMxyFLTxy bit field in the ATIM_FLTR register is set to 0x0, and the corresponding CHxy channel is forced to output low level; the OCMxyFLTxy bit field in the ATIM_FLTR register is set to 0x1, and the corresponding CHxy channel is forced to output high level.

In forced output mode, the ATIM_CHxCCRy register and the counter ATIM_CNT are continuously compared, and the comparison result will affect the corresponding flag bit, and also generate corresponding interrupts.

#### 13.3.3.3 Single pulse mode

The single pulse mode is a kind of PWM output mode. After the counter responds to the trigger, one pulse output is generated after a delay. The delay time and pulse width can be controlled by a program.

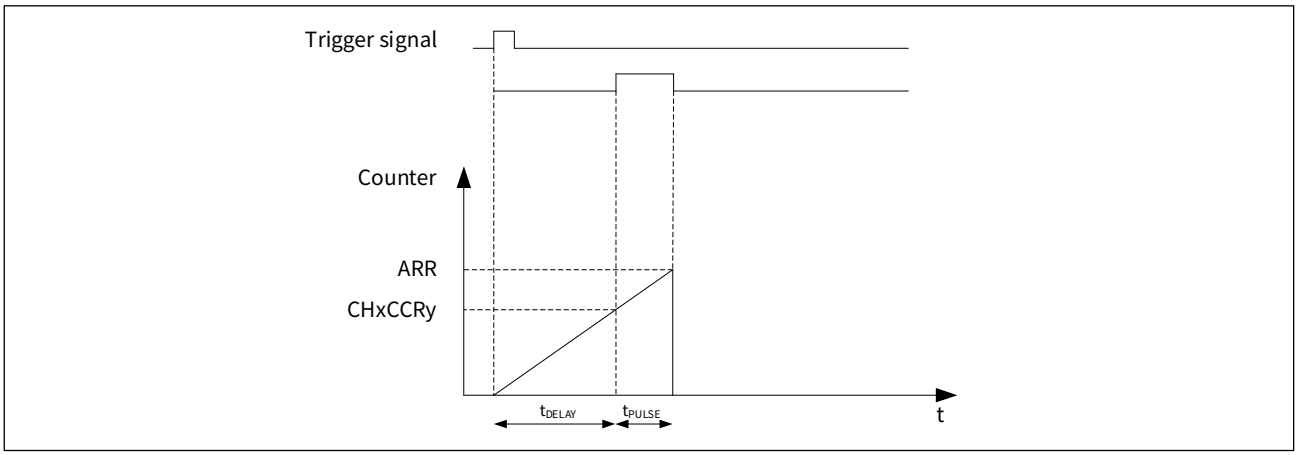
The single pulse mode needs to set the ATIM slave mode. For the slave mode and its trigger conditions, please refer to section *13.3.6 Slave mode*.

The OCMxyFLTxy bit field of the output control/input filter register ATIM_FLTR should be set to output compare mode or PWM output mode. At the same time, set the ONESHOT bit field of the control register ATIM_CR to 0x1, and select the single transmitting mode, the counter will generate the next update Stop counting on event UEV.

Before starting the single pulse mode, pay attention to the current value of the counter CNT, the reload value ARR, and the compare value CHxCCRy, which should meet the following conditions:

- Upcounting method: CNT < CHxCCRy <= ARR
- Downcounting method: CNT > CHxCCRy

Figure 13-22 Example of single pulse output (upcounting method)




After starting the timer with the slave mode trigger, the timer starts counting when it detects the valid edge of the trigger signal, and after a delay of $t_{DELAY}$, a positive pulse with a width of $t_{PULSE}$ is generated on the compare output port. Among them, $t_{DELAY}$ is determined by the difference between the initial value of the counting register ATIM_CNT and the compare/capture register ATIM_CHxCCRy, and $t_{PULSE}$ is determined by the reload register ATIM_ARR and the compare/capture register ATIM_CHxCCRy.

#### 13.3.3.4 Independent PWM output

The PWM (pulse width modulation) output mode of ATIM can generate a PWM waveform with programmable frequency and duty cycle. The frequency is determined by the reload register ATIM_ARR, and the duty cycle is determined by compare/capture register ATIM_CHxCCRy.

The PWM output mode needs to set the control register ATIM_CR, filter register ATIM_FLTR and dead-time register ATIM_DTR, as shown in the following table:

Table 13-7 PWM output mode

| PWM mode | Registers | Bit field | Bit field value | Description |
|---|---|---|---|---|
| PWM mode 1 | ATIM_CR | COMP | 0 | Independent PWM output mode |
| | ATIM_FLTR | OCMxyFLTxy | 110 | PWM mode 1 |
| | ATIM_DTR | MOE | 1 | Enable PWM output |
| PWM mode 2 | ATIM_CR | COMP | 0 | Independent PWM output mode |
| | ATIM_FLTR | OCMxyFLTxy | 111 | PWM mode 2 |
| | ATIM_DTR | MOE | 1 | Enable PWM output |

In PWM mode, channel A of the compare channel CHx can be configured as single-point compare or double-point compare through the PWM2S bit field of the control register ATIM_CR. In the single-point compare mode, use the compare/capture register ATIM_CHxCCRA to control the compare output; in the double-point compare mode, use the compare/capture registers ATIM_CHxCCRA and ATIM_CHxCCRB to control the compare output. The channel B of the compare channel can only use single-point compare, and the compare output is controlled by the compare/capture register ATIM_CHxCCRB.

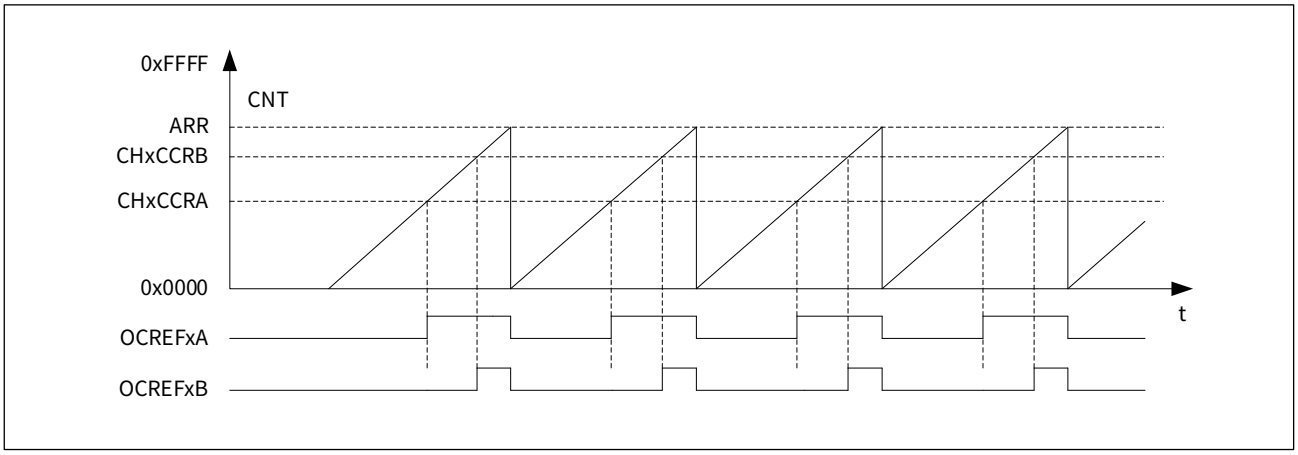
The output states of PWM modes 1 and 2 in different compare modes are shown in the following table:

Table 13-8 PWM mode 1, 2 output state in different compare modes

| Compare mode | Operating mode | Counting direction | PWM mode 1 | PWM mode 2 |
|---|---|---|---|---|
| Single-point | Edge-aligned mode, center-aligned mode | Upcounting | CNT < CHxCCRy, output high | CNT < CHxCCRy, output low |
| | | Downcounting | CNT > CHxCCRy, output low | CNT > CHxCCRy, output high |
| Double-point | Edge-aligned mode | Upcounting | CHxCCRA < CNT <= CHxCCRB, output low | CHxCCRA <= CNT < CHxCCRB, output high |
| | | Downcounting | CHxCCRA < CNT <= CHxCCRB, output high | CHxCCRA <= CNT < CHxCCRB, output low |
| | Center Aligned Mode | Upcounting | CNT < CHxCCRA, output high | CNT < CHxCCRA, output low |
| | | Downcounting | CNT > CHxCCRB, output low | CNT > CHxCCRB, output high |

The following are examples of independent PWM outputs of ATIM in different modes:

Figure 13-23 PWM mode 2, edge-aligned, upcounting, single-point comparison




The operation steps are as follows:

Step 1: Set the SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the register ATIM_CR.MODE to 0x2, the timer works in edge-aligned mode;

Step 3: Set the register ATIM_CR.COMP to 0x0, set as independent PWM output;

Step 4: Set the register ATIM_CR.DIR to 0x0, and the edge-aligned counting direction is upcounting;

Step 5: Set ATIM_ARR according to the period of PWM;

Step 6: Set the initial value of the counter ATIM_CNT (the initial value must be less than the value of ATIM_ARR);

Step 7: Set compare registers ATIM_CHxCCRA, ATIM_CHxCCRB according to the PWM pulse width;

Step 8: Set ATIM_FLTR.OCMxBFLTxB and ATIM_FLTR.OCMxAFLTxA to 0x7, select compare output mode as PWM mode 2;

Step 9: Set the relevant bits of the ATIM_ICR register and clear the relevant interrupt flag;

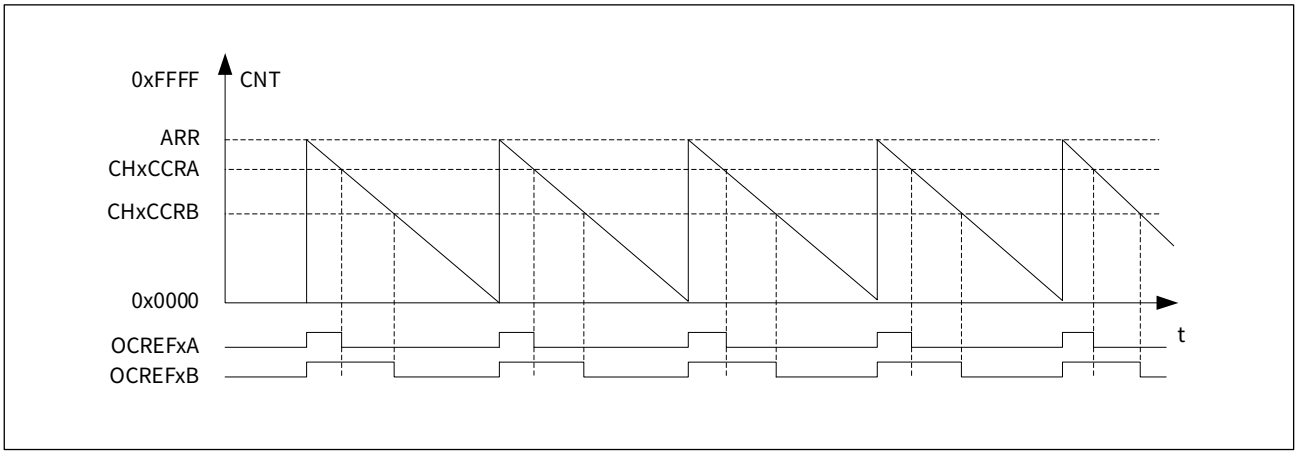
Step 10: If necessary, set the relevant interrupt enable;

Step 11: Set ATIM_FLTR.CCPxA and ATIM_FLTR.CCPxB to 0, select no inverting output;

Step 12: Set ATIM_DTR.MOE to 0x01 to enable PWM output;

Step 13: Set the register ATIM_CR.EN to 0x01 to start the timer.

Figure 13-24 PWM mode 2, edge-aligned, downcounting, single-point comparison




The operation steps are as follows:

Step 1: Set the SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the register ATIM_CR.MODE to 0x2, the timer works in edge-aligned mode

Step 3: Set the register ATIM_CR.COMP to 0x0, set as independent PWM output;

Step 4: Set the register ATIM_CR.DIR to 0x1, and the edge-aligned counting direction is downcounting;

Step 5: Set ATIM_ARR according to the period of PWM;

Step 6: Set the initial value of the counter ATIM_CNT (the initial value must be less than the value of ATIM_ARR);

Step 7: Set compare registers ATIM_CHxCCRA, ATIM_CHxCCRB according to the PWM pulse width;

Step 8: Set ATIM_FLTR.OCMxBFLTxB and ATIM_FLTR.OCMxAFLTxA to 0x7, so that the compare output mode is PWM mode 2;

Step 9: Set the relevant bits of the ATIM_ICR register and clear the relevant interrupt flag;

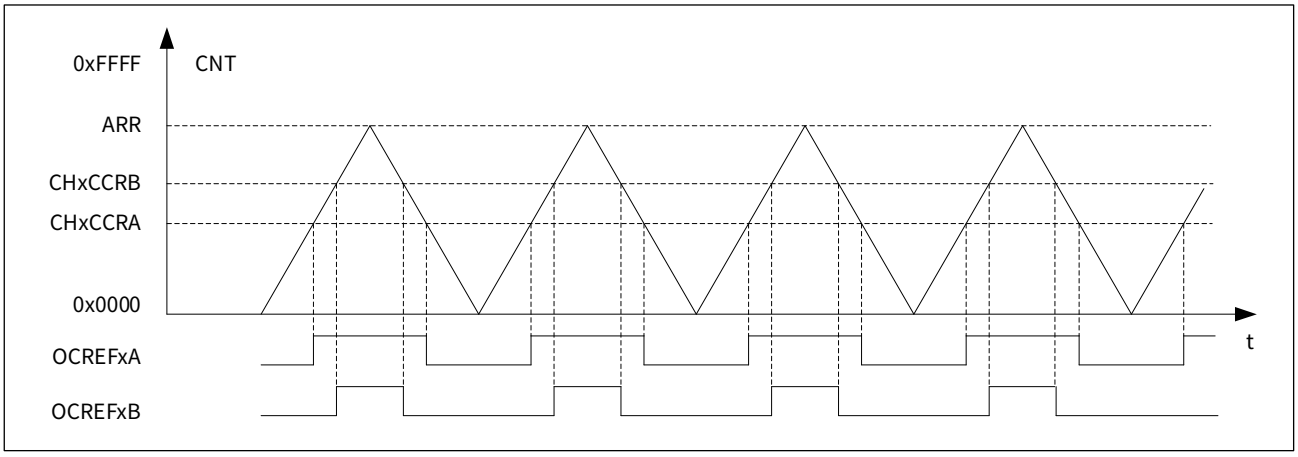
Step 10: If necessary, set the relevant interrupt enable;

Step 11: Set ATIM_FLTR.CCPxA and ATIM_FLTR.CCPxB to 0, no inverting output is required;

Step 12: Set ATIM_DTR.MOE to 0x01 to enable PWM output;

Step 13: Set the register ATIM_CR.EN to 0x01 to enable the timer.

Figure 13-25 PWM mode 2, center Aligned, single-point comparison




The operation steps are as follows:

Step 1: Set the SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the register ATIM_CR.MODE to 0x3, the timer works in center-aligned mode;

Step 3: Set register ATIM_CR.COMP to 0x0, set as independent PWM output;

Step 4: Set ATIM_ARR according to the period of the PWM;

Step 5: Set the initial value of the counter ATIM_CNT (the initial value must be less than the value of ATIM_ARR);

Step 6: Set compare registers ATIM_CHxCCRA, ATIM_CHxCCRB according to PWM pulse width;

Step 7: Set ATIM_FLTR.OCMxBFLTxB and ATIM_FLTR.OCMxAFLTxA to 0x7 to make the compare output mode PWM edge-aligned;

Step 8: Set the relevant bits of the ATIM_ICR register and clear the relevant interrupt flags;

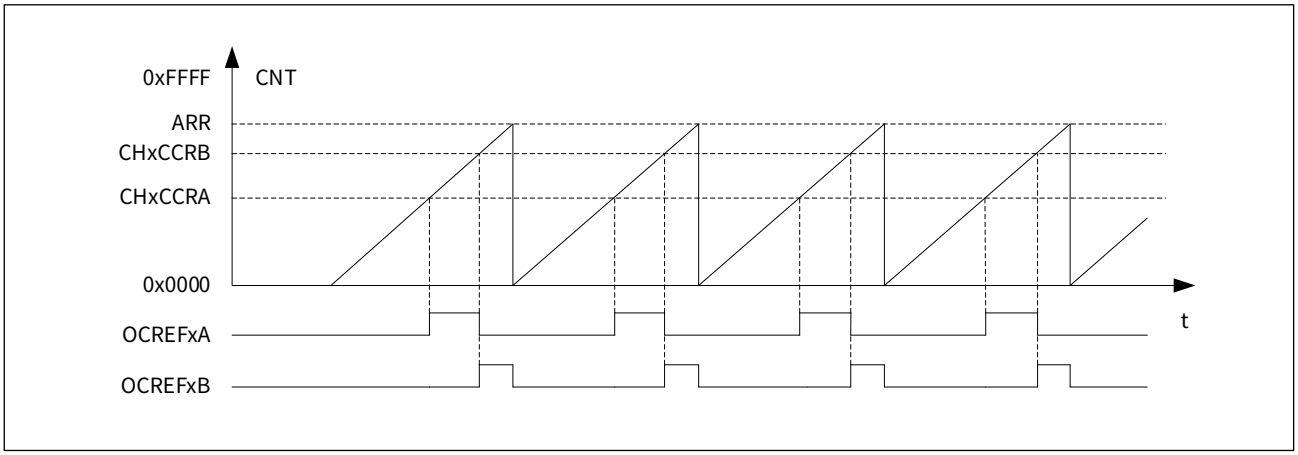
Step 9: Set related interrupt enable if needed;

Step 10: Set ATIM_FLTR.CCPxA and ATIM_FLTR.CCPxB to 0, no inverting output required;

Step 11: Set ATIM_DTR.MOE to 0x01 to enable PWM output;

Step 12: Set the register ATIM_CR.EN to 0x01 to enable the timer.

Figure 13-26 PWM mode 2, edge-aligned, two-point comparison




The operation steps are as follows:

Step 1: Set the system control register SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the register ATIM_CR.MODE to 0x2, the timer works in edge-aligned mode;

Step 3: Set register ATIM_CR.COMP to 0x0, set as independent PWM output;

Step 4: Set the register ATIM_CR.DIR to 0x0, and the edge-aligned counting direction is upcounting;

Step 5: Set ATIM_ARR according to the period of the PWM;

Step 6: Set the initial value of the counter ATIM_CNT (the initial value must be less than the value of ATIM_ARR);

Step 7: Set the compare registers ATIM_CHxCCRA, ATIM_CHxCCRB according to the PWM pulse width;

Step 8: Set ATIM_FLTR.OCMxBFLTxB and ATIM_FLTR.OCMxAFLTxA to 0x7 to make the compare output mode PWM mode 2;

Step 9: Set the relevant bits of the ATIM_ICR register and clear the relevant interrupt flags;

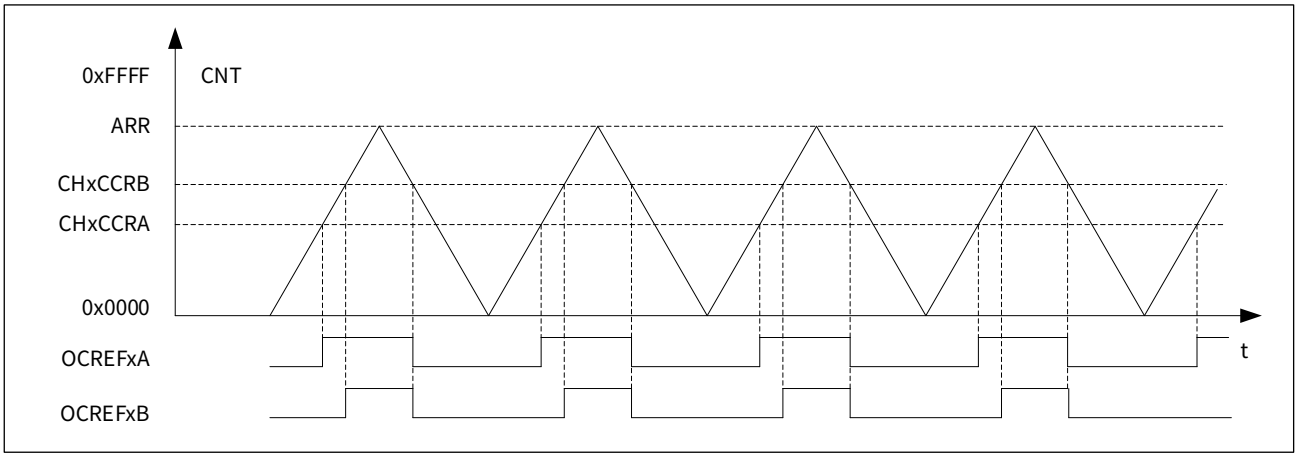
Step 10: Set related interrupt enable if needed;

Step 11: Set ATIM_FLTR.CCPxA and ATIM_FLTR.CCPxB to 0, no inverting output required;

Step 12: Set ATIM_DTR.MOE to 0x01 to enable PWM output;

Step 13: Set the register ATIM_CR.EN to 0x01 to enable the timer.

Figure 13-27 PWM mode 2, center-aligned, two-point comparison




*Caution:*

*OCREFxA is output by two-point comparison, and OCREFxB is controlled by CHxCCRB for single-point comparison output.*

The operation steps are as follows:

Step 1: Set the SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the register ATIM_CR.MODE to 0x3, the timer works in center-aligned mode;

Step 3: Set the register ATIM_CR.COMP to 0x0, set as independent PWM output;

Step 4: Set ATIM_ARR according to the period of PWM;

Step 5: Set the initial value of the counter ATIM_CNT (the initial value must be less than the value of ATIM_ARR);

Step 6: Set ATIM_CR.PWM2S to 0x0 to enable the two-point comparison function;

Step 7: Set the comparison registers ATIM_CHxCCRA, ATIM_CHxCCRB according to the PWM pulse width;

Step 8: Set ATIM_FLTR.OCMxBFLTxB and ATIM_FLTR.OCMxAFLTxA to 0x7, so that the compare output mode is PWM mode 2;

Step 9: Set the relevant bits of the ATIM_ICR register and clear the relevant interrupt flag;

Step 10: If necessary, set the relevant interrupt enable;

Step 11: Set ATIM_FLTR.CCPxA and ATIM_FLTR.CCPxB to 0, no inverting output is required;

Step 12: Set ATIM_DTR.MOE to 0x01 to enable PWM output;

Step 13: Set the register ATIM_CR.EN to 0x01 to enable the timer.

#### 13.3.3.5 Complementary PWM output and dead-time insertion

ATIM can output three complementary PWM signals, and can set dead-time.

Set the COMP bit field of the control register ATIM_CR to 1 to select the complementary PWM output mode, and compare the output channel CHxA and channel CHxB to generate a pair of complementary PWM. Set the OCMxAFLTxA bit field of the ATIM_FLTR register to 0x6 to select PWM mode 1, and to 0x7 to select PWM mode 2. Set the MOE bit field of the ATIM_DTR register to 1 to enable the PWM output.

Complementary PWM output mode, single-point comparison or two-point comparison can be selected through the PWM2S bit field of the control register ATIM_CR: use the comparison capture register ATIM_CHxCCRA to control the comparison output in single-point comparison; use the comparison capture register ATIM_CHxCCRA and ATIM_CHxCCRB to control the two-point comparison Compare output.

In complementary PWM output mode, channel A of channel CHx controls the output signal, and channel B compare capture register CHxCCRB no longer controls the CHxB output, but can still be used for internal control, such as triggering ADC.

Figure 13-28 Complementary PWM output waveform

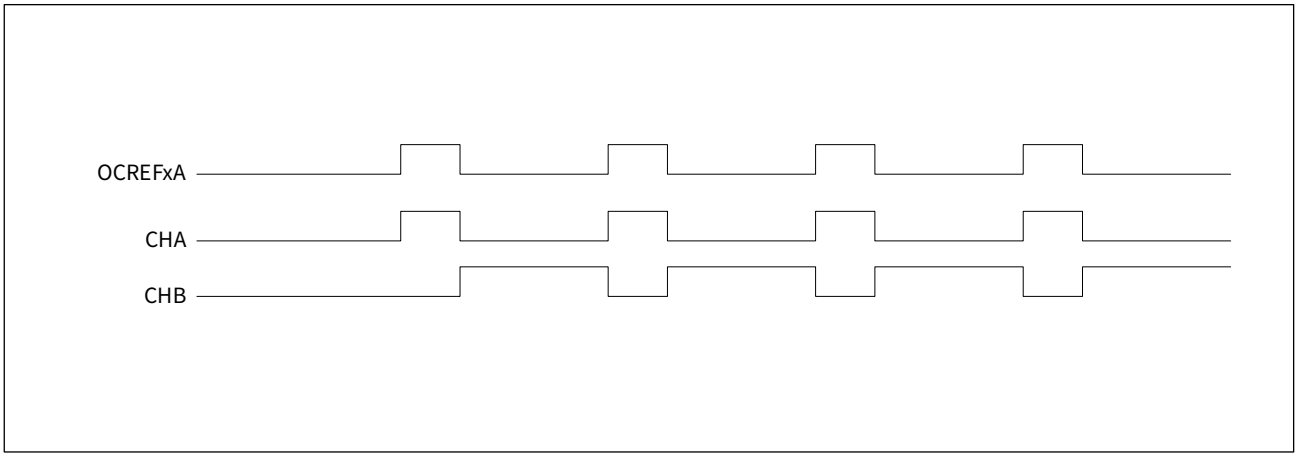
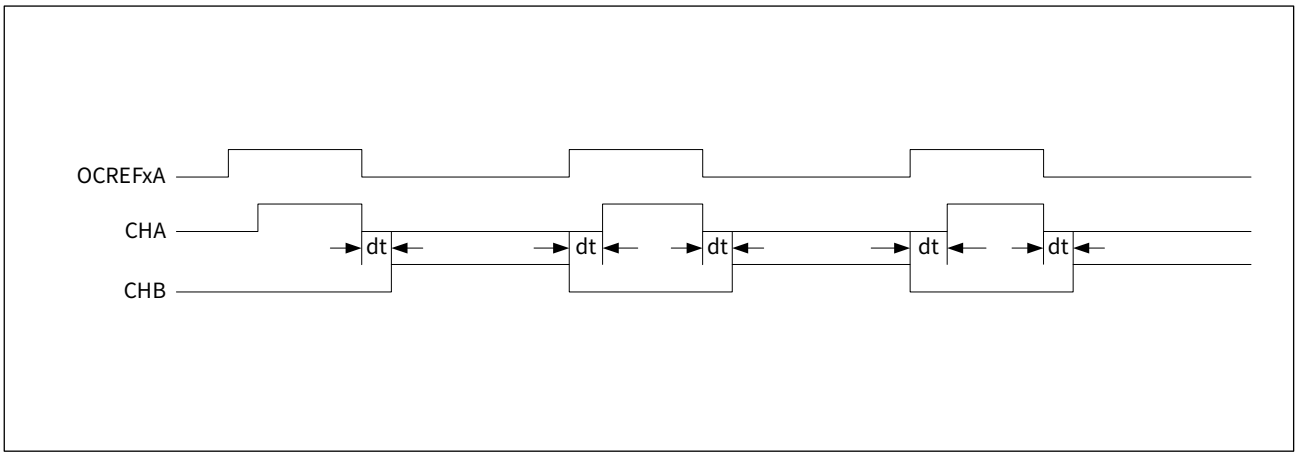



Figure 13-29 Complementary PWM output waveform with dead-time insertion




Set the DTEN bit field of the dead-time register ATIM_DTR to 1 to enable dead-time control.

The dead time of the three channels CHx is set uniformly and configured by the DTR bit field of the ATIM_DTR register, as shown in the following table:

Table 13-9 Dead-time settings of three channels CHx

| ATIM_DTR.DTR | Step length | Dead-time (unit: $T_{TCLK}$) | When $T_{TCLK}$ = 125 ns, dead-time range |
|---|---|---|---|
| DTR[7] = 0 | 1 $T_{TCLK}$ | DTR[6:0] + 2 | 0.25 ~ 16.125 μs |
| DTR[7:6] = 10 | 2 $T_{TCLK}$ | ( DTR[5:0] + 64 ) × 2 + 2 | 16.25 ~ 32 μs |
| DTR[7:5] = 110 | 8 $T_{TCLK}$ | ( DTR[4:0] + 32 ) × 8 + 2 | 32.5 ~ 63.25 μs |
| DTR[7:5] = 111 | 16 $T_{TCLK}$ | ( DTR[4:0] + 32 ) × 16 + 2 | 63.25 ~ 126.25 μs |

#### 13.3.3.6 Brake function

Set the BKE bit field of the dead-time register ATIM_DTR to 1 to enable the brake function.

The brake signal is active at high level. When the brake signal is active, the compare output channel CHxy will be immediately set to the output state set by the program. The specific output state is set by the BSKA and BSKB bit fields of the channel control register ATIM_CHxCR. The common brake control is BSKA and BAKB are high or low at the same time.

**Brake trigger source**

- Software brake, set the BG bit field of the control register ATIM_CR to 1 to trigger the software brake, and the BG bit is automatically cleared
- The compare output of the voltage comparator VC, the VCE bit field of the ATIM_DTR register is 1 to enable
- System abnormality, such as: abnormal clock, abnormal power supply, etc., the SAFEEN bit field of the ATIM_DTR register is 1 to enable
- External ATIM_BK pin input (GPIO function multiplexing setting), filter and set the brake input signal through the FLTBK bit field of the ATIM_FLTR register, and select the phase of the brake input signal in the BKP bit field of the ATIM_FLTR register

When the brake signal is valid, the brake interrupt flag ATIM_ISR.BIF will be set by hardware. If the brake interrupt is enabled (set ATIM_CR.BIE to 1), an interrupt request will be generated. Set ATIM_ICR.BIF to 0 to clear the flag.

#### 13.3.3.7 Compare match interrupt

After the ATIM timer is started, the count register ATIM_CNT and the compare/capture register ATIM_CHxCCRy begin to compare, when the two values are equal, a compare match event will be generated.

When working in edge-aligned mode, regardless of whether the counting direction is up or down, when a compare match occurs, the capture/compare match flag bit ATIM_ISR.CxyF of the corresponding channel will be set by hardware, if interrupts are enabled (set ATIM_CHxCR.CIEy to 1) , an interrupt request will be generated, set ATIM_ICR.CxyF to 0 to clear this flag.

When working in the center-aligned mode, the timing of the compare match interrupt of channel A and channel B of the compare channel is different:

- The channel A of compare channel is controlled by the CISA bit field of the control register ATIM_CR, and the timing of the compare match interrupt can be selected in the process of upcounting, downcounting or bidirectional counting. When the comparison matches, the capture/compare match flag bit ATIM_ISR. CxAF of the corresponding channel will be set by hardware. If the interrupt is enabled (set ATIM_CHxCR. CIEA to 1), an interrupt request will be generated, set ATIM_ICR.CxAF to 0 to clear the flag bit .
- The channel B of compare channel is controlled by the CISB bit field of the channel x control register ATIM_CHxCR, and the timing of the compare match interrupt can be selected in the process of upcounting, downcounting or bidirectional counting. When the comparison matches, the capture/compare match flag bit ATIM_ISR.CxBF of the corresponding channel will be set by hardware, if the interrupt is enabled (set ATIM_CHxCR.CIEB to 1), an interrupt request will be generated, set ATIM_ICR.CxBF to 0 to clear the flag bit . The compare match of channel B of the compare channel is controlled independently, which can trigger the ADC more flexibly. For details, see section *13.3.5 Trigger ADC*.

Table 13-10 Compare match interrupt settings

| Operating mode | Channel A/B | Counting direction | CISA/CISB bit fields | Whether to generate an interrupt on a compare match |
|---|---|---|---|---|
| Edge-aligned mode | Channel A/B | Down/up | - | Yes |
| Center-aligned mode | Channel A | Up | CISA=01 or 11 | Yes |
| | | | CISA=10 or 00 | No |
| | | Down | CISA=10 or 11 | Yes |
| | | | CISA=01 or 00 | No |
| | ChannelB | Up | CISB=01 or 11 | Yes |
| | | | CISB=10 or 00 | No |
| | | Down | CISB=10 or 11 | Yes |
| | | | CISB=01 or 00 | No |

The example of interrupt generated by compare match is shown in the following figure:

Figure 13-30 Edge-aligned mode compare match interrupt

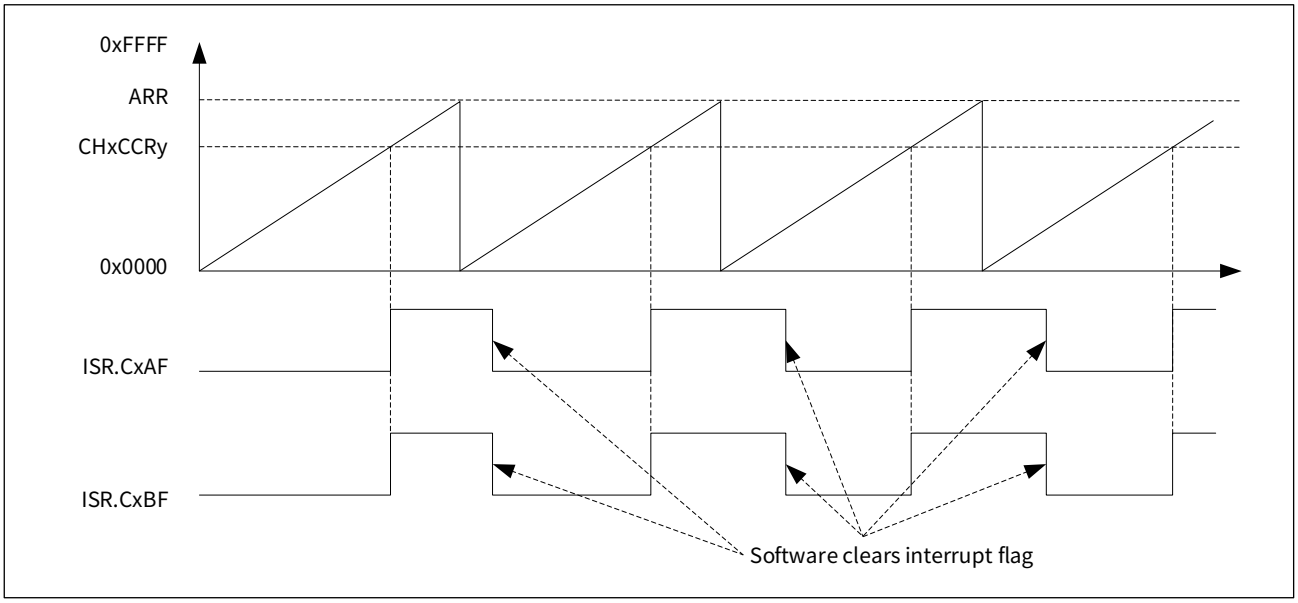
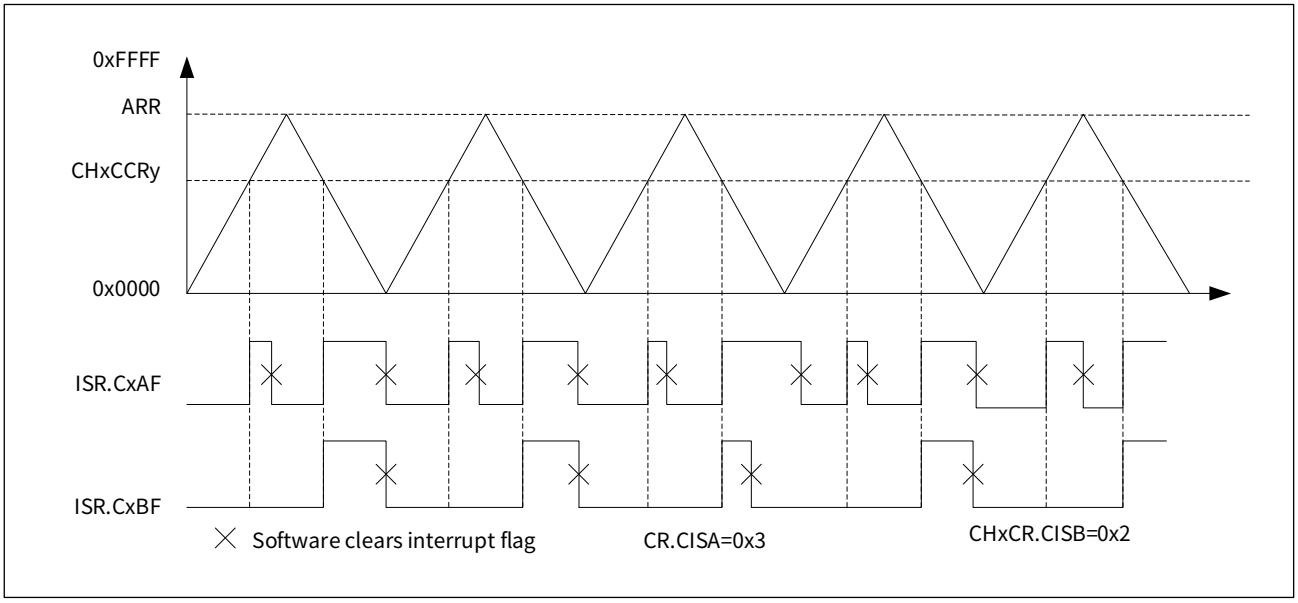


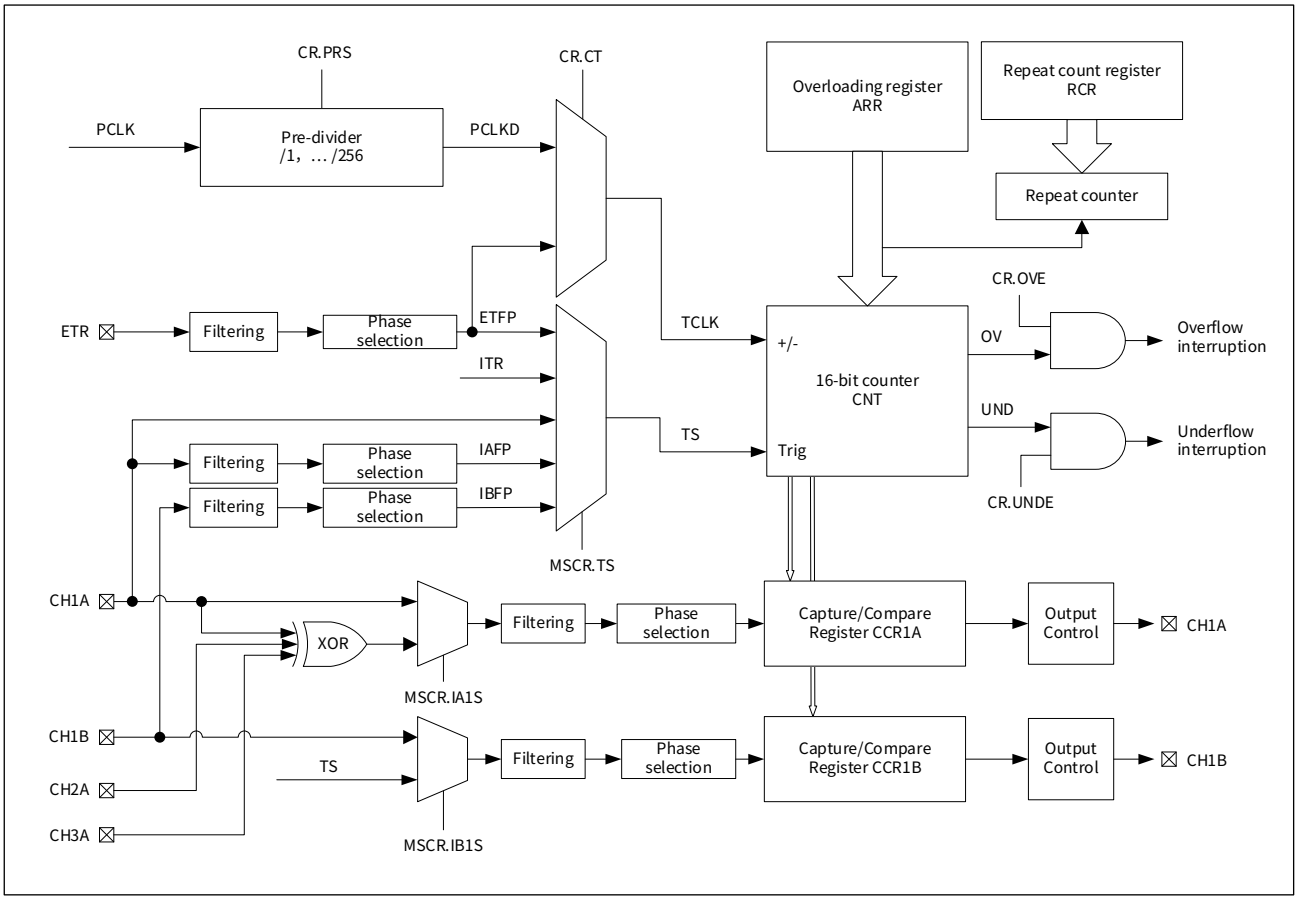

Figure 13-31 Center-aligned mode, channel A bidiectional compare match, channel B downcounting compare match



### 13.3.4 Quadrature encoding counting

When ATIM works in slave mode, it has the function of Quadrature encoding countinginging. It connects the external quadrature encoder through CH1A and CH1B. According to the jump sequence of the input signal, the counter can automatically count up or down. Its functional block diagram is shown in the following figure:

Figure 13-32 Quadrature encoding counting




The CH1A and CH1B input signals have filtering function, which are set by the OCM1AFLT1A and OCM1BFLT1B bit fields of the ATIM_FLTR register respectively; the configurable phase is controlled by the CCP1A and CCP1B bit fields of the ATIM_FLTR register respectively.

IAFP and IBFP are the internal signal names of CH1A and CH1B after filtering and phase selection, which are used as the input signal interface of the encoder. The phase relationship between IAFP and IBFP determines the counting direction, and also affects the direction bit field DIR of the control register ATIM_CR. .

The SMS bit fields of the master-slave mode control register ATIM_MSCR are 0x4, 0x5, and 0x6, which correspond to orthogonal encoding mode 1, mode 2, and mode 3 respectively. Quadrature encoding mode 1 uses the edge counting of CH1A; mode 2 uses the edge counting of CH1B; mode 3 uses the edge counting of CH1A and CH1B. The relationship between the counting direction and the encoder signal in the three modes of quadrature encoding is shown in the following table:

Table 13-11 Quadrature encoding mode counting direction

| Mode | Signal level | | IAFP | | IBFP | |
|---|---|---|---|---|---|---|
| | IAFP | IBFP | Rising | Falling | Rising | Falling |
| Mode 1 | High | - | Downcounting | Upcounting | No count | No count |
| | Low | - | Upcounting | Downcounting | No count | No count |
| Mode 2 | - | High | No count | No count | Upcounting | Downcounting |
| | - | Low | No count | No count | Downcounting | Upcounting |
| Mode 3 | High | High | Downcounting | Upcounting | Upcounting | Downcounting |
| | Low | Low | Upcounting | Downcounting | Downcounting | Upcounting |

*Caution:*

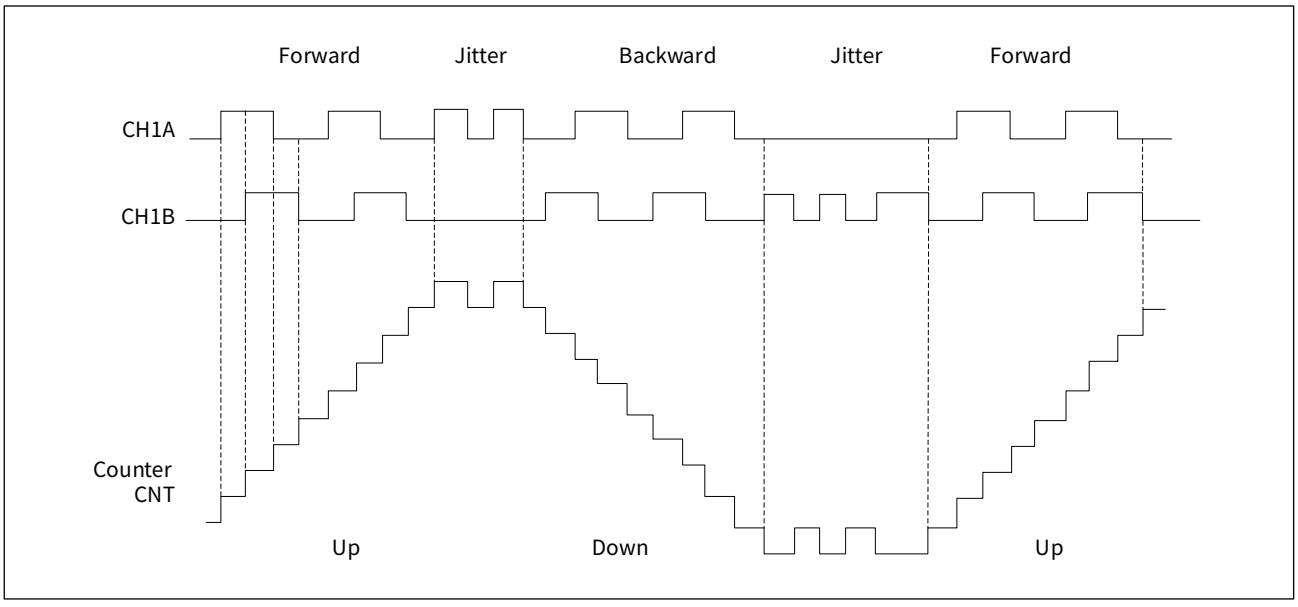
*To ensure the correctness of counting direction and speed, the phase difference between CH1A and CH1B should be greater than one TCLK pulse width; the signal pulse width of CH1A or CH1B should be greater than two TCLK pulse widths.*

The following figure is an example of encoding count operation, showing the generation and direction control of the count signal. (It also shows how to suppress input jitter when counting on both edges in center-aligned mode, which typically occurs when the encoder switches directions.)

The example configuration is as follows:

- CH1A is filtered and non-inverted into CH1CCRA
- CH1B is filtered and non-inverted into CH1CCRB
- Set the SMS bit field of the TIM_MSCR register to 0x6, select the quadrature encoding mode to 3, and count the rising and falling edges of CH1A and CH1B.
- Set ATIM_CR.EN to 0x01, start counter

Figure 13-33 Encoding counting operation example



### 13.3.5 Trigger ADC

Both ATIM edge-aligned and center-aligned modes support compare match events or update events to trigger the ADC. The ADC trigger function can be applied to motor FOC control.

Set the ADTE bit field of the trigger ADC control register ATIM_TRIG to 1, enable the ADC trigger global control, and allow the ADC to be triggered when a compare match or event update occurs.

Set ATIM_TRIG.UEVE to 1, and the ADC will be triggered synchronously when the update event occurs.

Set ATIM_TRIG.CMxyE to 1 to trigger the ADC when the channel CHxy compare match occurs. For the timing of the compare match, please refer to section *13.3.3.7 Compare match interrupt*.

### 13.3.6 Slave mode

The slave mode of the ATIM timer means that the operation mode of the timer is controlled by the trigger signal. The SMS bit field of the master-slave mode control register ATIM_MSCR can set the different working states of the ATIM timer in the slave mode. As shown in the following table:

Table 13-12 Working status of slave mode

| ATIM_MSCR.SMS | Slave mode function selection |
|---|---|
| 000 | Use internal clock |
| 001 | Reset function |
| 010 | Trigger mode |
| 011 | External clock mode |
| 100 | Quadrature encoding counting mode 1 |
| 101 | Quadrature encoding counting mode 2 |
| 110 | Quadrature encoding counting mode 3 |
| 111 | Gated function |

#### 13.3.6.1 Use internal clock (master mode)

When the SMS bit field of the ATIM master-slave mode control register ATIM_MSCR is 0x0, the timer slave mode is disabled. The counting clock source is selected by the CT bit field of the control register ATIM_CR. Write 1 to the ATIM_CR.EN bit field, and the timer starts counting immediately.

#### 13.3.6.2 Reset function (slave mode)

When the SMS bit field of the ATIM master-slave mode control register ATIM_MSCR is 0x1, the reset of the ATIM counter is controlled by the TS signal. When the TS signal is active, the ATIM counter CNT and the prescaler counter are initialized. If the URS bit field of the control register ATIM_CR is 0, an update event UEV is generated, the update event interrupt flag bit ATIM_ISR.UIF is set, and ATIM_ARR, ATIM_CHxCCRy are updated to its cache register at the same time.

The source of the TS signal is controlled by the TS bit of the master-slave mode control register ATIM_MSCR, as shown in the following table:

Table 13-13 TS signal sources

| ATIM_MSCR.TS | TS signal source |
|---|---|
| 000 | ETR filtered phase-selected signal ETFP |
| 001 | Internal interconnect signal ITR |
| 101 | Input signal of port CH1A |
| 110 | Filtered and phase-selected signal IAFP at port CH1A |
| 111 | Filtered and phase-selected signal IBFP at port CH1B |

The source of the ETR input signal can be the external ATIM_ETR pin or other on-chip peripherals, please refer to section *13.3.8 On-chip peripheral interconnect ETR*. When ETR is selected as the TS signal source, the filter control can be performed through ATIM_FLTR.FLTET, and the ETR input phase can be selected through ATIM_FLTR.ETP.

The source of the internal interconnection signal ITR is the overflow signal of BTIM and GTIM, which can be selected by the timer ITR source configuration register SYSCTRL_TIMITR, see *Table 13-14 ATIM internal cascade ITR*.

The external input ports CH1A and CH1B both have filtering and phase selection functions. CH1A and CH1B respectively use the OCM1AFLT1A and OCM1BFLT1B bit fields of the output control/input filtering register ATIM_FLTR for filtering control, and the CCP1A and CCP1B bit fields for phase control. The input CH1A and CH1B pins of ATIM need to be configured with multiplexing functions. For the supported pins, see *Table 13-3 ATIM compare/capture channel pins*.

#### 13.3.6.3 Trigger mode (slave mode)

The SMS bit field of the ATIM master-slave mode control register ATIM_MSCR is 0x2, which is configured as a trigger mode, and can be triggered by software or hardware to start the counter to start counting.

- Software trigger

  Set the TG bit field of the control register ATIM_CR to 1, start the counter immediately, and the TG bit is automatically cleared.

- Hardware trigger

  When the trigger signal TS is valid, the counter is started. For TS signal source and phase selection, please refer to section *13.3.6.2 Reset function (slave mode)*.

When a trigger is generated, the trigger interrupt flag bit ATIM_ISR.TIF will be set by hardware. If the trigger interrupt enable bit ATIM_CR.TIE is set to 1, an interrupt request will be generated, and ATIM_ICR.TIF is set to 0 to clear the interrupt flag bit.

*Caution:*

*If you use falling edge triggering, you need to select the trigger polarity first, and then select the trigger mode to avoid false triggering.*

#### 13.3.6.4 External clock mode (slave mode)

The SMS bit of the master-slave mode control register ATIM_MSCR is 0x3, and the ATIM counter counts on each valid edge of the TS signal. For TS signal source and phase selection, please refer to section *13.3.6.2 Reset function (slave mode)*.

*Caution:*

*In this mode, ATIM_CR.EN must be set to 1 to enable the timer.*

#### 13.3.6.5 Quadrature encoding mode (slave mode)

The SMS bits of the master-slave mode control register ATIM_MSCR are 0x4~0x6. The counter counts up or down according to the transition sequence of the channel CH1A and CH1B input signals. For details, please refer to section *13.3.4 Quadrature encoding counting*.

#### 13.3.6.6 Gated function (slave mode)

The SMS bit field of the master-slave mode control register ATIM_MSCR is 0x7, and the ATIM is configured in gated mode. When the gate control signal TS is valid and ATIM_CR.EN is 1, the counter starts counting; when the gate control signal TS is invalid or ATIM_CR.EN is 0, the counter counting is suspended.

For TS signal source and phase selection, please refer to section *13.3.6.2 Reset function (slave mode)*.

### 13.3.7 Internal cascade ITR

All timers (ATIM, GTIM, BTIM) of CW32F003 can be used in cascade, and the overflow signal of the previous timer is used as the ITR input of the next timer.

The ITR source of ATIM is the overflow signal of BTIM and GTIM, which can be selected by the timer ITR source configuration register SYSCTRL_TIMITR, as shown in the following table:

Table 13-14 ATIM internal cascade ITR

| SYSCTRL_TIMITR.ATIMITR | ITR signal sources of ATIM |
|---|---|
| 000 | Overflow signal of BTIM1 |
| 001 | Overflow signal of BTIM2 |
| 010 | Overflow signal of BTIM3 |
| 011 | Overflow signal of GTIM |

ATIM can also serve as an ITR source for BTIM and GTIM. Through the MMS bit field of the master-slave mode control register ATIM_MSCR, the master mode output of the ATIM can be selected and connected to the ITR input of other timers. As shown in the following figure:

Figure 13-34 ATIM internal cascade ITR

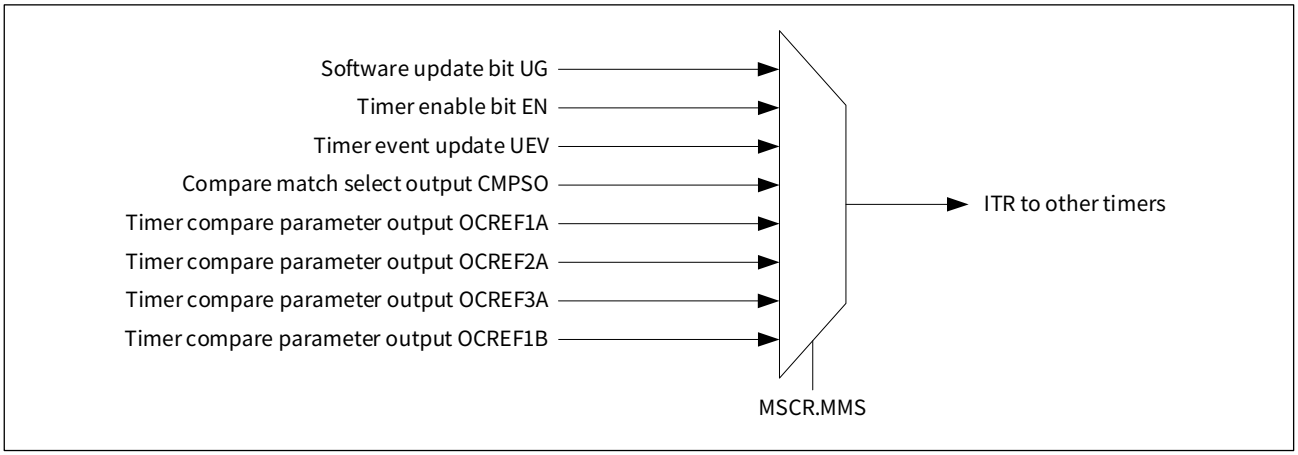


Table 13-15 ATIM master mode output selection

| ATIM_MSCR.MMS | ATIM master mode output |
|---|---|
| 000 | Software update UG, write ATIM_CR.UG |
| 001 | Timer enable signal EN (ATIM_CR.EN) |
| 010 | Timer update event UEV |
| 011 | Compare match select output CMPSO |
| 100 | Timer channel CH1A compare reference output OCREF1A |
| 101 | Timer channel CH2A compare reference output OCREF2A |
| 110 | Timer channel CH3A compare reference output OCREF3A |
| 111 | Timer channel CH1B compare reference output OCREF1B |

### 13.3.8 On-chip peripheral interconnect ETR

The source of the ETR signal of ATIM can be the ATIM_ETR pin or other on-chip peripherals, which can be selected through the advanced timer ETR source configuration register SYSCTRL_ATIMETR.

When SYSCTRL_ATIMETR.ATIMETR is 0x00, the ETR signal comes from the ATIM_ETR pin input; when ATIMETR is 0x01~0x06, the ETR signal comes from other on-chip peripherals to realize the interconnection between on-chip and peripherals, as shown in the following table:

Table 13-16 ETR sources of ATIM

| SYSCTRL_ATIMETR.ATIMETR | ETR sources of ATIM |
|---|---|
| 000 | Pin PA04 (AFR=0x07) |
| 001 | RXD signal of UART1 |
| 010 | RXD signal of UART2 |
| 100 | Compare output signal of VC1 |
| 101 | Compare output signal of VC2 |
| 110 | LVD output signal |

## 13.4 Debugging support

ATIM supports stop or continue counting in debug mode, which is set by the ATIM bit field of the debug status timer control register SYSCTRL_DEBUG.

Set SYSCTRL_DEBUG.ATIM to 1, then suspend the counter count of ATIM in the debug state;

If SYSCTRL_DEBUG.ATIM is set to 0, the counter of ATIM continues to count in the debug state.

## 13.5 Programming examples

### 13.5.1 Input capture

The following example shows how to capture the counter value into the ATIM_CH1CCRA register on the rising edge of the CH1A input signal, the steps are as follows:

Step 1: Set the SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the MODE bit of the register ATIM_CR to 0x02, so that the counting mode of the ATIM is edge-aligned mode, so that the ATIM can work in the input capture mode;

Step 3: Set the CSA bit of the register ATIM_CH1CR to 0x01, and configure CH1A to be the input capture mode;

Step 4: Set the BKSA bit of the register ATIM_CH1CR to 0x01, and configure the CH1A rising edge capture;

Step 5: According to the characteristics of the input signal, set the FLT1A bit of the ATIM_FLTR register, and configure the input filter to the required bandwidth;

Step 6: Set the relevant bits of the ATIM_ICR register to 0, and clear the flag to prevent the previous setting from affecting the result of this operation;

Step 7: According to the characteristics of the input signal, set the value of the ATIM_ARR register, and select the appropriate auto-reload value;

Step 8: Set the EN bit of the ATIM_CR register to 0x01 to allow the capture of the counter value into the capture register;

Step 9: If required, enable the relevant interrupt request by setting the CIEA bit in the ATIM_CH1CR register.

### 13.5.2 PWM input

The following example shows how to measure the pulse width and period of the PWM signal input from the CH3B channel. The steps are as follows:

Step 1: Set the SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the MODE bit of the register ATIM_CR to 0x02, so that the counting mode of the ATIM is edge-aligned mode, so that the ATIM can work in the input capture mode;

Step 3: Set the CSB bit of register ATIM_CH3CR to 0x01, and configure CH3B as input capture mode;

Step 4: Set the BKSB bit of the register ATIM_CH3CR to 0x11, and configure CH3B rising/falling edge capture;

Step 5: According to the characteristics of the input signal, set the FLT3B bit of the ATIM_FLTR register, and configure the input filter to the required bandwidth;

Step 6: Set the relevant bits of the ATIM_ICR register to 0, and clear the flag to prevent the previous setting from affecting the result of this operation;

Step 7: According to the characteristics of the input signal, set the value of the ATIM_ARR register, and select the appropriate auto-reload value;

Step 8: Set the EN bit of the ATIM_CR register to 0x01 to allow the capture of the counter value into the capture register;

Step 9: If required, enable the relevant interrupt request by setting the CIEA bit in the ATIM_CH3CR register;

Step 10: Wait for the capture to occur three times, read the value captured in the ATIM_CH3CCRB register after each capture occurs, and clear the capture interrupt flag bit;

Step 11: According to the value of the ATIM_CH3CCRB register read three times, it can be obtained that the period of PWM is the third value-first value, and the pulse width of PWM is the third value-first value or the second value- 1st value.

### 13.5.3 Output compare function

Configuration steps for output compare mode:

Step 1: Select the counter clock (internal, external, prescaler);

Step 2: Write the corresponding data into the ATIM_ARR and ATIM_CHxCCRy registers;

Step 3: If an interrupt request is to be generated, set the CIEy bit in the ATIM_CHxCR register;

Step 4: Select the output mode, if you need to flip the output, set ATIM_FLTR.OCMxy=0x4;

Step 5: Set the CSy bit in the ATIM_CHxCR register to use the corresponding channel as a compare output;

Step 6: Clear the relevant interrupt flag;

Step 7: Set the CCPxy bit in the ATIM_FLTR register to select the desired output phase;

Step 8: Set the EN bit in the ATIM_CR register to start the counter.

### 13.5.4 Complementary PWM output

The steps to set up a center-aligned complementary PWM output with dead-time are as follows:

Step 1: Set the SYSCTRL_APBEN2.ATIM bit to 1 and open the ATIM module;

Step 2: Set the register ATIM_CR.MODE to 0x3, the timer works in center-aligned mode;

Step 3: Set the register ATIM_CR.COMP to 0x01, and select the complementary PWM output;

Step 4: Set the register ATIM_CR.PWM2S to 0x0 and use the two-point compare function;

Step 5: Set ATIM_ARR according to the period of PWM;

Step 6: Set the initial value of the counter ATIM_CNT (the initial value must be less than the value of ATIM_ARR);

Step 7: Set the compare registers ATIM_CHxCCRA and ATIM_CHxCCRB according to the PWM pulse width;

Step 8: Set ATIM_FLTR.OCMxBFLTxB and ATIM_FLTR.OCMxAFLTxA to 0x6 or 0x7 to make the compare output mode PWM mode 1 or PWM mode 2;

Step 9: Set the relevant bits of the ATIM_ICR register and clear the relevant interrupt flag;

Step 10: If necessary, set the relevant interrupt enable;

Step 11: Set ATIM_FLTR.CCPxA and ATIM_FLTR.CCPxB to 0, no inverting output is required;

Step 12: Set ATIM_DTR.DTEN to 0x01 to enable dead-time insertion;

Step 13: Set ATIM_DTR.DTR, and configure the inserted dead-time duration;

Step 14: Set ATIM_DTR.MOE to 0x01 to enable PWM output;

Step 15: Set the register ATIM_CR.EN to 0x01 to enable the timer.

### 13.5.5 Trigger mode

The following is an example of the counter starting to count up on the rising edge of the CH1B input:

Step 1: Set ATIM_FLTR.FLT1B to 0x00, and configure the bandwidth of the input filter (in this example, no filtering is required);

Step 2: Set ATIM_FLTR.CCP1B to 0, select polarity (rising edge);

Step 3: Set the SMS bit of ATIM_MSCR to 0x2, and configure the timer as the trigger mode of slave mode;

Step 4: Set the TS bit of ATIM_MSCR to 0x7, and select CH1B as the input trigger source.

*Caution:*

*If you use falling edge triggering, you need to select the trigger polarity first, and then select the trigger mode to avoid false triggering.*

### 13.5.6 Gated Mode

The following is an example of the counter counting up when the CH1A input is low:

Step 1: Set ATIM_FLTR.FLT1A to 0x00, and configure the bandwidth of the input filter (in this example, no filtering is required);

Step 2: Set ATIM_FLTR.CCP1A to 0x1, select polarity (low level);

Step 3: Set the SMS bit of ATIM_MSCR to 0x7, and configure the timer to be the gated mode of the slave mode;

Step 4: Set the TS bit of ATIM_MSCR to 0x6, and select CH1A as the input trigger source;

Step 5: Set EN of ATIM_CR to 0x1 to enable counting. When the input level of CH1A is low, the counter starts to count the internal clock, and when the input level becomes high, the counter stops counting.

### 13.5.7 Internal cascade ITR

In this example, the enable bit EN of ATIM is used to trigger the start of BTIMx to realize the synchronous start of ATIM and BTIMx. The operation steps are as follows:

Step 1: Set the SYSCTRL_APBEN2.BTIM bit to 1, and open the configuration clock of the BTIM1-3 module;

*Caution:*

*BTIM1-3 share the SYSCTRL_APBEN2.BTIM bit.*

Step 2: Configure BTImx_BCR.TRS to 1, and select the trigger source from the internal cascade signal;

Step 3: Configure BTMx_BCR.PRS and select the prescaler ratio;

Step 4: Configure BTIMx_BCR.ONESHOT to select single or continuous counting mode. When configured as 0, it is continuous counting mode; when configured as 1, it is single counting;

Step 5: If you want the trigger event to trigger an interrupt, configure BTIMx_IER.IT to 1;

Step 6: If you want the ARR overflow event to trigger an interrupt, configure BTIMx_IER.OV to 1;

Step 7: Configure the BTIMx_ARR register and select the BTIMx count overflow time;

Step 8: Configure BTIMx_BCR.MODE to 0x10, select BTIMx to work in trigger start mode;

Step 9: Set the system control register SYSCTRL_APBEN2.ATIM bit to 1, and open the ATIM module;

Step 10: Configure ATIM_MSCR.MMS as 0x01, select ATIM main mode to output CTEN;

Step 11: Set the register ATIM_CR.MODE to 0x2, the timer works in edge-aligned mode;

Step 12: Set the register ATIM_CR.DIR to 0x0, and the edge-aligned counting direction is upcounting;

Step 13: Set ATIM_ARR according to the period;

Step 14: Set the register ATIM_CR.EN to 0x1, which will start ATIM and BTIMx synchronously.

## 13.6 List of registers

ATIM base address: ATIM_BASE = 0x4001 2C00

Table 13-17 List of ATIM registers

| Register name | Register address | Register description |
|---|---|---|
| ATIM_ARR | ATIM_BASE + 0x00 | Reload register |
| ATIM_CNT | ATIM_BASE + 0x04 | Count register |
| ATIM_CR | ATIM_BASE + 0x0C | Control register |
| ATIM_ISR | ATIM_BASE + 0x10 | Interrupt flag register |
| ATIM_ICR | ATIM_BASE + 0x14 | Interrupt flag clear register |
| ATIM_MSCR | ATIM_BASE + 0x18 | Master-slave mode control register |
| ATIM_FLTR | ATIM_BASE + 0x1C | Output control/Input filter register |
| ATIM_TRIG | ATIM_BASE + 0x20 | Trigger ADC control register |
| ATIM_CH1CR | ATIM_BASE + 0x24 | Channel 1 control register |
| ATIM_CH2CR | ATIM_BASE + 0x28 | Channel 2 control register |
| ATIM_CH3CR | ATIM_BASE + 0x2C | Channel 3 control register |
| ATIM_CH4CR | ATIM_BASE + 0x58 | Channel 4 control register |
| ATIM_DTR | ATIM_BASE + 0x30 | Dead-time register |
| ATIM_RCR | ATIM_BASE + 0x34 | Repeat count register |
| ATIM_CH1CCRA | ATIM_BASE + 0x3C | Channel 1 compare/capture register A |
| ATIM_CH1CCRB | ATIM_BASE + 0x40 | Channel 1 compare/capture register B |
| ATIM_CH2CCRA | ATIM_BASE + 0x44 | Channel 2 compare/capture register A |
| ATIM_CH2CCRB | ATIM_BASE + 0x48 | Channel 2 compare/capture register B |
| ATIM_CH3CCRA | ATIM_BASE + 0x4C | Channel 3 compare/capture register A |
| ATIM_CH3CCRB | ATIM_BASE + 0x50 | Channel 3 compare/capture register B |
| ATIM_CH4CCR | ATIM_BASE + 0x54 | Channel 4 compare/capture register |

## 13.7 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 13.7.1 ATIM_CR control register

Address offset: 0x0C Reset value: 0x0060 0008

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:30 | RFU | - | Reserved bits, please keep the default value |
| 29 | UNDE | RW | Underflow interrupt enabled<br>0: Disabled<br>1: Enabled |
| 28 | OVE | RW | Overflow interrupt enabled<br>0: Disabled<br>1: Enabled |
| 27 | DIR | RW/RO | Counting direction, only writable in edge-aligned mode. Read only in other modes, write invalid<br>0: upcounting<br>1: downcounting<br>Caution: DIR is automatically cleared to 0 when switching from other modes to center-aligned mode. Software event update and slave mode externally triggered reset mode DIR is automatically cleared. |
| 26 | BG | RW | Software brake, automatic reset<br>Write 1 to generate software brake<br>Write 0 invalid |
| 25 | UG | RW | Software update, auto reset<br>Write 1 to generate a software update<br>Write 0 invalid<br>Caution: After software update, initialize the counter and update ATIM_ARR, ATIM_CHxCCRy to its cache register (cache enable), the prescaler counter will also be cleared. |
| 24 | TG | RW | Triggered by software, automatically cleared. It needs to be triggered in trigger mode (SMS=2) and MODE=2/3.<br>Write 1 to generate software trigger<br>Write 0 invalid |
| 23 | OCCE | RW | OCREF clear enabled<br>0: OCREF output is not affected by OCREF_CLR<br>1: OCREF_CLR high level signal can clear OCREF output |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 22:21 | CISA | RW | In center-aligned mode, compare match interrupt settings for all A channels<br>00: No interrupt<br>01: Generate interrupt on match while counting up<br>10: Generate interrupt on match while counting down<br>11: Interrupt on match when up and down counting<br>Caution: In center-aligned mode, the compare match interrupt of the B channel is set in the CISB bit field of the ATIM_CHxCR register |
| 20 | BIE | RW | Brake interrupt enabled<br>0: Disabled<br>1: Enabled |
| 19 | TIE | RW | Trigger interrupt enabled<br>0: Disabled<br>1: Enabled |
| 18 | RFU | - | Reserved bits, please keep the default value |
| 17 | URS | RW | Update source<br>0: Overflow/underflow/software update UG/slave mode reset<br>1: Overflow/underflow |
| 16 | OCCS | RW | OCREF clear source selection<br>0: Voltage comparator VC output, VC selection is set in VCx_CR1 register<br>1: The ETR port filters the phase-selected signal<br>Caution: When OCCE is valid, the OCREF_CLR high level signal can clear the compare output signal of OCREF to zero (valid when OCMxyFLTxy>1), and continue to compare output after the next update event UEV. |
| 15 | RFU | - | Reserved bits, please keep the default value |
| 14 | ONESHOT | RW | Single trigger mode selection<br>0: Loop count<br>1: Timer stops after event update occurs |
| 13:12 | MODE | RW | Working mode<br>00: reserve 01: reserve<br>10: edge-aligned mode 11: center-aligned mode |
| 11 | RFU | - | Reserved bits, please keep the default value |
| 10 | UIE | RW | UIE update interrupt enabled<br>0: Disabled<br>1: Enabled |
| 9:8 | RFU | - | Reserved bits, please keep the default value |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 7 | BUFPEN | RW | Reload register cache enabled<br>0: The cache is invalid, and the period value is affected immediately after writing<br>1: The cache is enabled, and the period value will not be affected until the next event is updated after writing |
| 6:4 | PRS | RW | Internal clock PCLK frequency division selection<br>000: 1 001: 2 010: 4 011: 8<br>100: 16 101: 32 110: 64 111: 256 |
| 3 | PWM2S | RW | OCREFA two-point compare selection (default is 1)<br>0: Two-point compare enabled, use CHxCCRA, CHxCCRB compare to control OCREFA output<br>1: Single-point compare enabled, use only CHxCCRA compare to control OCREFA output<br>Caution: OCREFB is not affected, still use CHxCCRB to control OCREFB output |
| 2 | CT | RW | Count clock selection<br>0: internal count clock PCLK<br>1: external count clock ETR |
| 1 | COMP | RW | PWM complementary output mode selection<br>0: independent PWM output<br>1: complementary PWM output |
| 0 | EN | RW | Timer enabled<br>0: Disabled<br>1: Enabled<br>Caution: It can be enabled by external trigger, and this bit is automatically cleared when the ONESHOT mode ends. |

### 13.7.2 ATIM_ARR reload register

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | ARR | RW | Reload register/period register with cache function<br>When the counter is not enabled or the cache is not enabled, the cache register can be updated immediately |

### 13.7.3 ATIM_CNT count register

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CNT | RW | Reload timer count register |

*Caution:*

*When using the PWM compare output, the initialized CNT value needs to be smaller than the ARR value.*

### 13.7.4 ATIM_ISR interrupt flag register

Address offset: 0x10 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:19 | RFU | - | Reserved bits, please keep the default value |
| 18 | C4AF | RO | Channel CH4 compare match flag |
| 17 | UNDF | RO | Underflow interrupt flag |
| 16 | OVF | RO | Overflow interrupt flag |
| 15 | TIF | RO | Trigger interrupt flag |
| 14 | BIF | RO | Brake interruption flag |
| 13 | C3BE | RO | Channel CH3B capture data loss flag<br>0: no data loss<br>1: data loss |
| 12 | C2BE | RO | Channel CH2B capture data loss flag<br>0: no data loss<br>1: data loss |
| 11 | C1BE | RO | Channel CH1B capture data loss flag<br>0: no data loss<br>1: data loss |
| 10 | C3AE | RO | Channel CH3A capture data loss flag<br>0: no data loss<br>1: data loss |
| 9 | C2AE | RO | Channel CH2A capture data loss flag<br>0: no data loss<br>1: data loss |
| 8 | C1AE | RO | Channel CH1A capture data loss flag<br>0: no data loss<br>1: data loss |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 7 | C3BF | RO | Capture/compare match flag occurred on channel CH3B<br>0: not occur<br>1: occur |
| 6 | C2BF | RO | Capture/compare match flag occurred on channel CH2B<br>0: not occur<br>1: occur |
| 5 | C1BF | RO | Capture/compare match flag occurred on channel CH1B<br>0: not occur<br>1: occur |
| 4 | C3AF | RO | Capture/compare match flag occurred on channel CH3A<br>0: not occur<br>1: occur |
| 3 | C2AF | RO | Capture/compare match flag occurred on channel CH2A<br>0: not occur<br>1: occur |
| 2 | C1AF | RO | Capture/compare match flag occurred on channel CH1A<br>0: not occur<br>1: occur |
| 1 | RFU | - | Reserved bits, please keep the default value |
| 0 | UIF | RO | Event update interrupt flag<br>0: not occur<br>1: occur |

### 13.7.5 ATIM_ICR interrupt flag clear register

Address offset: 0x14 Reset value: 0x0007 FFFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:19 | RFU | - | Reserved bits, please keep the default value |
| 18 | C4AF | R1W0 | Channel CH4 compare match flag is cleared, write 0 to clear |
| 17 | UNDF | R1W0 | Underflow interrupt flag is cleared, write 0 to clear |
| 16 | OVF | R1W0 | Overflow interrupt flag is cleared, write 0 to clear |
| 15 | TIF | R1W0 | Trigger interrupt flag to clear, write 0 to clear |
| 14 | BIF | R1W0 | Brake interrupt flag is cleared, write 0 to clear |
| 13 | C3BE | R1W0 | Channel CH3B capture data loss flag clear, write 0 to clear |
| 12 | C2BE | R1W0 | Channel CH2B capture data loss flag clear, write 0 to clear |
| 11 | C1BE | R1W0 | Channel CH1B capture data loss flag clear, write 0 to clear |
| 10 | C3AE | R1W0 | Channel CH3A capture data loss flag clear, write 0 to clear |
| 9 | C2AE | R1W0 | Channel CH2A capture data loss flag clear, write 0 to clear |
| 8 | C1AE | R1W0 | Channel CH1A capture data loss flag clear, write 0 to clear |
| 7 | C3BF | R1W0 | Channel CH3B capture/compare match flag clear, write 0 to clear |
| 6 | C2BF | R1W0 | Channel CH2B capture/compare match flag clear, write 0 to clear |
| 5 | C1BF | R1W0 | Channel CH1B capture/compare match flag clear, write 0 to clear |
| 4 | C3AF | R1W0 | Channel CH3A capture/compare match flag clear, write 0 to clear |
| 3 | C2AF | R1W0 | Channel CH2A capture/compare match flag clear, write 0 to clear |
| 2 | C1AF | R1W0 | Channel CH1A capture/compare match flag clear, write 0 to clear |
| 1 | RFU | - | Reserved bits, please keep the default value |
| 0 | UIF | R1W0 | Event update interrupt flag clear, write 0 to clear |

### 13.7.6 ATIM_MSCR master-slave mode control register

Address offset: 0x18 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:13 | RFU | - | Reserved bits, please keep the default value |
| 12 | IB1S | RW | CH1B input selection<br>0: CH1B<br>1: Internal trigger TS signal |
| 11 | IA1S | RW | CH1A input selection<br>0: CH1A<br>1: Exclusive OR of CH1A, CH2A, CH3A<br>Caution: After setting to 1, any port change will cause the input to change |
| 10:8 | SMS | RW | Select from mode function<br>000: Use internal clock<br>001: Reset function<br>010: Trigger mode<br>011: External clock mode<br>100: Quadrature encoding counting mode 1<br>101: Quadrature encoding counting mode 2<br>110: Quadrature encoding counting mode 3<br>111: Gated function |
| 7:5 | TS | RW | Trigger selection<br>000: Filtered phase-selected signal ETFP at port ETR<br>001: Internal interconnect signal ITR<br>101: Edge signal of port CH1A<br>110: Filtered phase-selected signal IAFP for port CH1A<br>111: Filtered phase-selected signal IBFP for port CH1B |
| 4:3 | RFU | - | Reserved bits, please keep the default value |
| 2:0 | MMS | RW | Master mode output selection for internal cascade connection to ITR of other timers<br>000: Software update UG, write CR.UG<br>001: Timer enable signal EN<br>010: Timer update event UEV<br>011: Compare match selects output CMPSO<br>100: Timer channel CH1A compare reference output OCREF1A<br>101: Timer channel CH2A compare reference output OCREF2A<br>110: Timer channel CH3A compare reference output OCREF3A<br>111: Timer channel CH1B compare reference output OCREF1B |

### 13.7.7 ATIM_FLTR output control/input filter register

Address offset: 0x1C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31 | ETP | RW | ETR input phase selection<br>0: in phase<br>1: reverse input |
| 30:28 | FLTET | RW | ETR filter settings<br>0xx: invalid filtering<br>100: PCLK 3 consecutive valid<br>101: PCLK /4 3 consecutive valid<br>110: PCLK /16 3 consecutive valid<br>111: PCLK /64 3 consecutive valid |
| 27 | BKP | RW | Brake BK input phase selection<br>0: in phase<br>1: reverse input |
| 26:24 | FLTBK | RW | Brake input filter settings<br>0xx: invalid filtering<br>100: PCLK 3 consecutive valid<br>101: PCLK /4 3 consecutive valid<br>110: PCLK /16 3 consecutive valid<br>111: PCLK /64 3 consecutive valid |
| 23 | CCP3B | RW | Compare function: CH3B channel compare output phase control<br>0: normal output<br>1: reverse output |
| 22:20 | OCM3B<br>FLT3B | RW | Compare function: CH3B channel compare control, refer to OCM1BFLT1B<br>Capture function: CH3B input channel filter setting, refer to FLTBK |
| 19 | CCP3A | RW | Compare function: CH3A channel compare output phase control<br>0: normal output<br>1: reverse output |
| 18:16 | OCM3A<br>FLT3A | RW | Compare function: CH3B channel compare control, refer to OCM1BFLT1B<br>Capture function: CH3A input channel filter setting, refer to FLTBK |
| 15 | CCP2B | RW | Compare function: CH2B channel compare output phase control<br>0: normal output<br>1: reverse output |
| 14: 12 | OCM2B<br>FLT2B | RW | Compare function: CH2B channel compare control, refer to OCM1BFLT1B<br>Capture function: CH2B input channel filter setting, refer to FLTBK |
| 11 | CCP2A | RW | Compare function: CH2A channel compare output phase control<br>0: normal output<br>1: reverse output |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 10:8 | OCM2A<br>FLT2A | RW | Compare function: CH2A channel compare control, refer to OCM1BFLT1B<br>Capture function: CH2A input channel filter setting, refer to FLTBK |
| 7 | CCP1B | RW | Compare function: CH1B channel compare output phase control<br>0: normal output<br>1: reverse output<br>CH1B port input polarity control in slave mode:<br>0: normal output<br>1: reverse output |
| 6:4 | OCM1B<br>FLT1B | RW | Compare function: CH1B channel compare control<br>000: Force output low<br>001: Force output high<br>010: Set to 0 on compare match<br>011: Set to 1 on compare match<br>100: Flip output on compare match<br>101: Output a high level for one count period on a compare match<br>110: PWM mode 1<br>Single-point comparison:<br>CNT<CHxCCRy outputs high level when counting up;<br>CNT>CHxCCRy outputs a low level when counting down.<br>Two-point comparison :<br>1) CHxCCRA<CNT ⩽ CHxCCRB outputs low level when edge-aligned upcounting;<br>2) CHxCCRA<CNT ⩽ CHxCCRB outputs high level when edge-aligned downcounting;<br>3) CNT<CHxCCRA outputs high level when center-aligned upcounting, CNT>CHxCCRB outputs low level when center-aligned downcounting.<br>111: PWM mode 2<br>Single-point comparison:<br>CNT<CHxCCRy outputs low level when counting up;<br>CNT>CHxCCRy outputs a low level when counting down.<br>Two-point comparison :<br>1) CHxCCRA ⩽ CNT<CHxCCRB outputs high level when edge-aligned upcounting;<br>2) CHxCCRA ⩽ CNT<CHxCCRB outputs low level when edge-aligned downcounting;<br>3) CNT<CHxCCRA outputs low level when center-aligned downcounting, CNT>CHxCCRB outputs high level when center-aligned upcounting.<br>Capture function: CH1B input channel filter setting, refer to FLTBK |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 3 | CCP1A | RW | Compare function: CH1A channel compare output phase control<br>0: normal output<br>1: reverse output<br>CH1A port input polarity control in slave mode:<br>0: normal input<br>1: reverse input |
| 2:0 | OCM1A<br>FLT1A | RW | Compare function: A channel compare control, refer to OCM1BFLT1B<br>Capture function: CH1A input channel filter setting, refer to FLTBK |

### 13.7.8 ATIM_TRIG trigger ADC control register

Address offset: 0x20 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | ADTE | RW | ADC trigger global control enabled<br>0: Disabled<br>1: Enabled |
| 6 | CM3BE | RW | Channel CH3B compare match trigger ADC enabled<br>0: Disabled<br>1: Enabled |
| 5 | CM2BE | RW | Channel CH2B compare match trigger ADC enabled<br>0: Disabled<br>1: Enabled |
| 4 | CM1BE | RW | Channel CH1B compare match trigger ADC enabled<br>0: Disabled<br>1: Enabled |
| 3 | CM3AE | RW | Channel CH3A compare match trigger ADC enabled<br>0: Disabled<br>1: Enabled |
| 2 | CM2AE | RW | Channel CH2A compare match trigger ADC enabled<br>0: Disabled<br>1: Enabled |
| 1 | CM1AE | RW | Channel CH1A compare match trigger ADC enabled<br>0: Disabled<br>1: Enabled |
| 0 | UEVE | RW | Event update triggers ADC enabled<br>0: Disabled<br>1: Enabled |

### 13.7.9 ATIM_DTR dead-time register

Address offset: 0x30 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:15 | RFU | - | Reserved bits, please keep the default value |
| 14 | VCE | RW | VC compare output brake enabled<br>0: Disabled<br>1: Enabled |
| 13 | SAFEEN | RW | Safety brake enabled (osc fail,brown down,lockup)<br>0: Disabled<br>1: Enabled |
| 12 | MOE | RW | Output compare enabled<br>0: Disabled<br>1: Enabled |
| 11 | AOE | RW | PWM output auto-enabled<br>0: Disabled<br>1: Enabled |
| 10 | BKE | RW | Brake enabled<br>0: Disabled<br>1: Enabled |
| 9 | DTEN | RW | Dead-time control enabled<br>0: Disabled<br>1: Enabled<br>Caution: Disable dead-time control when independent PWM output is enabled. |
| 8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | DTR | RW | Dead-time register<br>DTR[7] =0 T=DTR[6:0]+2 2 ~ 129 step=1<br>DTR[7:6] =10 T={DTR[5:0]+64}×2 +2 130 ~ 256 step=2<br>DTR[7:5] =110 T={DTR[4:0]+32}×8 +2 258 ~ 506 step=8<br>DTR[7:5] =111 T={DTR[4:0]+32}×16 +2 514 ~ 1010 step=16 |

### 13.7.10 ATIM_RCR repeat count register

Address offset: 0x34 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | UD | RW | Repeat count controlled underflow shielding<br>0: Repeat counter count when downcounting overflows<br>1: Downcounting overflow shielding |
| 8 | OV | RW | Repeat count control overflow shielding<br>0: Repeat counter count when upcounting overflows<br>1: Upcounting overflow shielding |
| 7:0 | RCR | RW | Repeat cycle count value<br>Set the RCR+1 cycle to be updated by the overflow controlled by [9:8], and the internal RCR_CNT is decremented by 1 when the counter overflows or underflows. When the count reaches zero, the RCR_CNT reloads the RCR value and generates an event to update the UEV signal. |

### 13.7.11 ATIM_CH1CR channel 1 control register

Address offset: 0x24 Reset value: 0x0000 3000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15 | CCGB | RW | The capture/compare B channel is triggered by software, and the hardware is automatically cleared.<br>Only interrupt is generated in compare mode<br>Generate an interrupt in capture mode and capture the value of the counter into the capture register |
| 14 | CCGA | RW | The capture/compare A channel is triggered by software, and the hardware is automatically cleared.<br>Only interrupt is generated in compare mode<br>Generate an interrupt in capture mode and capture the value of the counter into the capture register |
| 13:12 | CISB | RW | In center-aligned mode, the compare match interrupt setting of the B channel<br>00: No interrupt<br>01: Interrupt is generated on match when counting up<br>10: Interrupt is generated on match when counting down<br>11: Interrupt is generated on match when counting up and down<br>Caution: in the center-aligned mode, the compare match interrupt of all A channels is set in the CISA bit field of the ATIM_CR register |
| 11:10 | RFU | - | Reserved bits, please keep the default value |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 9 | CIEB | RW | B channel capture compare trigger interrupt enabled<br>0: Disabled<br>1: Enabled |
| 8 | CIEA | RW | A channel capture compare trigger interrupt enabled<br>0: Disabled<br>1: Enabled |
| 7 | BUFEB | RW | Compare function: compare capture register B cache enable control<br>0: Disabled<br>1: Enabled |
| 6 | BUFEA | RW | Compare function: compare capture register A cache enable control<br>0: Disabled<br>1: Enabled |
| 5 | CSB | RW | B channel capture/compare function selection<br>0: Compare mode<br>1: Capture mode |
| 4 | CSA | RW | A channel capture/compare function selection<br>0: Compare mode<br>1: Capture mode |
| 3:2 | BKSB | RW | Compare mode B channel compare function output brake level control<br>00: High resistance output<br>01: No effect on output<br>10: Force output low level<br>11: Force output high level<br>Capture mode B channel capture enabled<br>00: Disabled<br>01: Rising edge capture enabled<br>10: Falling edge capture enabled<br>11: Rising/falling edge capture enabled |
| 1:0 | BKSA | RW | Compare mode A channel compare function output brake level control<br>00: High resistance output<br>01: No effect on output<br>10: Force output low level<br>11: Force output high level<br>Capture mode A channel capture enable<br>00: Disabled<br>01: Rising edge capture enabled<br>10: Falling edge capture enabled<br>11: Rising/falling edge capture enabled |

### 13.7.12 ATIM_CH2CR channel 2 control register

Address offset: 0x28 Reset value: 0x0000 3000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | | | See ATIM_CH1CR for details |

### 13.7.13 ATIM_CH3CR channel 3 control register

Address offset: 0x2C Reset value: 0x0000 3000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | | | See ATIM_CH1CR |

### 13.7.14 ATIM_CH4CR channel 4 control register

Address offset: 0x58 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:6 | RFU | - | Reserved bits, please keep the default value |
| 5 | C4EN | RW | Channel compare enable |
| 4:3 | CIS | RW | In center-aligned mode, compare match interrupt setting of channel CH4<br>00: No interrupt<br>01: Interrupt on match when counting up<br>10: Interrupt on match when counting down<br>11: Interrupt on match when counting up and down |
| 2 | RFU | - | Reserved bits, please keep the default value |
| 1 | CIE | RW | Compare trigger interrupt enabled<br>0: Disabled<br>1: Enabled |
| 0 | BUFE | RW | Compare function: channel 4 compare capture register buffer enable control<br>0: Disabled<br>1: Enabled |

### 13.7.15 ATIM_CH1CCRA channel 1 compare capture register A

Address offset: 0x3C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR1A | RW | Compare capture register, compare with cache function |

### 13.7.16 ATIM_CH1CCRB channel 1 compare capture register B

Address offset: 0x40 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR1B | RW | Compare capture register, compare with cache function |

### 13.7.17 ATIM_CH2CCRA channel 2 compare capture register A

Address offset: 0x44 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR2A | RW | Compare capture register, compare with cache function |

### 13.7.18 ATIM_CH2CCRB channel 2 compare capture register B

Address offset: 0x48 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR2B | RW | Compare capture register, compare with cache function |

### 13.7.19 ATIM_CH3CCRA channel 3 compare capture register A

Address offset: 0x4C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR3A | RW | Compare capture register, compare with cache function |

### 13.7.20 ATIM_CH3CCRB channel 3 compare capture register B

Address offset: 0x50 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR3B | RW | Compare capture register, compare with cache function |

### 13.7.21 ATIM_CH4CCR channel 4 compare capture register

Address offset: 0x54 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | CCR4 | RW | Compare capture register, compare with cache function |

# 14 Independent watchdog timer (IWDT)

## 14.1 Overview

The CW32F003 integrates an independent watchdog timer (IWDT) and uses a dedicated internal RC clock source RC10K to avoid external factors during operation. Once the IWDT is started, the user needs to reload the IWDT counter within the specified time interval, otherwise the counter overflow will trigger a reset or generate an interrupt signal. After the IWDT is started, the counting can be stopped. The IWDT can be selected to keep running or suspend counting during DeepSleep mode.

The specially set key-value register can lock the key registers of the IWDT to prevent the registers from being accidentally modified.

## 14.2 Main Features

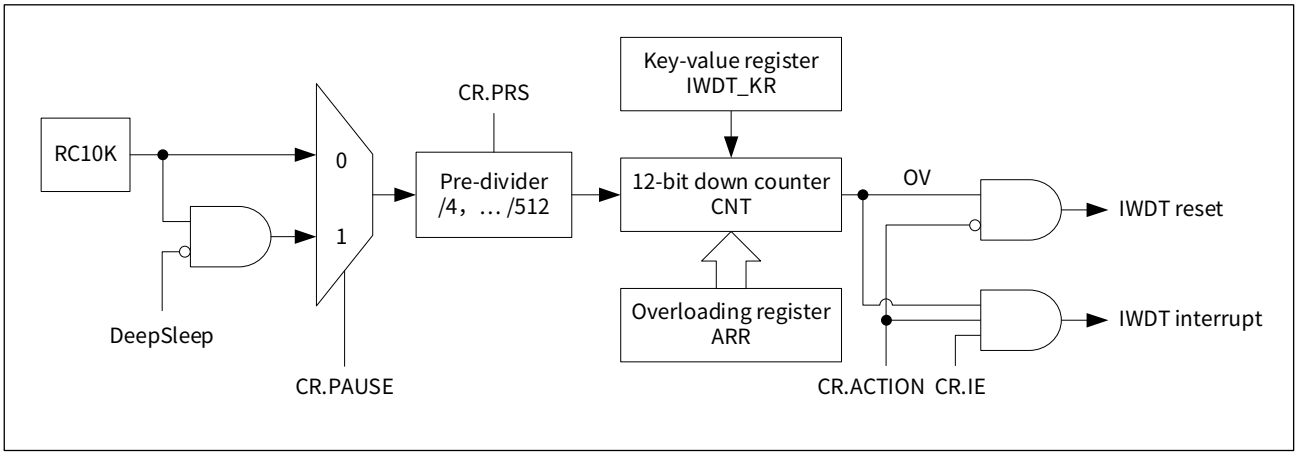
- 12-bit down counter
- Independent internal low-speed RC oscillator
- Programmable clock prescale period
- Overflow can trigger interrupt or reset
- Register protection lock function
- Counting can be suspended in DeepSleep mode

## 14.3 Functional description

### 14.3.1 Functional block diagram

The functional block diagram of IWDT is shown in the following figure:

Figure 14-1 IWDT functional block diagram




The IWDT is implemented by a 12-bit reloadable down-counter, and its counting clock source is the internal dedicated low-speed RC oscillator RC10K. The clock source RC10K signal can be prescaled by 4 to 512 through the PRS bit field of the control register IWDT_CR. When the IWDT counter overflows, it can optionally generate interrupt and reset signals.

### 14.3.2 Operating modes

To start the IWDT counter, you need to write 0xCCCC to the key value register IWDT_KR, and the counter starts to count down from 0xFFF.

Before the counter reduces to 0, writing 0xAAAA to the IWDT_KR register will trigger a counter reload, loading the ARR register value into the counter. When the IWDT counter value is reduced to 0, an overflow event will be generated. The overflow event can trigger the MCU reset or generate the IWDT interrupt signal, and trigger the counter reload at the same time. The ACTION and IE bit fields of the control register IWDT_CR are used to control whether interrupts and resets are generated when the watchdog overflows, as shown in the following table:

Table 14-1 Operating modes

| IWDT_CR.ACTION | IWDT_CR.IE | Action after IWDT overflows |
|---|---|---|
| 0 | X | Reset |
| 1 | 1 | No reset, only interrupt |
| 1 | 0 | No reset, no interrupt |

When the MCU enters the DeepSleep mode, the IWDT can choose to suspend the IWDT counting, so as to achieve lower overall system power consumption: the PAUSE bit field of the control register IWDT_CR is 0, and the IWDT timer is kept running in the DeepSleep mode; when it is 1, the counting is suspended, the IWDT automatically resumes counting when the MCU exits the DeepSleep mode.

### 14.3.3 Window options

Write a value smaller than the reload register IWDT_ARR to the window register IWDT_WINR to make the IWDT work in the window watchdog mode. The default value of the IWDT_WINR register is 0x0FFF, that is, the window option is closed by default.

Modify the value of the IWDT_WINR register will trigger a reload operation to load the ARR register value into the counter.

When using the IWDT window function, the user should perform a reload operation before the counter value is less than or equal to the IWDT_WINR window value and decrements to 0 to avoid reset or watchdog timer overflow. A reload operation when the watchdog counter value is greater than the window value will trigger a system reset.

### 14.3.4 Register lock function

Through the key value register IWDT_KR of the IWDT, the important registers IWDT_CR, IWDT_ARR, and IWDT_WINR of the IWDT can be configured to be locked or unlocked. The registers cannot be modified in the locked state.

Write 0x5555 to the IWDT_KR register to unlock the IWDT_CR, IWDT_ARR, and IWDT_WINR registers; write any other value to the IWDT_KR register to enable lock protection.

After the CW32F003 is powered on, the IWDT_CR, IWDT_ARR, and IWDT_WINR registers are locked by default, and the user needs to unlock them before modifying them.

### 14.3.5 Start refresh and stop

Configure the IWDT_KR register to start, refresh and stop the IWDT:

- Write 0xCCCC to start IWDT
- Write 0xAAAA, reload the counter, that is, refresh the IWDT
- Write 0x5A5A, 0xA5A5 sequentially, stop IWDT

*Caution:*

*The above operations will simultaneously activate the IWDT register lock protection.*

### 14.3.6 Status register

The status register IWDT_SR indicates the current running status or register update status of the IWDT, as shown in the following table:

Table 14-2 IWDT status indication

| IWDT_SR bit | Name | Description | 1 | 0 |
|---|---|---|---|---|
| 4 | RUN | Run flag | Running | Not running |
| 3 | OV | Overflow flag | Generate overflow | Not overflow |
| 5 | RELOAD | Counter reload flag | Reloading | Reload done |
| 2 | WINRF | WINR register update flag | Updating | Update completed |
| 1 | ARRF | ARR register update flag | Updating | Update completed |
| 0 | CRF | CR register update flag | Updating | Update completed |

To ensure the correct operation of the IWDT register, after updating IWDT_WINR, IWDT_ARR, and IWDT_CR, the user needs to check whether the WINRF, ARRF, and CRF flags are 0 to confirm whether the operation is completed.

To ensure the reload operation of the IWDT, the user should check whether the RELOAD flag is 0 after the reload operation.

### 14.3.7 Timing duration setting

The counting clock source of the IWDT is the dedicated low-speed clock RC10K (the clock frequency is about 10KHz, please refer to the datasheet for details). By controlling the PRS bit field of the IWDT_CR register, the frequency of the clock source RC10K signal can be divided, as shown in the following table:

Table 14-3 IWDT prescaler divider table

| IWDT_CR.PRS | Prescaler value |
|---|---|
| 000 | 4 |
| 001 | 8 |
| 010 | 16 |
| 011 | 32 |
| 100 | 64 |
| 101 | 128 |
| 110 | 256 |
| 111 | 512 |

Watchdog timer duration calculation formula:

$$T = ( 4 \times 2^{PRS} / f ) \times ( ARR + 1 )$$

Where, f is the frequency of the clock source RC10K, PRS is the prescaler divider, and ARR is the reload value.

Therefore, when the frequency of the clock source RC10K is 10000Hz, the longest and shortest timing ranges of the IWDT are:

$$\text{IWDT shortest timing} = ( 4 \times 2^{0} / 10000 ) \times ( 0x000 + 1 ) \approx 400\ \mu s$$

$$\text{IWDT longest timing} = ( 4 \times 2^{7} / 10000 ) \times ( 0xFFF + 1 ) \approx 209.7\ s$$

Example: When the frequency of the clock source RC10K is 10000Hz, set the prescale value to 64 and the reload value to 512, then:

$$\text{IWDT timing duration} = ( 4 \times 2^{4} / 10000 ) \times ( 512 + 1 ) = 3.28\ s$$

## 14.4 Programming examples

### 14.4.1 Configure IWDT as independent watchdog

Step 1: Set SYSCTRL_APBEN1.IWDT to 1 to enable the configuration clock of IWDT;

Step 2: Write 0xCCCC to the IWDT_KR register to start the IWDT;

*Caution:*

*The relevant registers can be modified only after the IWDT needs to be started.*

Step 3: Write 0x5555 to the IWDT_KR register to release the lock function of the IWDT register;

Step 4: Configure IWDT_CR, configure the prescale value of watchdog count clock and RC10K oscillator, action after overflow, whether to automatically suspend in DeepSleep mode;

Step 5: Configure IWDT_ARR and configure the overflow period of the watchdog;

Step 6: Wait for IWDT_SR.ARRF and IWDT_SR.CRF to become 0, wait for the reload value and CR register update to complete;

Step 7: Write 0xAAAA to the IWDT_KR register to load the ARR into the IWDT counter.

### 14.4.2 Configure IWDT as window watchdog

Step 1: Set SYSCTRL_APBEN1.IWDT to 1 to enable the configuration clock of IWDT;

Step 2: Write 0xCCCC to the IWDT_KR register to start the IWDT;

*Caution:*

*The relevant registers can be modified only after the IWDT needs to be started.*

Step 3: Write 0x5555 to the IWDT_KR register to release the lock function of the IWDT register;

Step 4: Configure IWDT_CR, configure the prescale value of watchdog count clock and RC10K oscillator, action after overflow, whether to automatically suspend in DeepSleep mode;

Step 5: Configure IWDT_ARR, configure the reload value of the watchdog;

Step 6: Configure IWDT_WINR, configure the window size, Caution that IWDT_WINR must be less than the overloaded value of IWDT_ARR;

Step 7: Wait for IWDT_SR.ARRF, IWDT_SR.WINRF and IWDT_SR.CRF to become 0, wait for the reload value, window register and CR register update to complete;

Step 8: Write 0xAAAA to the IWDT_KR register to load the ARR into the IWDT counter.

### 14.4.3 Refresh IWDT (feed dog operation)

Step 1: Write 0xAAAA to the IWDT_KR register to load ARR into the counter;

Step 2: Wait for IWDT_SR.RELOAD to become 0, and wait for the reload operation to complete.

## 14.5 List of registers

IWDT base address: IWDT_BASE = 0x4000 3000

Table 14-4 List of IWDT registers

| Register name | Register address | Register description |
|---|---|---|
| IWDT_KR | IWDT_BASE + 0x00 | Key-value register |
| IWDT_CR | IWDT_BASE + 0x04 | Control register |
| IWDT_ARR | IWDT_BASE + 0x08 | Reload register |
| IWDT_SR | IWDT_BASE + 0x0C | Status register |
| IWDT_WINR | IWDT_BASE + 0x10 | Window register |
| IWDT_CNT | IWDT_BASE + 0x24 | Count value register |

## 14.6 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 14.6.1 IWDT_KR key-value register

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | KR | WO | Write 0xCCCC: start the IWDT counter<br>Write 0xCCCC: start the IWDT counter<br>Write 0x5A5A, 0xA5A5 in sequence: stop watchdog<br>Write 0x5555: release the write protection of IWDT_ARR, IWDT_WINR, IWDT_CR<br>Write to non-0x5555: enable write protection of IWDT_ ARR, IWDT_WINR, IWDT_CR |

### 14.6.2 IWDT_CR control register

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:6 | RFU | - | Reserved bits, please keep the default value |
| 5 | PAUSE | RW | IWDT pause control in Deepsleep mode<br>0: IWDT continues to run in DeepSleep mode<br>1: IWDT auto-pause in DeepSleep mode |
| 4 | IE | RW | IWDT interrupt enable control<br>0: Watchdog interrupt disabled<br>1: Watchdog interrupt enabled |
| 3 | ACTION | RW | Action configuration after IWDT overflow<br>0: Reset after watchdog overflow<br>1: Interrupt after watchdog overflow |
| 2:0 | PRS | RW | Configure the divider ratio of the watchdog count clock to the RC10K oscillator<br>000: 4 frequency division<br>001: 8 frequency division<br>010: 16 frequency division<br>011: 32 frequency division<br>100: 64 frequency division<br>101: 128 frequency division<br>110: 256 frequency division<br>111: 512 frequency division |

*Caution:*

*This register can be modified only after writing 0x5555 to the IWDT_KR register; IWDT_CR can only be modified when IWDT_SR.CRF is 0.*

### 14.6.3 IWDT_ARR reload register

Address offset: 0x08 Reset value: 0x0000 0FFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:12 | RFU | - | Reserved bits, please keep the default value |
| 11:0 | ARR | RW | IWDT reload value |

*Caution:*

*This register can be modified only after writing 0x5555 to the IWDT_KR register; This register can be modified only when IWDT_SR.ARRF is 0.*

### 14.6.4 IWDT_CNT count value register

Address offset: 0x24 Reset value: 0x0000 0FFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:12 | RFU | - | Reserved bits, please keep the default value |
| 11:0 | CNT | RO | Count value register<br>Caution: The count value can only be considered to be read if the value read twice in a row is the same |

### 14.6.5 IWDT_WINR window register

Address offset: 0x10 Reset value: 0x0000 0FFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:12 | RFU | - | Reserved bits, please keep the default value |
| 11:0 | WINR | RW | Window comparator compare value |

### 14.6.6 IWDT_SR status register

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:6 | RFU | - | Reserved bits, please keep the default value |
| 5 | RELOAD | RO | WDT counter reload flag<br>0: WDT counter has completed reload operation<br>1: WDT counter is reloading<br>Caution: In window mode, you need to wait for the flag to become 0 after reloading |
| 4 | RUN | RO | WDT running flag<br>0: WDT is not running<br>1: WDT is running |
| 3 | OV | RW0 | WDT overflow flag<br>R0: WDT does not overflow<br>R1: WDT overflow, write 0 to clear |
| 2 | WINRF | RO | WINR register update flag<br>0: WINR register update completed<br>1: WINR register is updating<br>Caution: After writing the WINR register, you need to wait for this flag to become 0 |
| 1 | ARRF | RO | ARR register update flag<br>0: ARR register update completed<br>1: ARR register is updating<br>Caution: After writing the ARR register, you need to wait for this flag to become 0 |
| 0 | CRF | RO | CR register update flag<br>0: CR register update completed<br>1: CR register is updating<br>Caution: After writing the CR register, you need to wait for the flag to become 0 |

# 15 Window watchdog timer (WWDT)

## 15.1 Overview

The CW32F003 integrates a window watchdog timer (WWDT), the user needs to refresh within the set time window, otherwise it will trigger a system reset. WWDT is usually used to monitor the program execution flow with strict time requirements to prevent the abnormal execution of the application program caused by external interference or unknown conditions, resulting in system failure.

## 15.2 Main features

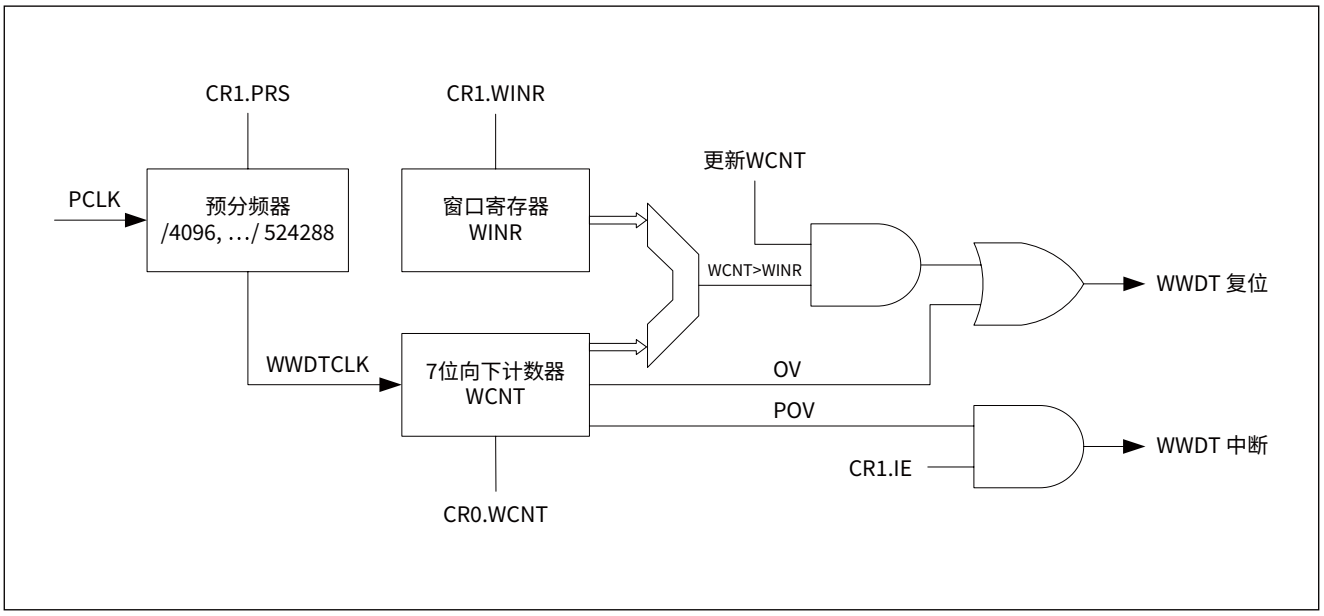
- 7-bit down counter
- PCLK clock driven, 8-level prescaler, maximum frequency division 524288
- Supports pre-overflow interrupt and count overflow, load count value error reset
- Can not be closed after opening, unless the system is reset

## 15.3 Functional description

### 15.3.1 Functional block diagram

The functional block diagram of WWDT is shown in the following figure:

Figure 15-1 WWDT functional block diagram




WWDT contains a 7-bit down counter. The counting clock source is the internal system clock PCLK. The clock source PCLK can be divided by the PRS bit field of the control register WWDT_CR1. After frequency division, the counting clock WWDTCLK is used to drive the counter to count.

The WWDT will stop counting in DeepSleep mode, and resume normal work after the CPU is woken up.

### 15.3.2 Operating mode

After the system is reset, the window watchdog WWDT is in a closed state, and the EN bit field of the control register WWDT_CR0 is set to 1 to start the WWDT. Once the WWDT is opened, it cannot be closed unless a reset occurs.

Before starting the WWDT, the user must preset the watchdog overflow time and window time. Setting the value of WWDT_CR0.WCNT can update the initial value of the counter, and setting the value of WWDT_CR1. WINR can configure the window value of the watchdog. The window value must be less than the initial value of the watchdog counter. After starting the WWDT, the counter counts down from the initial value.

When the counter decrements to 0x40, the pre-overflow signal POV will be generated, and setting WWDT_CR1. IE to 1 will generate a pre-overflow interrupt.

When the counter decrements to 0x3F, an overflow signal OV will be generated, which can trigger a system reset.

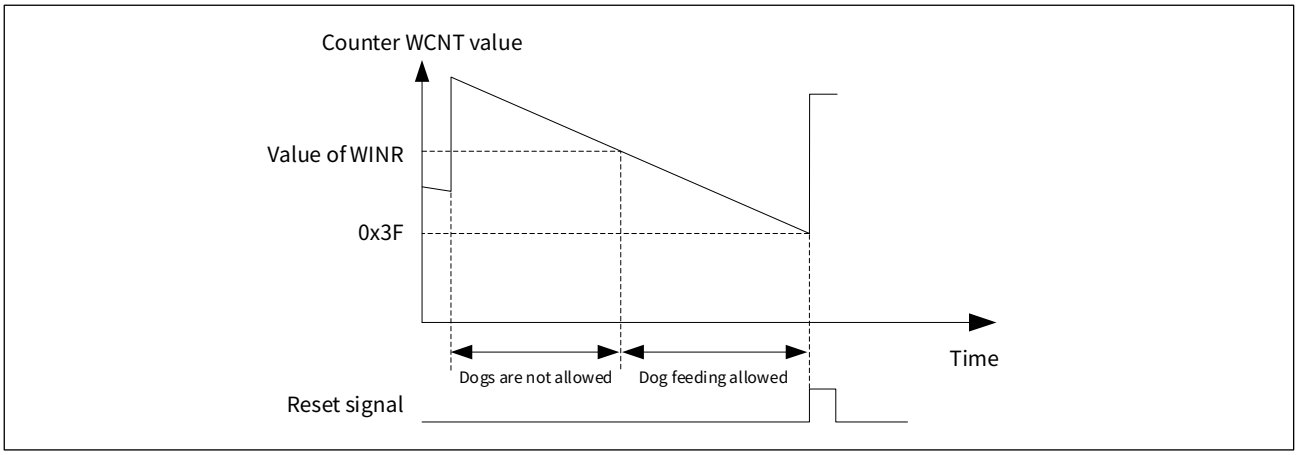
When the current count value WCNT is greater than the window value WINR, updating the watchdog counter will also trigger a system reset.

### 15.3.3 Refresh counter

When WWDT is running, the counter can be refreshed only when the count value is less than or equal to the window value and the count value is decremented to 0x3F, that is, the dog feeding operation, otherwise a system reset will occur.

Set the value of WWDT_CR1.WINR to configure the upper limit of the delay time window before reset, and the lower limit of the delay time window is fixed at 0x3F. The timing of feeding dog of the window watchdog is shown in the following figure:

Figure 15-2 Refresh (feed dog) timing of the window watchdog




### 15.3.4 Dog-feeding time

The calculation formula of the dog feeding time of the window watchdog:

Minimum time for dog-feeding:

$$T_{WWDT\_MIN} = T_{PCLK} \times 4096 \times 2^{PRS} \times (WCNT - WINR)$$

Maximum time for dog-feeding:

$$T_{WWDT\_MAX} = T_{PCLK} \times 4096 \times 2^{PRS} \times (WCNT - 0x3F)$$

Where:

$T_{PCLK}$: PCLK clock period

PRS: Prescale factor

WCNT: Counter current count value

WINR: Watchdog window value

Example:

When the PCLK frequency is 24MHz, the prescaler factor PRS is set to 0x01, WCNT is set to 0x6F, and the window value WINR is set to 0x4F, then the dog-feeding time is as follows:

$$T_{WWDT\_MIN} = 1 / 24MHz \times 4096 \times 2^{1} \times (0x6F - 0x4F) \approx 10.922\ ms$$

$$T_{WWDT\_MAX} = 1 / 24MHz \times 4096 \times 2^{1} \times (0x6F - 0x3F) = 16.384\ ms$$

That is, the dog-feeding time cannot be earlier than 10.922ms at the earliest, otherwise an error in the loading count value will occur and the system will reset; the dog-feeding time cannot be later than 16.384ms at the latest, otherwise an underflow will occur and the system will reset. The time window size for properly dog-feeding is about 5ms.

### 15.3.5 Reset and interrupt

When the value of the counter WCNT is decremented to 0x40, the pre-overflow interrupt flag bit WWDT_SR.POV is set by hardware. If the IE bit field of the control register WWDT_CR1 is set to 1 (Caution: this bit cannot be cleared to 0 after being set to 1), a pre-overflow interrupt flag will be generated. Overflow interrupt request. The user can update the counter WCNT in the interrupt service routine to avoid the reset of the WWDT.

A system reset can be triggered when one of the following conditions is met:

1. The value of the counter WCNT is decremented to 0x3F;
2. When updating the counter WCNT, the current count value is greater than the window value;
3. Write a value less than or equal to 0x3F to WWDT_CR0.WCNT.

## 15.4 Programming examples

### WWDT basic configuration process

Step 1: Set SYSCTRL_APBEN1.WWDT to 1, enable the configuration clock and working clock of WWDT;
Step 2: Configure the prescaler of the watchdog counter clock in the window through WWDT_CR1.PRS;
Step 3: Configure the compare value of the window watchdog count through WWDT_CR1.WINR;
Step 4: Configure the initial value of the counter through WWDT_CR0.WCNT;
Step 5: Configure WWDT_CR1.IE according to whether the pre-overflow interrupt needs to be enabled;
Step 6: Set WWDT_CR0.EN to 1 to start the window watchdog.

### WWDT dog-feeding

When the value of the counter WCNT is decremented to less than or equal to WWDT_CR1.WINR and greater than 0x3F, rewrite the initial value of the counter to WWDT_CR0.WCNT.

## 15.5 List of registers

WWDT base address: WWDT_BASE = 0x4000 2C00

Table 15-1 List of WWDT registers

| Register name | Register address | Register description |
|---|---|---|
| WWDT_CR0 | WWDT_BASE + 0x00 | Control register 0 |
| WWDT_CR1 | WWDT_BASE + 0x04 | Control register 1 |
| WWDT_SR | WWDT_BASE + 0x08 | Status register |

## 15.6 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 15.6.1 WWDT_CR0 control register 0

Address offset: 0x00 Reset value: 0x0000 007F

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | EN | RW1 | WWDT enable control, cannot be disabled after enabling<br>0: WWDT disabled<br>1: WWDT enabled |
| 6:0 | WCNT | RW | Read: WWDT counter current count value<br>Write: Update the current count value of the WWDT counter |

### 15.6.2 WWDT_CR1 control register 1

Address offset: 0x04 Reset value: 0x0000 007F

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:11 | RFU | - | Reserved bits, please keep the default value |
| 10 | IE | RW | WWDT pre-overflow interrupt enable control<br>0: Pre-overflow interrupt disabled<br>1: Pre-overflow interrupt enabled<br>Caution: It cannot be cleared after being set to 1 |
| 9:7 | PRS | RW | WWDT count clock prescale value<br>000: PCLK / 4096<br>001: PCLK / 8192<br>010: PCLK / 16384<br>011: PCLK / 32768<br>100: PCLK / 65536<br>101: PCLK / 131072<br>110: PCLK / 262144<br>111: PCLK / 524288 |
| 6:0 | WINR | RW | Watchdog window value |

### 15.6.3 WWDT_SR status register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:1 | RFU | - | Reserved bits, please keep the default value |
| 0 | POV | RW | WWDT pre-overflow flag<br>0: No pre-overflow occurred in WWDT<br>1: WWDT pre-overflow has occurred, write 0 to clear the flag |

# 16 Universal Asynchronous Receiver/Transmitter (UART)

## 16.1 Overview

CW32F003 integrates two universal asynchronous receiver/transmitter (UART), supports asynchronous full-duplex, synchronous half-duplex and single-wire half-duplex modes, supports hardware data flow control and multi-computer communication; programmable data frame structure can provide wide range of baud rate selection through wavelet baud rate generator.

The UART controller works in a dual clock domain, allowing data reception in DeepSleep mode, and the reception completion interrupt can wake the MCU back to Active mode.

## 16.2 Main features

- Supports dual clock domain drive
  - Configure clock PCLK
  - Transfer clock UCLK
- Programmable data frame structure
  - Data word length: 8/9 bits, LSB first
  - Parity bit: no parity/odd parity/even parity
  - Stop bit length: 1/1.5/2 bits
- 16-bit integer, 4-bit fractional baud rate generator
- Supports asynchronous full-duplex, synchronous half-duplex, single-wire half-duplex
- Supports hardware flow control RTS, CTS
- Supports multi-machine communication, automatic address recognition
- 6 interrupt sources with interrupt flags
- Error detection: parity error, frame structure error
- Send and receive data in low power mode, interrupt wake-up MCU

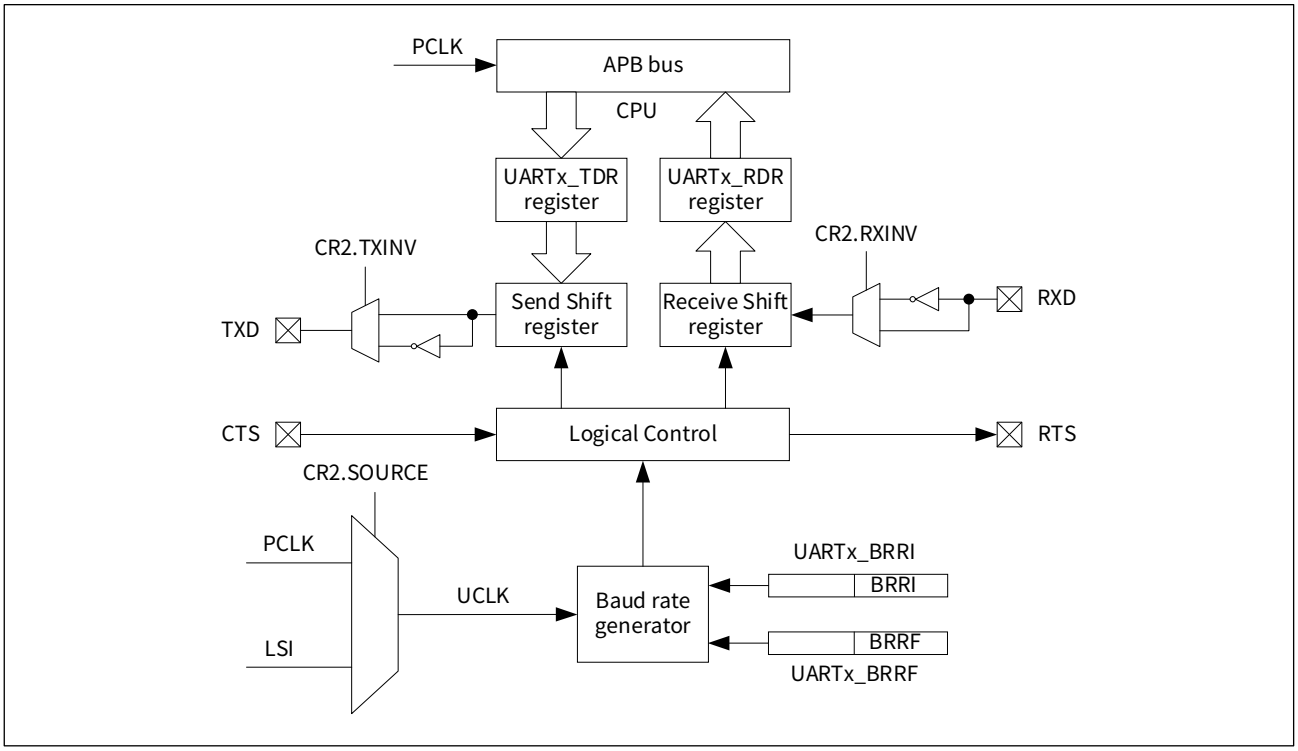
## 16.3 Functional description

### 16.3.1 Functional block diagram

The UART controller is mounted on the APB bus, and the clock domain PCLK is configured, which is fixed to the APB bus clock PCLK, which is used for register configuration logic work; transfer clock domain UCLK is used for data transceiver logic work, and its source can be selected from PCLK clock and internal low-speed clock (LSI). The dual clock domain design facilitates baud rate setting and supports waking up the controller from DeepSleep mode.

The functional block diagram of the UART controller is shown in the following figure:

Figure 16-1 UART functional block diagram




The UART controller supports multiple operating modes. In different operating modes, each pin has different function and different pin configuration, as shown in the following table (where x=1,2):

Table 16-1 UART ports configuration

| Operating mode | UART pins | Function | GPIO configuration |
|---|---|---|---|
| Asynchronous full duplex | UARTx_TXD | Data serial output | Digital, push-pull output/open-drain output, multiplexed |
| | UARTx_RXD | Data serial input | Digital, pull-up input, multiplexed |
| | UARTx_RTS [1] | Request to send | Digital, push-pull output/open-drain output, multiplexed |
| | UARTx_CTS [1] | Allow to send | Digital, pull-up input, multiplexed |
| Synchronous half duplex | UARTx_TXD | Clock signal output | Digital, push-pull output/open-drain output, multiplexed |
| | UARTx_RXD | Data transmitting and receiving | Digital, push-pull output/open-drain output, multiplexed |
| Single-wire half duplex | UARTx_TXD | Data transmitting and receiving | Digital, open-drain output, multiplexed (external pull-up required) |
| | UARTx_RXD | Do not use | Can be used as general IO |

*Caution 1:*

*Configuration is only required in hardware data flow control mode.*

### 16.3.2 Synchronous mode

The UART supports the synchronous half-duplex working mode. In this mode, the UARTx_TXD pin outputs a synchronous shift clock signal, and the UARTx_RXD pin transmits and receives data. See *Table 16-1 UART ports configuration* for the specific configuration of the UARTx_TXD and UARTx_RXD pins.

The synchronous mode of the UART can communicate with the single-wire half-duplex mode of the SPI. At this time, the SPI must be configured in a fixed level mode: the clock polarity CPOL is 1, and the clock phase CPHA is 1.

Set the SYNC bit field of the control register UARTx_CR1 to 1 to make the UART work in synchronous half-duplex mode. At this time, the UART can only be used as a host, the data word length can only be 8 bits, and UARTx_CR2.SIGNAL and UARTx_CR2.ADDREN must be cleared.

#### 16.3.2.1 Baud rate setting

In synchronous half-duplex mode, the baud rate calculation formula:

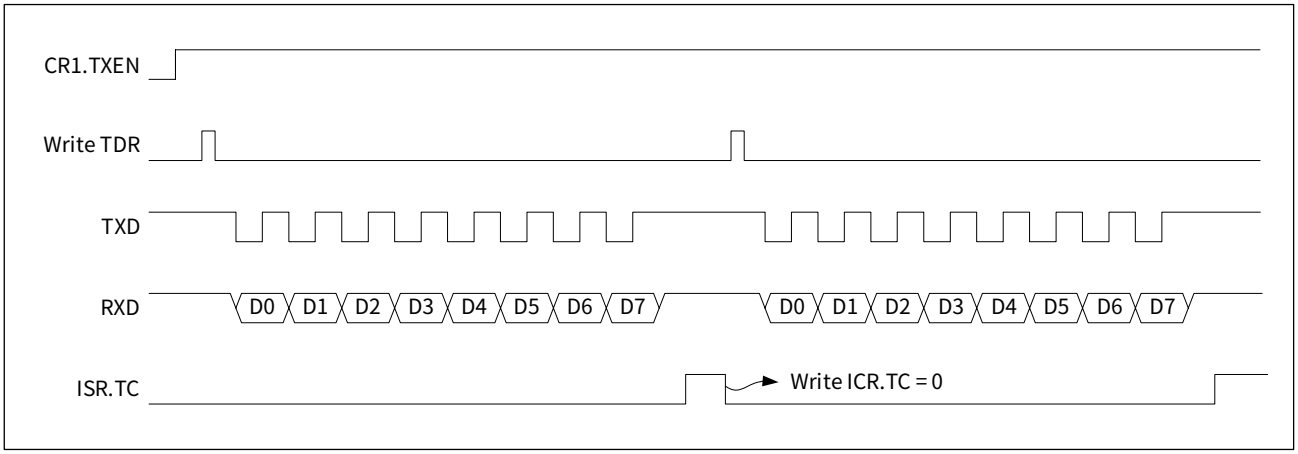
$$BaudRate = UCLK / 12$$

Where, UCLK is the transmission clock of the UART, and its source can be PCLK or LSI, which is selected by the SOURCE bit field of the control register UARTx_CR2.

#### 16.3.2.2 Data transmission and reception

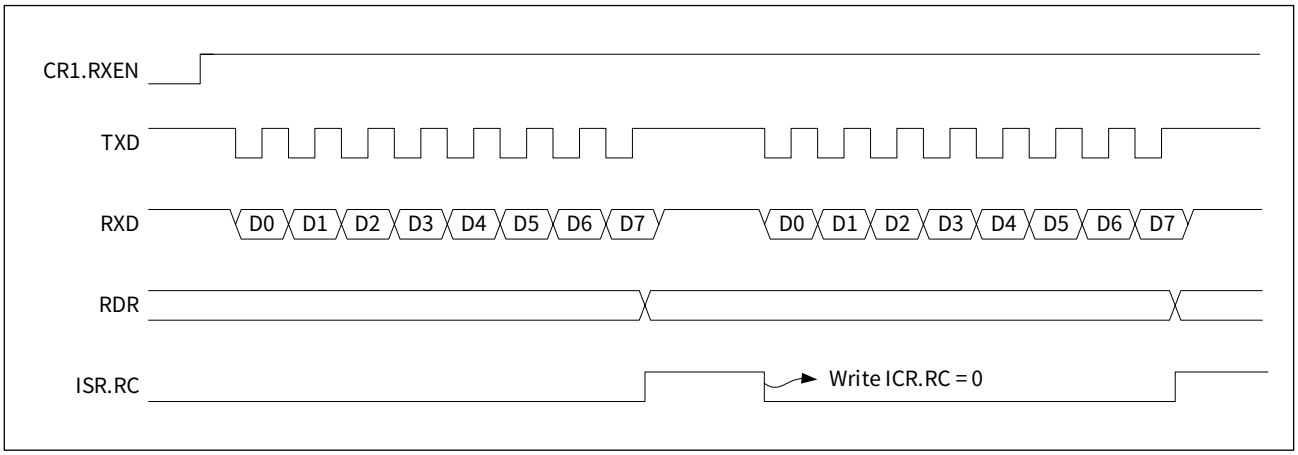
Set the UART transmitting enable bit UARTx_CR1.TXEN to 1 to make the UART work in the transmitting state. After writing the data to the UARTx_TDR register, the data will be serially output from the UARTx_RXD pin, and the UARTx_TXD pin will output a synchronous shift clock signal. When the data transmission is completed, the transmission completion interrupt flag UARTx_ISR.TC will be set by hardware.

Figure 16-2 Transmit timing in synchronous half-duplex mode (transmit two bytes of data)




Set the UART receiving enable bit UARTx_CR1.RXEN to 1 to make the UART work in the receiving state, the UARTx_TXD pin will output a synchronous shift clock signal, and the data will be serially input from the UARTx_RXD pin. After the data reception is completed, the reception completion interrupt flag UARTx_ISR.RC will be set by hardware, and the UARTx_RDR register can be read at this time. Before receiving the next byte, the UARTx_ISR.RC flag must be cleared to 0. After the RC flag is cleared, the next byte of data will be received immediately.

Figure 16-3 Receive timing in synchronous half-duplex mode (receive two bytes of data)



### 16.3.3 Asynchronous mode

#### 16.3.3.1 Data frame format

In UART asynchronous communication, data is transmited and received in the form of data frames, and a frame of data includes a start bit, a data field, an optional parity bit and a stop bit with a programmable width.

**Start bit**

The length of the start bit is fixed at 1 bit, and the judgment method of the start bit can be selected by falling edge or low level, which is selected by the START bit field of the control register UARTx_CR1. Generally, the falling edge is selected as the judgment method of the start bit. The low-level method to determine the start bit is mainly used in the low-power mode. Please refer to section *16.4 Low power mode*.

**Data field**

The data word length can be set to 8 bits or 9 bits, and is automatically configured by the presence or absence of parity bits. When set to no parity, the data word length is 8 bits, and when it is set to have parity (including custom parity, even parity, odd parity), the data word length is automatically set to 9 bits. Please refer to *Table 16-2 Data frame structure*.

**Parity bit**

The parity mode supports custom parity, even parity and odd parity, which is selected by the PARITY bit field of the control register UARTx_CR1.

Custom parity: The parity bit is determined by software writing to UARTx_TDR[8] and reading UARTx_RDR[8]. It is mainly used in multi-computer communication. Please refer to section *16.3.3.6 Multi-machine communicatio*.

Odd parity: The parity bit makes the total number of "1"s in the data bits and parity bits in a frame of data an odd number.

Even parity: The parity bit makes the total number of "1" in the data bits and the parity bit in a frame of data an even number.

UARTx_CR1.PARITY is used to set the parity mode, and the corresponding relationship with the data frame is shown in the following table:

Table 16-2 Data frame structure

| UARTx_CR1.PARITY | Checkout mode | Data frame |
|---|---|---|
| 00 | No parity | Start bit + 8-bit data + stop bit |
| 01 | Custom parity | Start bit + 8-bit data + custom parity bit + stop bit |
| 10 | Even parity | Start bit + 8-bit data + even parity bit + stop bit |
| 11 | Odd parity | Start bit + 8-bit data + odd parity bit + stop bit |

**Stop bit**

The length of the stop bit can be configured as 1, 1.5 or 2 bits, which is configured through the STOP bit field of the control register UARTx_CR1.

### 16.3.3.2 Fractional baud rate generator

#### Generation of baud rate

The receive and transmit baud rates of the UART are the same, generated by the same baud rate generator.

The baud rate generator supports 4 sampling modes: 16 times sampling, 8 times sampling, 4 times sampling and dedicated sampling. The specific sampling mode is selected by the OVER bit field of the control register UARTx_CR1.

1. OVER = 00, set 16 times sampling, baud rate calculation formula:

   $$BaudRate = UCLK / ( 16 \times BRRI + BRRF )$$

   UCLK is the transmission clock of UART, and its source can be PCLK or LSI. The specific source is selected by UARTx_CR2.SOURCE.

   BRRI (UARTx_BRRI［15:0］) is the integer part of the baud rate counter and can be set in the range of 1 ~ 65535.

   BRRF (UARTx_BRRF［3:0］) is the fractional part of the baud rate counter, and the settable range is 0 ~ 15.

   Example 1:

   When the frequency of the transmission clock UCLK is 24MHz, set BRRI=156 (ie UARTx_BRRI=0x9C), BRRF=4 (ie UARTx_BRRF=0x04), then:

   $$BaudRate = 24000000 / ( 16 \times 156 + 4 ) = 9600 \text{ bps}$$

   Example 2:

   When the frequency of the transmission clock UCLK is 24MHz, it is required to configure BaudRate=115200bps, and calculate

   $$16 \times BRRI + BRRF = 24000000 / 115200 = 208.33$$

   then:

   BRRI = 208.33 / 16 = 13.02, the nearest integer is: 13 (0x0D)

   BRRF = 0.02 × 16 = 0.32, the nearest integer is: 0 (0x00)

   That is, you need to set UARTx_BRRI to 0x0D and UARTx_BRRF to 0x00

   At this time, the actual baud rate BaudRate=115384.62bps, the error rate is 0.16%

2. OVER = 01, set 8 times sampling, baud rate calculation formula:

   $$BaudRate = UCLK / ( 8 \times BRRI )$$

3. OVER = 10, set 4 times sampling, baud rate calculation formula:

   $$BaudRate = UCLK / ( 4 \times BRRI )$$

4. OVER = 11, set dedicated sampling, baud rate calculation formula:

   $$BaudRate = ( 256 \times UCLK ) / BRRI$$

*Caution:*

*Dedicated sampling is only suitable for UART communication at 2400bps, 4800bps or 9600bps baud rate when the transmission clock source is LSI.*

Example 1:

The transmission clock UCLK selects LSI, the frequency is 32.8KHz, and the baud rate of 9600bps is required to be configured, then:

BRRI = 256×32800 / 9600 = 874.67 the nearest integer is: 875 (0x36B)

That is, you need to set UARTx_BRRI to 0x36B

At this time, the actual baud rate BaudRate=9596.34bps, the error is 0.039%

### Example of baud rate setting

Table 16-3 to Table 16-9 list the setting value of the baud rate count register and the baud rate error when the frequency of the common MCU is used.

Table 16-3 UCLK is 4MHz baud rate setting example (OVER=00)

| Baud rate (bps) | BRRI | BRRF | Actual baud rate | Error rate |
|---|---|---|---|---|
| 4800 | 52 | 1 | 4801.92 | 0.04% |
| 9600 | 26 | 1 | 9592.33 | -0.08% |
| 14400 | 17 | 6 | 14388.49 | -0.08% |
| 19200 | 13 | 0 | 19230.77 | 0.16% |
| 38400 | 6 | 8 | 38461.54 | 0.16% |
| 56000 | 4 | 7 | 56338.03 | 0.60% |
| 57600 | 4 | 5 | 57971.01 | 0.64% |
| 115200 | 2 | 3 | 114285.71 | -0.79% |
| 256000 | 1 | 0 | 250000 | -2.34% |

Table 16-4 UCLK is 8MHz baud rate setting example (OVER=00)

| Baud rate (bps) | BRRI | BRRF | Actual baud rate | Error rate |
|---|---|---|---|---|
| 4800 | 104 | 3 | 4799.04 | -0.02% |
| 9600 | 52 | 1 | 9603.84 | 0.04% |
| 14400 | 34 | 12 | 14388.49 | -0.08% |
| 19200 | 26 | 1 | 19184.65 | -0.08% |
| 38400 | 13 | 0 | 38461.54 | 0.16% |
| 56000 | 8 | 15 | 55944.06 | -0.10% |
| 57600 | 8 | 11 | 57553.96 | -0.08% |
| 115200 | 4 | 5 | 115942.03 | 0.64% |
| 500000 | 1 | 0 | 500000 | 0.00% |

Table 16-5 UCLK is 16MHz baud rate setting example (OVER=00)

| Baud rate (bps) | BRRI | BRRF | Actual baud rate | Error rate |
|---|---|---|---|---|
| 4800 | 208 | 5 | 4800.48 | 0.01% |
| 9600 | 104 | 3 | 9598.08 | -0.02% |
| 14400 | 69 | 7 | 14401.44 | 0.01% |
| 19200 | 52 | 1 | 19207.68 | 0.04% |
| 38400 | 26 | 1 | 38369.30 | -0.08% |
| 56000 | 17 | 14 | 55944.06 | -0.10% |
| 57600 | 17 | 6 | 57553.96 | -0.08% |
| 115200 | 8 | 11 | 115107.91 | -0.08% |
| 1000000 | 1 | 0 | 1000000 | 0.00% |

Table 16-6 UCLK is 24MHz baud rate setting example (OVER=00)

| Baud rate (bps) | BRRI | BRRF | Actual baud rate | Error rate |
|---|---|---|---|---|
| 4800 | 312 | 8 | 4800.00 | 0.00% |
| 9600 | 156 | 4 | 9600.00 | 0.00% |
| 14400 | 104 | 3 | 14397.12 | -0.02% |
| 19200 | 78 | 2 | 19200.00 | 0.00% |
| 38400 | 39 | 1 | 38400.00 | 0.00% |
| 56000 | 26 | 13 | 55944.06 | -0.10% |
| 57600 | 26 | 1 | 57553.96 | -0.08% |
| 115200 | 13 | 0 | 115384.62 | 0.16% |
| 1500000 | 1 | 0 | 1500000 | 0.00% |

Table 16-7 UCLK is 32MHz baud rate setting example (OVER=00)

| Baud rate (bps) | BRRI | BRRF | Actual baud rate | Error rate |
|---|---|---|---|---|
| 4800 | 416 | 11 | 4799.76 | 0.00% |
| 9600 | 208 | 5 | 9600.96 | 0.01% |
| 14400 | 138 | 14 | 14401.44 | 0.01% |
| 19200 | 104 | 3 | 19196.16 | -0.02% |
| 38400 | 52 | 1 | 38415.37 | 0.04% |
| 56000 | 35 | 11 | 56042.03 | 0.08% |
| 57600 | 34 | 12 | 57553.96 | -0.08% |
| 115200 | 17 | 6 | 115107.91 | -0.08% |
| 2000000 | 1 | 0 | 2000000 | 0.00% |

Table 16-8 UCLK is 48MHz baud rate setting example (OVER=00)

| Baud rate (bps) | BRRI | BRRF | Actual baud rate | Error rate |
|---|---|---|---|---|
| 4800 | 625 | 0 | 4800.00 | 0.00% |
| 9600 | 312 | 8 | 9600.00 | 0.00% |
| 14400 | 208 | 5 | 14401.44 | 0.01% |
| 19200 | 156 | 4 | 19200.00 | 0.00% |
| 38400 | 78 | 2 | 38400.00 | 0.00% |
| 56000 | 53 | 9 | 56009.33 | 0.02% |
| 57600 | 52 | 1 | 57623.05 | 0.04% |
| 115200 | 26 | 1 | 115107.91 | -0.08% |
| 3000000 | 1 | 0 | 3000000 | 0.00% |

Table 16-9 UCLK is 32.768KHz baud rate setting example (OVER=11)

| Baud rate (bps) | BRRI | Actual baud rate | Error rate |
|---|---|---|---|
| 2400 | 3499 | 2399.77 | -0.01% |
| 4800 | 1749 | 4800.91 | 0.02% |
| 9600 | 875 | 9596.34 | -0.04% |

### Automatic baud rate detection

When the CW32F003 uses the UART as the slave for communication, it can automatically adapt to the baud rate of the UART master by means of automatic baud rate detection. The input capture source of the general-purpose timer (GTIM) can be configured as the RXD signal of the UART, or the gating signal of the GTIM can be configured as the RXD signal of the UART, and the baud rate of the UART can be measured by using related software algorithms to achieve the baud rate. Adaptive. Please refer to the relevant application Cautions for details.

#### 16.3.3.3 Transmit control

**Data transmission**

Set UARTx_CR1.TXEN to 1 to enable transmit circuit. After writing the data to the UARTx_TDR register, the data will be transferred to the transmission shift register by hardware, and the transmission shift register will serially shift the data out from the lowest bit. For the register configuration of the specific transmission process, please refer to section *16.6 Programming examples*.

**Transmit interrupt**

In the transmission control link, there are three flag bits UARTx_ISR.TXE, UARTx_ISR.TC and UARTx_ISR.TXBUSY, among which UARTx_ISR.TXE and UARTx_ISR.TC can generate interrupt requests.

When the data is transferred to the transmission shift register by hardware, the transmit buffer empty interrupt flag UARTx_ISR.TXE will be set by hardware, indicating that the UARTx_TDR register is empty, and new data to be sent is allowed to be written to the UARTx_TDR register. While writing data to the UARTx_TDR register, the UARTx_ISR.TXE flag will be automatically cleared. If there is still untransmitted data in the transmission shift register at this time, the newly written data will be temporarily stored in the UARTx_TDR register. After the current data transmission is completed, the data in the UARTx_TDR register will be transferred by hardware to the transmission shift register. continue to transmit in the register.

When the transmission of the data frame in the transmission shift register is completed, the transmission completion interrupt flag bit UARTx_ISR.TC will be set by hardware, indicating that a frame of data has been transmitted.

When the data frame in the transmission shift register is transmitted, if there is no data to be transmitted in the UARTx_TDR register (ie UARTx_ISR.TXE=1), the UART transmit busy flag UARTx_ISR.TXBUSY will be cleared by hardware, indicating that the data in the UARTx_TDR register and the transmission shift register has been transmitted.

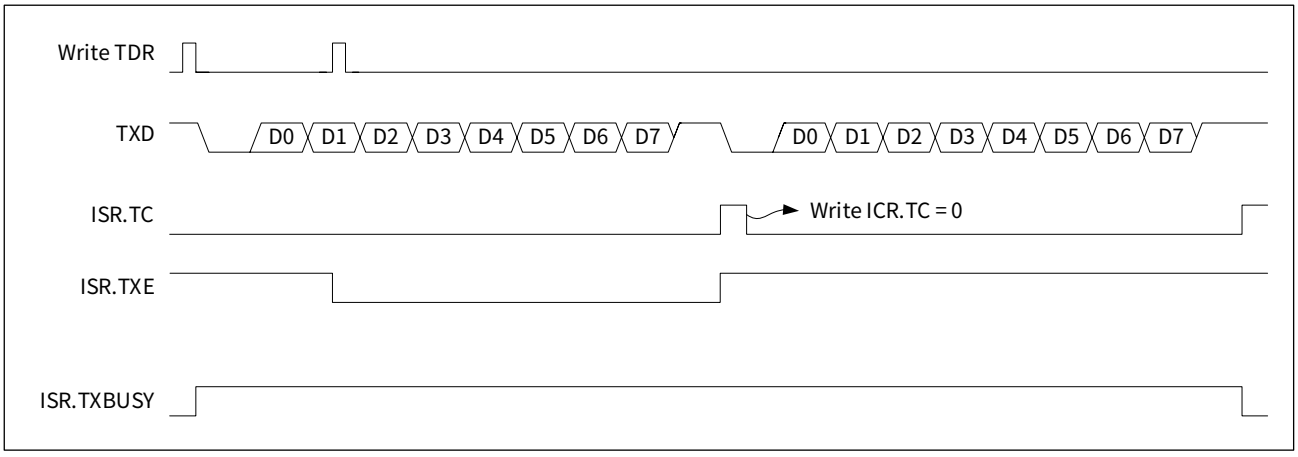
In user applications, before closing the UART, it is recommended to check whether the UARTx_ISR.TXBUSY flag has been cleared to ensure that all data has been transmitted to avoid data loss.

The UARTx_ISR.TXE flag bit and the UARTx_ISR.TC flag bit are generated by hardware, and both can generate interrupt requests, which are controlled by the interrupt enable bits UARTx_IER.TXE and UARTx_IER.TC respectively.

In the UART interrupt service routine, the interrupt source is determined by reading the UARTx_ISR.TXE and UARTx_ISR.TC flag bits. The corresponding flag bits should be cleared before exiting the interrupt service routine to avoid repeated entry into the interrupt service routine.

The timing diagram of the UART transmission control link is shown in the following figure:

Figure 16-4 Transmit control sequence in asynchronous working mode (transmitting two bytes of data)



#### 16.3.3.4 Receive control

**Data reception**

Set UARTx_CR1.RXEN to 1 to enable the receive circuit. When the receiver detects the first falling edge of the start bit, the hardware will sample the port at 16 times the set baud rate, and at the same time, it will make a comprehensive judgement on the intermediate three sampling results of each bit, and finally form the receive data, which will be shifted into the receive shift register serially, and transferred to the UARTx_RDR register in parallel, and at this time, the data can be be read out. For the register configuration of the specific receiving process, please refer to *16.6 Programming examples*.

**Receive interrupt**

In the receiving control link, there are three flag bits UARTx_ISR.RC, UARTx_ISR.PE and UARTx_ISR.FE, and all can generate interrupt requests.

When the received data is transferred from the shift register to the UARTx_RDR register, the reception completion interrupt flag bit UARTx_ISR.RC will be set by hardware, indicating that the reception of a frame of data has been completed. In the user program, once the UARTx_ISR.RC flag is detected as 1, the UARTx_RDR register should be read as soon as possible, and the UARTx_ISR.RC flag should be cleared. If the UARTx_RDR register is not read in time, the newly received data will overwrite the data in the UARTx_RDR register, resulting in data loss.

When a data frame is received, the parity check will be performed automatically. If a parity check error is detected, the UARTx_ISR.PE flag will be set by hardware, indicating a parity check error.

When receiving a data frame, if the correct stop bit is not recognized within the expected time, it is determined that a frame structure error has occurred, and the UARTx_ISR.FE flag will be set by hardware.

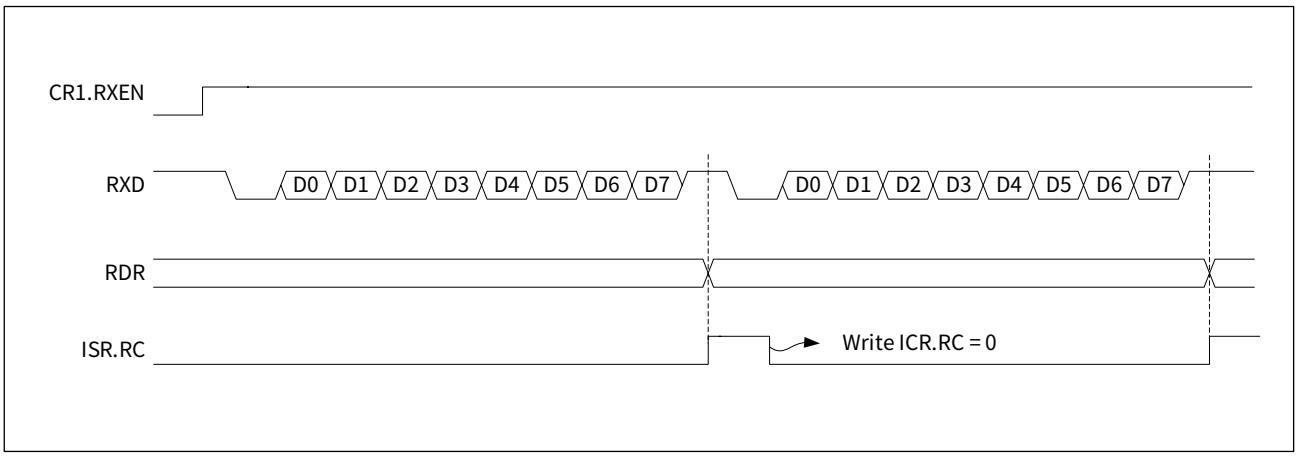
The UARTx_ISR.RC, UARTx_ISR.PE and UARTx_ISR.FE flag bits are generated by hardware and can generate interrupt requests, which are controlled by the interrupt enable bits UARTx_IER.RC, UARTx_IER.PE and UARTx_IER.FE respectively.

In the UART interrupt service routine, the interrupt source is determined by reading the UARTx_ISR.RC, UARTx_ISR.PE and UARTx_ISR.FE flag bits, and the corresponding flag bits should be cleared before exiting the interrupt service routine to avoid repeatedly entering the interrupt service routine.

In user applications, before closing the UART, it is recommended to check whether the UARTx_ISR.RC flag has been set to 1 to ensure that all data has been received to avoid data loss.

The timing diagram of the UART receiving control link is shown in the following figure:

Figure 16-5 Receive control timing in asynchronous working mode (receive two bytes of data)



#### 16.3.3.5 Single-wire half-duplex mode

Set UARTx_CR2.SIGNAL to 1 to make the UART work in single-wire half-duplex mode. In this mode, the UARTx_TXD pin is used for data transmission and reception without occupying the UARTx_RXD pin (UARTx_RXD can be used as general IO). See *Table 16-1 UART ports configuration* for configuration.

After writing data to the UARTx_TDR register, the UARTx_TXD pin immediately enters the transmit state and the output UARTx_TDR data in the register. After the data transmission is complete, the UARTx_TXD pin returns to its normal receive state.

When no data is sent, the UARTx_TXD pin is in the receiving state. After the data is received, the reception completion flag UARTx_ISR.RC will be set by hardware. At this time, the UARTx_RDR register should be read as soon as possible, and the UARTx_ISR.RC flag should be cleared.

*Caution:*

*Users should take appropriate application layer protection mechanisms to ensure that multiple masters do not send data to the bus at the same time.*

#### 16.3.3.6 Multi-machine communicatio

UART supports multi-machine communication mode. In this mode, there is one master and multiple slaves on the UART bus. Each slave has a unique slave address. During communication, the master will first send an address frame to address the slave. Only the slave with the matching address is activated and receives the data frame sent by the subsequent master.

**Master transmits**

In the multi-machine communication mode, the master needs to set UARTx_CR1.PARITY to 1, configure it as a custom check, and the frame data length is automatically set to 9 bits.

The highest bit of the UARTx_TDR register determines whether the master sends an address frame or a data frame. When UARTx_TDR[8] is 1, it means that the master sends an address frame. When UARTx_TDR[8] is 0, it means that the master sends a data frame.

**Slave receives**

In the multi-machine communication mode, the slave needs to set UARTx_CR1.PARITY to 1, configure it as a custom check, and set UARTx_CR2.ADDREN to 1 to enable the slave address recognition, and the slave hardware automatically detects the address sent by the host and the local machine. whether the address matches.

If the address matches, the slave will save the received address frame into the UARTx_RDR register, the UARTx_ISR.RC flag is set by hardware, and the UARTx_ISR.MATCH flag is set by hardware, and the slave receives the subsequent data frame sent by the master. During the communication process, the slave should pay attention to:

1. The application program queries UARTx_RDR[8] in the reception completion interrupt RC to determine whether it is an address frame or a data frame.
2. When the slave sends a data frame, it needs to set UARTx_TDR[8] to 0 to avoid being regarded as an address frame by other slaves.

If the addresses do not match, the slave will not receive the data frame sent by the master, nor will it generate a reception completion interrupt, and the set UARTx_ISR.MATCH flag will be cleared.

### Slave address and address mask

The slave address is configured by the UARTx_ADDR register, and the slave address should be configured to be unique. The UARTx_MASK register is an address mask. The slave address bit corresponding to the bit '1' in UARTx_MASK[7:0] participates in the slave address matching operation, and the slave address bit corresponding to the bit '0' does not participate in the slave address matching operation.

When all bits in UARTx_MASK[7:0] are set to '1', that is, write 0xFF to the UARTx_MASK register, then all 8-bit address bits of the slave address participate in the slave address matching, and the master can uniquely identify the slave.

When some bits in UARTx_MASK[7:0] are set to '0', the corresponding slave address bits do not participate in slave address matching, so as to realize that multiple slaves respond to the same address frame sent by the master, that is, the master addresses multiple slaves at the same time.

Example 1:

Set the slave A: UARTx_ADDR = 0xA0, UARTx_MASK = 0xFE

Set the slave B: UARTx_ADDR = 0xA1, UARTx_MASK = 0xFE

Set the slave C: UARTx_ADDR = 0xA2, UARTx_MASK = 0xFC

Set the slave D: UARTx_ADDR = 0xA3, UARTx_MASK = 0xFC

Set the slave E: UARTx_ADDR = 0xA5, UARTx_MASK = 0xFF

When the master transmits a 0xA0 or 0xA1 address frame, it can address slave A, slave B, slave C, and slave D

When the master transmits 0xA2 or 0xA3 address frame, it can address slave C, slave D

When the master transmits a 0xA4 address frame, no slave is addressed

When the master transmits a 0xA5 address frame, it can address the slave E

### Broadcast address

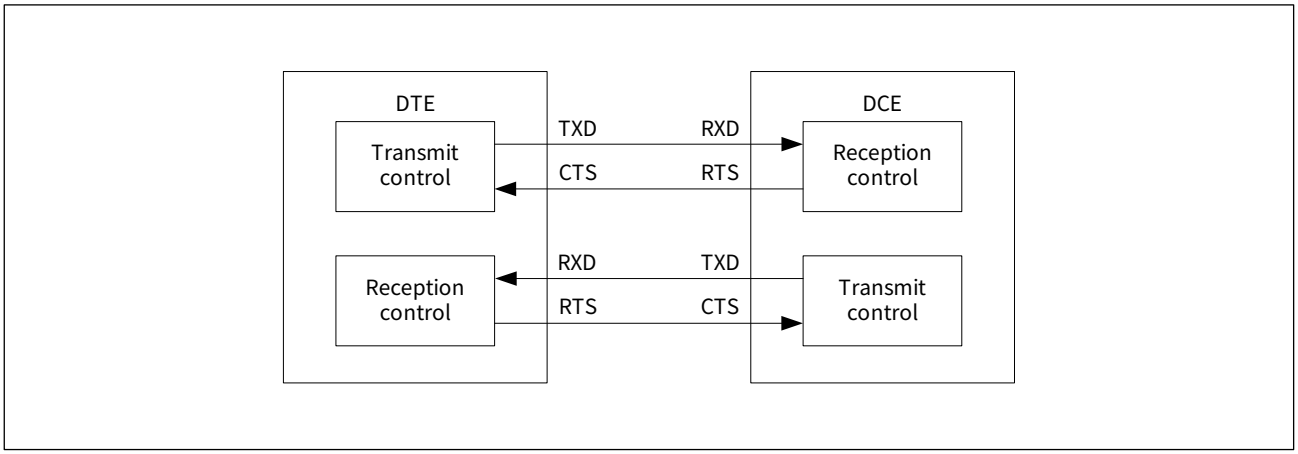
In multi-machine communication, the address 0xFF is defined as the broadcast address. When the master transmits the broadcast address frame, the slaves of all addresses are addressed.

### 16.3.3.7 Hardware flow control

CW32F003 supports UART hardware flow control mode, that is, it supports request to transmit RTS and allow to transmit CTS, which is used to automatically control the transmission and reception of data.

In the data communication system, the schematic diagram of the communication hardware flow control between the data terminal equipment DTE and the data circuit terminating equipment DCE is shown in the following figure.

Figure 16-6 Schematic diagram of hardware flow control




**Request to transmit RTS**

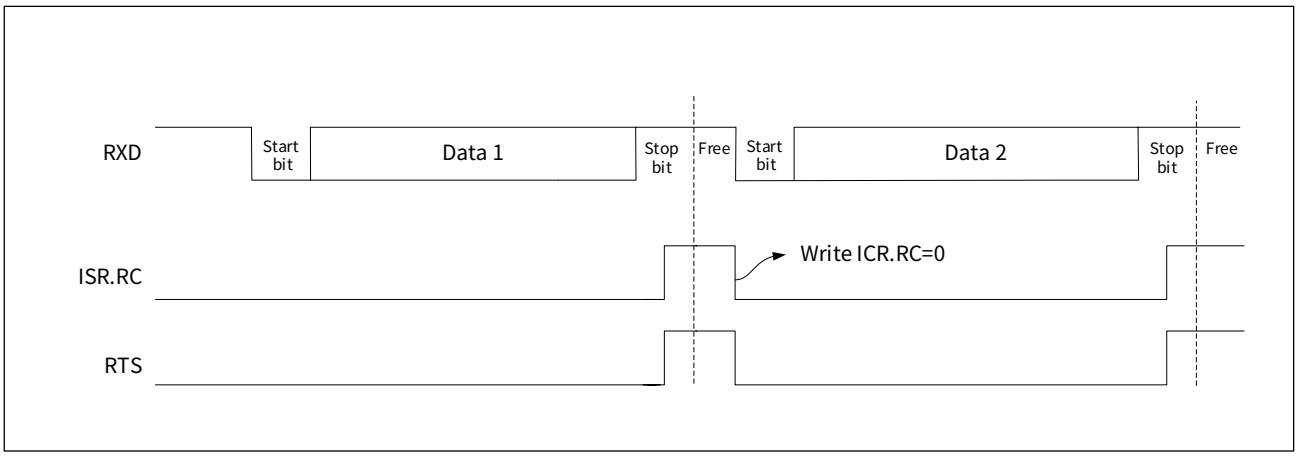
Set UARTx_CR2.RTSEN to 1 to enable RTS flow control, RTS means request to transmit, and low level is active.

When the receiver is ready to receive new data (ie UARTx_ISR.RC=0), the RTS pin will output a low level, requesting the transmitting terminal to transmit a frame of data.

When the receiving terminal receives a frame of data (ie UARTx_ISR.RC=1), the RTS pin will output a high level to notify the transmitting terminal to suspend transmiting.

In the user application, after reading the data in the UARTx_RDR register, if you want to continue to receive the next frame of data, the software must clear the UARTx_ISR.RC flag bit. At this time, the RTS pin will output a low level, requesting the transmitting terminal to transmit the next frame of data.

Figure 16-7 RTS flow control sequence



### Allow transmitting CTS

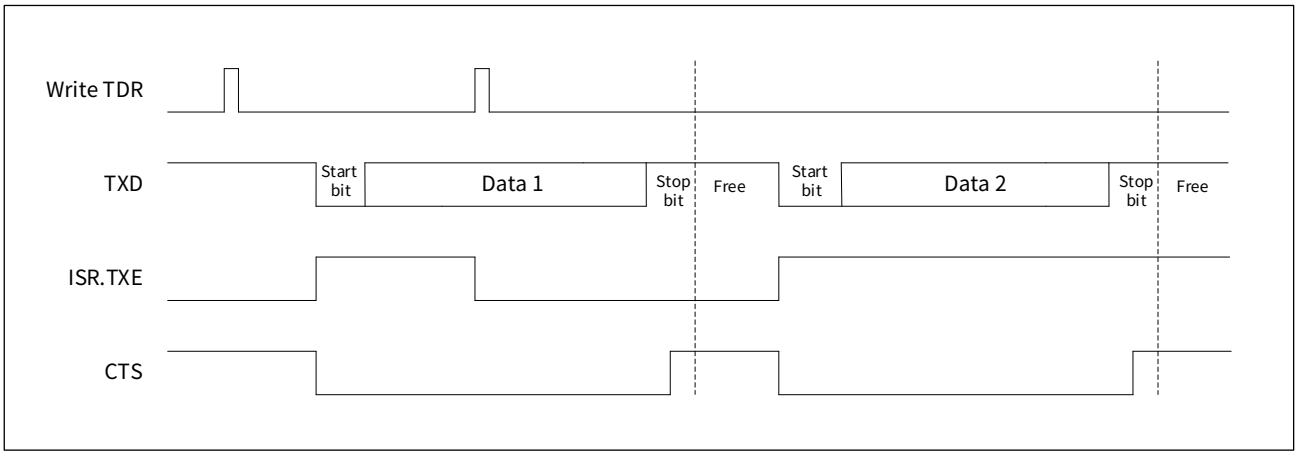
Set UARTx_CR.CTSEN to 1 to enable CTS flow control, CTS means transmission is allowed, and low level is active..

When the transmitting terminal detects that the CTS signal is low, if there is data to be sent (ie UARTx_ISR. TXE=0), and there is no data currently being transmitted, this frame of data will be transmitted.

When the transmitting terminal detects that the CTS signal is at a high level, if there is ongoing data transmission, the current transmission will continue, and the transmission will be stopped after the current transmission is completed.

In the CTS flow control process, the CTS signal level flag bit UARTx_ISR.CTSLV indicates the current level state of the CTS pin; the CTS signal change flag bit UARTx_ISR.CTS indicates whether the CTS signal has changed, when the CTS signal changes, The UARTx_ISR.CTS flag will be set by hardware. When the CTS interrupt is enabled (that is, set UARTx_IER.CTS to 1), an interrupt will be generated. Write UARTx_ICR.CTS to 0 to clear the interrupt flag.

Figure 16-8 CTS flow control sequence



## 16.4 Low power mode

The UART controller works in a dual clock domain, supports normal data transmission and reception in DeepSleep mode, and wakes up the MCU back to Active mode through a reception completion interrupt.

If the source of the transmission clock UCLK is set as the low-speed clock, when the system enters the DeepSleep mode, the high-speed clock will stop, the low-speed clock will keep running, and the UART can still transmit and receive data normally (the baud rate only supports 2400bps, 4800bps and 9600bps).

To use the UART wake-up function in DeepSleep mode, you need to enable the UART reception completion interrupt (that is, set UARTx_IER.RC to 1) before entering DeepSleep mode. When data reception is completed, the reception completion interrupt will wake up the MCU and return to Active mode.

If the source of the transmission clock UCLK is set as the high-speed clock, when the system enters the DeepSleep mode, the high-speed clock will stop running and the UART will not receive data. At this time, the MCU can still be woken up through the GPIO interrupt to receive data in DeepSleep mode. The reference configuration steps are as follows:

Step 1: Enable the GPIO falling edge interrupt of the corresponding pin of UARTx_RXD;

Step 2: Set UARTx_CR1.START to 1, and select the RXD signal start bit determination method to be low level;

Step 3: Enable UART reception (that is, set UARTx_CR1.RXEN to 1);

Step 4: Enter DeepSleep mode;

Step 5: Wait for the master to transmit data, generate a GPIO falling edge interrupt, and wake up the MCU;

Step 6: Closed the GPIO interrupt function of the corresponding pin to RXD and wait for the completion of RXD reception.

For more information on the optimized design of low power mode entry and wake-up, please refer to chapter *3 Power Control (PWR) and Power Consumption*.

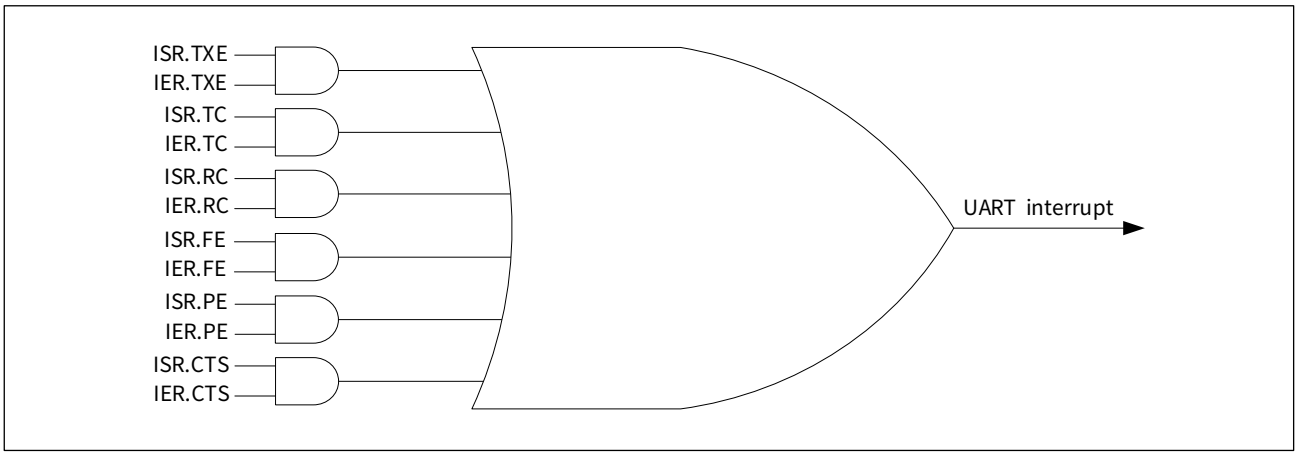
## 16.5 UART interrupts

The UART controller supports 6 interrupt sources. When the UART interrupt trigger event occurs, the interrupt flag bit will be set by hardware. If the corresponding interrupt enable control bit is set, an interrupt request will be generated.

A UART module of CW32F003 uses a same system UART interrupt. Whether the UART interrupt generates interrupt jump is controlled by the corresponding bit of the interrupt enable setting register NVIC_ISER of the Nested Vectored Interrupt Controller (NVIC).

The schematic diagram of the system UART interrupt is shown in the following figure:

Figure 16-9 Schematic diagram of UART interrupt structure




In the user's UART interrupt service routine, the relevant UART interrupt flag bit should be queried for corresponding processing. Before exiting the interrupt service routine, the interrupt flag bit should be cleared to avoid repeatedly entering the interrupt routine.

The flag bits, interrupt enable bits, interrupt flag clear bits and clear methods of each interrupt source of the UART are shown in the following table:

Table 16-10 UART interrupt control

| Interrupt event | Interrupt flag bit | Interrupt enable bit | Flag clear method |
|---|---|---|---|
| Transmit buffer empty | ISR.TXE | IER.TXE | Write UARTx_TDR register |
| Complete transmit-ting | ISR.TC | IER.TC | Write 0 to ICR.TC |
| Complete receiving | ISR.RC | IER.RC | Write 0 to ICR.RC |
| Frame structure er-ror | ISR.FE | IER.FE | Write 0 to ICR.FE |
| Parity error | ISR.PE | IER.PE | Write 0 to ICR.PE |
| CTS signal changes | ISR.CTS | IER.CTS | Write 0 to ICR.CTS |

## 16.6 Programming examples

### 16.6.1 Asynchronous full-duplex programming examples

#### 16.6.1.1 Transmit data in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable the GPIO clock and UART configuration clock corresponding to the UART pin;

Step 2: Configure the UARTx_TXD pin to the push-pull multiplexed output mode. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR1.SYNC to 0, and configure UARTx to be asynchronous full-duplex communication mode;

Step 4: Configure the data frame;

1. Start bit determination method: configure UARTx_CR1.START
2. Parity bit: configure UARTx_CR1.PARITY
3. Stop bit: configure UARTx_CR1.STOP

Step 5: Configure UARTx_CR2.SOURCE and select the transmission clock source;

Step 6: Configure UARTx_CR1.OVER and select sampling mode;

Step 7: Configure the UARTx_BRRI and UARTx_BRRF registers and configure the baud rate. For details, please refer to section *16.3.3.2 Fractional baud rate generator*;

Step 8: Set UARTx_CR1.TXEN to 1 to enable transmit;

Step 9: Set UARTx_ICR.TC to 0, clear the transmit completion flag;

Step 10: Write a frame of data to be transmitted into the UARTx_TDR register;

Step 11: Query and wait for the UARTx_ISR.TC flag to be set to 1 to confirm that one frame of data is transmitted;

Step 12: Repeat steps 9 ~ 12 to transmit the next frame of data.

#### 16.6.1.2 Receive data in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable the GPIO clock and UART configuration clock corresponding to the UART pin;

Step 2: Configure the RXD pin as the pull-up input multiplexed mode. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR1.SYNC to 0, and configure UARTx to be asynchronous full-duplex communication mode;

Step 4: Configure the data frame;

1. Start bit determination method: configure UARTx_CR1.START
2. Parity bit: configure UARTx_CR1.PARITY
3. Stop bit: configure UARTx_CR1.STOP

Step 5: Configure UARTx_CR2.SOURCE and select the transmission clock source;

Step 6: Configure UARTx_CR1.OVER and select sampling mode;

Step 7: Configure the UARTx_BRRI and UARTx_BRRF registers and configure the baud rate. For details, please refer to section *16.3.3.2 Fractional baud rate generator*;

Step 8: Write 0x00 to the UARTx_ICR register to clear all flags;

Step 9: Set UARTx_CR1.RXEN to 1 to enable receive;

Step 10: Query and wait for the UARTx_ISR.RC flag to be set to 1 to confirm that one frame of data has been received;

Step 11: Query the error flags UARTx_ISR.PE and UARTx_ISR.FE to confirm whether the data is valid. If the data is invalid, perform error processing. If the data is valid, read the UARTx_RDR register and save the data;

Step 12: Set UARTx_ICR.RC to 0, clear the reception completion flag;

Step 13: Repeat steps 10 ~ 13 to receive the next frame of data.

#### 16.6.1.3 Transmit data in interrupt mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable GPIO clock and UART configuration clock corresponding to UART pins;

Step 2: Configure the TXD pin to the push-pull multiplexed output mode. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR1.SYNC to 0, and configure UARTx to be asynchronous full-duplex communication mode;

Step 4: Configure the data frame;

1. Start bit determination method: configure UARTx_CR1.START
2. Parity bit: configure UARTx_CR1.PARITY
3. Stop bit: configure UARTx_CR1.STOP

Step 5: Configure UARTx_CR2.SOURCE and select the transmission clock source;

Step 6: Configure UARTx_CR1.OVER and select sampling mode;

Step 7: Configure the UARTx_BRRI and UARTx_BRRF registers and configure the baud rate. For details, For details, please refer to section *16.3.3.2 Fractional baud rate generator*;

Step 8: Configure the NVIC controller, see chapter *5 Interrupt*;

Step 9: Set UARTx_IER.TXE to 1 to enable transmit buffer empty interrupt;

Step 10: Set UARTx_CR1.TXEN to 1 to enable transmit;

Step 11: Write a frame of data to be transmitted into the UARTx_TDR register;

Step 12: The data starts to transmit, the transmit buffer is empty, and the interrupt service function is entered: query and judge the UARTx_ISR.TXE flag bit, if the flag bit is 1, write a new frame of data to the UARTx_TDR register;

Step 13: Query and wait for the UARTx_ISR.TXBUSY flag to be cleared and turn off the UART.

#### 16.6.1.4 Receive data in interrupt mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable the GPIO clock and UART configuration clock corresponding to the UART pin;

Step 2: Configure the RXD pin as the pull-up input multiplexed mode. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR1.SYNC to 0, and configure UARTx to be asynchronous full-duplex communication mode;

Step 4: Configure the data frame;

1. Start bit determination method: configure UARTx_CR1.START
2. Parity bit: configure UARTx_CR1.PARITY
3. Stop bit: configure UARTx_CR1.STOP

Step 5: Configure UARTx_CR2.SOURCE and select the transmission clock source;

Step 6: Configure UARTx_CR1.OVER and select sampling mode;

Step 7: Configure the UARTx_BRRI and UARTx_BRRF registers and configure the baud rate. For details, please refer to section *16.3.3.2 Fractional baud rate generator*;

Step 8: Configure the NVIC controller, see chapter *5 Interrupt*;

Step 9: Set UARTx_IER.RC to 1 to enable receive completion interrupt;

Step 10: Set UARTx_CR1.RXEN to 1 to enable receive;

Step 11: Wait for the completion of receiving a frame of data, and enter the interrupt service function: query the UARTx_ISR.PE and UARTx_ISR.FE flag bits to confirm whether the data is valid, if invalid, perform error handling; if valid, query and judge the UARTx_ISR.RC flag bit, If the flag bit is 1, read the UARTx_RDR register and save the data, and clear the flag bit.

### 16.6.2 Synchronous half-duplex programming examples

#### 16.6.2.1 Transmit data in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable GPIO clock and UART configuration clock;

Step 2: Configure the UARTx_TXD and UARTx_RXD pins as push-pull multiplexed output mode, and configure the slave chip select pin NCS as push-pull output mode. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR1.SYNC to 1, and configure UARTx to be synchronous half-duplex communication mode;

Step 4: Configure UARTx_CR2.SOURCE, select the transmission clock source, and configure the baud rate. For details, please refer to chapter *16.3.2 Synchronous mode*;

Step 5: Pull down the NCS signal and select a slave;

Step 6: Set UARTx_CR1.TXEN to 1 to enable transmit;

Step 7: Set UARTx_ICR.TC to 0, and clear the transmission completion flag;

Step 8: Write the 8-bit data to be transmitted to the UARTx_TDR register;

Step 9: Query and wait for the UARTx_ISR.TC flag bit to be set to 1 to confirm that the data transmission is completed;

Step 10: Repeat steps 7 ~ 10 to transmit the next frame of data;

Step 11: Pull up the NCS signal to end the communication.

#### 16.6.2.2 Receive data in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable GPIO clock and UART configuration clock;

Step 2: Configure the UARTx_TXD and UARTx_RXD pins as push-pull multiplexed output mode, and configure the slave chip select pin NCS as push-pull output mode. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR1.SYNC to 1, and configure UARTx to be synchronous half-duplex communication mode;

Step 4: Configure UARTx_CR2.SOURCE, select the transmission clock source, and configure the baud rate. For details, please refer to section *16.3.2 Synchronous mode*;

Step 5: Pull down the NCS signal and select a slave;

Step 6: Write 0x00 to UARTx_ICR to clear all flags;

Step 7: Set UARTx_CR1.RXEN to 1 to enable reception;

Step 8: Query and wait for the UARTx_ISR.RC flag to be set to 1 to confirm that the data reception is completed;

Step 9: Read the UARTx_RDR register and save the data;

Step 10: Set UARTx_ICR.RC to 0, clear the reception completion flag;

Step 11: Repeat steps 8 ~ 11 to receive the next 8-bit data;

Step 12: Pull up the NCS signal to end the communication.

### 16.6.3 Single-wire half-duplex programming examples

#### 16.6.3.1 Transmit data in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable the GPIO clock and UART configuration clock corresponding to the UART pin;

Step 2: Configure the UARTx_TXD pin to open-drain multiplexed output mode, and connect an external pull-up resistor. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR2.SIGNAL to 1, and configure UARTx to be single-wire half-duplex communication mode;

Step 4: Configure the data frame;

1. Start bit determination method: configure UARTx_CR1.START
2. Parity bit: configure UARTx_CR1.PARITY
3. Stop bit: configure UARTx_CR1.STOP

Step 5: Configure UARTx_CR2.SOURCE and select the transmission clock source;

Step 6: Configure UARTx_CR1.OVER and select sampling mode;

Step 7: Configure the UARTx_BRRI and UARTx_BRRF registers and configure the baud rate. For details, please refer to section *16.3.3.2 Fractional baud rate generator*;

Step 8: Set UARTx_CR1.TXEN to 1 and UARTx_CR1.RXEN to 1 to enable transmit and receive;

Step 9: Set UARTx_ICR.TC to 0, and clear the transmission completion flag;

Step 10: Write a frame of data to be transmitted into the UARTx_TDR register;

Step 11: Query and wait for the UARTx_ISR.TC flag bit to be set to 1 to confirm that the data transmission is completed;

Step 12: Repeat steps 9 ~ 12 to transmit the next frame of data.

#### 16.6.3.2 Receive data in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.UARTx to 1, enable the GPIO clock and UART configuration clock corresponding to the UART pin;

Step 2: Configure the UARTx_TXD pin to open-drain multiplexed output mode, and connect an external pull-up resistor. For the specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set UARTx_CR2.SIGNAL to 1, and configure UARTx to be single-wire half-duplex communication mode;

Step 4: Configure the data frame;

1. Start bit determination method: configure UARTx_CR1.START
2. Parity bit: configure UARTx_CR1.PARITY
3. Stop bit: configure UARTx_CR1.STOP

Step 5: Configure UARTx_CR2.SOURCE and select the transmission clock source;

Step 6: Configure UARTx_CR1.OVER and select sampling mode;

Step 7: Configure the UARTx_BRRI and UARTx_BRRF registers and configure the baud rate. For details, please refer to section *16.3.3.2 Fractional baud rate generator*;

Step 8: Write 0x00 to UARTx_ICR to clear all flags;

Step 9: Set UARTx_CR1.TXEN to 1 and UARTx_CR1.RXEN to 1 to enable transmit and receive;

Step 10: Query and wait for the UARTx_ISR.RC flag to be set to 1 to confirm that one frame of data has been received;

Step 11: Query the error flags UARTx_ISR.PE and UARTx_ISR.FE to confirm whether the data is valid. If the data is invalid, perform error processing. If the data is valid, read the UARTx_RDR register and save the data;

Step 12: Set UARTx_ICR.RC to 0, clear the reception completion flag;

Step 13: Repeat steps 10 ~ 13 to receive the next frame of data.

## 16.7 List of registers

UART1 base address: UART1_BASE = 0x4001 3800

UART2 base address: UART2_BASE = 0x4000 4400

Table 16-11 List of UART registers

| Register name | Register address | Register description |
|---|---|---|
| UARTx_CR1 | UARTx_BASE + 0x00 | Control register 1 |
| UARTx_CR2 | UARTx _BASE + 0x04 | Control register 2 |
| UARTx_IER | UARTx_BASE + 0x08 | Interrupt enable register |
| UARTx_BRRI | UARTx_BASE + 0x0C | Baud rate counter integer part register |
| UARTx_BRRF | UARTx_BASE + 0x10 | Baud rate counter fractional part register |
| UARTx_ISR | UARTx_BASE + 0x1C | Interrupt flag register |
| UARTx_ICR | UARTx_BASE + 0x20 | Interrupt flag clear register |
| UARTx_RDR | UARTx_BASE + 0x24 | Receive data register |
| UARTx_TDR | UARTx_BASE + 0x28 | Transmit data register |
| UARTx_ADDR | UARTx_BASE + 0x30 | Slave address register |
| UARTx_MASK | UARTx_BASE + 0x34 | Slave address mask register |

## 16.8 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 16.8.1 UARTx_CR1 control register 1

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:11 | RFU | - | Reserved bits, please keep the default value |
| 10:9 | OVER | RW | Sampling mode settings<br>00: 16 times sampling, Baud = UCLK/( BRRI×16+BRRF )<br>01: 8 times sampling, Baud = UCLK/( BRRI×8 )<br>10: 4 times sampling, Baud = UCLK/( BRRI×4 )<br>11: dedicated sampling, Baud = UCLK×256/BRRI<br>Caution 1: Dedicated sampling is only suitable for 2400, 4800, 9600 communication when the transmission clock is LSI;<br>Caution 2: When BRRF is a non-zero value, 16 times sampling is automatically adopted. |
| 8 | START | RW | RXD signal START bit determination mode settings<br>0: RXD signal falling edge<br>1: RXD signal low level |
| 7 | RFU | - | Reserved bits, please keep the default value |
| 6 | SYNC | RW | Transmission sync-async mode settings<br>0: Working in asynchronous mode<br>1: Working in synchronous mode |
| 5:4 | STOP | RW | Stop bit length settings<br>00: 1 bit<br>01: 1.5 bit<br>10: 2 bit<br>11: reserve<br>Caution: STOP bit must be cleared in synchronous mode |
| 3:2 | PARITY | RW | Check digit settings<br>00: no parity<br>01: custom check<br>10: even parity<br>11: odd parity<br>Caution 1: When there is no parity check, the data word length is 8 bits; when there is a check bit, the data word length is 9 bits;<br>Caution 2: In the multi-computer communication mode, both the master and the slave are configured as custom check. |
| 1 | RXEN | RW | Receive circuit enable control<br>0: Disabled<br>1: Enabled |
| 0 | TXEN | RW | Transmit circuit enable control<br>0: Disabled<br>1: Enabled |

### 16.8.2 UARTx_CR2 control register 2

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9:8 | SOURCE | RW | UART transmission clock source configuration<br>00: UCLK is from PCLK<br>01: UCLK is from PCLK<br>11: UCLK is from LSI |
| 7:6 | RFU | - | Reserved bits, please keep the default value |
| 5 | TXINV | RW | TXD pin outputs signal inversion control<br>0: The TXD pin outputs the non-inverting signal<br>1: The TXD pin outputs the inverted signal |
| 4 | RXINV | RW | RXD pin inputs signal inversion control<br>0: The RXD pin signal is directly transmitted to the receiving circuit<br>1: The RXD pin signal is inverted and transmitted to the receiving circuit |
| 3 | RTSEN | RW | RTS hardware signal enable control<br>0: Disabled<br>1: Enabled |
| 2 | CTSEN | RW | CTS hardware signal enable control<br>0: Disabled<br>1: Enabled |
| 1 | SIGNAL | RW | Single bus mode enable control<br>0: Disabled, transmit data through TXD port and receive data through RXD port<br>1: Enabled, transmit and receive data through the TXD port |
| 0 | ADDREN | RW | Slave address recognition enable control<br>0: Disabled<br>1: Enabled |

### 16.8.3 UARTx_BRRI baud rate counter integer part register

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | BRRI | RW | Baud rate counter integer part |

### 16.8.4 UARTx_BRRF baud rate counter fractional register

Address offset: 0x10 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:4 | RFU | - | Reserved bits, please keep the default value |
| 3:0 | BRRF | RW | Baud rate counter fractional part |

### 16.8.5 UARTx_TDR transmit data register

Address offset: 0x28 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:9 | RFU | - | Reserved bits, please keep the default value |
| 8:0 | TDR | WO | Data to be transmitted<br>Caution: When UARTX_CR1.PARITY is set to a custom verification, the TXD pin will send UARTx_TDR[8] |

### 16.8.6 UARTx_RDR receive data register

Address offset: 0x24 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:9 | RFU | - | Reserved bits, please keep the default value |
| 8:0 | RDR | RO | Received data<br>Caution: When UARTx_CR1.PARITY is set to a custom verification, the receiving circuit will receive UARTx_TDR[8] |

### 16.8.7 UARTx_IER interrupt enable register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | CTS | RW | CTS signal change interrupt enable control<br>0: Disabled<br>1: Enabled |
| 5 | RFU | - | Reserved bits, please keep the default value |
| 4 | PE | RW | Parity check error interrupt enable control<br>0: Disabled<br>1: Enabled |
| 3 | FE | RW | Frame structure error interrupt enable control<br>0: Disabled<br>1: Enabled |
| 2 | RC | RW | Receive completion interrupt enable control<br>0: Disabled<br>1: Enabled |
| 1 | TC | RW | Transmit completion interrupt enable control<br>0: Disabled<br>1: Enabled |
| 0 | TXE | RW | Transmit buffer empty interrupt enable control<br>0: Disabled<br>1: Enabled |

### 16.8.8 UARTx_ISR interrupt flag register

Address offset: 0x1C Reset value: 0x0000 0001

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:9 | RFU | - | Reserved bits, please keep the default value |
| 8 | TXBUSY | RO | TXD transmitting status<br>0:TDR register and transmit shift register are empty<br>1:TDR register or transmit shift register is not empty |
| 7 | CTSLV | RO | CTS signal level status<br>0: CTS signal is low level<br>1: CTS signal is high level |
| 6 | CTS | RO | CTS signal change interrupt flag<br>0: No CTS level change detected<br>1: CTS level change has been detected |
| 5 | MATCH | RO | Slave address match flag<br>0: No match address was detected.<br>1: The match address has been detected |
| 4 | PE | RO | Parity check error interrupt flag<br>0: No parity check error was detected<br>1: Parity check error has been detected |
| 3 | FE | RO | Frame structure error interrupt flag<br>0: No frame structure error was detected<br>1: The frame structure error has been detected |
| 2 | RC | RO | Receive completion interrupt flag<br>0: The reception of a frame of data has not been completed<br>1: The reception of a frame of data has been completed |
| 1 | TC | RO | Transmit completion interrupt flag<br>The transmit shift register is set by hardware when the transmission of a frame of data is completed.<br>0: The transmission of a frame of data has not been completed<br>1: The transmission of a frame of data has been completed |
| 0 | TXE | RO | Transmit buffer empty flag<br>0: There is data in the transmit buffer<br>1: No data in transmit buffer<br>Caution: Write the uartx_tdr register will clear the flag bit |

### 16.8.9 UARTx_ICR interrupt flag clear register

Address offset: 0x20 Reset value: 0x0000 00FF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | CTS | R1W0 | Clear control of CTS signal change interrupt flag<br>W0: CTS signal change interrupt flag cleared<br>W1: No function |
| 5 | RFU | - | Reserved bits, please keep the default value |
| 4 | PE | R1W0 | Clear control of parity check error interrupt flag<br>W0: Parity check error interrupt flag cleared<br>W1: No function |
| 3 | FE | R1W0 | Clear control of frame error interrupt flag<br>W0: Frame structure error interrupt flag cleared<br>W1: No function |
| 2 | RC | R1W0 | Clear control of receiving completion interrupt flag<br>W0: Receive completion interrupt flag cleared<br>W1: No function |
| 1 | TC | R1W0 | Clear control of transmitting completion interrupt flag<br>W0: Transmit completion interrupt flag cleared<br>W1: No function |
| 0 | RFU | - | Reserved bits, please keep the default value |

### 16.8.10 UARTx_ADDR slave address register

Address offset: 0x30 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | ADDR | RW | Slave address register |

### 16.8.11 UARTx_MASK slave address mask register

Address offset: 0x34 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | MASK | RW | Slave address mask register |

# 17 Serial peripheral interface (SPI)

## 17.1 Overview

Serial peripheral interface (SPI) is a synchronous serial data communication interface, which is often used for synchronous serial communication between MCU and external devices. CW32F003 integrates one serial peripheral SPI interfaces, supports bidirectional full-duplex, single-wire half-duplex and simplex communication modes, can configure MCU as master or slave, supports multi-master communication mode.

## 17.2 Main features

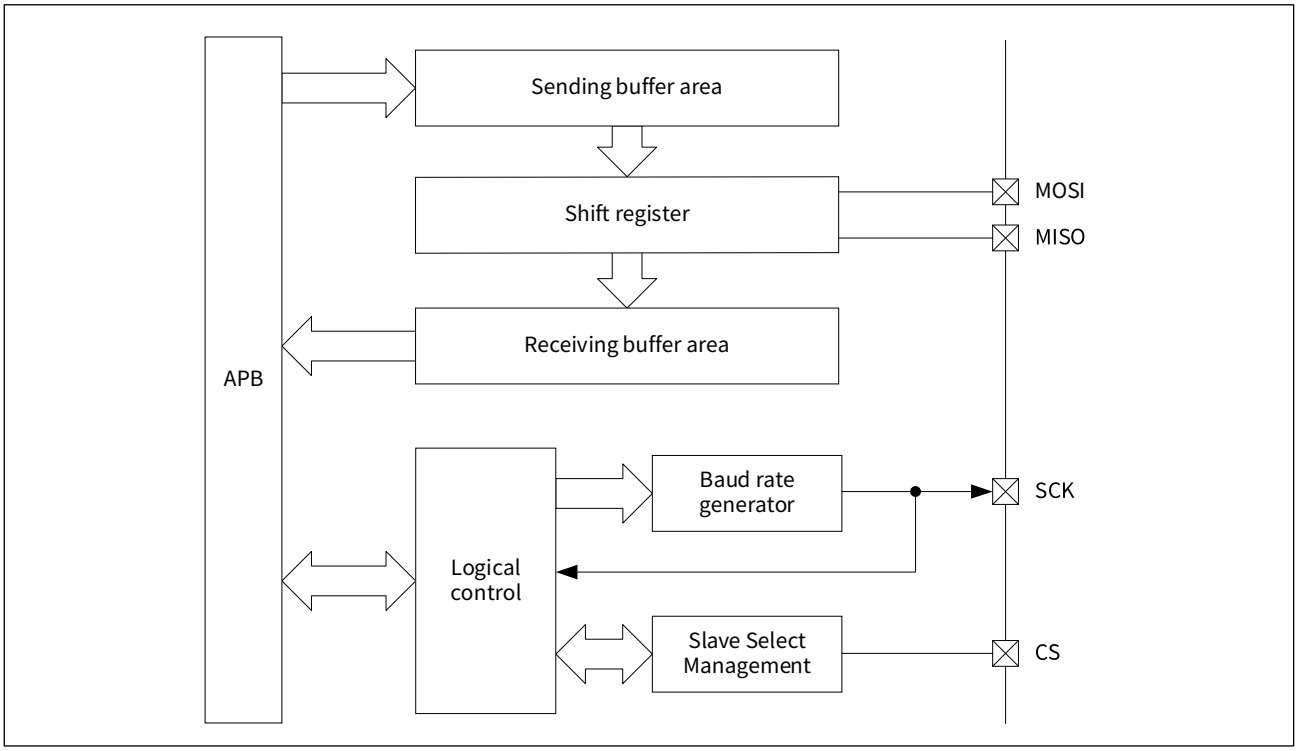
- Supports master mode, slave mode
- Supports full-duplex, single-wire half-duplex, simplex
- Selectable 4 to 16 bit data frame width
- Supports sending and receiving data LSB or MSB first
- Programmable clock polarity and clock phase
- Communication rates up to PCLK/4 in master mode
- Communication rates up to PCLK/4 in slave mode
- Supports multi-computer communication mode
- 8 interrupt sources with flag bits

## 17.3 Functional description

### 17.3.1 Functional block diagram

The functional block diagram of the SPI controller is shown in the following figure:

Figure 17-1 SPI functional block diagram




SPI is generally connected to external devices through 4 pins:

- MOSI

  Master output/slave input, used for data transmission in master mode and data reception in slave mode;

- MISO

  Master input/slave output, used for data reception in master mode and data transmission in slave mode;

- SCK

  Synchronous serial clock, master clock output and slave clock input, transmit or receive master synchronous clock;

- CS

  Slave selection is also used for bus conflict detection in multi-computer communication. Refer to section *17.3.3 Slave select pin CS*.

CW32F003 supports the user to flexibly select GPIO as the SPI communication pin, which is realized by multiplexing the AFR function, as shown in the following table:

Table 17-1 SPI pins selection

| SPI | GPIO | AFR |
|---|---|---|
| SPI_CS | PB00/PB02/PB06 | 0x03 |
| | PC03 | 0x02 |
| SPI_MISO | PA04/PC01/PC03 | 0x03 |
| SPI_MOSI | PA01/PC02/PC04 | 0x03 |
| SPI_SCK | PA00/PB07/PC00 | 0x03 |

SPI supports a variety of operating modes. In different operating modes, each pin has different functions. The pin configuration is shown in the following table:

Table 17-2 SPI ports configuration

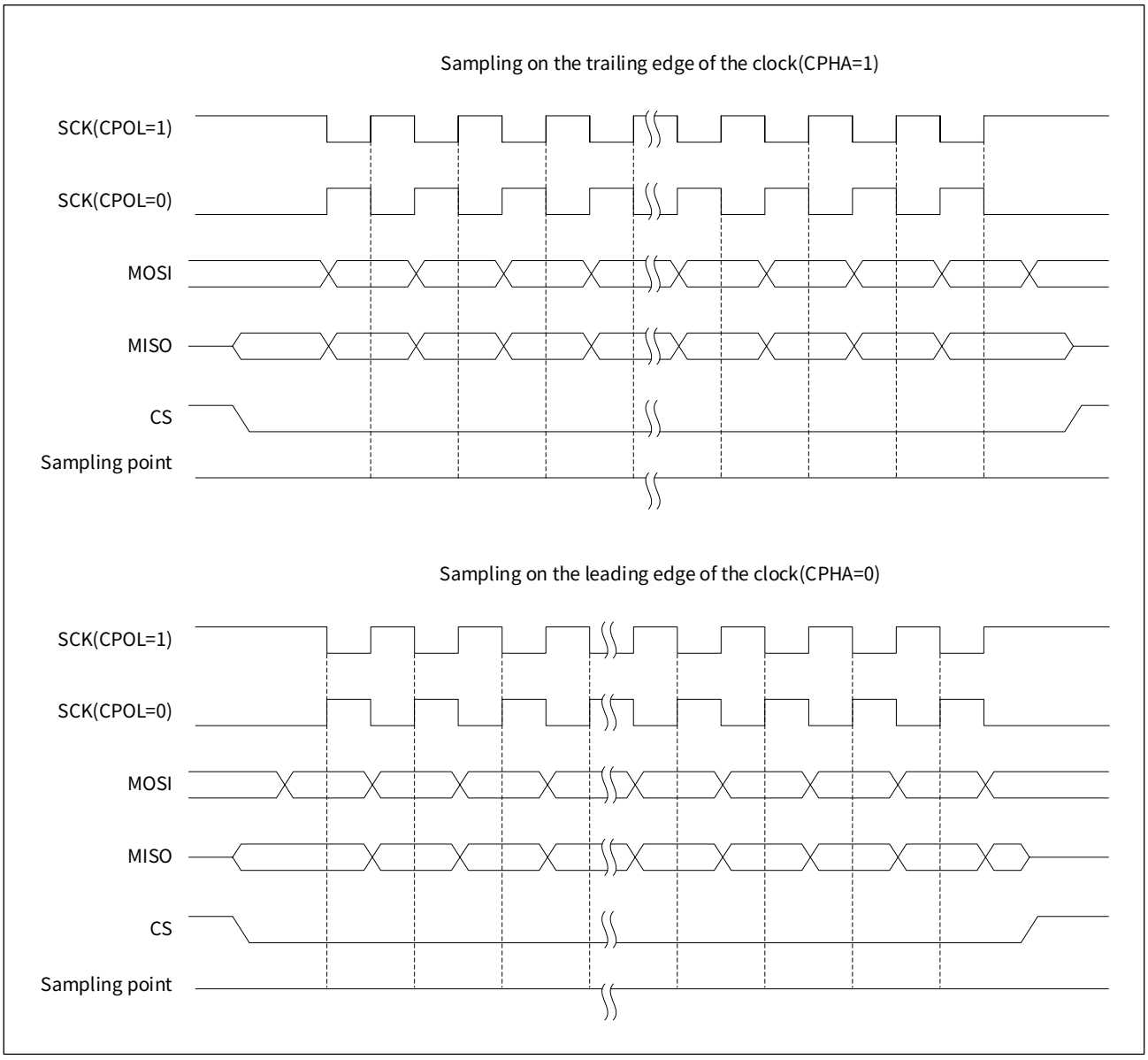
| SPI pins | Mode configuration | Function | GPIO configuration |
|---|---|---|---|
| SPI_SCK | Master | Serial clock output | Digital, push-pull output, multiplexed |
| | Slave | Serial clock input | Digital, floating input, multiplexed |
| SPI_MOSI | Full duplex/master | Master data output | Digital, push-pull output, multiplexed |
| | Full duplex/slave | Slave data input | Digital, floating input, multiplexed |
| | Single-wire half-duplex/ master | Master data transmitting and receiving | Digital, push-pull output, multiplexed |
| | Single-wire half-duplex/ slave | Do not use | Can be used as general purpose I/O |
| | Simplex single transmitting/master | Master data output | Digital, push-pull output, multiplexed |
| | Simplex single transmitting/slave | Do not use | Can be used as general purpose I/O |
| | Simplex single receiving/ master | Do not use | Can be used as general purpose I/O |
| | Simplex single receiving/ slave | Slave data input | Digital, floating input, multiplexed |
| SPI_MISO | Full duplex/master | Master data input | Digital, floating input, multiplexed |
| | Full duplex/slave | Slave data output | Digital, push-pull output, multiplexed |
| | Single-wire half-duplex/ master | Do not use | Can be used as general purpose I/O |
| | Single-wire half-duplex/ slave | Slave data transmission and reception | Digital, push-pull output, multiplexed |
| | Simplex single transmitting/master | Do not use | Can be used as general purpose I/O |
| | Simplex single transmitting/slave | Slave data output | Digital, push-pull output, multiplexed |
| | Simplex single receiving/ master | Master data output | Digital, push-pull output, multiplexed |
| | Simplex single receiving/ slave | Do not use | Can be used as general purpose I/O |
| SPI_CS | Master (SSM=0) | Slave select input | Digital, floating input, multiplexed |
| | Master (SSM=1) | Slave select output | Digital, push-pull output, multiplexed |
| | Slave (SSM=0) | Slave select input | Digital, floating input, multiplexed |
| | Slave (SSM=1) | Do not use | Can be used as general purpose I/O |

### 17.3.2 Communication timing and data format

#### Communication timing

The following figure shows the communication timing of SPI. The CS, SCK, and MOSI signals are all controlled by the master, and the MISO signal is the slave response signal. The slave selects the CS signal from high level to low level, which is the start signal of SPI communication. When the slave detects the start signal, it starts to communicate with the master. In each SCK clock cycle, the MOSI and MISO signal lines transmit one bit of data. CS changes from low level to high, which is the stop signal of communication, indicating the end of this communication.

Figure 17-2 SPI communication timing




#### Data frame format

The data frame width is configured by the WIDTH bit field of the control register SPI_CR1, which can be set to any data bit width from 4 to 16 bits.

The size of the data is configured by the LSBF bit field of the control register SPI_CR1, and the most significant bit first (MSB) or the least significant bit first (LSB) can be selected.

### Clock frequency

The synchronous serial clock SCK signal is controlled and generated by the SPI host, and its clock source is PCLK. The frequency division factor is set by configuring the BR bit field of the control register SPI_CR1, and the frequency division can be selected from 2 to 128. For SPI slaves, configuring SPI_CR1.BR has no effect.

### Clock polarity, clock phase

Clock polarity CPOL refers to the level state of the SCK serial clock line when the device is in an idle state without data transmission. Configure through the CPOL bit field of the control register SPI_CR1: set SPI_CR1. CPOL to 0, and the SCK clock line is low when it is idle; set SPI_CR1.CPOL to 1, and the SCK clock line is high when it is idle.

The clock phase CPHA refers to the sampling and shifting moment of the data. Configure through the CPHA of the control register SPI_CR1: set SPI_CR1.CPHA to 0, sample on the front edge of SCK (the clock edge when SCK changes from idle state to non-idle state), and the trailing edge (SCK changes from non-idle state to idle state). Clock edge) shift; set SPI_CR1.CPHA to 1, shift on the leading edge of SCK, and sample on the trailing edge.

According to the different configurations of clock polarity CPOL and clock phase CPHA, SPI can set 4 level modes. The master and slave need to be configured to the same level mode to ensure normal communication.

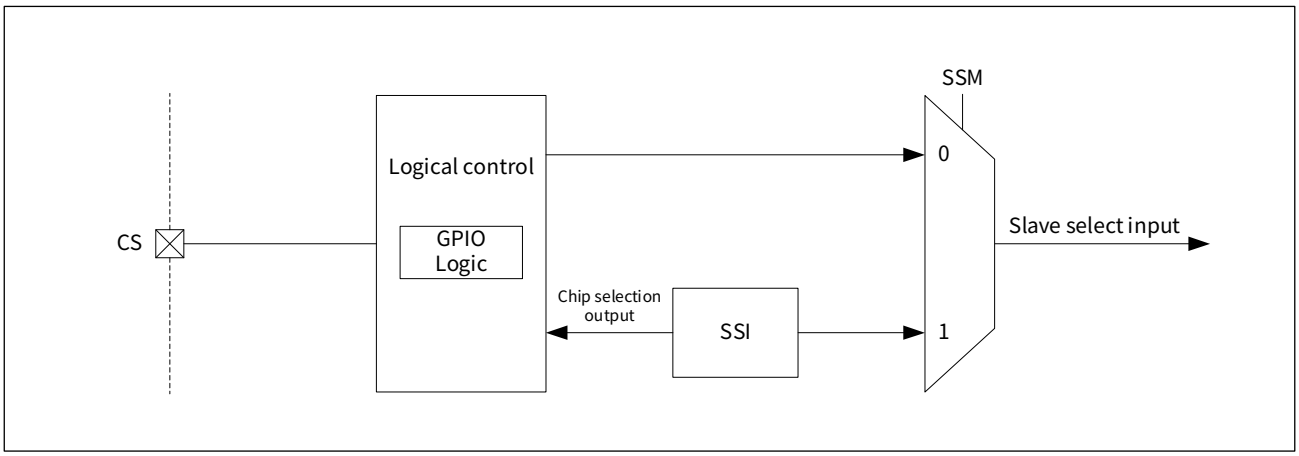
Table 17-3 SPI level mode

| CPOL | CPHA | SCK clock when idle | Sampling time |
|---|---|---|---|
| 0 | 0 | Low level | Front edge |
| 0 | 1 | Low level | Back edge |
| 1 | 0 | High level | Front edge |
| 1 | 1 | High level | Back edge |

### 17.3.3 Slave select pin CS

The following figure shows a schematic diagram of the configuration management of slave select pin CS. In master mode, CS can be selected as input or output. When used as input, it is used to detect bus conflicts in multi-master mode, and when used as output, it is used to generate slave chip select signals. In slave mode, CS acts as a device chip select input.

Figure 17-3 Slave select pin management




In master and slave mode, the slave select CS is managed by the SSM bit field of the control register SPI_CR1.

#### CS pin in master mode

Set SPI_CR1.SSM to 0, and CS is configured as an input to detect whether there is a bus conflict in the multi-computer communication system (Caution that the master needs to select the slave to communicate through other GPIOs at this time), please refer to section *17.3.7 Multi-machine communication*.

Setting SPI_CR1.SSM to 1, the slave select pin CS is configured as an output for generating the slave select signal. The CS pin output level is determined by the SSI bit field of the slave select register SPI_SSI: set SPI_SSI. SSI to 1, CS outputs a high level; set SPI_SSI.SSI to 0, CS outputs a low level.

#### CS pin in slave mode

Set SPI_CR1.SSM to 0 to use CS signal as device select. The CS level determines whether this unit is selected: when CS is low level, this unit is selected; when CS is high level, this unit is deselected.

Set SPI_CR1.SSM to 1, do not use CS as the device selection, the slave select is determined by the SSI bit field value of the SPI_SSI register. Set SPI_SSI.SSI to 0, this machine is selected; set SPI_SSI.SSI to 1, this machine is unchecked. In this configuration, the user program sets whether the machine is selected by software, and the CS pin can be used as a common IO.

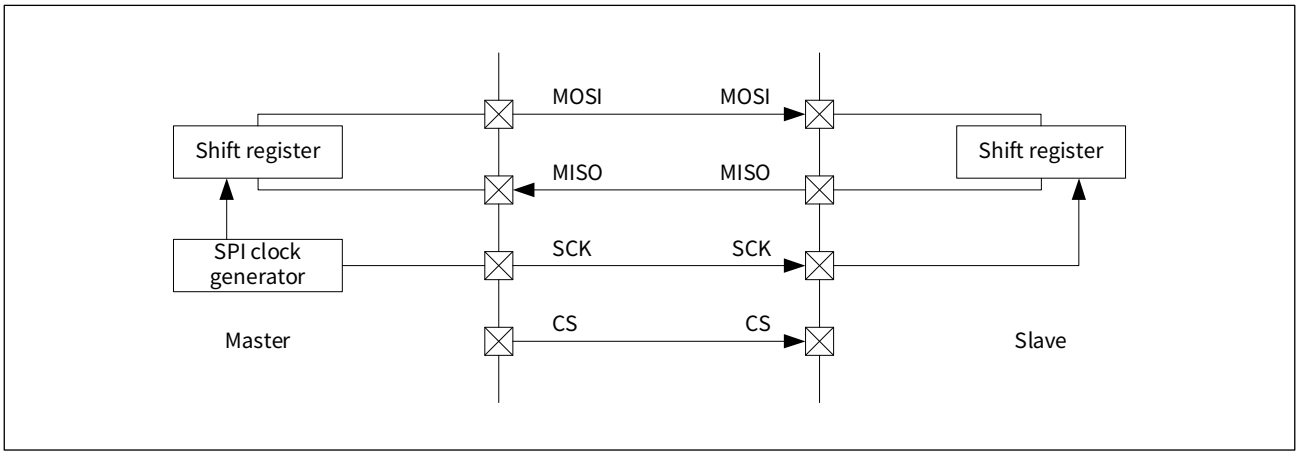
### 17.3.4 Full-duplex mode

SPI supports full-duplex communication mode. In this mode, the shift registers of the master and slave are connected through two unidirectional data lines, MOSI and MISO, and two lines of data are simultaneously performed under the control of the SCK clock signal provided by the host. transmission.

Set the MODE bit field of the control register SPI_CR1 to 0x0 to make the SPI work in full-duplex communication mode.

A typical application block diagram of full-duplex mode is shown in the following figure:

Figure 17-4 Full-duplex communication mode




#### Master transmits and receives

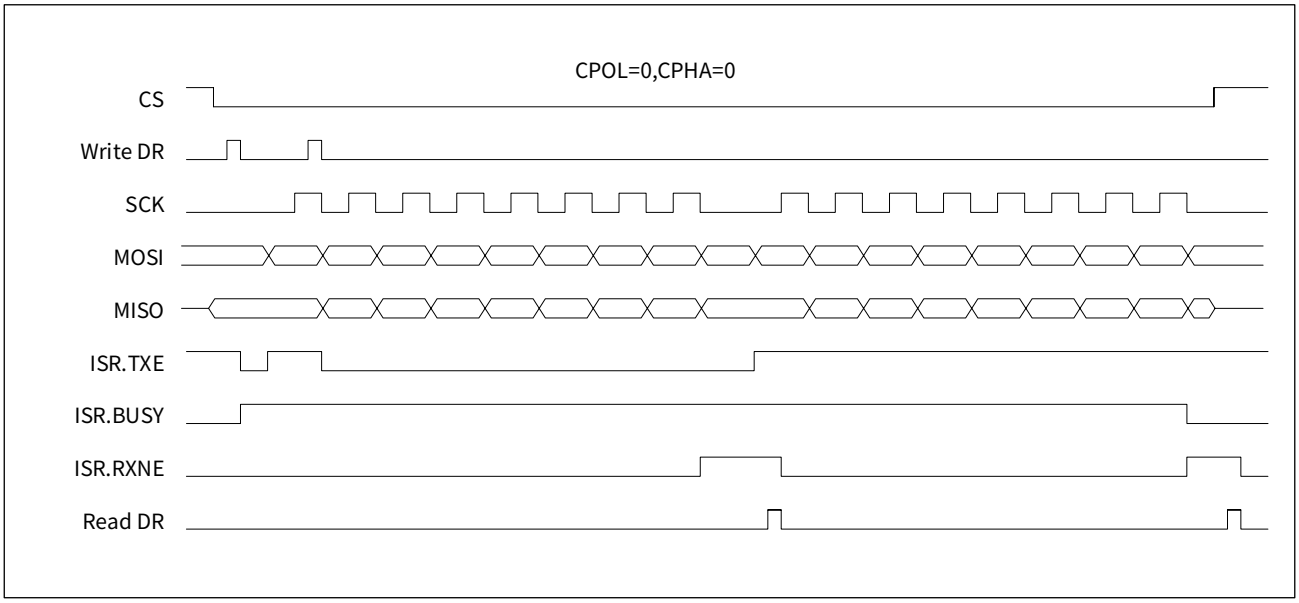
Set SPI_CR1.MSTR to 1, SPI works in master mode.

The master sets SPI_SSI.SSI to 0, and the slave selects the CS pin to output a low level as the communication start signal.

When the transmit buffer is empty, that is, the SPI_ISR.TXE flag is 1, write a frame of data to be sent into the SPI_DR register, the data is output from the MOSI pin under the control of the synchronous shift clock signal, and the MISO pin is simultaneously pin data is received into the shift register. At the end of transmitting a data frame, the non-empty flag bit SPI_ISR.RXNE of the receive buffer is set to 1 by hardware, indicating that a frame of data has been received, and the SPI_DR register can be read at this time.

Set SPI_SSI.SSI to 1, and the slave select pin CS outputs a high level to end this communication.

Figure 17-5 Master transmit and receive timing in full-duplex mode (two bytes)




### Slave transmits and receives

Set SPI_CR1.MSTR to 0, SPI works in slave mode.

Before the slave select CS signal is pulled low, the slave needs to set SPI_ICR.FLUSH to 0 to clear the transmit buffer and shift register, and write the first frame data to be transmitted into the SPI_DR register.

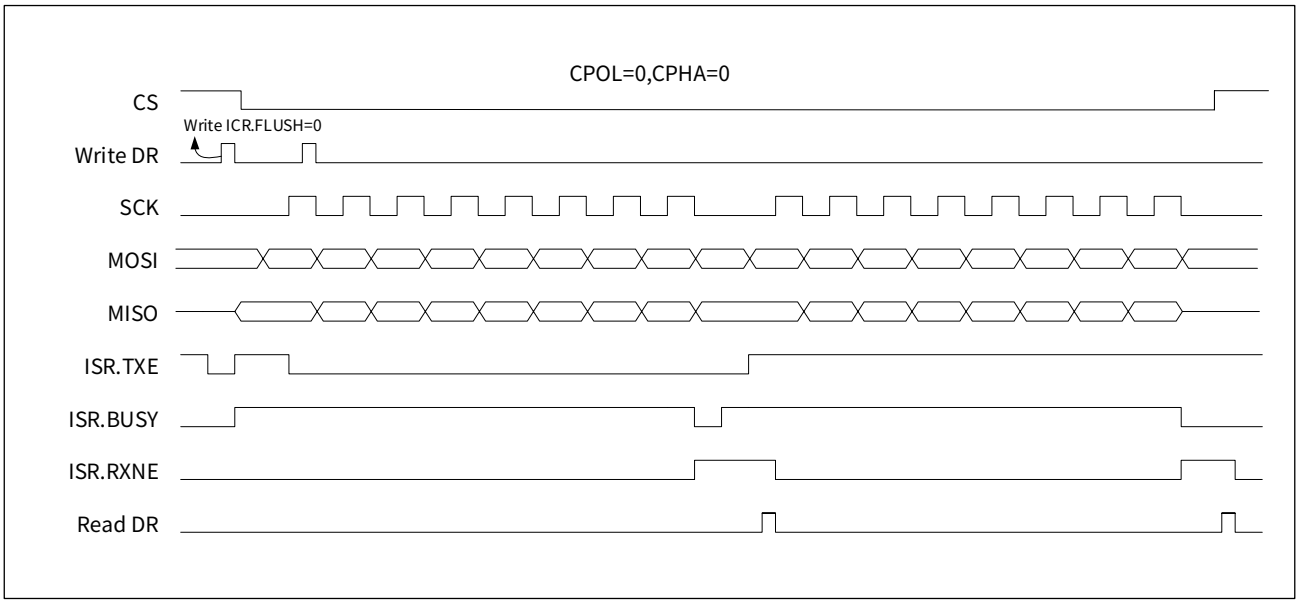
When the CS signal is pulled low, the written data will be output from the MISO pin under the control of the master's synchronous shift clock signal, and at the same time receive the data from the MOSI pin to the shift register.

If it is continuous communication of multiple data frames, the user should continuously query the SPI_ISR.TXE flag bit. Once the flag is 1, immediately write the data to be transmitted into the SPI_DR register to avoid data leakage.

When the receive buffer is not empty, that is, the SPI_ISR.RXNE flag is 1, indicating that a frame of data has been received, and the SPI_DR register can be read at this time.

This communication ends when it is detected that the CS pin changes to a high level.

Figure 17-6 Slave transmit and receive timing in full-duplex mode (two bytes)



### 17.3.5 Single-wire half-duplex mode

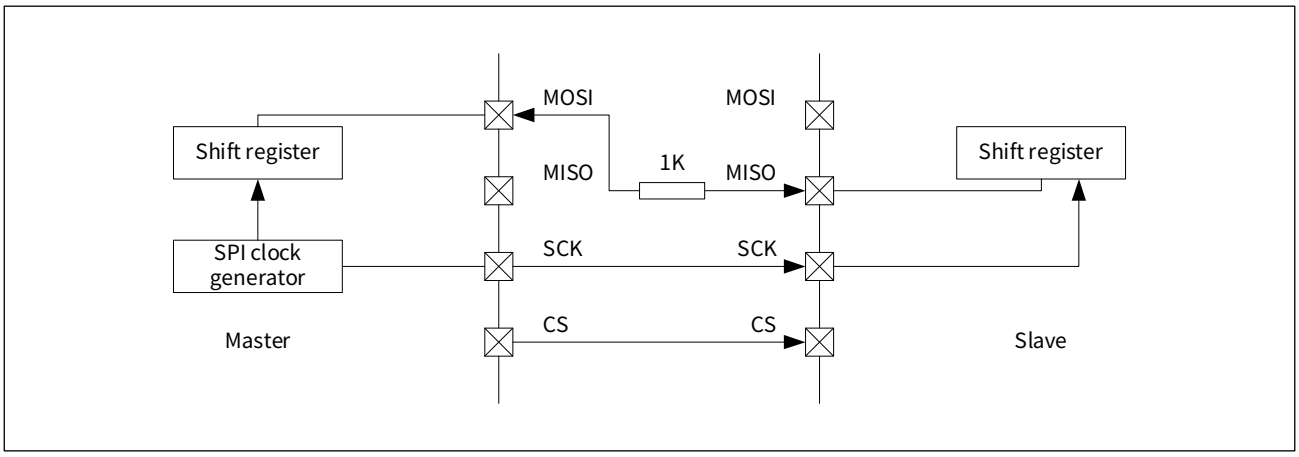
SPI supports single-wire half-duplex communication mode. In this mode, the master and slave communicate data through a bidirectional data line. The master uses MOSI, the slave uses MISO, and other unused SPI signal lines can be used by other functions.

Set the MODE bit field of the control register SPI_CR1 to 0x3 to make the SPI work in single-wire half-duplex mode.

The HDOE bit field of the control register SPI_CR2 selects the transmitting and receiving direction. When HDOE is 0, it is only received, and when HDOE is 1, it is only transmitted.

A typical application block diagram of single-wire half-duplex mode is shown in the following figure:

Figure 17-7 Single-wire half-duplex mode




During single-wire half-duplex communication, since the communication directions of the master and the slave cannot be guaranteed to switch synchronously, there may be a driving conflict and cause device damage. It is recommended to connect a resistor in series on the data line between the master and the slave to limit the current. .

When the SPI is used as a slave, the single-wire half-duplex mode can be used to communicate with a standard UART transceiver in synchronous mode. At this time, the SPI must be configured as a fixed level mode: the clock polarity CPOL is 1, and the clock phase is CPHA. is 1.

#### Master transmits

Set SPI_CR1.MSTR to 1, SPI works in master mode; set SPI_CR2.HDOE to 1, the master only transmits data.

The master sets SPI_SSI.SSI to 0, and outputs a low level on the slave selection pin CS as a communication start signal.

When the transmit buffer is empty, that is, the SPI_ISR.TXE flag bit is 1, a frame of data to be transmitted is written into the SPI_DR register, and the data is output from the MOSI pin under the control of the synchronous shift clock signal.

After writing the last frame of data, you must wait for the transmit buffer to be empty, that is, the SPI_ISR.TXE flag becomes 1, and the SPI_ISR.BUSY flag becomes 0 to ensure that the data is transmitted. Then set SPI_SSI. SSI to 1, select the CS pin to output a high level in the slave, and end this communication.

### Master receives

Set SPI_CR1.MSTR to 1, SPI works in master mode; set SPI_CR2.HDOE to 0, the master only receives data.

The master sets SPI_SSI.SSI to 0, and the slave selects the CS pin to output a low level as the communication start signal.

When the transmit buffer is empty, that is, the SPI_ISR.TXE flag is 1, write a frame of dummy data to the SPI_DR register to start the transmission.

When the receive buffer is not empty, that is, the SPI_ISR.RXNE flag is 1, indicating that a frame of data has been received, and the SPI_DR register can be read at this time.

If you want to receive multiple frames of data, repeat the above steps to write to SPI_DR and read the received data from SPI_DR.

When all data frames are received, set SPI_SSI.SSI to 1, select the CS pin to output a high level in the slave, and end this communication.

### Slave transmits

Set SPI_CR1.MSTR to 0, SPI works in slave mode; set SPI_CR2.HDOE to 1, the slave only transmits data.

Before the slave select CS signal is pulled low, the slave needs to set SPI_ICR.FLUSH to 0 to clear the transmit buffer and shift register, and write the first frame data to be transmitted into the SPI_DR register.

When the CS signal is pulled low, the written data will be output from the MISO pin under the control of the master's synchronous shift clock signal.

If it is continuous communication of multiple data frames, the user should continuously query the SPI_ISR.TXE flag bit. Once the flag is 1, immediately write the data to be transmitted into the SPI_DR register to avoid data leakage.

This communication ends when it is detected that the CS pin changes to a high level.

### Slave receives

Set SPI_CR1.MSTR to 0, SPI works in slave mode; set SPI_CR2.HDOE to 0, the slave only receives data.

When the CS pin is detected to go low, the slave starts to communicate with the master.

When the receive buffer is not empty, that is, the SPI_ISR.RXNE flag becomes 1, indicating that a frame of data has been received, and the SPI_DR register can be read at this time.

This communication ends when it is detected that the CS pin changes to a high level.

### 17.3.6 Simplex mode

SPI supports simplex communication mode, and the master and slave communicate with single transmitting or single receiving through a unidirectional data line.

The master uses the MOSI signal line for single transmitting communication, and the MISO signal line for single receiving communication; the slave uses the MOSI signal line for single receiving communication, and the MISO signal line for single transmitting communication, and the unused signal line can be used for other functions.

Set the MODE bit field of the control register SPI_CR1 to 0x1, and the SPI works in the simplex single transmitting communication mode; set the MODE bit field to 0x2, and the SPI works in the simplex single receiving communication mode.

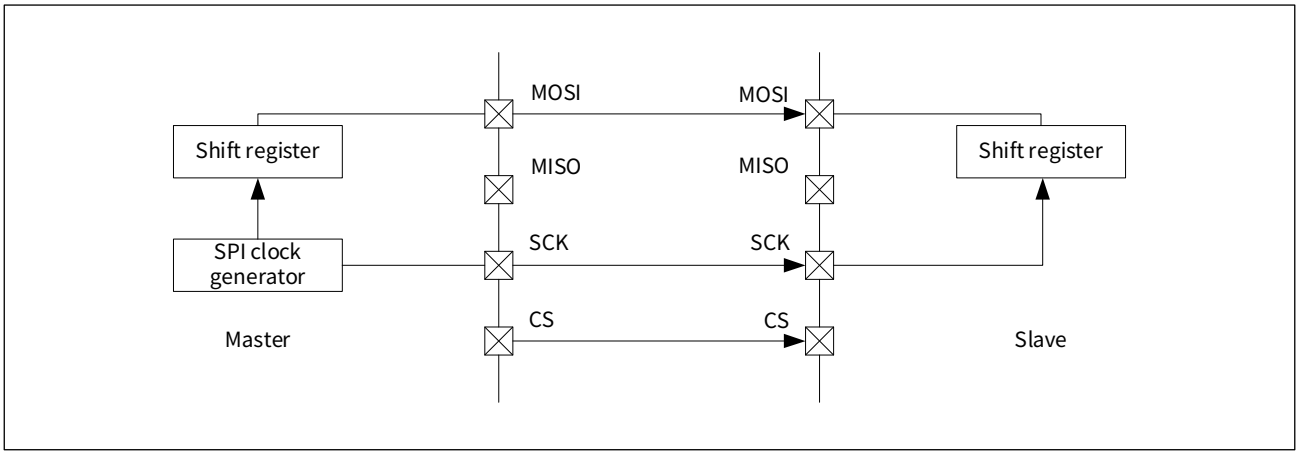
*Caution:*

*In simplex single receiving mode, all events related to the receive stream on the transmitter side, such as the receive buffer overflow error flag bit SPI_ISR.OV, must be ignored.*

#### 17.3.6.1 Master single transmitting/slave single receiving

In the application scenario of single transmitting by the master and single receiving of the slave, the master and the slave use a MOSI data line to communicate. The application block diagram is shown in the following figure:

Figure 17-8 Master single transmitting, slave single receiving mode




**Master single transmitting**

Set SPI_CR1.MODE to 0x1, SPI works in simplex single transmitting communication mode; set SPI_CR1.MSTR to 1, SPI works in master mode.

Set SPI_SSI.SSI to 0, and select the CS pin to output a low level in the slave as the communication start signal.

When the transmit buffer is empty, that is, the SPI_ISR.TXE flag bit is 1, a frame of data to be sent is written into the SPI_DR register, and the data is output from the MOSI pin under the control of the synchronous shift clock signal.

After writing the last frame of data, you must wait for transmit buffer empty flag SPI_ISR.TXE to become 1, and the SPI bus busy flag SPI_ISR.BUSY to become 0 to ensure that the data is transmitted. Then set SPI_SSI.SSI to 1, so that the slave selects the CS pin to output a high level, and ends this communication.

**Slave single receiving**

Set SPI_CR1.MODE to 0x2, SPI works in simplex single receiving communication mode; set SPI_CR1.MSTR to 0, SPI works in slave mode.

When the CS pin is detected to go low, the slave starts to communicate with the master.

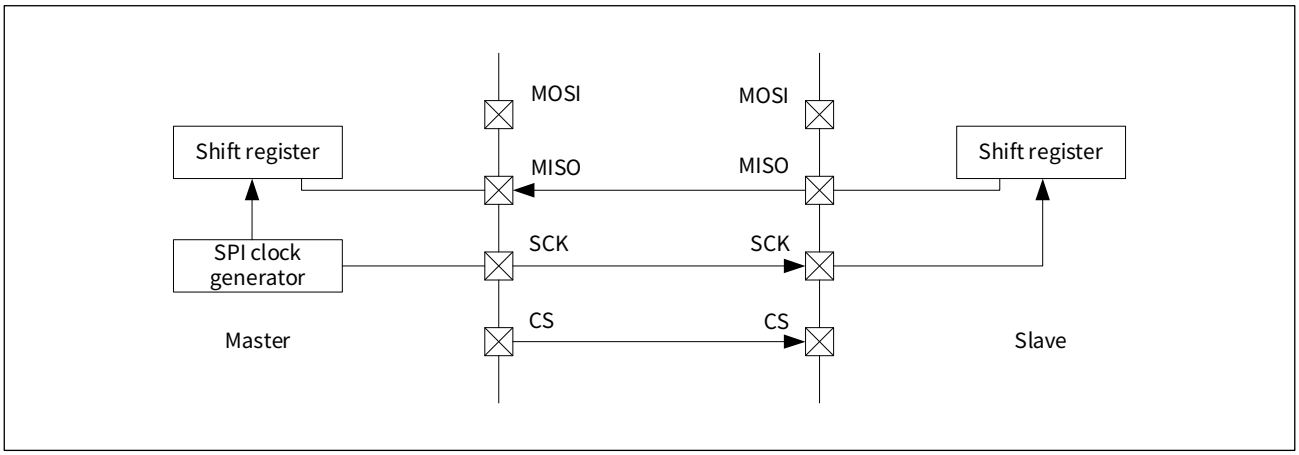
When the receive buffer is not empty, that is, the SPI_ISR.RXNE flag is 1, indicating that a frame of data has been received, and the SPI_DR register can be read at this time.

This communication ends when it is detected that the CS pin changes to a high level.

### 17.3.6.2 Master single receiving/slave single transmitting

In the application scenario of the master single receiving and the slave single transmitting, the master and the slave use a MISO data line to communicate. The application block diagram is shown in the following figure:

Figure 17-9 Master single receiving, slave single transmitting mode




#### Master single receiving

Set SPI_CR1.MODE to 0x2, SPI works in simplex single receiving communication mode; set SPI_CR1.MSTR to 1, SPI works in master mode.

Set SPI_SSI.SSI to 0, and select the CS pin to output a low level in the slave as the communication start signal.

When the transmit buffer is empty, that is, the SPI_ISR.TXE flag is 1, write a frame of dummy data to the SPI_DR register to start the transmission.

When the receive buffer is not empty, that is, the SPI_ISR.RXNE flag is 1, indicating that a frame of data has been received, and the SPI_DR register can be read at this time.

If you want to receive multiple frames of data, repeat the above steps to write to SPI_DR and read the received data from SPI_DR.

When all data frames are received, set SPI_SSI.SSI to 1, and the slave selects the CS pin to output a high level to end this communication.

#### Slave single transmitting

Set SPI_CR1.MODE to 0x1, SPI works in simplex single transmitting communication mode; set SPI_CR1.MSTR to 0, SPI works in slave mode.

Before the slave selection signal CS is pulled low, the slave needs to set SPI_ICR.FLUSH to 0 to clear the transmit buffer and shift register, and write the first frame data to be transmitted into the SPI_DR register.

When the CS signal is pulled low, the written data is output from the MISO pin under the control of the master's synchronous shift clock signal.

If it is continuous communication of multiple data frames, the user should continuously query the SPI_ISR.TXE flag bit. Once the flag is 1, immediately write the data to be transmitted into the SPI_DR register to avoid data leakage.

This communication ends when it is detected that the CS pin changes to a high level.

### 17.3.7 Multi-machine communication

SPI supports multi-machine communication mode. In this mode, the slave select CS pin of the master should be configured as an input and connected to the bus application signal of other masters to detect whether there is a conflict on the SPI bus.

If the slave select CS pin of a master is pulled to a low level, it means that another master is occupying the bus, the master will generate a mode error (please refer to section *17.3.9 Error flags*), the master SPI will be disabled, and the SPI will be disabled. Avoid SPI bus conflicts.

If the slave of a master selects the CS pin to detect a high level, it means that the SPI bus is idle. At this time, if the master wants to communicate with the slave, it must first apply for a signal on the bus to output a low level to other masters. CS signal, get SPI bus control, and then select a slave to start communication.

*Figure 17-10 Multi-machine communication system* shows a typical multi-machine communication system. Taking the communication between master A and slave 1 as an example, the operation process is as follows:

Step 1: Master A detects the CS pin until a high level occurs;

Step 2: GPIO0 of master A outputs a low level to obtain bus control, and master B is disabled;

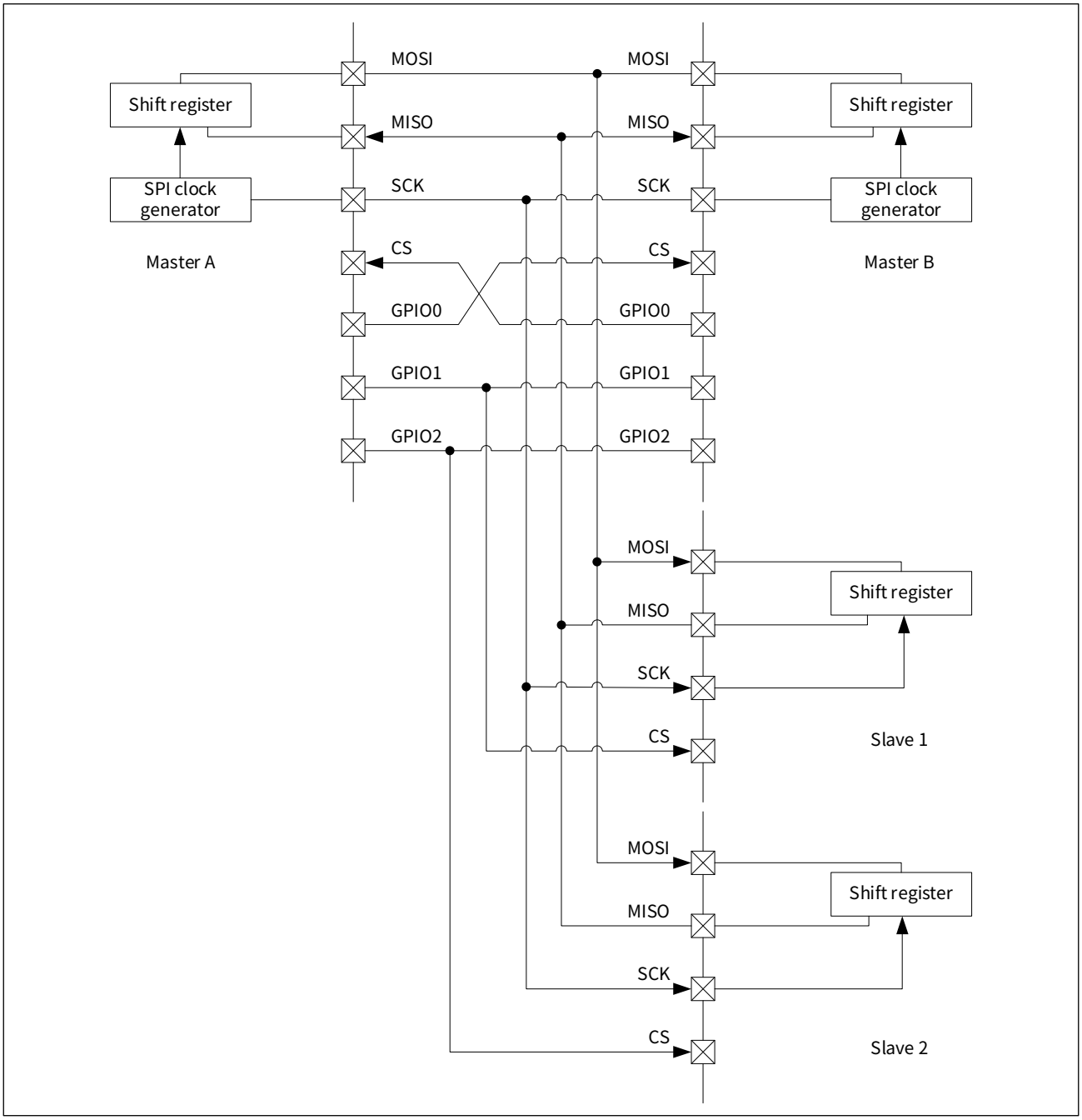
Step 3: GPIO1 of master A outputs low level, and slave 1 is selected;

Step 4: The master A communicates with the slave 1;

Step 5: GPIO1 of master A outputs a high level, releasing slave 1;

Step 6: GPIO0 of master A outputs a high level, releasing the SPI bus.

Figure 17-10 Multi-machine communication system




In order to ensure the communication reliability in the multi-machine communication mode, the MISOHD bit field of the slave control register SPI_CR1 should be set to 1. When the slave is selected, the MISO pin is CMOS output; when it is not selected, the MISO pin is high resistance output.

### 17.3.8 Status flags

There are 6 status flags in the SPI controller to indicate the general working status of the SPI.

**Transmit buffer empty flag (SPI_ISR.TXE)**

When the transmit data is transferred from the SPI_DR register to the shift register, the SPI_ISR.TXE flag will be set by hardware, indicating that the transmit buffer is empty, and new data to be transmitted is allowed to be written to the SPI_DR register. While writing data to the SPI_DR register, the SPI_ISR.TXE flag will be automatically cleared.

**Receive buffer not empty flag (SPI_ISR.RXNE)**

When the received data is transferred from the shift register to the SPI_DR register, the SPI_ISR.RXNE flag will be set by hardware, indicating that the reception of a frame of data has been completed. At this time, the SPI_DR register can be read, and the SPI_ISR will be cleared while reading the SPI_DR register. RXNE flag bit.

Users can also manually clear the SPI_ISR.RXNE flag by software.

**Bus busy flag bit (SPI_ISR.BUSY)**

The SPI_ISR.BUSY flag bit, set and cleared by hardware, is used to indicate the current SPI communication status.

When preparing or performing data transmission (there is data in the transmit buffer or data in the shift register), the BUSY flag will be set by hardware, indicating that the SPI interface is in a busy state;

When there is no data transmission (no data in the send buffer and no data in the shift register), the BUSY flag is automatically cleared by hardware, indicating that the SPI interface is in an idle state.

**Slave select signal status (SPI_ISR.SSLVL)**

The SPI_ISR.SSLVL status bit, set and cleared by hardware, indicates the level status of the slave select input signal.

SPI_ISR.SSLVL is 0, the slave select CS input is low, and the slave is selected; SPI_ISR.SSLVL is 1, the slave select CS input is high, and the slave is not selected.

**Slave select input rising edge flag (SPI_ISR.SSR)**

The SPI_ISR.SSR flag bit is used to indicate whether the slave select CS input signal has a rising edge.

SPI_ISR.SSR is 0, indicating that the slave select CS input has no rising edge; SPI_ISR.SSR is 1, indicating that the slave select input has a rising edge.

**Slave select input falling edge flag (SPI_ISR.SSF)**

The SPI_ISR.SSF flag bit is used to indicate whether the slave select CS input signal has a falling edge.

SPI_ISR.SSF is 0, indicating that the slave select CS input has no falling edge; SPI_ISR.SSF is 1, indicating that the slave select CS input has a falling edge.

### 17.3.9 Error flags

There are 4 error flags in the SPI controller to indicate whether an error has occurred in the SPI.

#### Mode error flag (SPI_ISR.MODF)

SPI_ISR.MODF flag bit, used to detect bus conflict in SPI multi-machine communication mode.

When the SPI works in master mode and the slave select CS pin is configured as an input, if the CS pin input is low, a mode error will occur, and the SPI_ISR.MODF flag is set by hardware, indicating that other SPI masters are occupying the bus at this time.

After a mode error occurs, SPI_CR1.EN is cleared and the SPI is forced to shut down.

Setting SPI_ICR.MODF to 0 can clear the SPI_ISR.MODF flag. After clearing, the user needs to reconfigure the SPI controller.

After a mode error occurs, if the SPI controller is re-enabled, the SPI can still perform normal data transmission, but the SPI_ISR.MODF flag will remain set until cleared by user software.

#### Slave select error flag in slave mode (SPI_ISR.SSERR)

When the SPI is working in slave mode and a valid data transfer is in progress, if the slave select CS input is pulled high, a slave select error will occur, and the SPI_ISR.SSERR flag will be set by hardware.

The occurrence of a slave select error will cause an error in the current data transmission, and the slave will enter the unselected state.

After a slave select error occurs, if the slave select CS input is pulled low again, the SPI can still perform normal data transmission, but the SPI_ISR.SSERR flag will remain set until cleared by user software.

#### Receive buffer overflow error flag (SPI _ISR.OV)

When the master or slave does not clear the SPI_ISR.RXNE flag after receiving a frame of data, that is, neither read the SPI_DR register nor clear the SPI_ICR.RXNE to 0 by software, and and then when a new frame of data is received, a receive buffer overflow error will occur, and the SPI_ISR.OV flag is set by hardware.

After a receive buffer overflow error occurs, the newly received data will overwrite the original data in the buffer.

After a receive buffer overflow error occurs, the SPI can still perform normal data transmission, but the SPI_ISR.OV flag will remain set until cleared by user software.

#### Transmit buffer underflow error flag in slave mode (SPI_ISR.UD)

When the SPI works in slave mode, if the transmit buffer is empty and the transmission of a new frame has already started, a transmit buffer underflow error will occur, and the SPI_ISR.UD flag will be set by hardware, indicating that the slave did not send the data in time. The SPI_DR register writes data and misses transmitting data to the host.

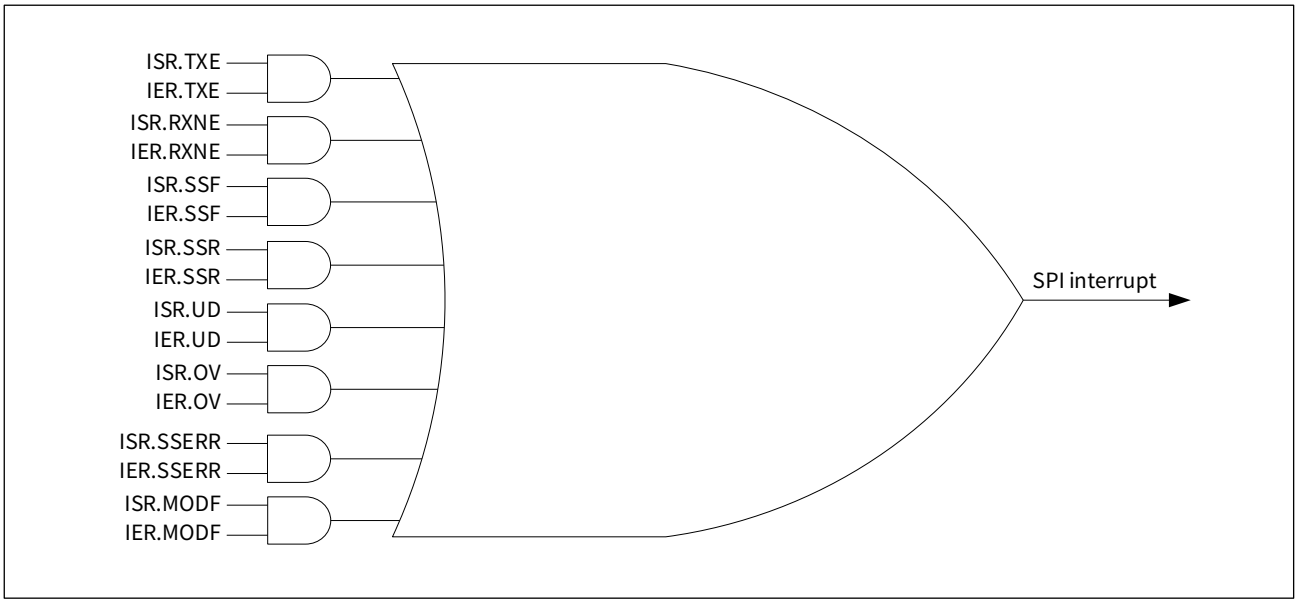
After the transmit buffer underflow error occurs, the SPI can still perform normal data transmission, but the SPI_ISR.UD flag will remain set until cleared by user software.

## 17.4 SPI interrupts

The SPI controller supports 8 interrupt sources. When the SPI interrupt trigger event occurs, the interrupt flag bit will be set by hardware. If the corresponding interrupt enable control bit is set, an interrupt request will be generated.

An SPI module of CW32F003 uses a same system SPI interrupt, whether the SPI interrupt generates an interrupt jump is controlled by the corresponding bit of the interrupt enable setting register NVIC_ISER. The schematic diagram of the system SPI interrupt is shown in the following figure:

Figure 17-11 SPI interrupts




In the user's SPI interrupt service routine, the relevant SPI interrupt flag bit should be queried for corresponding processing. Before exiting the interrupt service routine, the interrupt flag bit should be cleared to avoid repeatedly entering the interrupt routine.

The flag bit, interrupt enable bit, interrupt flag clear bit and clear method of each interrupt source of SPI are shown in the following table:

Table 17-4 SPI interrupt control

| Interrupt event | Interrupt flag bit | Interrupt enable bit | Flag clear method |
|---|---|---|---|
| Transmit buffer empty | ISR.TXE | IER.TXE | Write SPI_DR register |
| Receive buffer not empty | ISR.RXNE | IER.RXNE | Read SPI_DR register, or write 0 to ICR.RXNE |
| Falling edge appears on the slave select input | ISR.SSF | IER.SSF | Write 0 to ICR.SSF |
| Rising edge appears on the slave select input | ISR.SSR | IER.SSR | Write 0 to ICR.SSR |
| Transmit buffer underflow in slave mode | ISR.UD | IER.UD | Write 0 to ICR.UD |
| Receive buffer overflow | ISR.OV | IER.OV | Write 0 to ICR.OV |
| Slave select error in slave mode | ISR.SSERR | IER.SSERR | Write 0 to ICR.SSERR |
| Mode error | ISR.MODF | IER.MODF | Write 0 to ICR.MODF |

## 17.5 Programming examples

### 17.5.1 Full duplex/master mode

#### 17.5.1.1 Transmit and receive in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.SPI to 1, and enable the GPIO clock and SPI working clock corresponding to the SPI pin;

Step 2: Configure the SPI_SCK, SPI_MOSI and SPI_CS pins as push-pull multiplexed output mode, and configure the SPI_MISO pin as floating input and multiplexed mode. For specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set SPI_CR1.MODE to 0x0, and configure SPI to be bidirectional full-duplex communication mode;

Step 4: Set SPI_CR1.MSTR to 1, and configure SPI to master mode;

Step 5: Configure SPI_CR1.WIDTH and set the width of each frame of data;

Step 6: Configure SPI_CR1.LSBF and set the data bit sending and receiving sequence;

Step 7: Configure SPI_CR1.CPOL and set the clock polarity;

Step 8: Configure SPI_CR1.CPHA and set the clock phase;

Step 9: Set SPI_CR1.SSM to 1, and determine the slave select output value through the SSI register;

Step 10: Configure SPI_CR1.BR and set the SCK baud rate;

Step 11: Set SPI_CR1.EN to 1 to enable the SPI controller;

Step 12: Set SPI_SSI.SSI to 0, communication starts;

Step 13: Query and wait for the SPI_ISR.TXE flag to be set to 1, and confirm that the transmit buffer is empty;

Step 14: Write a frame of data to be transmitted into the SPI_DR register;

Step 15: Query and wait for the SPI_ISR.RXNE flag to be set to 1 to confirm that one frame of data has been received;

Step 16: Read the SPI_DR register and save the data;

Step 17: Repeat steps 13-17 to send and receive the next frame of data until all data is transmitted and received;

Step 18: Set SPI_SSI.SSI to 1, communication ends.

#### 17.5.1.2 Transmit and receive in interrupt mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.SPI to 1, and enable the GPIO clock and SPI working clock corresponding to the SPI pin;

Step 2: Configure the SPI_SCK, SPI_MOSI and SPI_CS pins as push-pull multiplexed output mode, and configure the SPI_MISO pin as floating input and multiplexed mode. For specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set SPI_CR1.MODE to 0x0, and configure SPI to be bidirectional full-duplex communication mode;

Step 4: Set SPI_CR1.MSTR to 1, and configure SPI to master mode;

Step 5: Configure SPI_CR1.WIDTH and set the width of each frame of data;

Step 6: Configure SPI_CR1.LSBF and set the data bit sending and receiving sequence;

Step 7: Configure SPI_CR1.CPOL and set the clock polarity;

Step 8: Configure SPI_CR1.CPHA and set the clock phase;

Step 9: Set SPI_CR1.SSM to 1, and determine the slave selection output value through the SSI register;

Step 10: Configure SPI_CR1.BR and set the SCK baud rate;

Step 11: Set SPI_CR1.EN to 1 to enable the SPI controller;

Step 12: Write 0x00 to SPI_ICR to clear all flags;

Step 13: Configure the NVIC controller, please refer to chapter *5 Interrupt*;

Step 14: Set SPI_SSI.SSI to 0, communication starts;

Step 15: Set SPI_IER.RXNE to 1 to enable receive buffer non-empty interrupt;

Step 16: Set SPI_IER.TXE to 1 to enable transmit buffer empty interrupt;

Step 17: Enter the interrupt service function: query and judge the SPI_ISR.TXE flag bit, if the flag bit is 1, write a frame of data to be transmitted into the SPI_DR register; query and judge the SPI_ISR.RXNE flag bit, if the flag bit is 1, then read the SPI_DR register and save the data;

Step 18: Set SPI_SSI.SSI to 1, communication ends.

### 17.5.2 Single-wire half-duplex/master mode

#### 17.5.2.1 Transmit in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.SPI to 1, and enable the GPIO clock and SPI working clock corresponding to the SPI pin;

Step 2: Configure the SPI_SCK, SPI_MOSI and SPI_CS pins as push-pull multiplexed output mode. For specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set SPI_CR1.MODE to 0x3, and configure SPI to be single-wire half-duplex communication mode;

Step 4: Set SPI_CR1.MSTR to 1, and configure SPI to master mode;

Step 5: Configure SPI_CR1.WIDTH and set the width of each frame of data;

Step 6: Configure SPI_CR1.LSBF and set the data bit transmitting and receiving sequence;

Step 7: Configure SPI_CR1.CPOL and set the clock polarity;

Step 8: Configure SPI_CR1.CPHA and set the clock phase;

Step 9: Set SPI_CR1.SSM to 1, and determine the slave select output value through the SSI register;

Step 10: Configure SPI_CR1.BR and set the SCK baud rate;

Step 11: Set SPI_CR1.EN to 1 to enable the SPI controller;

Step 12: Set SPI_SSI.SSI to 0, communication starts;

Step 13: Set SPI_CR2.HDOE to 1, only transmit;

Step 14: Query and wait for the SPI_ISR.TXE flag to be set to 1, and confirm that the transmit buffer is empty;

Step 15: Write a frame of data to be transmitted into the SPI_DR register;

Step 16: If there is still data to be transmitted, repeat steps 14-16 to transmit the next frame of data;

Step 17: Query and wait for the SPI_ISR.TXE flag bit to be 1, and the last frame of data starts to be transmitted;

Step 18: Query and wait for the SPI_ISR.BUSY flag to become 0, and the SPI bus is idle;

Step 19: Set SPI_SSI.SSI to 1, communication ends.

#### 17.5.2.2 Receive in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.SPI to 1, and enable the GPIO clock and SPI working clock corresponding to the SPI pin;

Step 2: Configure the SPI_SCK, SPI_MOSI and SPI_CS pins as push-pull multiplexed output mode. For specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set SPI_CR1.MODE to 11, and configure SPI to be single-wire half-duplex communication mode;

Step 4: Set SPI_CR1.MSTR to 1, and configure SPI to master mode;

Step 5: Configure SPI_CR1.WIDTH and set the width of each frame of data;

Step 6: Configure SPI_CR1.LSBF and set the data bit transmitting and receiving sequence;

Step 7: Configure SPI_CR1.CPOL and set the clock polarity;

Step 8: Configure SPI_CR1.CPHA and set the clock phase;

Step 9: Set SPI_CR1.SSM to 1, and determine the slave select output value through the SSI register;

Step 10: Configure SPI_CR1.BR and set the SCK baud rate;;

Step 11: Set SPI_CR1.EN to 1 to enable the SPI controller;

Step 12: Set SPI_SSI.SSI to 0, communication starts;

Step 13: Set SPI_CR2.HDOE to 0, only receive;

Step 14: Query and wait for the SPI_ISR.TXE flag to be set to 1, and confirm that the transmit buffer is empty;

Step 15: Write a frame of dummy data into the SPI_DR register;

Step 16: Query and wait for the SPI_ISR.RXNE flag to be set to 1 to confirm that one frame of data has been received;

Step 17: Read the SPI_DR register and save the data;

Step 18: Repeat steps 14-18 to receive the next frame of data until all data is received;

Step 19: Set SPI_SSI.SSI to 1, communication ends.

### 17.5.3 Simplex mode/master mode

#### 17.5.3.1 Transmit in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.SPI to 1, and enable the GPIO clock and SPI working clock corresponding to the SPI pin;

Step 2: Configure the SPI_SCK, SPI_MOSI and SPI_CS pins as push-pull multiplexed output mode. For specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set SPI_CR1.MODE to 01, and configure SPI as simplex single-transmit communication mode;

Step 4: Set SPI_CR1.MSTR to 1, and configure SPI to master mode;

Step 5: Configure SPI_CR1.WIDTH and set the width of each frame of data;

Step 6: Configure SPI_CR1.LSBF and set the data bit transmitting and receiving sequence;

Step 7: Configure SPI_CR1.CPOL and set the clock polarity;

Step 8: Configure SPI_CR1.CPHA and set the clock phase;

Step 9: Set SPI_CR1.SSM to 1, and determine the slave select output value through the SSI register;

Step 10: Configure SPI_CR1.BR and set the SCK baud rate;

Step 11: Set SPI_CR1.EN to 1 to enable the SPI controller;

Step 12: Set SPI_SSI.SSI to 0,communication starts;

Step 13: Query and wait for the SPI_ISR.TXE flag to be set to 1, and confirm that the transmit buffer is empty;

Step 14: Write a frame of data to be transmitted into the SPI_DR register;

Step 15: If there is still data to be transmitted, repeat steps 13-15 to transmit the next frame of data;

Step 16: Query and wait for the SPI_ISR.TXE flag bit to be 1, and the last frame of data starts to be transmitted;

Step 17: Query and wait for the SPI_ISR.BUSY flag to become 0, and the SPI bus is idle;

Step 18: Set SPI_SSI.SSI to 1, communication ends.

#### 17.5.3.2 Receive in query mode

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBENx.SPI to 1, and enable the GPIO clock and SPI working clock corresponding to the SPI pin;

Step 2: Configure the SPI_SCK and SPI_CS pins as push-pull multiplexing output mode, and configure the SPI_MISO pin as floating input and multiplexing mode. For specific register configuration steps, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set SPI_CR1.MODE to 10, and configure SPI as simplex single receiving communication mode;

Step 4: Set SPI_CR1.MSTR to 1, and configure SPI to master mode;

Step 5: Configure SPI_CR1.WIDTH and set the width of each frame of data;

Step 6: Configure SPI_CR1.LSBF and set the data bit transmitting and receiving sequence;

Step 7: Configure SPI_CR1.CPOL and set the clock polarity;

Step 8: Configure SPI_CR1.CPHA and set the clock phase;

Step 9: Set SPI_CR1.SSM to 1, and determine the slave select output value through the SSI register;

Step 10: Configure SPI_CR1.BR and set the SCK baud rate;

Step 11: Set SPI_CR1.EN to 1 to enable the SPI controller;

Step 12: Set SPI_SSI.SSI to 0, communication starts;

Step 13: Query and wait for the SPI_ISR.TXE flag to be set to 1, and confirm that the transmit buffer is empty;

Step 14: Write a frame of dummy data into the SPI_DR register;

Step 15: Query and wait for the SPI_ISR.RXNE flag to be set to 1 to confirm that one frame of data has been received;

Step 16: Read the SPI_DR register and save the data;

Step 17: Repeat steps 13-17 to receive the next frame of data until all data is received;

Step 18: Set SPI_SSI.SSI to 1, communication ends.

## 17.6 List of registers

SPI base address: SPI_BASE = 0x4001 3000

Table 17-5 List of SPI registers

| Register name | Register address | Register description |
|---|---|---|
| SPI_CR1 | SPI_BASE + 0x00 | Control register 1 |
| SPI_CR2 | SPI_BASE + 0x08 | Control register 2 |
| SPI_SSI | SPI_BASE + 0x0C | Slave select register |
| SPI_IER | SPI_BASE + 0x04 | Interrupt enable register |
| SPI_ISR | SPI_BASE + 0x10 | Interrupt flag register |
| SPI_ICR | SPI_BASE + 0x14 | Interrupt flag clear register |
| SPI_DR | SPI_BASE + 0x18 | Data register |

## 17.7 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 17.7.1 SPI_CR1 control register 1

Address offset: 0x00 Reset value: 0x0000 1C04

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:19 | RFU | - | Reserved bits, please keep the default value |
| 18 | MISOHD | RW | Slave MISO output configuration<br>0: MISO is always output for COMS<br>1: When the slave is selected, MISO is COMS output, and when it is not selected, it is high resistance output<br>Caution: This bit needs to remain 0 when SSM is set to 1 in slave mode. |
| 17:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | MODE | RW | Communication mode configuration<br>00: bidirectional full duplex<br>01: simplex single transmitting<br>10: simplex single receiving<br>11: single-wire half-duplex |
| 13:10 | WIDTH | RW | The width of each frame of data is WIDTH+1, for example<br>0011: 4bit<br>0100: 5bit<br>…<br>1110: 15bit<br>1111: 16bit |
| 9 | SSM | RW | Slave select configuration (Master mode)<br>0: The SPI_CS pin is in input mode, and the low level of the pin can select the machine<br>1: The SPI_CS pin is in output mode, and the output level comes from SSI<br>Slave select configuration (slave mode)<br>0: SPI_CS pin level determines whether the machine is selected<br>1: The value of the SSI register determines whether the machine is selected |
| 8 | SMP | RW | Master mode delayed sampling<br>0: In the traditional way of SPI, the data received by the master is sampled according to the settings of CPOL and CPHA<br>1: Delayed sampling method, the sampling time is delayed by half SCK cycle compared with the traditional method |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 7 | LSBF | RW | Data frame high and low sequence selection<br>0: Most significant bit MSB is sent and received first<br>1: Least significant bit MSB is sent and received first |
| 6 | EN | RW | Enable control<br>0: SPI module disabled<br>1: SPI module enabled |
| 5:3 | BR | RW | Master mode SCK baud rate configuration<br>000: PCLK/2<br>001: PCLK/4<br>010: PCLK/8<br>011: PCLK/16<br>100: PCLK/32<br>101: PCLK/64<br>110: PCLK/128<br>111: reserve |
| 2 | MSTR | RW | Work mode configuration<br>0: slave mode<br>1: master mode |
| 1 | CPOL | RW | Serial clock polarity configuration<br>0: Low level in standby<br>1: High level in standby |
| 0 | CPHA | RW | Serial clock phase configuration<br>0: Leading edge sampling/trailing edge shift<br>1: Leading edge shift/trailing edge sampling |

### 17.7.2 SPI_CR2 control register 2

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:1 | RFU | - | Reserved bits, please keep the default value |
| 0 | HDOE | RW | When single-wire half-duplex, data transmission/reception status control<br>0: Only received<br>1: Only transmitted<br>Caution: Only valid for single-wire half-duplex communication |

### 17.7.3 SPI_SSI slave select register

Address offset: 0x0C Reset value: 0x0000 0001

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:1 | RFU | - | Reserved bits, please keep the default value |
| 0 | SSI | RW | Slave select, valid when SPI_CR1.SSM is 1 |

| SSM | MSTR | SSI | Description |
|---|---|---|---|
| 0 | - | - | Invalid |
| 1 | 0(slave) | 0 | Set the machine to selected state |
| | | 1 | Set this machine to unselected state |
| | 1(master) | 0 | SPI_CS pins output low |
| | | 1 | SPI_CS pins output high |

### 17.7.4 SPI_IER interrupt enable register

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | MODF | RW | Mode error interrupt enable control<br>0: Disabled<br>1: Enabled |
| 6 | SSERR | RW | Slave select error interrupt enable control in slave mode<br>0: Disabled<br>1: Enabled |
| 5 | OV | RW | Receive buffer overflow error interrupt enable control<br>0: Disabled<br>1: Enabled |
| 4 | UD | RW | Transmit buffer underflow error interrupt enable control in slave mode<br>0: Disabled<br>1: Enabled |
| 3 | SSR | RW | Slave select input rising edge interrupt enable control<br>0: Disabled<br>1: Enabled |
| 2 | SSF | RW | Slave select input falling edge interrupt enable control<br>0: Disabled<br>1: Enabled |
| 1 | RXNE | RW | Receive buffer non-empty interrupt enable control<br>0: Disabled<br>1: Enabled |
| 0 | TXE | RW | Transmit buffer empty interrupt enable control<br>0: Disabled<br>1: Enabled |

### 17.7.5 SPI_ISR interrupt flag register

Address offset: 0x10 Reset value: 0x0000 0001

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | SSLVL | RO | Slave select signal state<br>0: Slave select signal is low level<br>1: Slave select signal is high level |
| 8 | BUSY | RO | Bus busy flag<br>0: bus idle<br>1: bus busy |
| 7 | MODF | RO | Mode error flag<br>0: normal<br>1: error<br>Caution: In master mode, the slave select input is low to trigger MODF |
| 6 | SSERR | RO | Slave select error flag in slave mode<br>0: normal<br>1: error |
| 5 | OV | RO | Receive buffer overflow error flag bit<br>0: normal<br>1: error |
| 4 | UD | RO | Transmit buffer underflow error flag in slave mode<br>0: normal<br>1: error |
| 3 | SSR | RO | Slave select input rising edge flag bit<br>0: No rising edge occurred<br>1: Rising edge has occurred |
| 2 | SSF | RO | Slave select input falling edge flag bit<br>0: No falling edge occurred<br>1: Falling edge occurred |
| 1 | RXNE | RO | Receive buffer not empty flag bit<br>0: Receive buffer is empty<br>1: Receive buffer is not empty<br>Caution: This flag can be cleared by reading the DR register or writing 0 to ICR.RXNE |
| 0 | TXE | RO | Send buffer empty flag bit<br>0: Send buffer is not empty<br>1: Send buffer is empty |

### 17.7.6 SPI_ICR interrupt flag clear register

Address offset: 0x14 Reset value: 0x0000 00FF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | MODF | R1W0 | Mode error flag cleared<br>W0: Mode error flag cleared<br>W1: No function |
| 6 | SSERR | R1W0 | Slave select error flag cleared in slave mode<br>W0: Slave select error flag in slave mode cleared<br>W1: No function |
| 5 | OV | R1W0 | Receive buffer overflow error flag cleared<br>W0: Receive buffer overflow error flag cleared<br>W1: No function |
| 4 | UD | R1W0 | Transmit buffer underflow error flag cleared in slave mode<br>W0: Transmit buffer underflow error flag in slave mode cleared<br>W1: No function |
| 3 | SSR | R1W0 | Slave select input rising edge flag cleared<br>W0: Slave select input rising edge flag cleared<br>W1: No function |
| 2 | SSF | R1W0 | Slave select input falling edge flag cleared<br>W0: Slave select input falling edge flag cleared<br>W1: No function |
| 1 | RXNE | R1W0 | Receive buffer not empty flag cleared<br>W0: Receive buffer not empty flag cleared<br>W1: No function |
| 0 | FLUSH | R1W0 | Clear transmit buffer and shift register<br>W0: The transmit buffer and shift register (ISR.TXE is set to 1 after clearing) cleared<br>W1: No function |

### 17.7.7 SPI_DR data register

Address offset: 0x18 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | DR | RW | Write data to be transmitted<br>Read as received data |

# 18 I2C interface

## 18.1 Overview

CW32F003 integrates one I2C controller, which can serially transmit the data to be transmitted to the I2C bus according to the I2C specification according to the set transmission rate (standard, fast, high-speed), and detect the state during the communication process. Bus collision and arbitration handling in multi-master communication is also supported.

## 18.2 Main features

- Supports master transmit/receive and slave transmit/receive four working modes
- Supports clock stretching (clock synchronization) and multi-master communication conflict arbitration
- Supports standard (100Kbps)/fast (400Kbps)/high speed (1Mbps) three working rates
- Supports 7-bit addressing function
- Supports 3 slave addresses
- Supports broadcast address
- Supports input signal noise filtering function
- Supports interrupt status query function

## 18.3 Protocol description

The I2C bus uses two signal lines (the data line SDA and the clock line SCL) to transfer data between devices. SCL is a unidirectional clock line, which is fixedly driven by the master. SDA is a bidirectional data line, which is driven by the transmitting and receiving terminals in time sharing during data transmission.

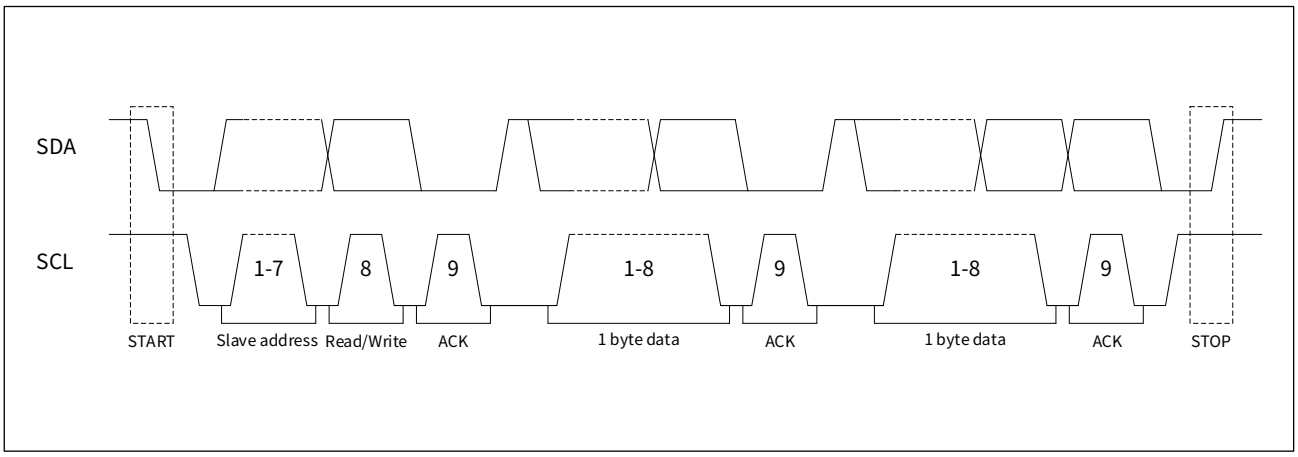
Multiple devices can be connected to the I2C bus. All devices are in an idle state when they are not transmitting data (the slave receiving mode is not addressed). Any device can send a START signal as a master to start data transmission. The bus is occupied until the STOP signal appears on the bus.

I2C communication adopts the master-slave structure, and the communication is initiated and terminated by the master. The master initiates communication by transmitting the START signal, then transmits a total of 8 bits of SLA+W/R data (where SLA is the 7-bit slave address, and W/R is the read-write bit), and releases the SDA bus at the ninth SCL clock, the corresponding slave occupies the SDA bus at the ninth SCL clock and outputs an ACK response signal to complete the slave addressing. After that, the transmitter and receiver of data communication are determined according to the W/R bit of the first byte transmitted by the master. For each byte of data transmitted by the transmitter, the receiver must respond with an ACK response signal. After the data transmission is completed, the master transmits a STOP signal to end the communication.

### 18.3.1 Protocol frame format

The standard I2C transmission protocol frame consists of four parts: start signal (START) or repeated start signal (Repeated START), slave address and read and write bits, data transmission, and stop signal (STOP). As shown in the following figure:

Figure 18-1 I2C protocol frame




- Start signal (START)

  When the bus is in an idle state (SCL and SDA lines are high at the same time), a falling edge signal from high to low appears on the SDA line, which is defined as a start signal. After the master transmits a start signal to the bus, data transmission starts and the bus is occupied.

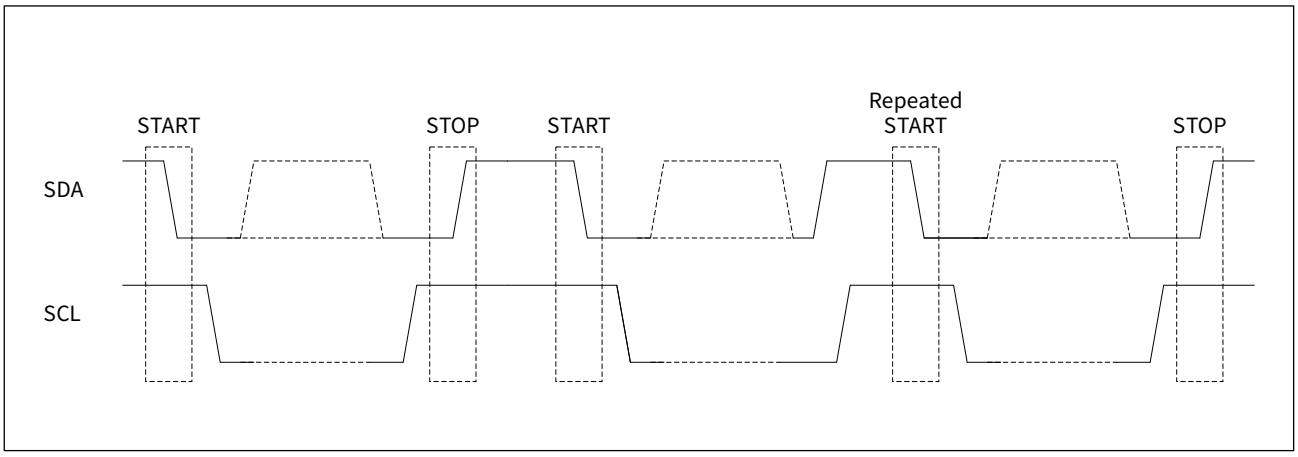
- Repeated start signal (Repeated START)

  When a start signal does not appear before a stop signal, a new start signal appears, and the new start signal is defined as a repeated start signal. Before the master transmits the stop signal, the SDA bus is always in the occupied state, and other masters cannot occupy the bus.

- Stop signal (STOP)

  When the SCL line is high, a rising edge signal from low to high appears on the SDA line, which is defined as a stop signal. The master transmits a stop signal to the bus to end the data transfer and releases the bus.

Figure 18-2 Start and stop signals



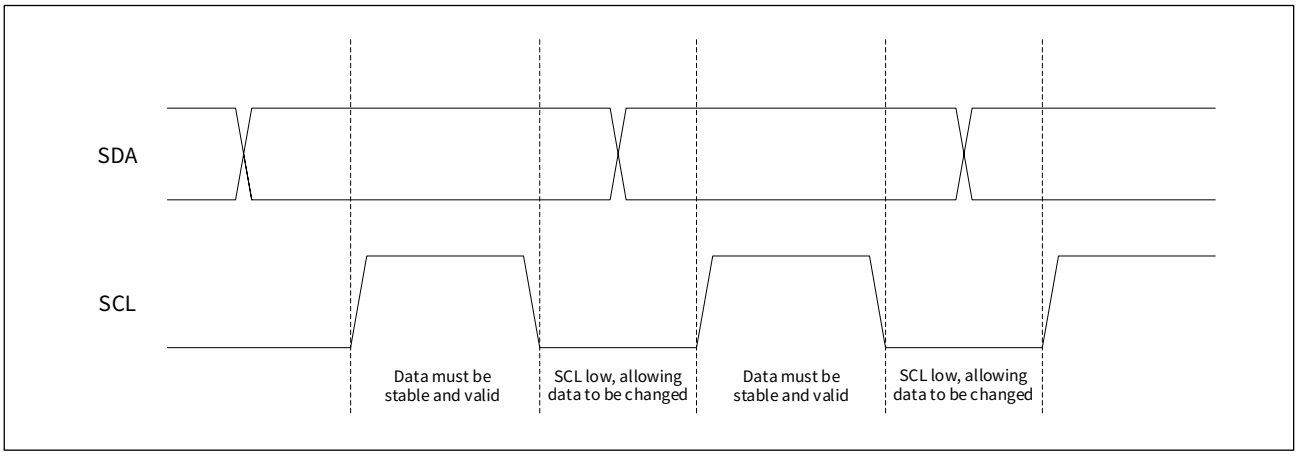

- Slave address and read/write bit

  When the start signal is generated, the master starts to transmit the first byte of data: 7 bit slave address + read and write bits. The read and write bits (1: read; 0: write) control the direction of data transfer on the bus. The addressed slave occupies the SDA bus during the 9th SCL clock cycle and acknowledges the ACK by asserting SDA low.

- Data transmission

  The master outputs the serial clock signal on the SCL line, and the master and slave devices transmit data through the SDA line. During data transmission, 1 SCL clock pulse transmits 1 data bit (the most significant bit MSB is first), and the data on the SDA line changes only when SCL is low, and each transmitted 1 byte is followed by an acknowledge bit. As shown in the following figure:

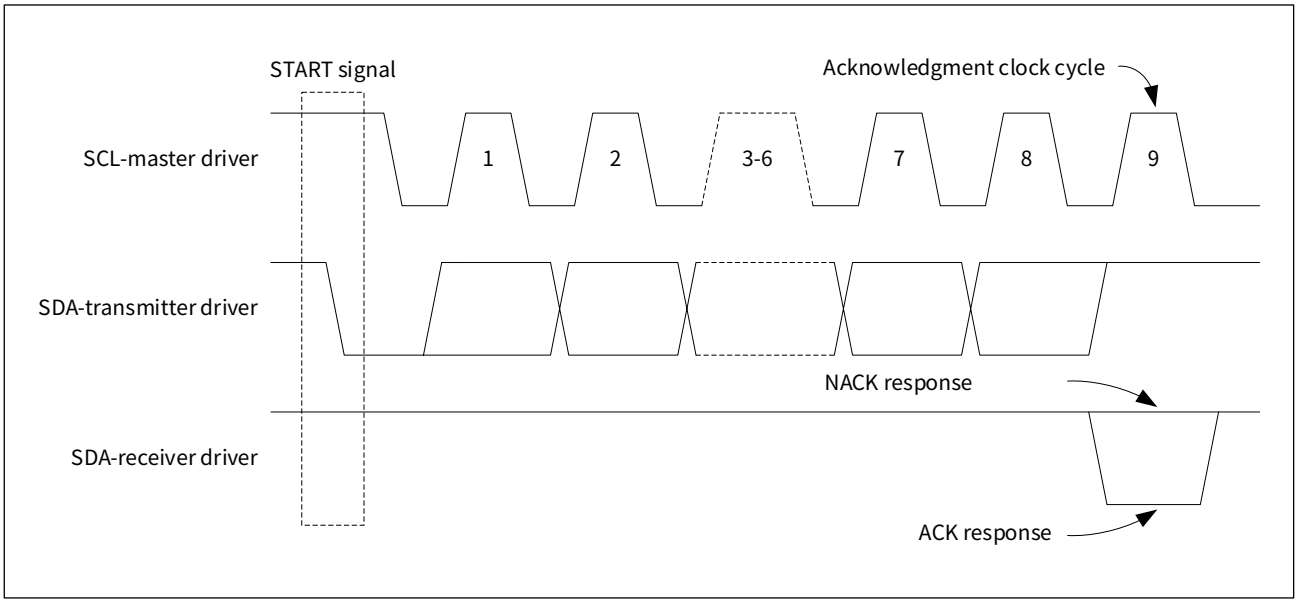
Figure 18-3 Data transmission



### 18.3.2 Transmission response

When transmitting data on the bus, the transmitter gives up the control of SDA in the 9th SCL clock cycle after transmitting 1 byte of data, and the receiver must reply a response bit in the 9th SCL clock cycle: the reception is successful, Send ACK response, receive exception send NACK response.

Figure 18-4 Transmission response




### 18.3.3 Conflict detection and arbitration

In a multi-master communication system, each node on the bus has a slave address. Each node can be accessed by other nodes as a slave node, and can also be used as a master node to send control bytes and transmit data to other nodes. A bus collision occurs when two or more nodes send a start signal to the bus at the same time and start transmitting data. The I2C controller has a built-in arbiter that can detect and arbitrate I2C bus conflicts to ensure the reliability and integrity of data communication.

- Conflict detection principle

  In the physical realization, the SDA and SCL pins have the same circuit structure, and the output drive of the pin is connected with the input buffer. The output structure is an open-drain field effect transistor, and the input structure is an inverter with high input impedance. Based on this structure:

  1. Since SDA and SCL are open-drain structures, the "wire-and" logic of the signal is realized by means of an external pull-up resistor;
  2. The device reads data while writing data to the bus, which can be used to detect bus conflicts and realize "clock synchronization" and "bus arbitration".

  According to the "wire-and" logic, if two masters send logic 1 or logic 0 at the same time, neither of the two masters can detect the conflict, and it needs to wait until the next bit of data is transmitted before continuing to detect the conflict; if one of the two hosts sends a logic 1, When a logic 0 is transmitted, the bus is logic 0 at this time, the master transmitting logic 1 detects a conflict, and the master transmitting logic 0 does not detect a conflict.

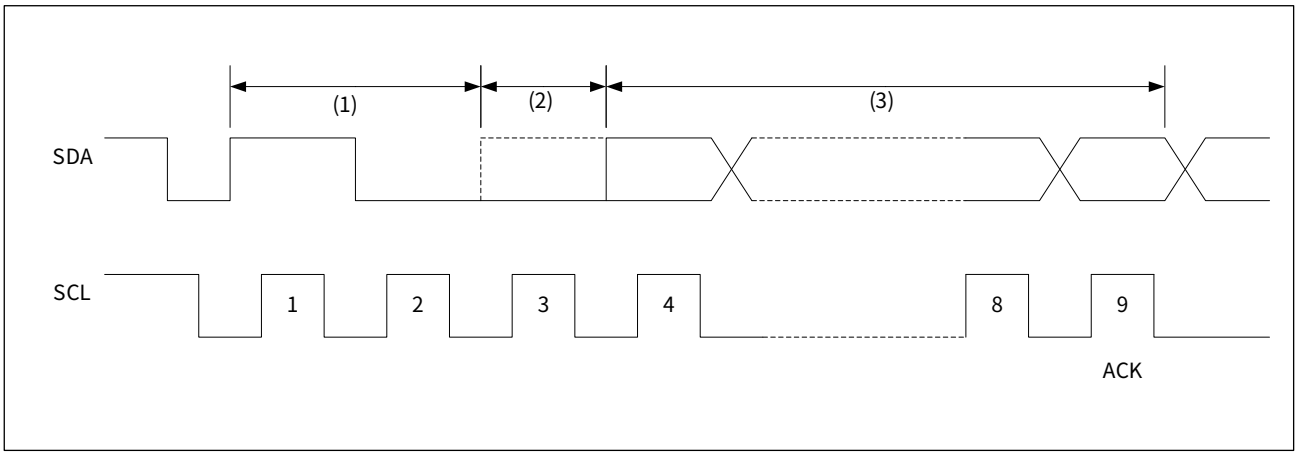
- Conflict arbitration principles

  When the master detects a bus conflict, the master loses arbitration, exits the master transmit mode, enters the unaddressed slave mode, releases the SDA data line, and returns to the address detection state, and then enters the state according to the received SLA+W/R corresponding slave mode (SLA address match enters addressed slave mode, SLA address mismatch enters unaddressed slave mode). The master that loses arbitration will still send the SCL serial clock until the end of the current byte transfer.

  When the master does not detect a bus conflict, the master wins the arbitration and continues to dominate the data transfer until the communication is complete.

The following figure is a schematic diagram of transmitting conflict and arbitration between master A and master B on an I2C bus (assuming that the transmitting clocks of the two masters A and B are synchronized):

Figure 18-5 Conflict detection and arbitration



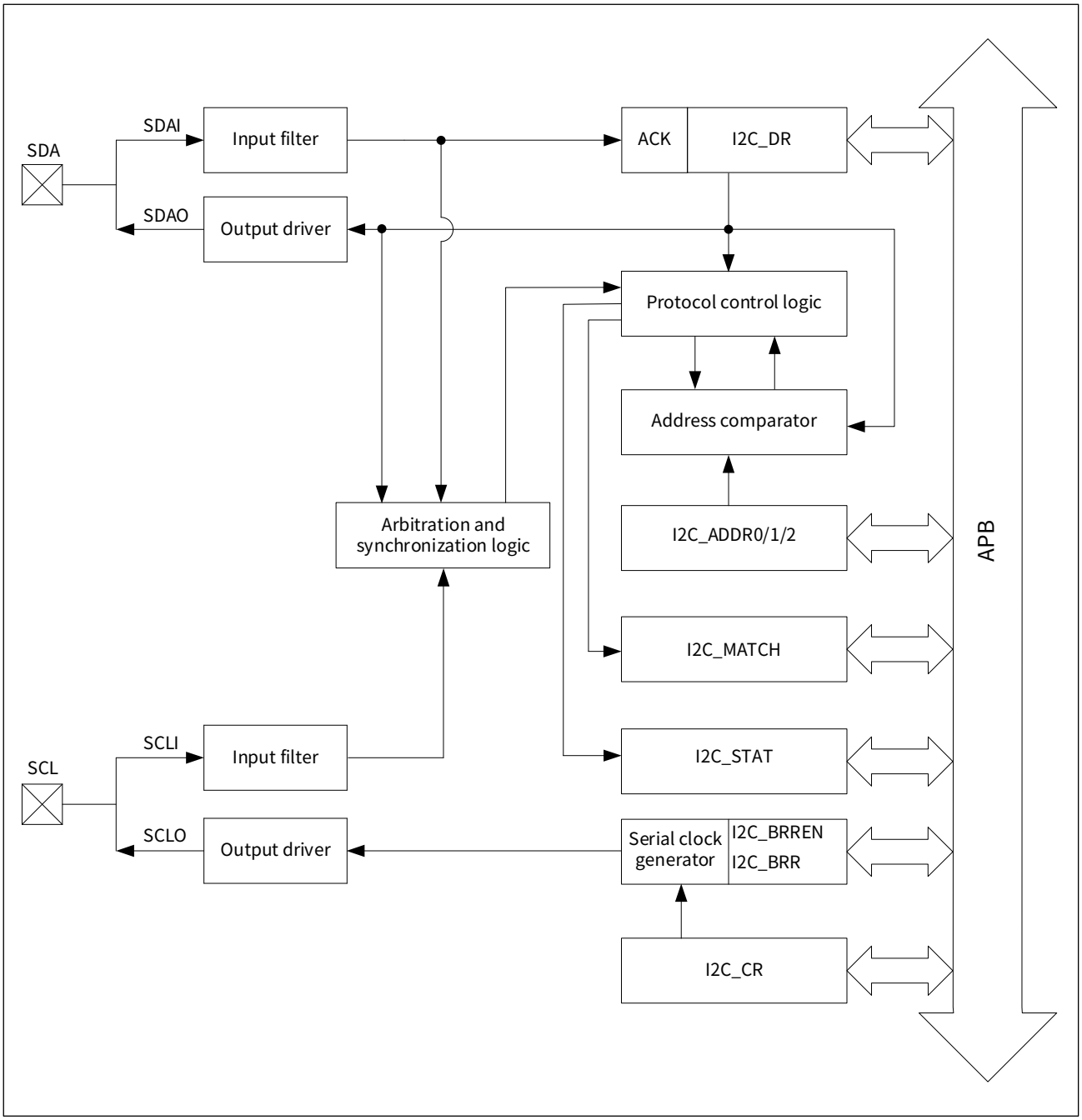

1. In stage (1), both master A and master B transmit logic 1 and logic 0 successively. Since the data is the same and synchronous, neither host will detect a bus conflict.
2. In stage (2), master A transmits a logic 1, master B transmits a logic 0, and the data on the SDA bus is a logic 0. The A host controller detects a data error and exits the transmission competition, that is, the arbitration is lost, and the A master enters the unaddressed slave receiver mode. Host B does not detect a bus conflict, wins the arbitration, and continues this data transfer.
3. In phase (3), the A master is in unaddressed slave receive mode, but still generates SCL clock pulses until the end of the current byte transfer. After that, the A host I2C controller no longer transmits the clock signal. Since the B master wins the arbitration, both SCL and SDA are dominated by the B master to control the transfer.

## 18.4 Functional description

### 18.4.1 Functional block diagram

The I2C module mainly includes clock generator, input filter, address comparator, protocol control logic, arbitration and synchronization logic, and related registers. Its functional block diagram is shown in the following figure:

Figure 18-6 I2C functional block diagram




CW32F003 supports users to flexibly select GPIO as I2C communication pin, as shown in the following table:

Table 18-1 I2C pin selection

| | I2C1 |
|---|---|
| SCL | PA05/PB01/PB04/PB06 |
| SDA | PA02/PA06/PB00/PB03/PB05 |

### 18.4.2 Serial clock generator

The serial clock generator is used to generate the baud rate clock SCL for I2C communication. The serial clock generator uses PCLK as the input clock, counts through an 8-bit counter, and outputs the I2C clock signal with the required baud rate.

SCL clock frequency calculation formula:

$$f_{SCL} = f_{PCLK} / 8 / ( BRR + 1 )$$

Where, BRR is configured through the baud rate counter configuration register I2C_BRR, and the effective range of BRR is 1~255.

The counter count of the serial clock generator is enabled by the EN bit field of the I2C_BRREN register. EN is 1 to enable and 0 to prohibit. When the master is used, EN should be set to 1, and this bit has no effect when the slave is used.

The corresponding relationship between the combination of PCLK and BRR and the SCL clock frequency is shown in the following table:

Table 18-2 I2C transfer rate and configuration example

| $f_{SCL}$(kHz) / $f_{PCLK}$ (kHz) | I2C_BRR | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1000 | 62 | 41 | 31 | 25 | 20 | 17 | 15 |
| 2000 | 125 | 83 | 62 | 50 | 41 | 35 | 31 |
| 4000 | 250 | 166 | 125 | **100** | 83 | 71 | 62 |
| 6000 | 375 | 250 | 187 | 150 | 124 | 107 | 93 |
| 8000 | 500 | 333 | 250 | 200 | 166 | 142 | 125 |
| 10000 | 625 | 416 | 312 | 250 | 208 | 178 | 156 |
| 12000 | 750 | 500 | 375 | 300 | 250 | 214 | 187 |
| 14000 | 875 | 583 | 437 | 350 | 291 | 250 | 218 |
| 16000 | **1000** | 666 | 500 | **400** | 333 | 285 | 250 |

### 18.4.3 Input filter

The input filter can filter the SDA and SCL input signals, and the spike signal less than 1 PCLK cycle will be filtered out.

The input filter can be configured in two modes: simple filter mode and advanced filter mode. Simple filtering has faster communication rate; advanced mode has higher anti-jamming performance. The filtering mode is configured through the FLT bit field of the I2C control register I2C_CR. Setting I2C_CR.FLT to 1 is the simple filtering mode, and setting I2C_CR.FLT to 0 is the advanced filtering mode.

As a master, if the value of BRR is less than or equal to 9, I2C_CR.FLT should be set to 1; if the value of BRR is greater than 9, then I2C_CR.FLT should be set to 0.

As a slave, if the frequency ratio of PCLK to SCL is less than or equal to 40, then I2C_CR.FLT should be set to 1; if the frequency ratio of PCLK to SCL is greater than 40, then I2C_CR.FLT should be set to 0.

### 18.4.4 Address comparator

Each device on the I2C bus has a slave address, and each slave address is different. The master addresses the slave according to the slave address, and the slave automatically detects whether the 7bit addressing address transmitted by the master matches the local address through the address comparator to determine whether to communicate with the master.

The I2C controller supports 3 programmable slave addresses, and the specific address information is configured through the slave address registers I2C_ADDR0/I2C_ADDR1/I2C_ADDR2.

The address comparator of the slave machine compares the received 7-bit addressing address with the three slave addresses and the broadcast address (0x00). If it matches any one of the 4 addresses, it is considered that the address matches, and the I2C interrupt flag bit I2C_CR.SI will be set to 1, and an interrupt request will be generated. The application program can obtain the successfully matched address serial number by querying the slave address matching register I2C_MATCH.

If the address matches I2C_ADD0/1/2, the slave enters the corresponding addressed slave receive mode (receives SLA+W) or addressed slave transmit mode (receives SLA+R); if the address matches to If the broadcast address is 0x00 (SLA+W is received), the slave enters the broadcast receiving mode.

### 18.4.5 Arbitration and synchronization logic

#### 18.4.5.1 SCL synchronization

I2C supports clock synchronization (clock stretching) function, the time of SCL clock low level is determined by the device with the longest SCL clock low level width, and the SCL clock high level time is determined by the device with the shortest clock high level width.

If the slave wants the master to reduce the transmission speed, it can keep I2C_CR.SI 1 after receiving the data and responding to ACK, and the slave I2C controller will keep SCL low to notify the master. When the master is ready for the next byte data transfer (transmit or receive), it detects that the level of SCL is pulled low, and waits until the slave completes the operation and clears I2C_CR.SI to 0, and the slave controller releases the pull of SCL. Low control, the master continues to transmit the next byte of data after detecting that SCL is high.

#### 18.4.5.2 SDA arbitration

I2C supports SDA conflict detection and arbitration, which can ensure that the data on the I2C bus is not destroyed when multiple masters attempt to control the I2C bus.

When each master transmits data, it will simultaneously compare whether the data on the bus is consistent with the data transmitted by itself. If they are inconsistent, it is considered that a bus conflict is detected, and the transmitting competition will be withdrawn, that is, the arbitration will be lost. A master that loses arbitration will immediately switch to an unaddressed slave state to ensure that it can be addressed by a master that succeeds in arbitration. A master that loses arbitration continues to output the SCL serial clock until the current byte transfer is complete.

SDA arbitration generally occurs when the master transmits SLA+W/R data. If two masters transmit data to a slave at the same time, that is, the slave addresses transmitted by the two masters are the same, the arbitration will continue in the second byte.

### 18.4.6 Response control

The receiver of the I2C data transmission must give an ACK or NACK response to the transmitter in the 9th SCL clock cycle of each byte. The receiver obtains the current state of the receiver through the response bit: responding with an ACK response means that the receiver has received it correctly. This byte of data can continue to receive the next byte of data; responding to NACK generally indicates that the receiver no longer receives any data.

Whether the receiver sends ACK or NACK is controlled by the AA bit field of the receiver's I2C control register I2C_CR. When I2C_CR.AA is set to 1, the I2C module will respond with an ACK response every time it receives 1 byte of data. When I2C_CR.AA is set to 0, the I2C module will respond with a NACK response every time it receives 1 byte of data.

In the process of receiving data by the master, the master as the initiator of communication, controls the number of bytes transmitted and received. The master (receiver) responds to the NACK response to the slave (transmitter) after receiving the last byte of data, and the slave receives the NACK. After the slave receives the NACK response, it will switch to unaddressed slave receive mode and release the SDA bus so that the master can transmit a STOP signal or a Repeated START signal.

In the process of transmitting data from the slave, if its own I2C_CR.AA response control bit is cleared by the application program, the slave will switch itself to the unaddressed slave receiving mode after transmitting the last 1 byte of valid data, and release the SDA bus, the master will get 0xFF when reading data from the bus. At this time, the master should be able to judge that the slave is in an unresponsive state, and send a STOP signal or a Repeated START signal.

In the process of receiving data from the slave, if it responds to NACK to the master (transmitter), it means that the slave (receiver) actively ends the communication, no longer receives the data transmits by the master, and switches itself to the unaddressed slave to receive mode, and release the SDA bus, at this time the master should send a STOP signal or a Repeated START signal.

### 18.4.7 I2C interrupts

The SI bit field of the I2C control register I2C_CR is the interrupt flag bit. When the value of the STAT bit field of the I2C status register I2C_STAT changes (except for 0xF8), the I2C_CR.SI flag will be set and an interrupt request will be generated.

In the user I2C interrupt service routine, you should query the STAT bit field value of the I2C status register I2C_STAT to obtain the status of the I2C bus to determine the cause of the interrupt. Set I2C_CR.SI to 0 to clear this flag.

### 18.4.8 Operating modes

The I2C controller supports 4 operating modes: master transmit mode, master receive mode, slave transmit mode, and slave receive mode. In addition, the broadcast receive mode is also supported, and its working method is similar to that of the slave receive mode.

#### 18.4.8.1 Master transmit mode

In master transmit mode, the master actively transmits multiple bytes to the slave.

The SCL serial clock is controlled by the master, so it is necessary to set the I2C_BRR register according to the transmission baud rate and set the EN bit field of the I2C_BRREN register to 1, and then start the transmission.

The master sets I2C_CR.STA to 1, and informs the controller to transmit a START signal. After the controller receives the notification, it detects whether the bus is free. When the bus is free, the master controller transmits a START signal to the bus. If the transmission is successful, the status code I2C_STAT becomes 0x08, and the interrupt flag bit I2C_CR.SI is set to 1.

After the master transmits the START signal, software needs to set I2C_CR.STA to 0, and then write the slave address and write flag bit (SLA+W) to the I2C data register I2C_DR; clear the I2C_CR.SI bit, the master controller SLA+W will be transmitted to the I2C bus; when the master transmits SLA+W and receives the ACK response signal from the slave, the status code I2C_STAT becomes 0x18, and the interrupt flag bit I2C_CR.SI is set to 1.

After that, the master transmits multiple bytes of user-defined data and detects the ACK response signal according to the needs of the application. The transmitting process of each byte is similar to that of sending SLA+W data frames.

In the process of transmitting data, if the master receives a NACK response signal, it indicates that the slave no longer receives the data sent by the host (I2C_CR.AA of the slave is cleared), the master should send a STOP signal to end the data transfer, or send Repeated START repeats the start signal to start a new round of transmission.

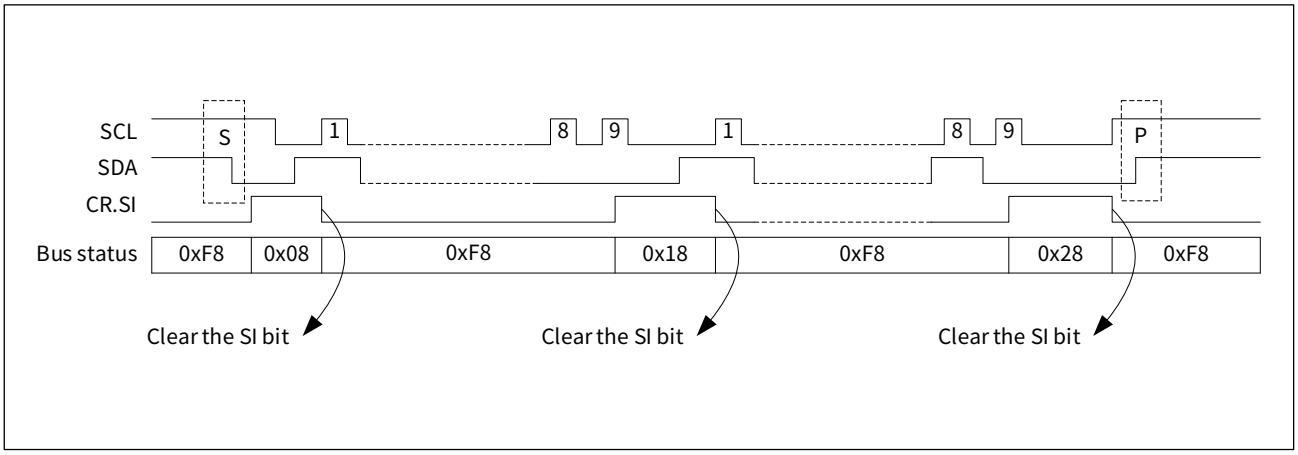
When the master completes transmitting all the data, set I2C_CR.STO to 1 to notify the controller that the data has been transmitted and the STOP signal is to be transmitted. Clear the I2C_CR.SI bit, and the master controller will transmit a STOP signal to the I2C bus to complete this data transfer and release the bus.

When the master has finished transmitting all the data, it can also not transmit the STOP signal, but directly transmit the Repeated START signal, and continue to occupy the bus for a new round of data transmission.

When the master loses arbitration due to a bus conflict, it enters the unaddressed slave receive mode (status code I2C_STAT=0x38).

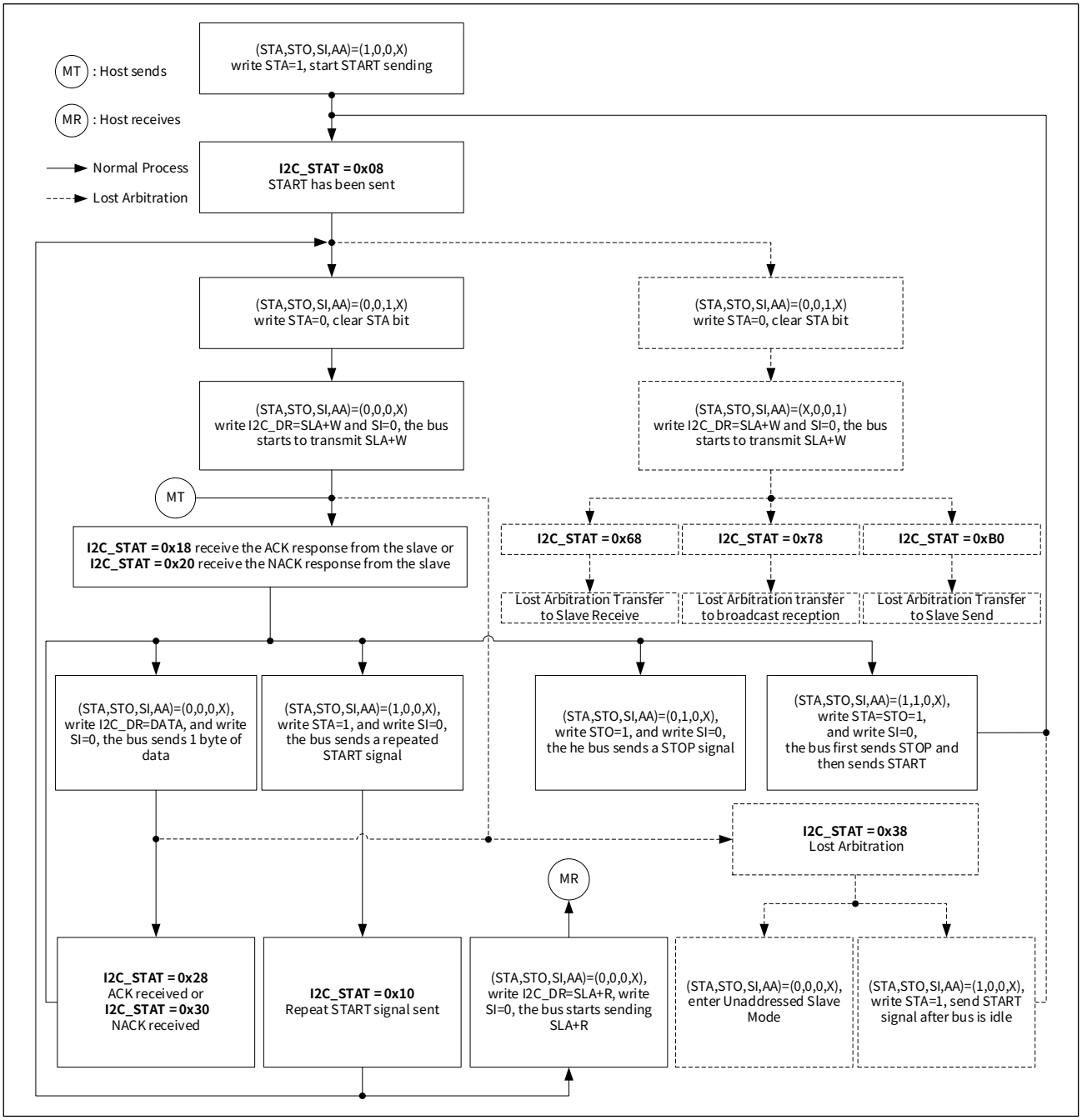
The state synchronization diagram in the master transmit mode is shown in the following figure:

Figure 18-7 The status synchronization diagram of master transmit mode




The data transmission flow chart and status register values in the master transmit mode are shown in the following figure:

Figure 18-8 Master transmit mode flow and register status



#### 18.4.8.2 Master receive mode

The master receive mode is used to receive multiple data transmitted by the slave, and the master will respond with an ACK response signal every time it receives 1 byte of data.

The SCL serial clock is controlled and generated by the master, so it is necessary to set the I2C_BRR register of the master according to the transmission baud rate and set the EN bit field of the I2C_BRREN register to 1, and then start the transmission.

The master sets I2C_CR.STA to 1, and informs the controller to send a START signal. After the controller receives the notification, it detects whether the bus is free. When the bus is free, the master controller sends a START signal to the bus. If the transmission is successful, the status code I2C_STAT becomes 0x08, and the interrupt flag bit I2C_CR.SI is set to 1.

After the master sends the START signal, the software needs to set I2C_CR.STA to 0, and then write the slave address and read flag bit (SLA+R) to the I2C data register I2C_DR; clear the I2C_CR.SI flag bit, the master controls The controller will send SLA+R to the I2C bus; when the master sends SLA+R and receives the ACK response signal from the slave, the status code I2C_STAT becomes 0x40, and the interrupt flag bit I2C_CR.SI is set to 1.

After that, the master sets I2C_CR.AA to 1, clears the I2C_CR.SI bit, and starts to receive the data transmitted by the slave. After receiving 1 byte of data, it will reply an ACK response signal. During the master receiving process, it should be cautiond that:

1. In order to ensure that the I2C_CR.SI interrupt signal can be correctly generated after receiving 1 byte of data, it is necessary to clear the I2C_CR.SI bit in time after receiving 1 byte of data.
2. Before receiving the last byte, I2C_CR.AA needs to be cleared, that is, the master does not generate an ACK response signal when receiving the last byte, so as to notify the slave to stop data transmission.

In the process of receiving data by the master, if the slave no longer transmits the data required by the master for some reason (the I2C_CR.AA of the slave is cleared), the master will receive all 1 signals later. At this time, the master needs to be in the application layer. Judgment is made on the data, and it is determined that the slave is in an unresponsive state. It should send a STOP signal to end the current transmission, or send a Repeated START signal to start a new round of transmission.

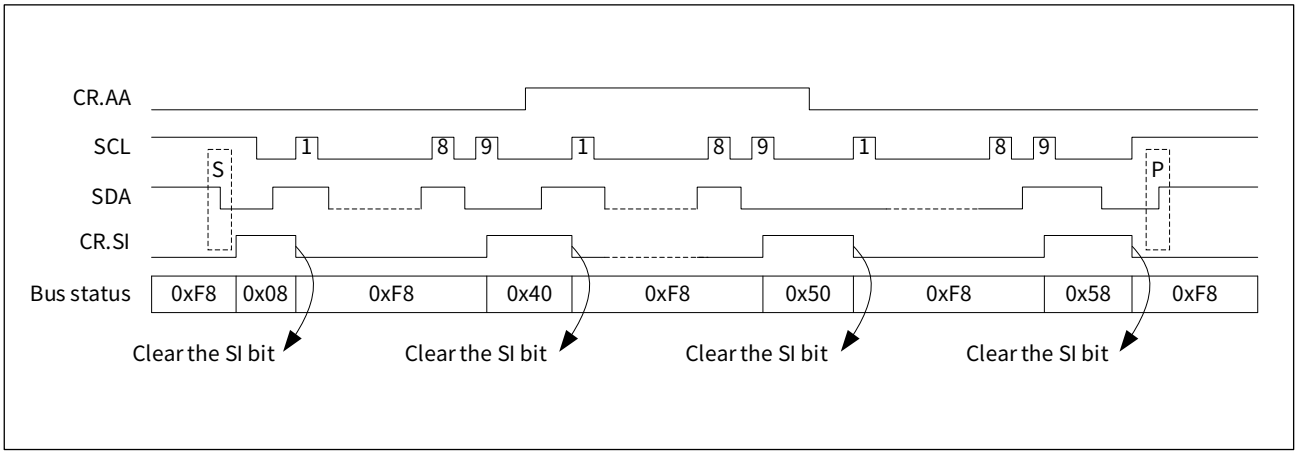
When the master completes receiving all the data, set I2C_CR.STO to 1 to notify the controller that the data has been received and to send a STOP signal. Clear the I2C_CR.SI bit, and the master controller sends a STOP signal to the I2C bus to complete the data transfer and release the bus.

When the master has finished receiving all the data, it can also not send the STOP signal, but directly send the Repeated START signal, and continue to occupy the bus for a new round of data transmission.

When the master loses arbitration due to a bus conflict, it enters the unaddressed slave receive mode (status code I2C_STAT=0x38).

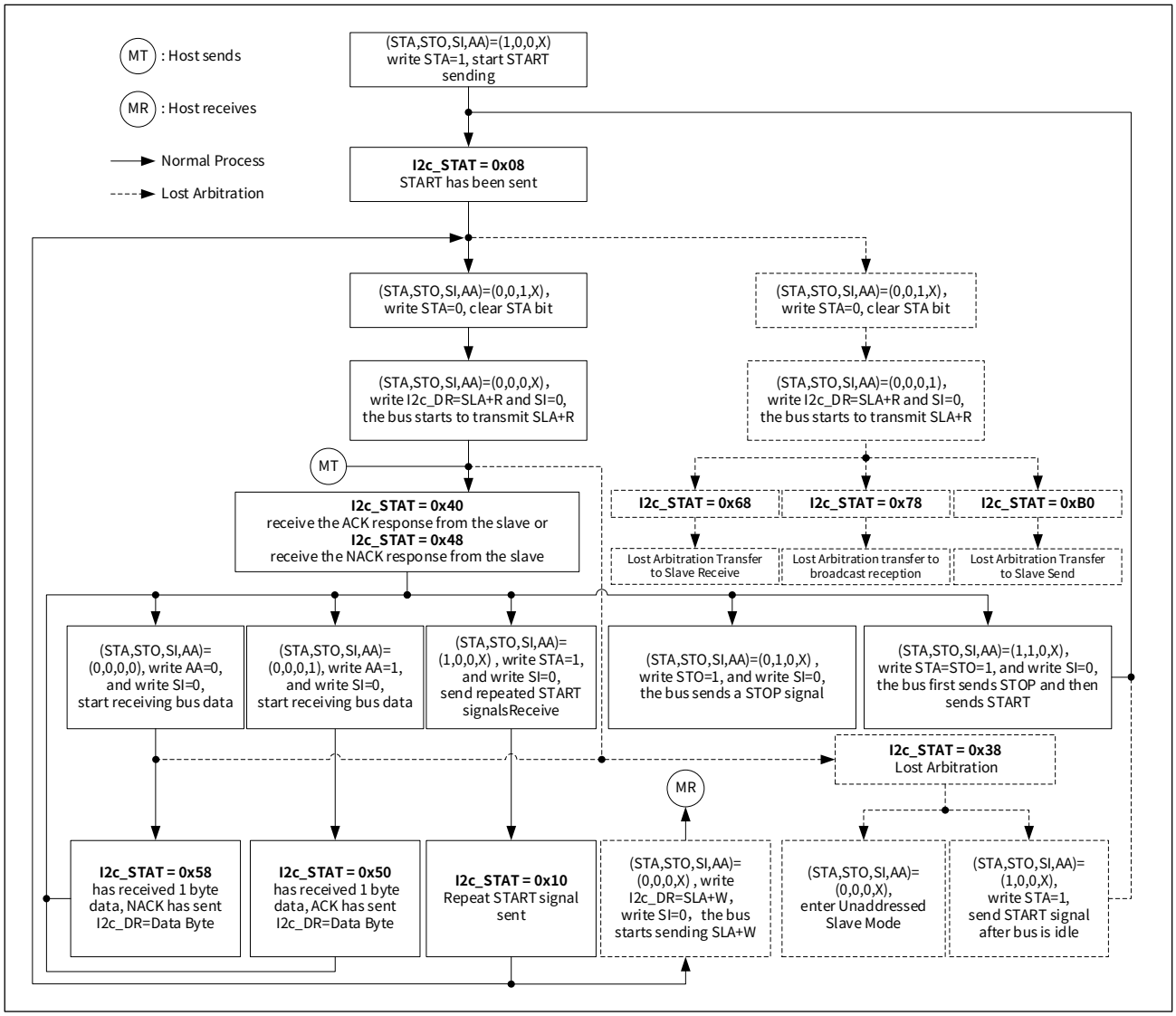
The state synchronization diagram in the master receive mode is shown in the following figure:

Figure 18-9 The status synchronization diagram of master receive mode




The data reception flow chart and status register value in the master receive mode are shown in the following figure:

Figure 18-10 Master receive mode flow and register status



### 18.4.8.3 Slave receive mode

In slave receive mode, the slave receives the data transmitted by the master.

Before transmission, the slave needs to set the slave address: write the slave address to any one of the three slave address registers I2C_ADDR0, I2C_ADDR1, I2C_ADDR2; set I2C_CR.AA to 1 to respond to the master's addressing; I2C_BRR The set value of the register is invalid, so it is not necessary to set it.

After completing the above initialization work, the slave machine enters the idle state (the slave receiving mode is not addressed), waiting to be addressed by the write signal (SLA+W) sent by the master. After the slave receives SLA+W, if the address matches I2C_ADDR0/1/2, the slave responds with ACK and enters the addressed slave receiving mode, the status code I2C_STAT becomes 0x60, and the interrupt flag bit I2C_CR.SI Also set to 1. The I2C_CR.SI bit must be cleared at this time in order to receive data from the master from the bus.

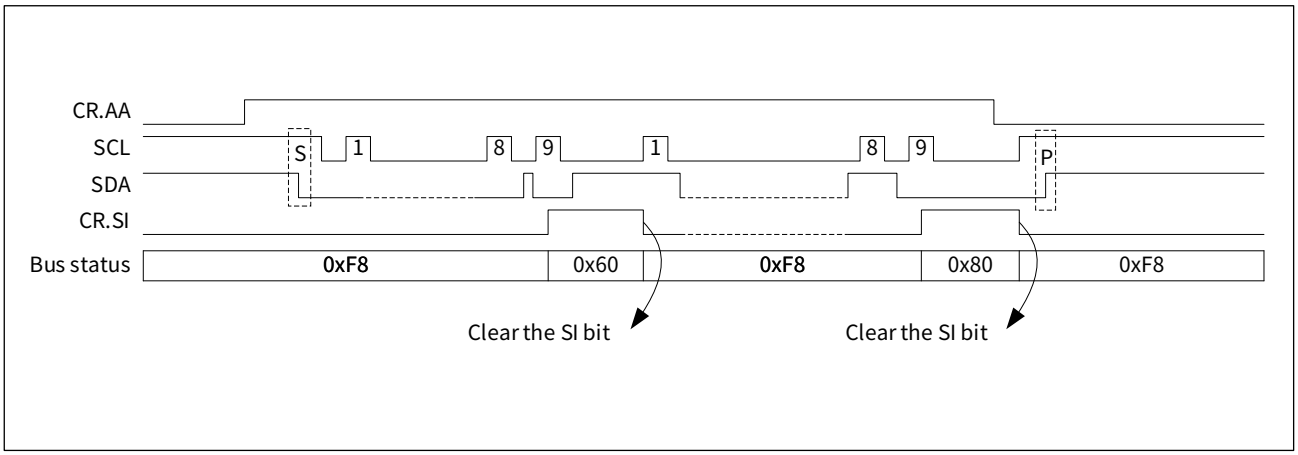
Every time the slave receives 1 byte of data, it must respond with an ACK response. After the application reads the byte of data, it must clear the I2C_CR.SI bit to prepare for receiving the next byte of data.

In the process of receiving data from the slave, if I2C_CR.AA is cleared, the slave will return a NACK signal when it receives the next byte, and the state of the slave will also switch to the unaddressed slave receiving mode, ending with the master. communication, no more data is received, and the I2C_DR register holds the previously received data. It can be seen from this feature that the slave application can actively switch the slave from the addressed slave receive mode to the unaddressed slave receive mode by setting I2C_CR.AA to 0.

When the master loses arbitration due to bus conflict in the SLA+ read and write stage, it will enter the unaddressed slave receiving mode, and then if it receives SLA+W that matches the local address and responds with ACK (status code I2C_STAT=0x68), it will enter addressed slave receive mode.

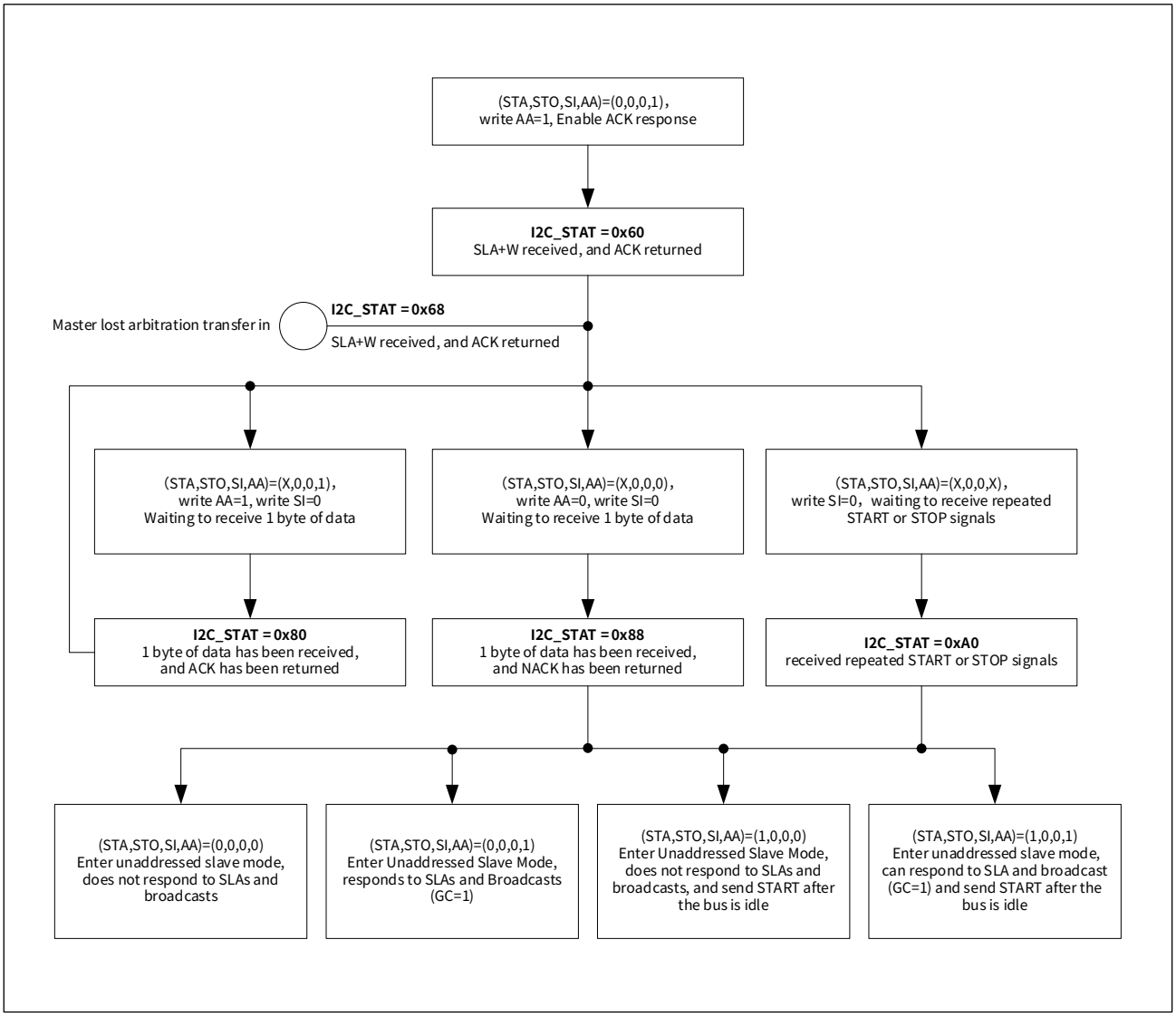
The state synchronization diagram in slave receive mode is shown in the following figure:

Figure 18-11 The status synchronization diagram of slave receive mode




The data receive flow chart and status register value in slave receive mode are shown in the following figure:

Figure 18-12 Slave receive mode flow and register status



#### 18.4.8.4 Slave transmit mode

In slave transmit mode, data is transmitted from the slave to the master.

Before transmission, the slave needs to set the slave address: write the slave address to any one of the three slave address registers I2C_ADDR0, I2C_ADDR1, I2C_ADDR2; set I2C_CR.AA to 1 to respond to the master's addressing; I2C_BRR The set value of the register is invalid, so it is not necessary to set it.

After completing the above initialization work, the slave machine enters the idle state (the slave receive mode is not addressed), waiting to be addressed by the read signal (SLA+R) transmitted by the master. After the slave receives SLA+R, if the address matches I2C_ADDR0/1/2, the slave responds with ACK and enters the addressed slave transmit mode, the status code I2C_STAT becomes 0xA8, and the interrupt flag bit I2C_CR.SI Also set to 1. At this time, the data to be transmitted should be written into the I2C_DR register in time, and the I2C_CR.SI bit should be cleared, waiting for 1 byte of data to be transmitted, and ACK confirmation after each byte of data is transmitted until all data is transmitted.

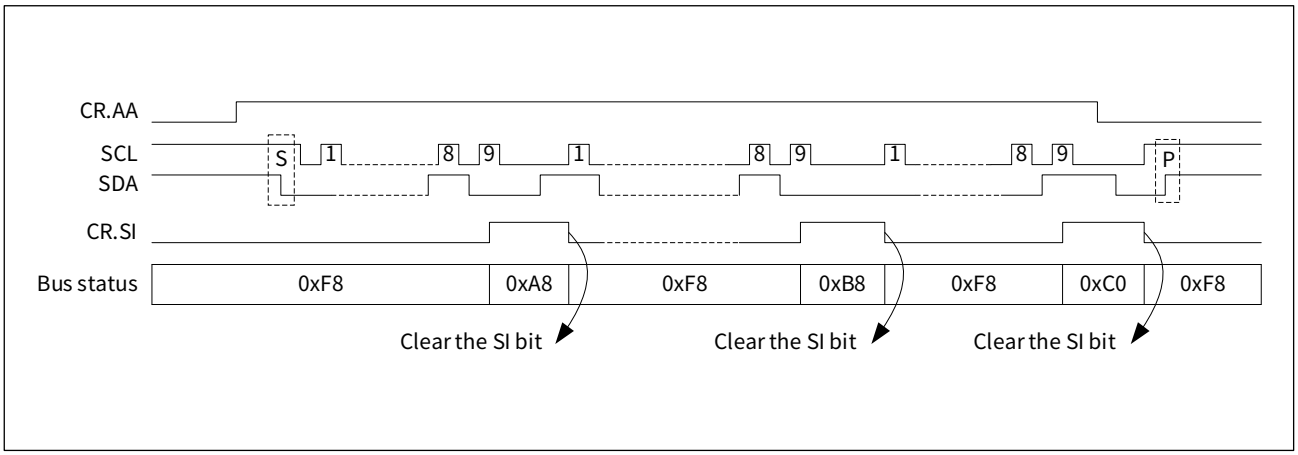
If the slave receives a NACK response during transmission, the slave no longer transmits data and enters the unaddressed slave receive mode.

If the slave actively sets I2C_CR.AA to 0 during the transmission process, the slave switches itself to the unaddressed slave receive mode after transmitting the last 1 byte of valid data, after which the master will get 0xFF when reading data from the bus.

When the master loses arbitration due to bus conflict in the SLA+ read and write stage, it will enter the unaddressed slave receive mode, and then if it receives SLA+R that matches the local address and responds with ACK (status code I2C_STAT=0xB0), it will enter addressed slave transmit mode.

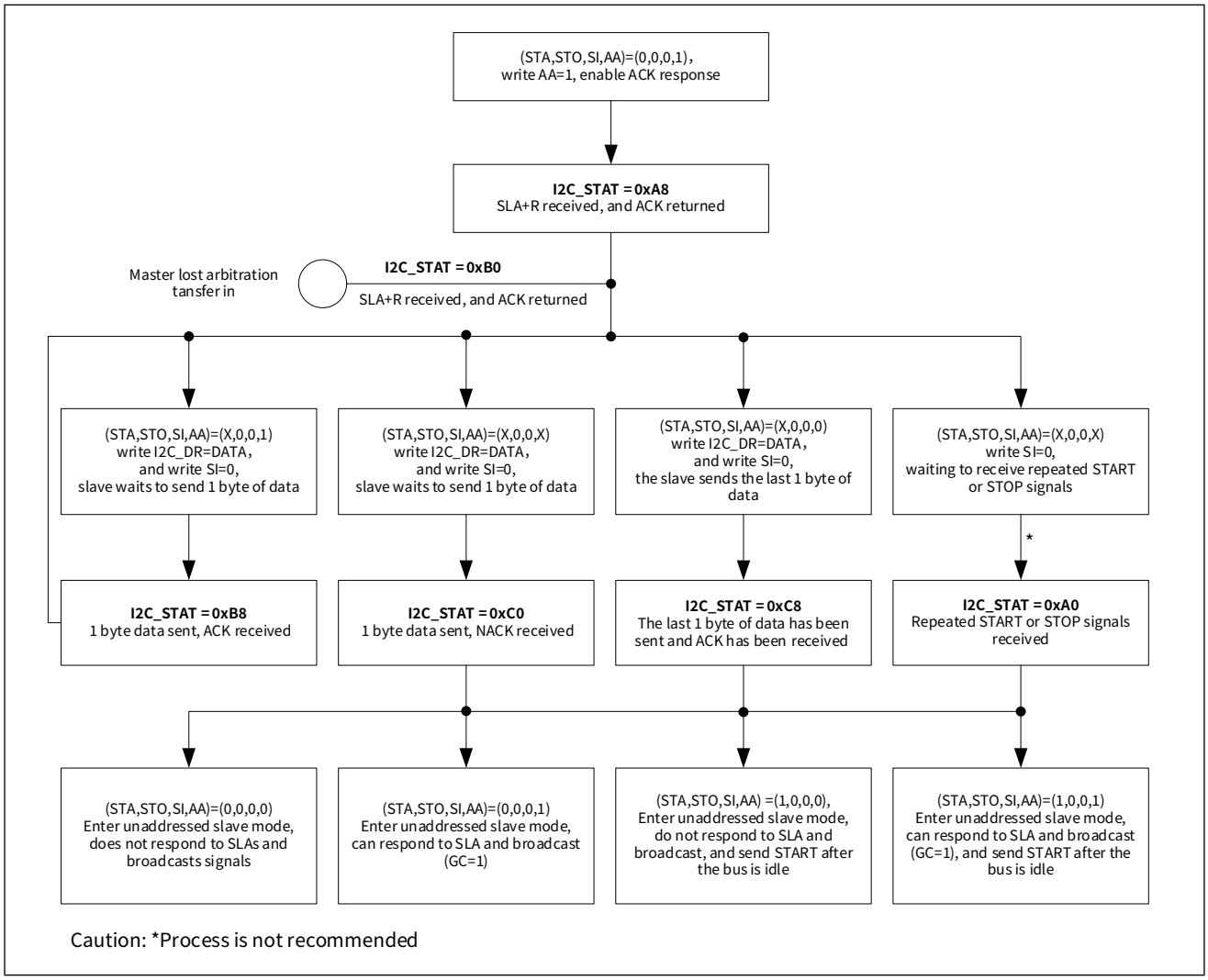
The state synchronization diagram in slave transmit mode is shown in the following figure:

Figure 18-13 The status synchronization diagram of slave transmit mode




The data transmission flow chart and status register value in slave transmit mode are shown in the following figure:

Figure 18-14 Slave transmit mode flow and register status



#### 18.4.8.5 Broadcast receive mode

The broadcast receive mode is a special slave receive mode. When the slave is in an idle state (the slave receive mode is not addressed), the broadcast receive mode is entered when the SLA+W transmitted by the master is received, and the SLA is broadcast address 0x00. In this mode, the data reception process is the same as the slave receive mode, but the I2C bus status code is different.

To receive the broadcast information transmitted by the master, the slave needs to set I2C_CR.AA to 1 and I2C_ADDR0.GC to 1 to respond to the master's broadcast addressing and broadcast data; the three slave address registers I2C_ADDR0/1/2 do not need to be set; The value of I2C_BRR is invalid and does not need to be set.

After completing the above initialization work, the slave machine enters the idle state (the slave receive mode is not addressed), waiting to be addressed by the write signal (SLA+W) transmitted by the master. When the slave receives SLA+W, if the address matches the broadcast address 0x00, the slave responds with an ACK response and enters the broadcast receive mode, the status code I2C_STAT becomes 0x70, and the interrupt flag bit I2C_CR.SI is set to 1 at the same time. At this time, the I2C_CR.SI bit must be cleared in time to receive the data transmitted by the master from the bus.

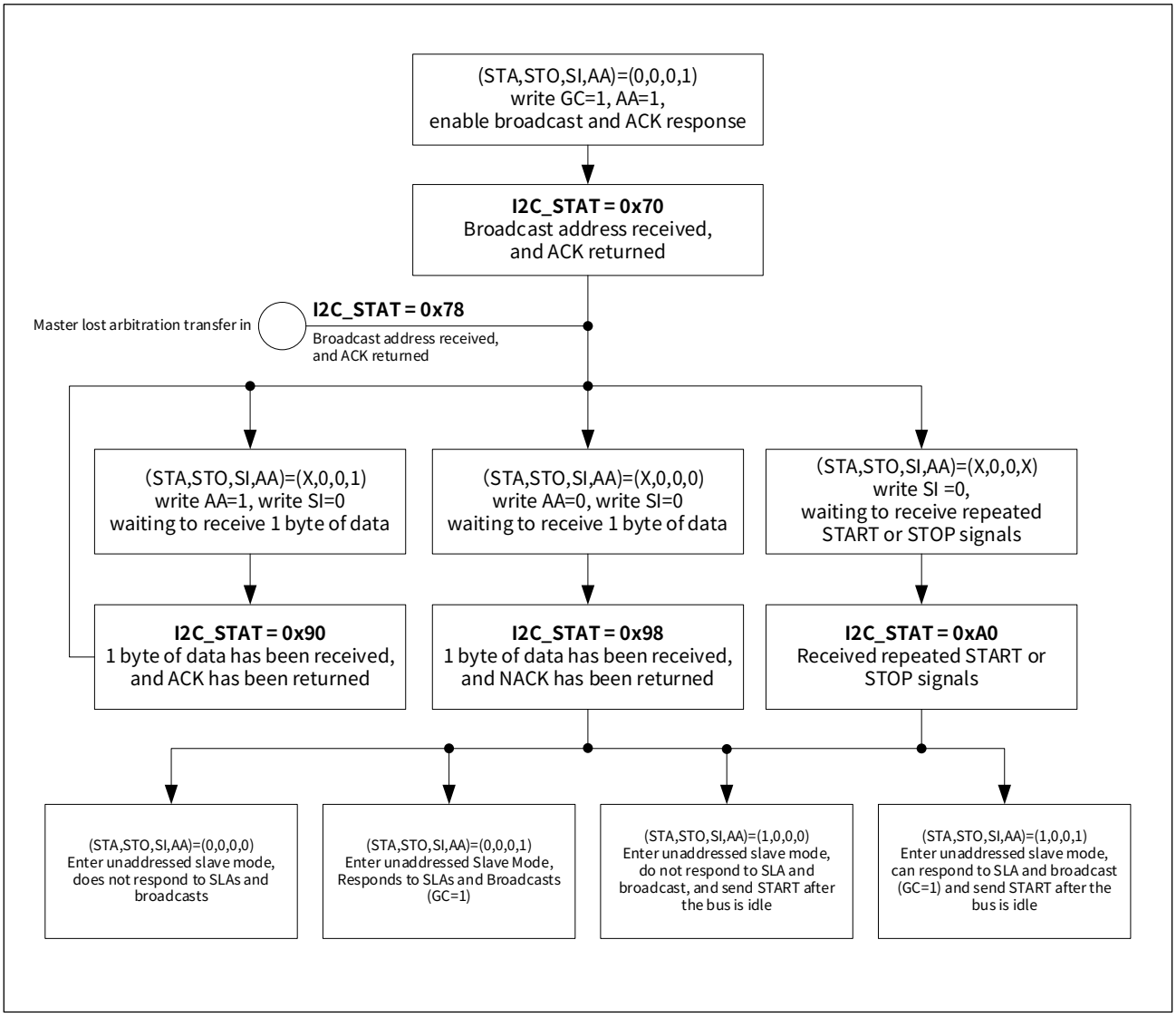
Every time the slave receives 1 byte of data, it must respond with an ACK response. After the application reads the byte of data, it must clear the I2C_CR.SI bit to prepare for receiving the next byte of data.

In the process of receiving data from the slave, if I2C_CR.AA is cleared, the slave will return the NACK signal when it receives the next byte, and the state of the slave will also switch to the unaddressed slave receive mode, ending communication with the master, no more data is received, and the I2C_DR register holds the previously received data. It can be seen from this feature that the slave application can actively switch from the addressed broadcast reception mode to the unaddressed slave receive mode by setting I2C_CR.AA to 0.

When the master loses arbitration due to bus conflict in the SLA+ read and write stage, it will enter the unaddressed slave receive mode, and then if it receives SLA+W that conforms to the broadcast address and responds with ACK (status code I2C_STAT=0x78), it will enter the addressed broadcast receive mode.

The data reception flow chart and status register value in broadcast receive mode are shown in the following figure:

Figure 18-15 Broadcast receive mode flow and register status



### 18.4.9 Multi-master communication

In some applications, there are two or more masters accessing the slave at the same time on one I2C bus, and it is possible to transmit data at the same time. At this time, there will be data conflicts on the SDA bus.

The I2C of the CW32F003 can perform data conflict detection and arbitration on the SDA bus to achieve multi-master applications. If two masters transmit data at the same time, the master that detects the conflict will lose arbitration and enter the unaddressed slave mode, and the master that has not detected the conflict will win the arbitration and continue to dominate the data communication flow.

### 18.4.10 I2C status codes

The I2C bus status is identified by the I2C status register I2C_STAT, with a total of 26 normal receiving or transmitting statuses, and 2 special statuses (0xF8: no information available on the I2C bus; 0x00: bus error).

Whether I2C is in master transmit, master receive, slave receive, slave transmit or broadcast receive mode, when the content of the status register I2C_STAT changes, I2C_CR.SI will be set and an I2C interrupt will be generated.

The I2C status codes are shown in the following table:

Table 18-3 I2C status codes

| Operating mode | Status code | Meaning |
|---|---|---|
| Master transmit mode | 08H | Start signal has been transmitted |
| | 10H | Repeated start signal has been transmitted |
| | 18H | SLA+W has been transmitted, ACK has been received |
| | 20H | SLA+W has been transmitted, NACK has been received |
| | 28H | Data in I2C_DR has been transmitted, ACK has been received |
| | 30H | Data in I2C_DR has been transmitted, NACK has been received |
| | 38H | The master loses arbitration during sending SLA+W phase or during transmitting data phase |
| Master receive mode | 08H | Start signal has been transmitted |
| | 10H | Repeated start signal has been transmitted |
| | 38H | Host loses arbitration during transmitting SLA+R phase or responding to NACK phase |
| | 40H | SLA+R has been transmitted, ACK has been received |
| | 48H | SLA+R has been transmitted, NACK has been received |
| | 50H | Data bytes have been received, ACK has been returned |
| | 58H | Data bytes have been received, NACK has been returned |

| Operating mode | Status code | Meaning |
|---|---|---|
| Slave receive mode | 60H | Its own SLA+W has been received, ACK has been returned |
| | 68H | When the master loses arbitration in the SLA+read-write phase, its own SLA+W has been received, and ACK has been returned |
| | 80H | The previous addressing used its own slave address, the data byte has been received, and ACK has been returned |
| | 88H | The previous addressing used its own slave address, the data byte has been received, and NACK has been returned |
| | A0H | STOP or Repeated START condition is received while the addressed slave is waiting to receive data |
| Slave transmit mode | A8H | Its own SLA+R has been received, ACK has been returned |
| | B0H | When the master loses arbitration in the SLA+read-write phase, its own SLA+R has been received, and ACK has been returned |
| | B8H | Data bytes have been transmitted, ACK has been received |
| | C0H | Data bytes have been transmitted, NACK has been received |
| | C8H | Slave last data byte has been transmitted and ACK has been received |
| Broadcast receive mode | 70H | The broadcast address (0x00) has been received, ACK has been returned |
| | 78H | When the master loses arbitration in the SLA + read and write phase, the broadcast address has been received, and ACK has been returned |
| | 90H | The previous addressing used the broadcast address, the data byte has been received, and the ACK has been returned |
| | 98H | The previous addressing used the broadcast address, the data byte has been received, and NACK has been returned |
| | A0H | STOP or Repeated START condition is received while the addressed slave is waiting to receive data |
| Others | F8H | No relevant status information available, I2C_CR.SI=0 |
| | 00H | A bus error occurred during the transfer, or external interference caused the I2C to enter an undefined state |

The special status code F8H indicates that there is no useful information at the current moment, and the current bus status cannot be determined, because I2C_CR.SI has not been set, and no interrupt is generated. This condition occurs before other states and the I2C module has started to perform a serial transfer.

The special status code 00H indicates that there is a bus error during the I2C serial transmission, such as the START or STOP signal appears in the wrong position of the data frame (including the address byte, data byte or response bit in the serial transmission process) or when external interference affects the internal I2C module signal, etc. When a bus error occurs, the I2C_CR.SI flag is immediately set, and the device is immediately switched to unaddressed slave receive mode, releasing SDA and SCL, and clearing the I2C_DR register.

When it is detected that the STAT of the I2C bus is a bus error (status code is 00H), since the I2C bus is continuously enabled, and the error has not been cleared for the I2C module, SI will continue to remain 1, that is, SI cannot be cleared.

Bus error clearing method:

1. Transmit a STOP signal to the bus: set the STO bit and clear the SI bit (due to the current bus error state, the controller does not actually transmit the STOP signal to the bus), the STO bit will be automatically cleared by hardware, releasing the bus to normal idle state.
2. If the SI bit cannot be cleared by setting the STO bit, it means that the timing has been disrupted. It is necessary to set I2C_CR.EN to 0 and 1 in turn, that is, shut down and restart the I2C module, and then set SI to 0 to clear the SI bit.

In each operating mode, the I2C bus state transition diagram is shown in the following figure. Caution that when transitioning between the two normal states, when the action is not completed (such as in the process of transmitting SLA), that is, before entering a new state, the status code will appear a short-term transition state, 0xF8. The user does not care, and no interruption is generated.

Figure 18-16 The status transition diagram of master transmit/receive mode

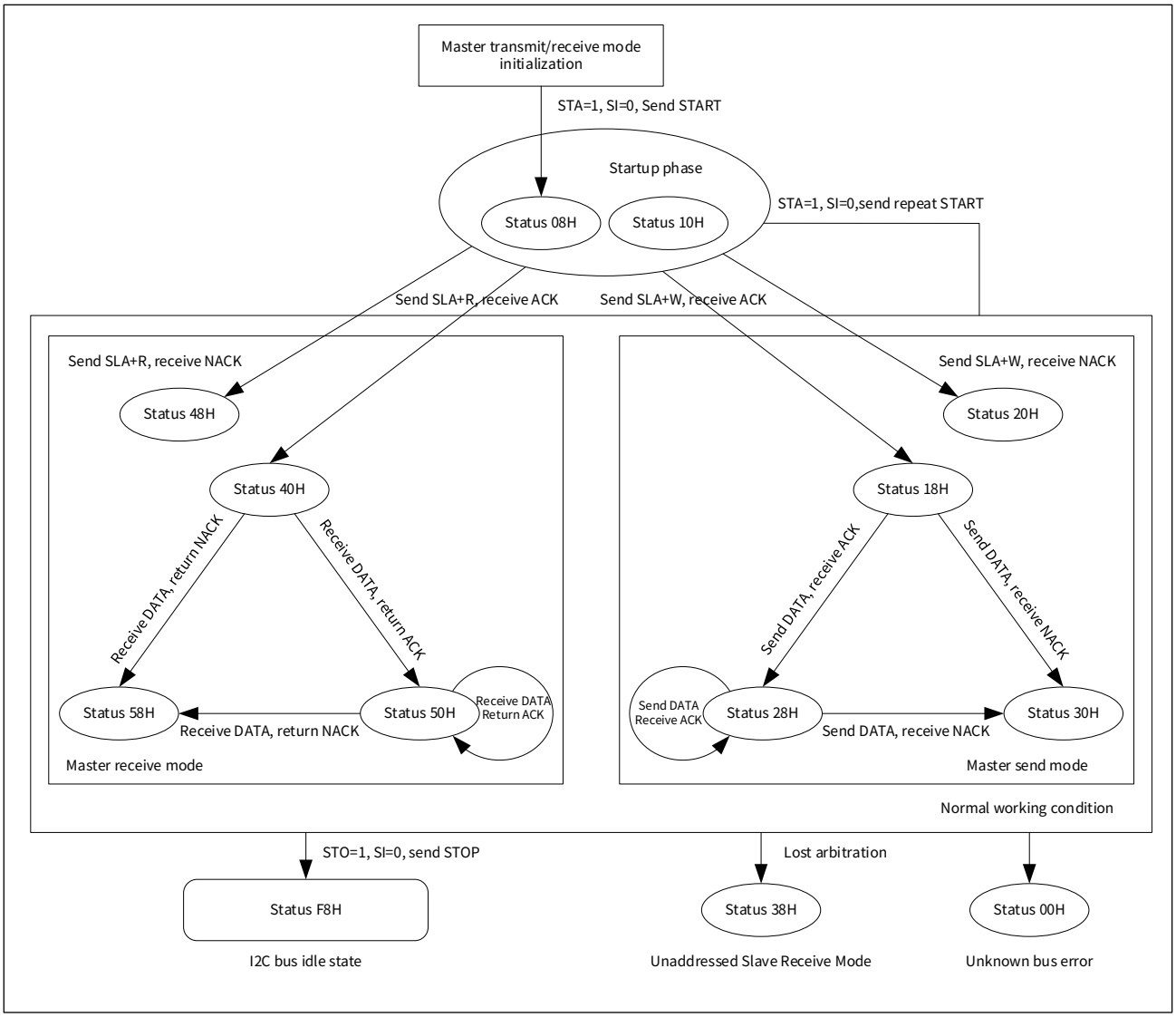
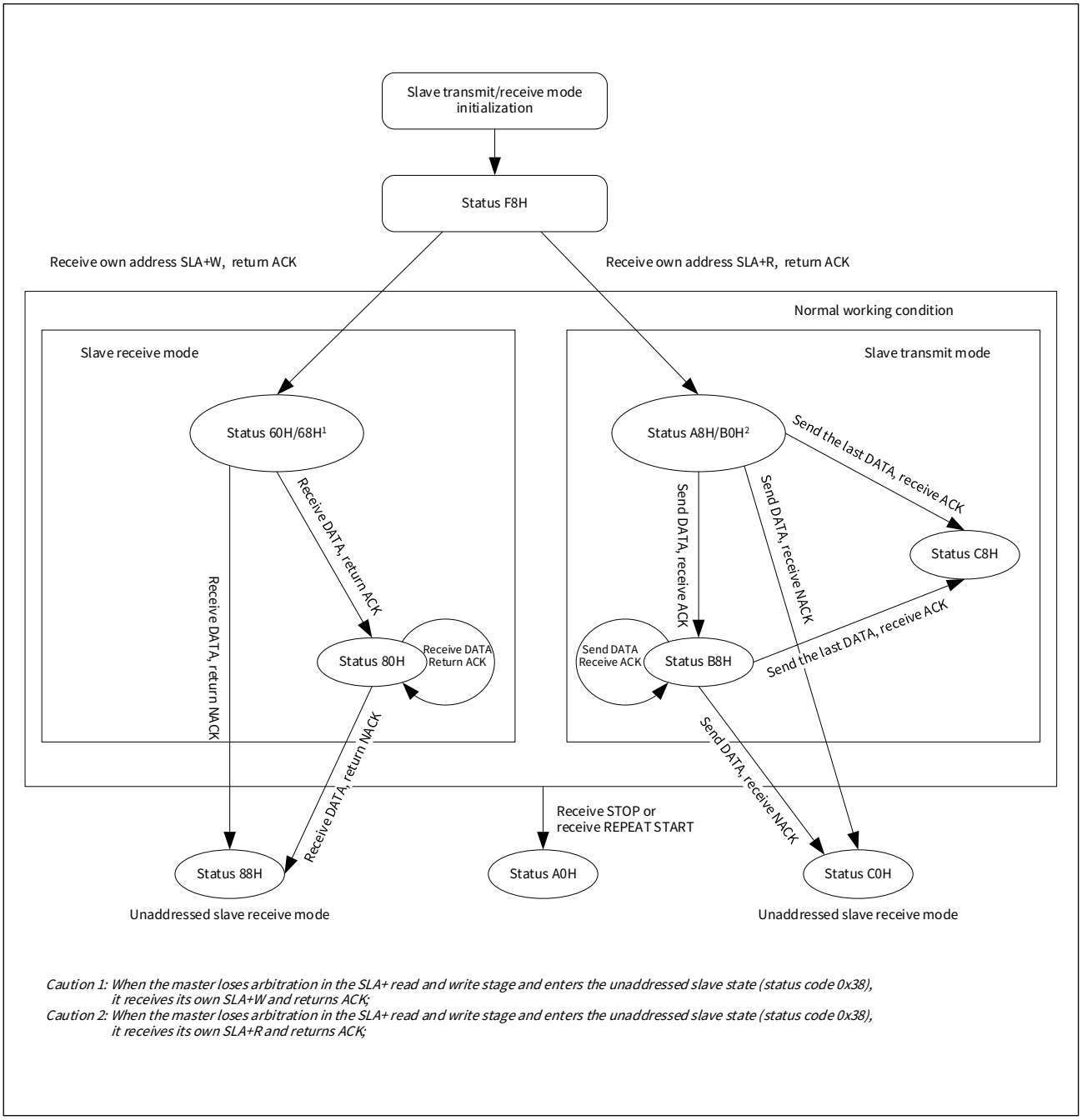
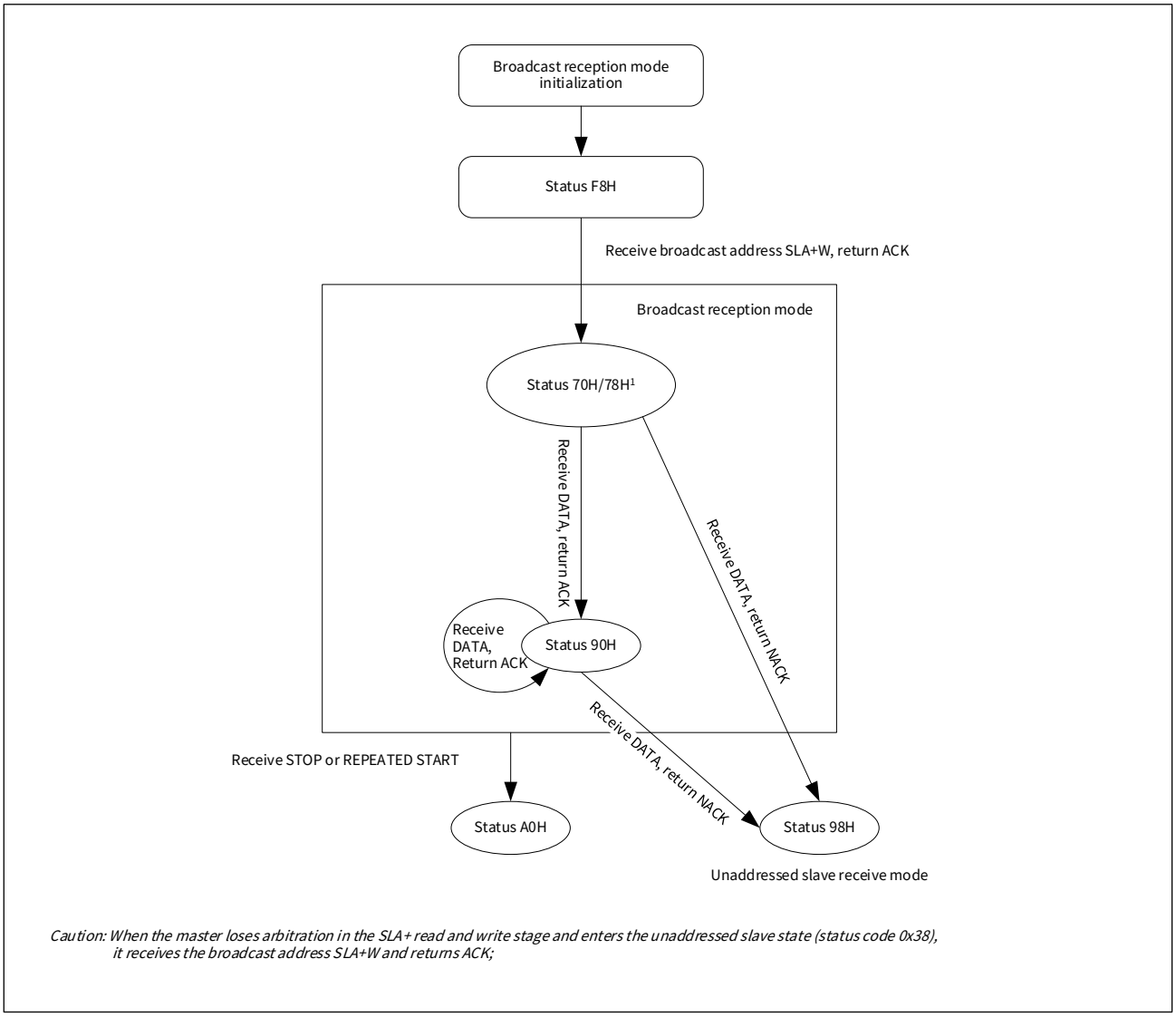


Figure 18-17 The status transition diagram of slave transmit/receive mode




*Caution 1: When the master loses arbitration in the SLA+ read and write stage and enters the unaddressed slave state (status code 0x38), it receives its own SLA+W and returns ACK;*
*Caution 2: When the master loses arbitration in the SLA+ read and write stage and enters the unaddressed slave state (status code 0x38), it receives its own SLA+R and returns ACK;*

Figure 18-18 The status transition diagram of broadcast receive mode




*Caution: When the master loses arbitration in the SLA+ read and write stage and enters the unaddressed slave state (status code 0x38), it receives the broadcast address SLA+W and returns ACK;*

## 18.5 Programming examples

### 18.5.1 Master transmits example

Step 1: According to the relevant description of the pin digital multiplexed function in the GPIO chapter, map SCL and SDA to the required pins, and configure the SCL and SDA pins as open-drain output mode;

Step 2: Set SYSCTRL_APBEN1.I2C to 1 to enable the I2C module clock;

Step 3: Write 0 and 1 to SYSCTRL_APBRST1.I2C in turn to reset the I2C module;

Step 4: Configure I2C_BRR to make the clock rate of SCL meet the application requirements;

Step 5: Set I2C_BRREN to 1 to enable the SCL clock generator;

Step 6: Set I2C_CR.EN to 1 to enable the I2C module;

Step 7: Set I2C_CR.STA to 1, the bus tries to transmit a START signal;

Step 8: Wait for I2C_CR.SI to become 1, START signal has been transmitted to the bus;

Step 9: Query I2C_STAT, if the register value is 0x08 or 0x10, continue to the next step, otherwise, perform error processing;

Step 10: Write SLA+W to I2C_DR, set I2C_CR.STA to 0, set I2C_CR.SI to 0, and transmit SLA+W;

Step 11: Wait for I2C_CR.SI to become 1, SLA+W has been transmitted to the bus;

Step 12: Query I2C_STAT, if the register value is 0x18, continue to the next step, otherwise, perform error processing;

Step 13: Write the data to be transmitted to I2C_DR, set I2C_CR.SI to 0, and transmit the data;

Step 14: Wait for I2C_CR.SI to become 1, the data has been transmitted to the bus;

Step 15: Query I2C_STAT, if the register value is 0x28, continue to the next step, otherwise, perform error processing;

Step 16: If the data to be transmitted is not completed, then jump to step 13 and continue to execute;

Step 17: Set I2C_CR.STO to 1, set I2C_CR.SI to 0, transmit STOP signal, and end this data transfer.

### 18.5.2 Master receives example

Step 1: According to the relevant description of the pin digital multiplexed function in the GPIO chapter, map SCL and SDA to the required pins, and configure the SCL and SDA pins as open-drain output mode;

Step 2: Set SYSCTRL_APBEN1.I2C to 1 to enable the I2C module clock;

Step 3: Write 0 and 1 to SYSCTRL_APBRST1.I2C in turn to reset the I2C module;

Step 4: Configure I2C_BRR to make the clock rate of SCL meet the application requirements;

Step 5: Set I2C_BRREN to 1 to enable the SCL clock generator;

Step 6: Set I2C_CR.EN to 1 to enable the I2C module;

Step 7: Set I2C_CR.STA to 1, the bus tries to transmit a START signal;

Step 8: Wait for I2C_CR.SI to become 1, START signal has been transmitted to the bus;

Step 9: Query I2C_STAT, if the register value is 0x08 or 0x10, continue to the next step, otherwise, perform error processing;

Step 10: Write SLA+W to I2C_DR, set I2C_CR.STA to 0, set I2C_CR.SI to 0, and transmit SLA+R;

Step 11: Wait for I2C_CR.SI to become 1, SLA+R has been transmitted to the bus;

Step 12: Query I2C_STAT, if the register value is 0x40 (ACK has been received), continue to the next step, otherwise, perform error processing;

Step 13: Set I2C_CR.AA to 1, enable the response flag;

Step 14: Set I2C_CR.SI to 0, waiting to receive 1 byte of data (the master transmits the clock, and the slave transmits data under the action of the clock);

Step 15: Wait for I2C_CR.SI to become 1 (the host has completed the 1-byte data reception and has responded to the ACK signal), and read the received data from I2C_DR;

Step 16: Query I2C_STAT, if the register value is 0x50 or 0x58, continue to the next step, otherwise, perform error processing;

Step 17: If the data to be received is only missing the last byte, set I2C_CR.AA to 0, and enable the non-response flag;

Step 18: If the data to be received is not completed, then jump to step 14 and continue to execute;

Step 19: Set I2C_CR.STO to 1, set I2C_CR.SI to 0, transmit STOP signal, and end this data transfer.

### 18.5.3 Slave receives example

Step 1: According to the relevant description of the pin digital multiplexed function in the GPIO chapter, map SCL and SDA to the required pins, and configure the SCL and SDA pins as open-drain output mode;

Step 2: Set SYSCTRL_APBEN1.I2C to 1 to enable the I2C module clock;

Step 3: Write 0 and 1 to SYSCTRL_APBRST1.I2C in turn to reset the I2C module;

Step 4: Set I2C_CR.EN to 1 to enable the I2C module;

Step 5: Configure I2C_ADDR0 as the slave address;

Step 6: Set I2C_CR.AA to 1, enable the response flag;

Step 7: Wait for I2C_CR.SI to become 1 and be addressed by SLA+W;

Step 8: Query I2C_STAT, if the register value is 0x60, continue to the next step, otherwise, perform error processing;

Step 9: Set I2C_CR.SI to 0, wait for the master to transmit data, and respond to the ACK signal;

Step 10: Wait for I2C_CR.SI to become 1, and read the received data from I2C_DR;

Step 11: Query I2C_STAT, if the register value is 0x80, continue to the next step, otherwise, perform error processing;

Step 12: If the data to be received is not completed, then jump to step 9 and continue to execute;

Step 13: Set I2C_CR.AA to 0, set I2C_CR.SI to 0, the slave switches to the unaddressed slave receive mode, and does not respond to master addressing.

### 18.5.4 Slave transmits example

Step 1: According to the relevant description of the pin digital multiplexed function in the GPIO chapter, map SCL and SDA to the required pins, and configure the SCL and SDA pins as open-drain output mode;

Step 2: Set SYSCTRL_APBEN1.I2C to 1 to enable the I2C module clock;

Step 3: Write 0 and 1 to SYSCTRL_APBRST1.I2C in turn to reset the I2C module;

Step 4: Set I2C_CR.EN to 1 to enable the I2C module;

Step 5: Configure I2C_ADDR0 as the slave address;

Step 6: Set I2C_CR.AA to 1, enable the response flag;

Step 7: Wait for I2C_CR.SI to become 1 and be addressed by SLA+R;

Step 8: Query I2C_STAT, if the value of this register is 0xA8, continue to the next step, otherwise, perform error processing;

Step 9: Write the data to be transmitted to I2C_DR, set I2C_CR.SI to 0, and prepare to transmit data;

Step 10: Wait for I2C_CR.SI to become 1, indicating that the data has been transmitted to the bus and received an ACK or NACK response;

Step 11: Query I2C_STAT, if the value of this register is 0xB8, continue to the next step, otherwise, perform error processing;

Step 12: If the data to be transmitted is not completed, then jump to step 9 to continue execution;

Step 13: Set I2C_CR.AA to 0, set I2C_CR.SI to 0, the slave switches to the unaddressed slave receive mode, and does not respond to master addressing.

## 18.6 List of registers

I2C base address: I2C_BASE = 0x4000 5400

Table 18-4 List of I2C registers

| Register name | Register address | Register description |
|---|---|---|
| I2C_BRREN | I2C_BASE + 0x00 | Baud rate counter enable register |
| I2C_BRR | I2C_BASE + 0x04 | Baud rate counter configuration register |
| I2C_CR | I2C_BASE + 0x08 | Control register |
| I2C_DR | I2C_BASE + 0x0C | Data register |
| I2C_ADDR0 | I2C_BASE + 0x10 | Slave address 0 register |
| I2C_STAT | I2C_BASE + 0x14 | Status register |
| I2C_ADDR1 | I2C_BASE + 0x20 | Slave address 1 register |
| I2C_ADDR2 | I2C_BASE + 0x24 | Slave address 2 register |
| I2C_MATCH | I2C_BASE + 0x28 | Slave address match flag register |

## 18.7 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 18.7.1 I2C_BRREN baud rate counter enable register

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:1 | RFU | - | Reserved bits, please keep the default value |
| 0 | EN | RW | I2C bus SCL baud rate counter enable control<br>0: Disabled<br>1: Enabled<br>Caution: EN should be enabled when the master, this bit does not affect the slave |

### 18.7.2 I2C_BRR baud rate counter configuration register

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | BRR | RW | I2C bus SCL baud rate configuration<br>$f_{SCL} = f_{PCLK} / 8 / (BRR+1)$ , where BRR > 0 |

### 18.7.3 I2C_CR control register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | EN | RW | Module enable control<br>0: Disabled<br>1: Enabled |
| 5 | STA | RW | Bus Status Control<br>W0: Clear STA<br>W1: Send START signal to the bus<br>Caution 1: After setting STA to 1, if the bus is idle, send the START signal, if the bus is busy, wait for the I2C to stop, and then send the START signal.<br>Caution 2: If the device is already in master mode and has sent one or more bytes, and then set STA at this time, the I2C bus will generate a Repeated START signal.<br>Caution 3: STA can be set at any time, including slave mode. However, the hardware will not automatically clear 0 after the completion of the START or repeat START signal transmission, and the user needs to manually clear STA. |
| 4 | STO | RW | Bus Status Control<br>W0: No function<br>W1: Send STOP signal to bus<br>Caution 1: The hardware will automatically clear STO to 0 after the completion of the STOP signal transmission.<br>Caution 2: If STA and STO are set at the same time in master mode, the I2C bus sends START immediately after sending STOP. In slave mode, it is forbidden to set STA and STO at the same time to avoid sending illegal I2C frames.<br>Caution 3: STO is also set to 1 when an error state occurs on the bus (STAT status word is 00H), but the I2C bus will not send a STOP signal in this case. |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 3 | SI | RW | I2C interrupt flags<br>R0: No I2C interrupt occurred<br>R1: I2C interrupt has occurred<br>W0: Clear the I2C interrupt flag and cause the status machine to execute the next action<br>W1: No function<br>Caution 1: If one of the 26 states of I2C occurs, the hardware will set this bit (except F8H) to 1. At this time, the software confirms the current status of the bus through the I2C_STAT register value.<br>Caution 2: SI needs to be cleared by software.<br>Caution 3: Before SI is cleared to 0, the SCL low level period is extended and the transmission is suspended. This status is very useful for the slave to process the received data, which can ensure that the previous data is accurately processed before the next data is received.<br>Caution 4: Before software clears SI, software should prepare appropriate register settings. After SI is cleared, the I2C bus will perform the corresponding operation according to the register settings. |
| 2 | AA | RW | Response control<br>0: Transmit NACK in response phase<br>1: Transmit ACK in response phase<br>Caution 1: For the addressed slave, if no ACK response bit is returned in slave receive mode or no ACK response bit is received in slave transmit mode, the slave will switch to unaddressed slave receive mode, cannot receive data until its AA is set and re-addressed by the master.<br>Caution 2: Special case: in slave transmit mode, before the slave transmits the last byte to the master, clear AA. After transmitting the last byte, the slave will switch to the unaddressed slave mode and be disconnected from the master, the status register I2C_STAT is C8H. If the master reads data from the bus again, it will get 0xFF. |
| 1 | RFU | - | Reserved bits, please keep the default value |
| 0 | FLT | RW | I2C filter parameter configuration<br>0: Advanced filtering, higher anti-interference performance<br>1: Simple filtering, faster communication rate<br>Caution: See section *18.4.3 Input filter*. |

### 18.7.4 I2C_DR data register

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | DR | RW | Data register<br>In transmit mode, write the transmitted data<br>In receive mode, read the received data |

### 18.7.5 I2C_STAT status register

Address offset: 0x14 Reset value: 0x0000 00F8

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:0 | STAT | RO | I2C status register, see section *18.4.10 I2C status codes* for the specific definition of status value; when STAT = F8H, it means that there is no relevant status information available, and SI will remain at 0. For the other 26 states, SI will be set to 1 and an interrupt request will be generated. |

### 18.7.6 I2C_ADDR0 slave address 0 register

Address offset: 0x10 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:1 | ADDR0 | RW | Slave mode address 0<br>Caution 1: The master mode is invalid.<br>Caution 2: The master needs to address the slave, and the address information must be the same as this address through the first byte value address information after START or Repeated START. If AA is 1, the slave responds to the master and becomes the addressed slave, otherwise the master broadcast addressing information will be ignored.<br>Caution 3: I2C_ADDR0[7:1] cannot be written as all 0, because 0x00 is reserved for broadcast addressing. |
| 0 | GC | RW | Broadcast address response enabled<br>0: Disabled<br>1: Enabled<br>Caution 1: The master mode is invalid.<br>Caution 2: After enabling GC, if AA is set to 1, the broadcast receive mode is enabled, and if AA is cleared to 0, the broadcast addressing information on the bus is ignored. |

### 18.7.7 I2C_ADDR1 slave address 1 register

Address offset: 0x20 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:1 | ADDR1 | RW | Slave mode address 1<br>Caution: The master mode is invalid. |
| 0 | RFU | - | Reserved bits, please keep the default value |

### 18.7.8 I2C_ADDR2 slave address 2 register

Address offset: 0x24 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7:1 | ADDR2 | RW | Slave mode address 2<br>Caution: The master mode is invalid. |
| 0 | RFU | - | Reserved bits, please keep the default value |

### 18.7.9 I2C_MATCH slave address match register

Address offset: 0x28 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:3 | RFU | - | Reserved bits, please keep the default value |
| 2 | ADDR2 | RO | I2C slave mode address 2 match flag bit<br>0: The device address received from the bus is not the same as ADDR2<br>1: The device address received from the bus is the same as ADDR2 |
| 1 | ADDR1 | RO | I2C slave mode address 1 match flag bit<br>0: The device address received from the bus is not the same as ADDR1<br>1: The device address received from the bus is the same as ADDR1 |
| 0 | ADDR0 | RO | I2C slave mode address 0 match flag bit<br>0: The device address received from the bus is not the same as ADDR0<br>1: The device address received from the bus is the same as ADDR0 |

*Caution:*

*The address match flag bit will be cleared in the following three cases:*

- When the module is reset;
- When START/STOP is transmitted.

# 19 Infrared modulation transmitter (IR)

## 19.1 Overview

The CW32F003 integrates an infrared modulation transmitter (IR), which can be used in conjunction with a general-purpose timer and a basic timer, a general-purpose timer and a soft control bit, or a timer and a UART, which can easily implement various standard PWM or PPM encoding methods, and can and also realize infrared modulation transmission of UART data.

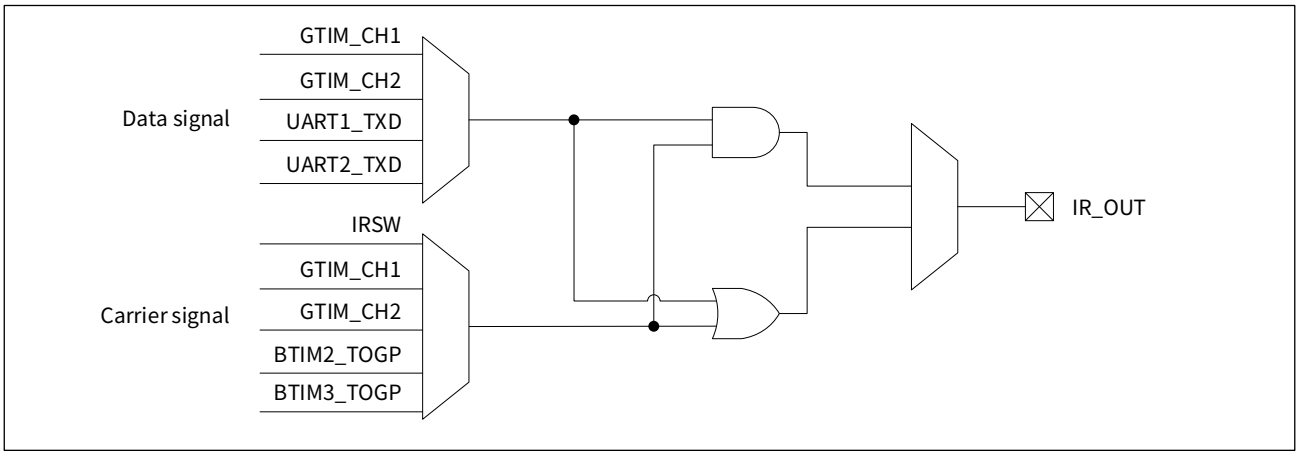
## 19.2 Main features

- Supports SIR of IrDA standard 1.0
- Maximum data rate 115.2kbps
- Can adapt to high and low level infrared emission tube

## 19.3 Functional description

When the infrared modulation transmitter is implemented, a timer channel is used to generate a square wave signal with a fixed frequency, and another timer, soft control bit IRSW or UART is used to generate modulation data. After 'AND' or 'OR' operation of the two, it is converted from IR_OUT pin output, user can select PA04/PB03/PC02/PC04 as IR_OUT output. The schematic diagram of internal connection of IR is shown in the following figure:

Figure 19-1 Schematic diagram of internal connection of IR




The IR infrared modulation control register SYSCTRL_IRMOD is used to select the source of the carrier signal and the data signal, as well as the 'AND' or 'OR' operation of the two. The choice of 'AND' and 'OR' is determined by the drive level of the user's hardware infrared emitter.

The carrier signal frequency can be set by the user, the most common carrier frequency is 38KHz.

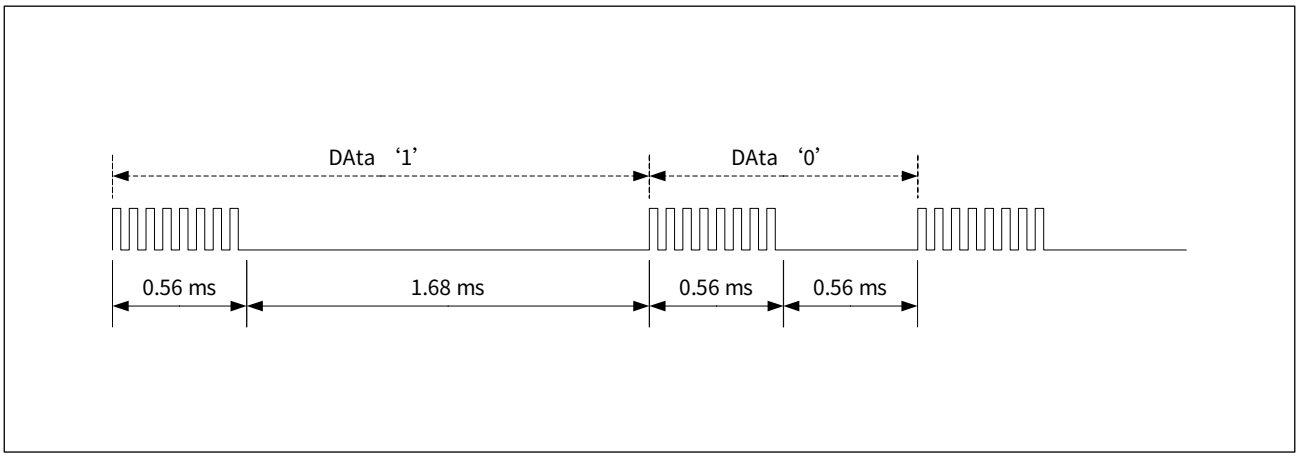
In order to reduce the transmission power consumption, the duty cycle of the carrier is generally set to about 20%, and the timer that generates the carrier can use the PWM mode or the flip output mode. In order to increase the transmission distance, the duty cycle can be appropriately increased. In the IrDA standard 1.0, in order to achieve a fast IR communication rate, the width of the pulse is specified as 3/16 of a bit period or a fixed 1.63 μs, and the minimum cannot be lower than 1.41μs.

### 19.3.1 Infrared modulation mode

There are two common IR infrared encoding methods, pulse width modulation PWM and pulse position modulation PPM.

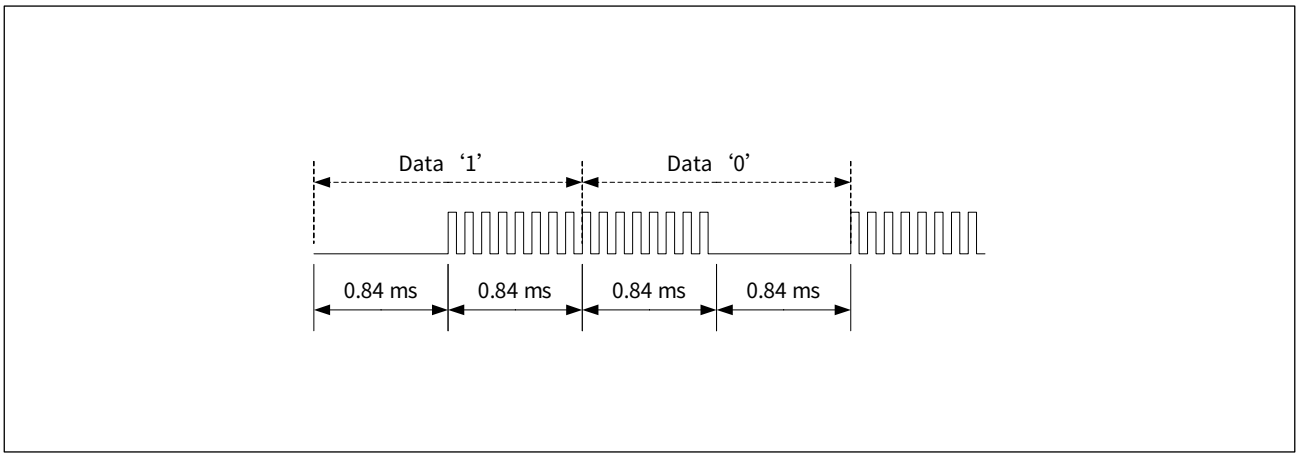
PWM pulse width modulation to represent '0' and '1' with the duty cycle of the transmitted infrared carrier. For example, UPD6121, the carrier transmits 0.56ms, and does not transmit 0.56ms, which means '0'; the carrier transmits 0.56ms, and does not transmit 1.68ms, which means '1'. The reference diagram is shown in the following figure:

Figure 19-2 PWM modulation mode



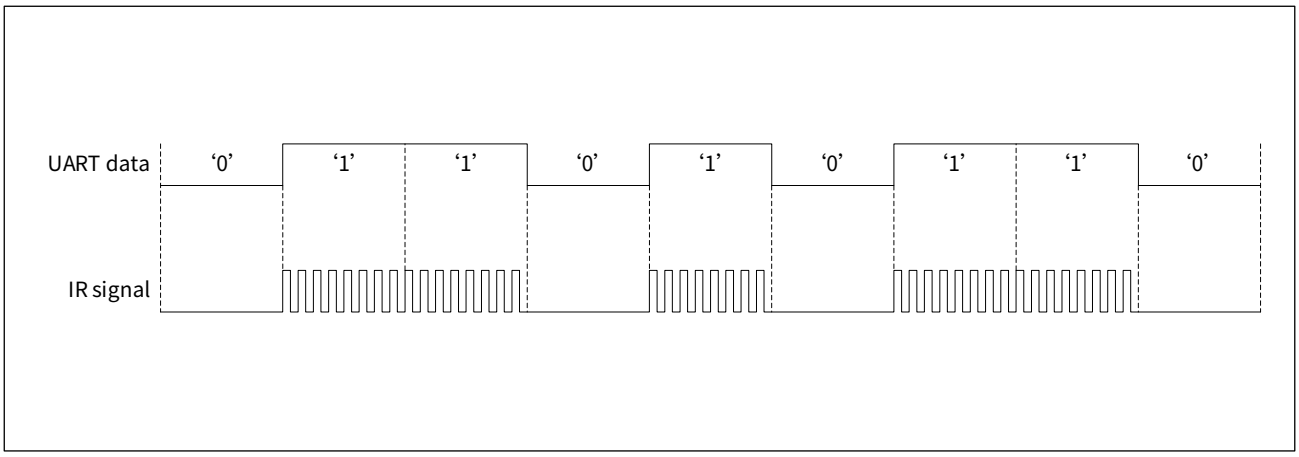

PPM pulse position modulation to represent '0' and '1' with the position of the transmitted carrier. It is '0' from transmitting carrier to not transmitting carrier, and '1' from never transmitting carrier to transmitting carrier. For example, SAA3010, the carrier transmits 0.84ms, and does not transmit 0.84ms, which means '0'; does not transmit 0.84ms, and the carrier transmits 0.84ms, means '1'. The reference diagram is shown in the following figure:

Figure 19-3 PPM modulation mode



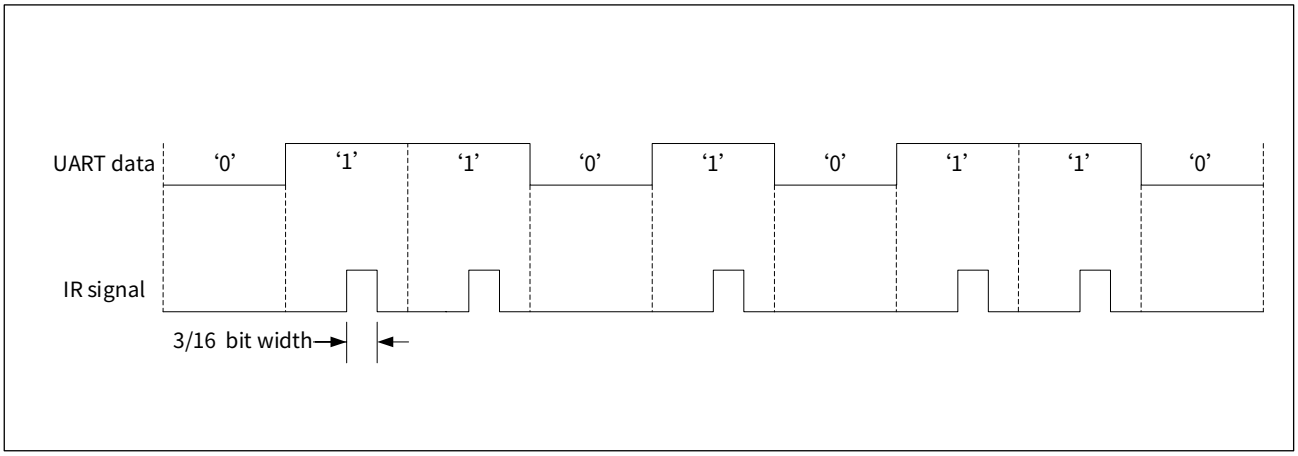

UART modulation is a relatively simple infrared modulation method, which directly performs 'AND' or 'OR' operation on serial data and carrier signal. When the UART serial baud rate is lower than the carrier frequency, the reference schematic diagram is shown in the following figure:

Figure 19-4 UART modulation mode




When the UART serial baud rate is higher than the carrier frequency, the carrier pulse width should be set to 3/16 of the bit width, and the carrier frequency should be kept consistent with the serial frequency. For example, when the UART baud rate is set to 115.2Kbps, the carrier frequency is also set to 115.2KHz, and the width of the carrier pulse is 1.63μs (3/16 * 1/115200Hz = 1.63μs). The reference schematic diagram is shown in the following figure:

Figure 19-5 High-speed UART modulation mode




For the convenience of debugging, usually a standard data frame also includes boot code, end bit, retransmission code, etc. Please refer to the relevant data manual by yourself.

### 19.3.2 Infrared modulation initialization configuration

The initialization process of a complete IR infrared modulation transmitter should include the configuration of timer, UART, and GPIO. The reference steps are as follows:

Step 1: Set the SYSCTRL_APBEN1 and SYSCTRL_APBEN2 registers to enable the timer and UART peripheral clock to be selected;

Step 2: Set the GPIO clock enable bit to be used in SYSCTRL_AHBEN;

Step 3: Configure the timer selected as the carrier, set the carrier frequency, duty cycle (common frequency 38KHz, 1/3 duty cycle), and set the timer working mode (PWM output is recommended);

Step 4: Configure the timer or UART selected as data, set the bit width, and UART working mode (no need to define output pins);

Step 5: Set GPIOx_ANALOG as IR_OUT output port as digital mode, set GPIOx_DIR as output, and set appropriate GPIOx_OPENDRAIN, GPIOx_PUR and GPIOx_PDR;

Step 6: Start the timer and UART as carrier and data;

Step 7: Set the MOD bit field of the IR infrared modulation control register SYSCTRL_IRMOD, and set the infrared modulation mode configuration;

Step 8: Set the GPIOx_AFRL of the IR_OUT output port to IR_OUT, and enable the IR multiplexing function.

### 19.3.3 Infrared reception

CW32F003 does not have an IR receiving demodulation module inside. In IR receiving applications, it is necessary to use an integrated infrared receiving head with demodulation function. With the capture function of GTIM (UART mode can directly use the RXD pin input), it can be easily realized IR receiving function.

## 19.4 SYSCTRL_IRMOD infrared modulation control register

Address: 0x4001 0000 + 0x74 Reset value: 0x0000 0000

| Bit field | Name | Permission | Functional description |
|---|---|---|---|
| 31:5 | RFU | - | Reserved bits, please keep the default value |
| 4 | IRSW | RW | Infrared modulation soft control bit<br>The functions are shown in the following |
| 3:0 | MOD | RW | Infrared modulation mode configuration<br>Infrared modulation mode configuration<br>0000: GTIM_CH1 & BTIM2_TOGP<br>0001: GTIM_CH1 & IRSW<br>0010: GTIM_CH2 & BTIM3_TOGP<br>0011: GTIM_CH2 & IRSW<br>0100: GTIM_CH1 \| BTIM2_TOGP<br>0101: GTIM_CH1 \| IRSW<br>0110: GTIM_CH2 \| BTIM3_TOGP<br>0111: GTIM_CH2 \| IRSW<br>1000: UART1_TXD & GTIM_CH1<br>1001: UART1_TXD \| GTIM_CH1<br>1010: UART1_TXD & GTIM_CH2<br>1011: UART1_TXD \| GTIM_CH2<br>1100: UART2_TXD & BTIM2_TOGP<br>1101: UART2_TXD \| BTIM2_TOGP<br>1110: UART2_TXD & BTIM3_TOGP<br>1111: UART2_TXD \| BTIM3_TOGP |

# 20 Analog-to-digital converter (ADC)

## 20.1 Overview

CW32F003 integrates a successive approximation analog-to-digital converter (SAR ADC) with a 12-bit precision and a maximum conversion speed of 1M SPS, which can convert up to 16 analog signals into digital signals. The vast majority of signals in the real world are analog, such as light, electricity, sound, image signals, etc., which must be converted into digital signals by ADC before they can be digitally processed by MCU.

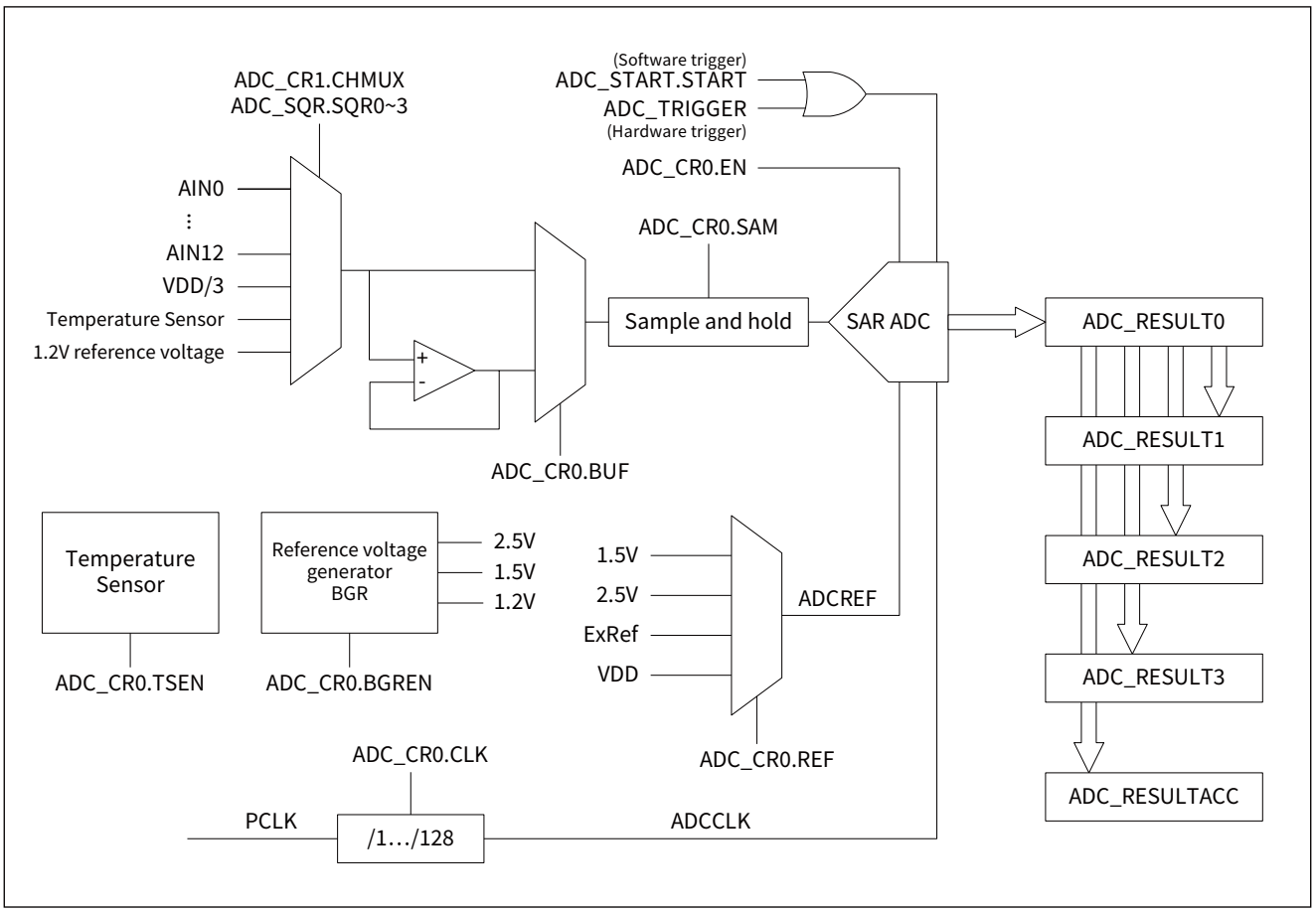
## 20.2 Main features

- 12-bit precision
- Programmable conversion speed, up to 1M SPS
- 16 channels of input conversion
  - 13 external pin inputs
  - Built-in temperature sensor
  - Built-in BGR 1.2V reference
  - 1/3 VDD supply voltage
- 4-channel reference voltage source (Vref)
  - VDD supply voltage
  - ExRef (PB04) pin voltage
  - Built-in 1.5V reference voltage
  - Built-in 2.5V reference voltage
- Sampling voltage input range: 0 ~ $V_{ref}$
- Multiple conversion modes, all support conversion accumulation function
  - Single conversion
  - Multiple conversions
  - Continuous conversion
  - Sequence scan conversion
  - Sequence discontinuous conversion
- Supports single channel and sequence channel selection, and supports up to 4 sequences at the same time
- Supports input channel voltage threshold monitoring
- Built-in signal follower to convert high impedance input signals
- Supports on-chip peripherals to automatically trigger ADC conversion

## 20.3 Functional block diagram

The ADC functional block diagram is shown in the following figure:

Figure 20-1 ADC functional block diagram

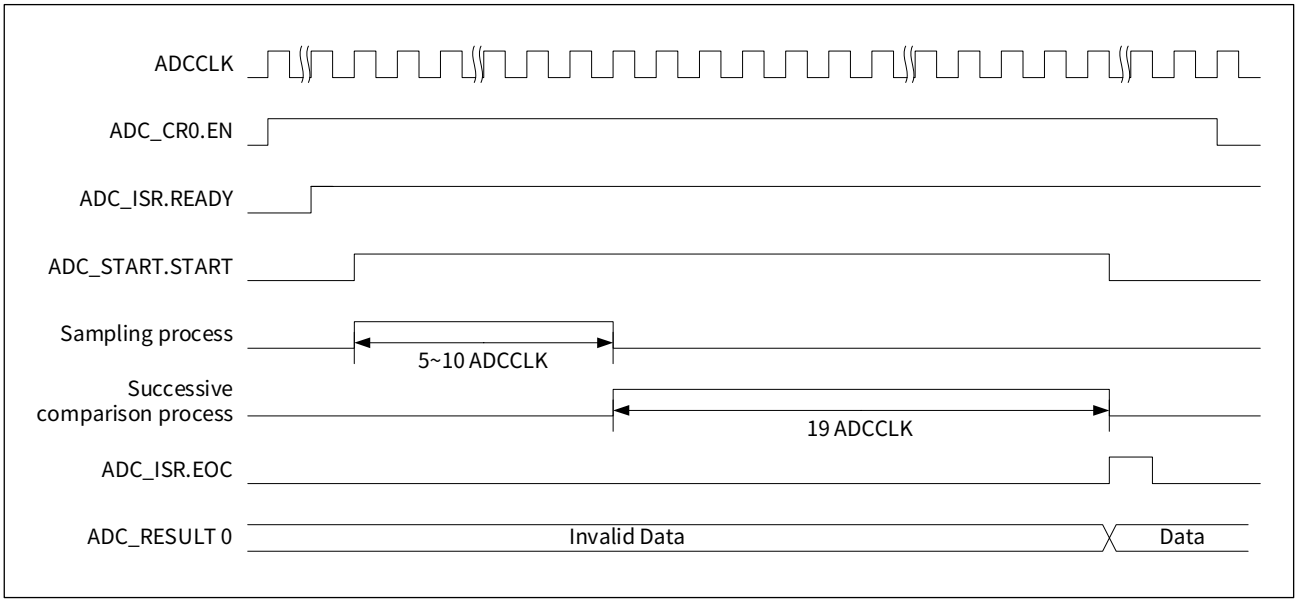


## 20.4 Conversion timing, conversion speed, conversion accuracy, and conversion results

### 20.4.1 Conversion timing

The conversion timing of the ADC is shown in the following figure:

Figure 20-2 ADC conversion timing diagram




Write 1 to the EN bit field of the ADC control register ADC_CR0 to enable the ADC module.

About 40μs after ADC_CR0.EN changes from 0 to 1, the ADC_ISR.READY flag is set to 1, indicating that the initialization of the analog circuit is completed, and the ADC conversion can be started.

Write 1 to the START bit field of the ADC start register ADC_START to start the ADC conversion, and the hardware will automatically clear it after the conversion is completed.

The ADC working clock ADCCLK is obtained by dividing the system clock PCLK by the prescaler. The CLK bit field of the control register ADC_CR0 can be selected to divide the frequency from 1 to 128, as shown in the following table:

Table 20-1 ADC clock configuration table

| ADC_CR0.CLK | ADCCLK |
|---|---|
| 000 | PCLK |
| 001 | PCLK/2 |
| 010 | PCLK/4 |
| 011 | PCLK/8 |
| 100 | PCLK/16 |
| 101 | PCLK/32 |
| 110 | PCLK/64 |
| 111 | PCLK/128 |

A complete ADC conversion requires 24~29 ADCCLK clock cycles, including sampling phase and successive comparison phase:

1. Sampling phase: 5~10 ADCCLK clock cycles are required. The sampling period is configured by the SAM bit field of the control register ADC_CR0, as shown in the following table:

Table 20-2 ADC sampling period selection table

| ADC_CR0.SAM | ADC sampling period (ADCCLK number) |
|---|---|
| 00 | 5 |
| 01 | 6 |
| 10 | 8 |
| 11 | 10 |

The length of the ADC sampling period is determined by the user's sampling speed requirements and the electrical characteristics of the sampling signal. The user should select an appropriate sampling period to achieve the best conversion effect.

2. Successive comparison phase: 19 ADCCLK clock cycles are required.

After the ADC conversion is completed, the conversion completion flag ADC_ISR.EOC will be set to 1 by hardware, and the ADC conversion result is stored in the corresponding ADC conversion result register ADC_RESULTy (y=0,1,2,3), the user can set ADC_ICR.EOC to 0 to clear this flag.

### 20.4.2 Conversion speed

The ADC conversion speed is closely related to the ADC reference voltage and VDD supply voltage, and the maximum conversion speed under various conditions is shown in the following table:

Table 20-3 ADC conversion speed and voltage comparison table

| ADC reference voltage | VDD voltage | Maximum conversion speed | Maximum ADCCLK frequency |
|---|---|---|---|
| Internal 1.5V | 1.8V ~ 2V | 100K SPS | 2MHz |
| Internal 1.5V | 2V ~ 5.5V | 200K SPS | 4MHz |
| Internal 2.5V | 2.8V ~ 5.5V | 200K SPS | 4MHz |
| VDD/ExRef | 1.65V ~ 1.8V | 25K SPS | 500kHz |
| VDD/ExRef | 1.8V ~ 2V | 100K SPS | 2MHz |
| VDD/ExRef | 2V ~ 2.4V | 200K SPS | 4MHz |
| VDD/ExRef | 2.4V ~ 2.7V | 500K SPS | 12MHz |
| VDD/ExRef | 2.7V ~ 5.5V | 1M SPS | 24MHz |

The corresponding relationship between the conversion speed of the ADC and the working clock ADCCLK is as follows:

$$\text{ADC conversion speed} = f_{ADCCLK} / N_T$$

Where, $f_{ADCCLK}$ is the clock frequency of ADCCLK, and $N_T$ is ADCCLK number required for one ADC conversion.

### 20.4.3 Conversion accuracy

When the ADC external input signal driving capability is insufficient, or the ADC input comes from the chip (built-in temperature sensor voltage, built-in 1.2V reference voltage or 1/3 VDD voltage), the built-in signal follower of the ADC module must be enabled, and a single channel must be used. Single conversion mode. The built-in signal follower is controlled by the BUF bit field of the control register ADC_CR0. Set BUF to 1 to enable the follower; set BUF to 0 to disable the follower.

When multi-channel ADC conversion is selected, and the signal driving capability of some of the channels is weak, in order to avoid the interference of the input channel with weak ADC driving capability, the signal follower must be enabled, and the ADC conversion speed must be no higher than 200K SPS.

To further improve the ADC conversion accuracy, the user can use the ADC conversion accumulation function to sample the same channel multiple times, and use the arithmetic mean of the accumulated results as the final measurement value. For details, please refer to section *20.6 Accumulation conversion function*.

### 20.4.4 Conversion results

After the ADC conversion is completed, the 12-bit ADC conversion result is stored in the corresponding conversion result register ADC_RESULTy.

When the ADC works in single-channel conversion mode, the conversion result is stored in the ADC_RESULT0 register.

When the ADC works in the sequence conversion mode, the conversion result of the conversion sequence SQRy (y=0,1,2,3) is stored in the corresponding ADC_RESULTy (y=0,1,2,3) register.

The conversion result register ADC_RESULTy is 16 bits wide, and the user can choose left-aligned or right-aligned, which is determined by the ALIGN bit field of the control register ADC_CR1:

- The ALIGN bit is 0, select right alignment, the effective value is stored in the lower 12 bits (bit 11:0) of the ADC_RESULTy register, and the higher bits (bit 15:12) are automatically filled with 0;
- When the ALIGN bit is 1, the left alignment is selected, and the effective value is stored in the higher 12 bits (bit 15:4) of the ADC_RESULTy register, and the lower bits (bit 3:0) are automatically filled with 0.

## 20.5 Operating modes

The MODE bit field of the ADC control register ADC_CR0 configures the ADC operating mode, as shown in the following table:

Table 20-4 ADC operating mode configuration

| ADC_CR0.MODE | ADC operating mode |
|---|---|
| 000 | Single channel single conversion mode |
| 001 | Single channel multiple conversion mode, see CR2.CNT for conversion times |
| 010 | Single channel continuous conversion mode |
| 011 | Sequence continuous conversion mode |
| 100 | Sequence scan conversion mode |
| 101 | Sequence multiple conversion mode, see CR2.CNT for conversion times. |
| 110 | Sequence intermittent conversion mode |

The ADC conversion can be started by writing 1 to the START bit field of the ADC start register ADC_START; it can also be triggered by other peripherals, see section *20.8 External trigger source*.

### 20.5.1 Single channel single conversion mode (MODE=0)

After the ADC is started, a conversion is performed on a specified channel.

The ADC has 16 channels that can be selected, which is determined by the CHMUX bit field of the control register ADC_CR1, as shown in the following table. Among them, AIN0~AIN12 are external pin inputs, and the analog function of the corresponding GPIO must be enabled before use (GPIOx_ANALOG.PINy = 1).

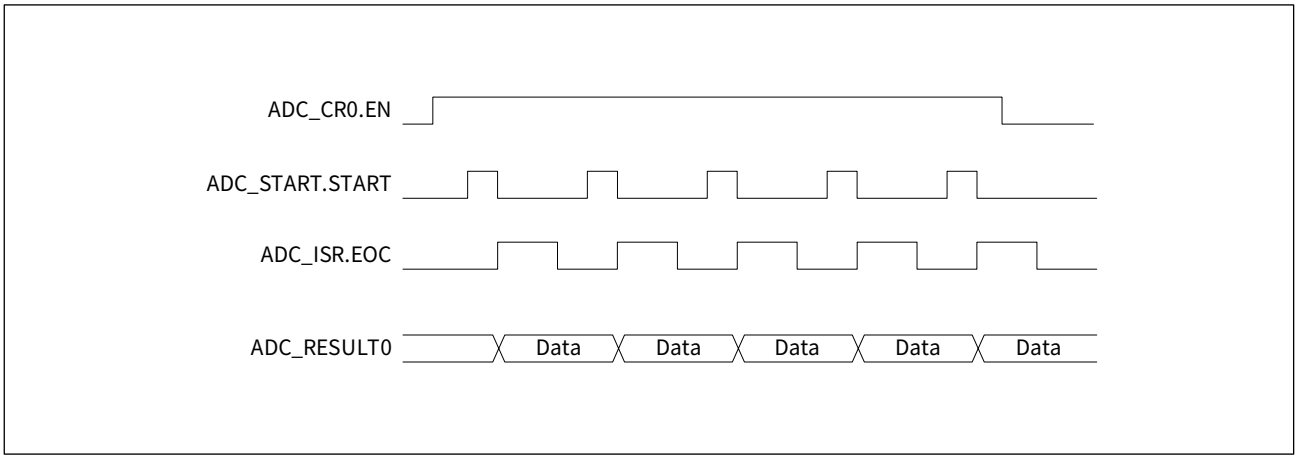
Table 20-5 Single channel configuration

| ADC_CR1.CHMUX | Channel selection | GPIO |
|---|---|---|
| 0000 | AIN0 | PB02 |
| 0001 | AIN1 | PA01 |
| 0010 | AIN2 | PA04 |
| 0011 | AIN3 | PA06 |
| 0100 | AIN4 | PA07 |
| 0101 | AIN5 | PC00 |
| 0110 | AIN6 | PC01 |
| 0111 | AIN7 | PC02 |
| 1000 | AIN8 | PB00 |
| 1001 | AIN9 | PB01 |
| 1010 | AIN10 | PB06 |
| 1011 | AIN11 | PB05 |
| 1100 | AIN12 | PB03 |
| 1101 | VDD/3 | - |
| 1110 | TS built-in temperature sensor | - |
| 1111 | 1.2V core voltage reference source | - |

In the single channel single conversion mode, after the ADC conversion is completed, the conversion completion flag ADC_ISR.EOC will be automatically set to 1 by the hardware, the conversion result is stored in the ADC_RESULT0 register, and the START bit of the ADC start register ADC_START is automatically cleared to 0, and the ADC conversion stop.

The timing of single channel single conversion mode is shown in the following figure:

Figure 20-3 Single channel single conversion timing diagram




The single conversion of the ADC single channel to the external analog input signal is started through the START bit. The reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1 and SYSCTRL_APBEN2.ADC to 1 to enable the GPIO clock and ADC working clock corresponding to the ADC channel;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set ADC_CR0.EN to 1 to enable the ADC module;

Step 4: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 5: Set ADC_CR0.MODE to 0, select single channel single conversion mode;

Step 6: Configure ADC_CR0.REF and select the reference voltage source of ADC;

*Caution:*

*If an external reference voltage pin is selected, this pin must be configured as an analog function first.*

Step 7: Configure ADC_CR0.SAM and ADC_CR0.CLK, set ADC sampling speed and clock selection;

Step 8: Configure ADC_CR1.CHMUX and select the channel to be converted;

Step 9: Set ADC_START.START to 1 to start ADC conversion;

Step 10: Wait for ADC_ISR.EOC to become 1, and read the ADC conversion result in the ADC_RESULT0 register;

Step 11: If it is necessary to convert other channels, repeat steps 8 to 10;

Step 12: Set ADC_CR0.EN to 0 to turn off the ADC module.

Start a single conversion of the ADC single channel to the external analog input signal through an external trigger, and use the ADC conversion completion interrupt to read the conversion result. The reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1 and SYSCTRL_APBEN2.ADC to 1 to enable the GPIO clock and ADC working clock corresponding to the ADC channel;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set ADC_CR0.EN to 1 to enable the ADC module;

Step 4: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 5: Set ADC_CR0.MODE to 0, select single channel single conversion mode;

Step 6: Configure ADC_CR0.REF and select the reference voltage of ADC;

*Caution:*

*If an external reference voltage pin is selected, this pin must be configured as an analog function first.*

Step 7: Configure ADC_CR0.SAM and ADC_CR0.CLK, set ADC sampling speed and clock selection;

Step 8: Set ADC_IER.EOC to 1 to enable ADC conversion complete interrupt;

Step 9: Enable ADC interrupt in NVIC interrupt vector table;

Step 10: Set ADC_ICR to 0x00, clear ADC interrupt flag;

Step 11: Configure the external trigger register ADC_TRIGGER and select the external trigger source;

Step 12: Configure ADC_CR1.CHMUX and select the channel to be converted;

Step 13: Wait for the ADC to be triggered to start. When the ADC conversion is completed, the ADC_ISR.EOC flag is set to 1 by hardware, and the MCU responds to the ADC interrupt and enters the ADC interrupt service routine. The user reads the ADC conversion result in the ADC_RESULT0 register; when exiting the service routine, the ADC_ISR.EOC flag should be cleared first.;

Step 14: If it is necessary to convert other channels, repeat steps 11 to 13;

Step 15: Set ADC_CR0.EN to 0 to turn off the ADC module.

### 20.5.2 Single channel multiple conversion mode (MODE=1)

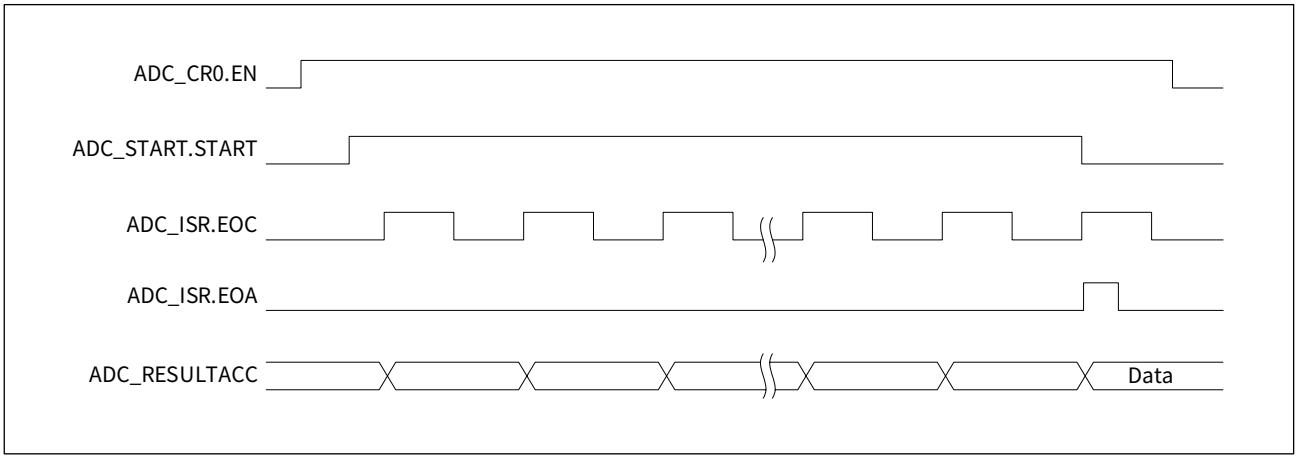
After the ADC is started, it performs multiple conversions for a specified channel. The number of conversions is determined by the ADC_CR2.CNT bit field value. The default number of conversion is 1.

In multiple conversion mode, the accumulation conversions function must be enabled, that is, setting ADC_CR2.ACCRST to 1 clears the ADC_RESULTACC register, while setting ADC_CR2.ACCEN to 1 enables the automatic accumulation function of ADC conversion results.

After each ADC conversion is completed, the ADC_ISR.EOC flag is automatically set to 1, the conversion result is stored in the ADC_RESULT0 register, and the conversion result is automatically accumulated, and the accumulated value is stored in the ADC_RESULTACC register. If the set ADC conversion times are not reached, the ADC will continue to convert; when the set ADC conversion times are reached, the multiple conversion completion flag ADC_ISR.EOA is automatically set to 1, and the ADC_START.START bit is automatically cleared to 0, the ADC conversion stops.

The timing of single channel multiple conversion mode is shown in the following figure:

Figure 20-4 Single channel multiple conversion timing diagram



The ADC single channel multiple conversion are started through the START bit. The reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBEN2.ADC to 1, enable the GPIO clock corresponding to the ADC channel, the GPIO clock corresponding to the external reference voltage pin, and the ADC working clock;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set the GPIO pin corresponding to the external reference voltage to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

*Caution:*

*If the ADC reference voltage does not select the external reference voltage pin, skip this step.*

Step 4: Set ADC_CR0.EN to 1 to enable the ADC module;

Step 5: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 6: Set ADC_CR0.MODE to 1 and select single channel multiple conversions mode;

Step 7: Configure ADC_CR0.REF and select the reference voltage of ADC;

Step 8: Configure ADC_CR0.SAM and ADC_CR0.CLK, set ADC sampling speed and clock selection;

Step 9: Configure ADC_CR1.CHMUX and select the channel to be converted;

Step 10: Configure ADC_CR2.CNT and set the conversion times;

Step 11: Set ADC_CR2.ACCRST and ADC_CR2.ACCEN to 1 to enable the conversion result accumulation function;

Step 12: Set ADC_IER.EOA to 1 to enable multiple conversion completion interrupts;

Step 13: Enable ADC interrupt in NVIC interrupt vector table;

Step 14: Set ADC_ICR.EOA to 0, clear the EOA interrupt flag;

Step 15: Set ADC_START.START to 1 to start ADC conversion;

Step 16: Query and wait for ADC_ISR.EOA to become 1, indicating that multiple conversions are completed, ADC_START.START bit is automatically cleared to 0, and ADC conversion stops. At this time, the user can read the ADC_RESULTACC register to obtain the accumulated value of the ADC conversion result, and divide it by the conversion times to obtain the ADC conversion result;

Step 17: If you need to convert other channels, repeat steps 9 to 16;

Step 18: Set ADC_CR0.EN to 0 to turn off the ADC module.

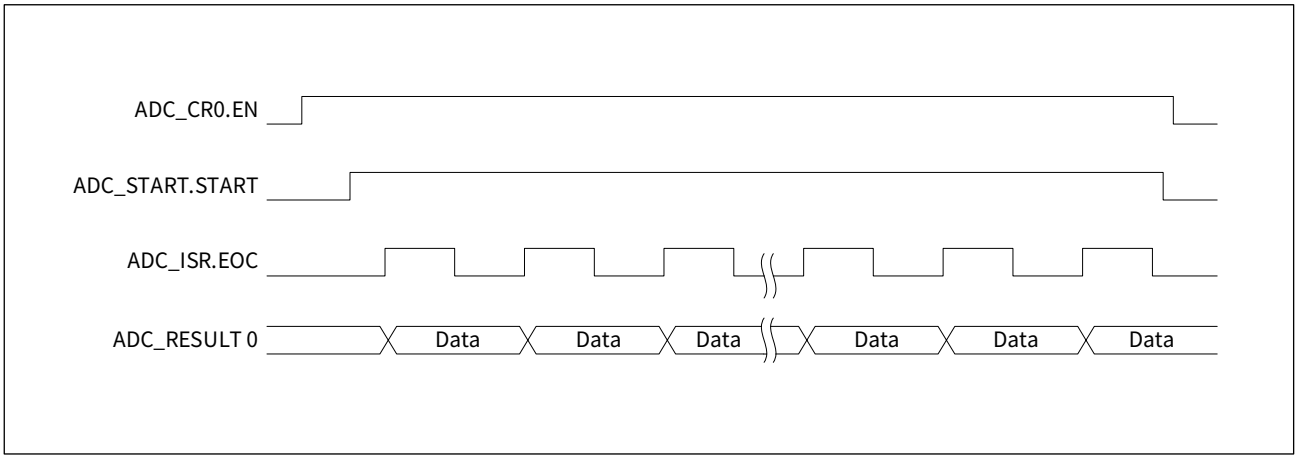
### 20.5.3 Single channel continuous conversion mode (MODE=2)

In this mode, whether the ADC is started by the software START bit or the external trigger, once the ADC is started, it will continue to convert a specified channel until the ADC_START.START bit is cleared to 0, and the conversion will not be stopped.

After each ADC conversion is completed, the ADC_ISR.EOC flag is automatically set to 1, and the conversion result is stored in the ADC_RESULT0 register. The user should read the conversion result in ADC_RESULT0 in time to avoid conversion result overflow. User writes 0 to ADC_START.START bit to stop conversion.

The timing of single channel continuous conversion mode is shown in the following figure:

Figure 20-5 Single channel continuous conversion timing diagram




The ADC single channel continuous conversion is started through the START bit, and the reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBEN2.ADC to 1, enable the GPIO clock corresponding to the ADC channel, the GPIO clock corresponding to the external reference voltage pin, and the ADC working clock;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set the GPIO pin corresponding to the external reference voltage to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

*Caution:*

*If the ADC reference voltage does not select the external reference voltage pin, skip this step.*

Step 4: Set ADC_CR0.EN to 1 to enable ADC module;

Step 5: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 6: Set ADC_CR0.MODE to 2, select single channel continuous conversion mode;

Step 7: Configure ADC_CR0.REF and select the reference voltage of ADC;

Step 8: Configure ADC_CR0. SAM and ADC_CR0. CLK, set ADC sampling speed and clock selection;

Step 9: Configure ADC_CR1.CHMUX and select the channel to be converted;

Step 10: Set ADC_START.START to 1 to start ADC conversion;

Step 11: Query and wait for ADC_ISR.EOC to become 1, and continuously read the ADC_RESULT0 register to obtain the ADC conversion result;

Step 12: Set ADC_START.START to 0 to stop ADC conversion;

Step 13: If you need to convert other channels, repeat steps 9 to 13;

Step 14: Set ADC_CR0.EN to 0 to turn off the ADC module.

### 20.5.4 Sequence continuous conversion mode (MODE=3)

Sequence continuous conversion mode is similar to single channel continuous conversion mode, except that the sequence continuous conversion mode can alternately convert up to four sequence channels instead of a single channel, and each sequence SQRy can select one of 16 conversion channels, which is configured by the SQRy bit field of the sequence configuration register ADC_SQR. The sequence configuration to be converted is determined by the ENS bit field of the ADC sequence configuration register ADC_SQR, as shown in the following table:

Table 20-6 Sequence configuration to be converted

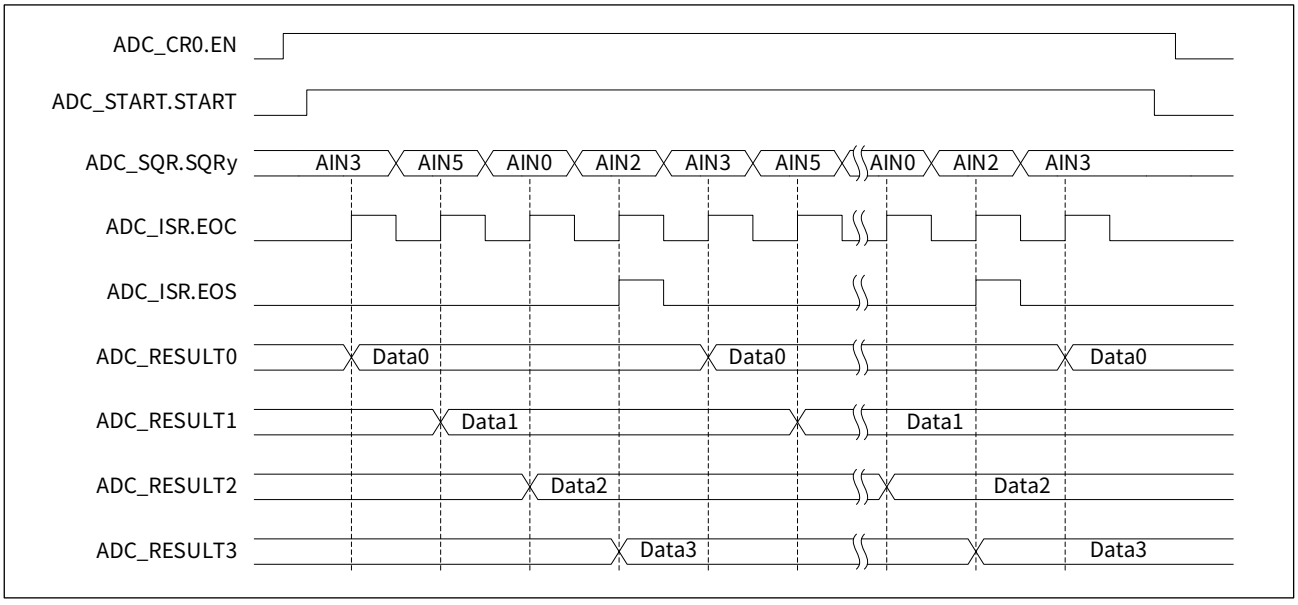
| ADC_SQR.ENS | The sequence configuration to be converted |
|---|---|
| 00 | Only SQR0 is converted |
| 01 | Convert SQR0, SQR1 |
| 10 | Convert SQR0, SQR1, SQR2 |
| 11 | Convert SQR0, SQR1, SQR2, SQR3 |

In this mode, whether the ADC is started through the software START bit or an external trigger, once the ADC is started, it will continue to convert the selected conversion sequence until the ADC_START.START bit is cleared to 0, and the conversion will not stop.

After each ADC conversion is completed, the ADC_ISR.EOC flag is automatically set to 1, and the conversion result is stored in the conversion result registers ADC_RESULT0~ADC_RESULT3 with the same sequence number as the sequence SQR0~SQR3. When all conversions of the selected conversion sequence are completed, the sequence conversion completion flag ADC_ISR.EOS is set to 1. The user should read the conversion result in time to avoid overflow of the conversion result. User clears ADC_START.START bit to 0 to stop conversion.

The timing of sequential continuous conversion mode is shown in the following figure:

Figure 20-6 Sequence continuous conversion timing diagram



The ADC sequence continuous conversion is started by the START bit, and the reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBEN2.ADC to 1, enable the GPIO clock corresponding to the ADC channel, the GPIO clock corresponding to the external reference voltage pin, and the ADC working clock;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set the GPIO pin corresponding to the external reference voltage to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

*Caution:*

*If the ADC reference voltage does not select the external reference voltage pin, skip this step.*

Step 4: Set ADC_CR0.EN to 1 to enable ADC module;

Step 5: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 6: Set ADC_CR0.MODE to 3, select sequential continuous conversion mode;

Step 7: Configure ADC_CR0.REF and select the reference voltage of ADC;

Step 8: Configure ADC_CR0. SAM and ADC_CR0. CLK, set ADC sampling speed and clock selection;

Step 9: Configure ADC_SQR.ENS, select the sequence to be converted, as shown in *Figure 20-6*, set ADC_SQR. ENS to 3, and the conversion sequence to be SQR0~SQR3;

Step 10: Configure ADC_SQR.SQR0, select the channel to be converted for the sequence SQR0 to be converted, as shown in *Figure 20-6*, set ADC_SQR.SQR0 to 3, and the channel to be converted for the sequence SQR0 to AIN3;

Step 11: Configure ADC_SQR.SQR1, select the channel to be converted for the sequence SQR1 to be converted, as shown in *Figure 20-6*, set ADC_SQR.SQR1 to 5, and the channel to be converted for the sequence SQR1 to AIN5;

Step 12: Configure ADC_SQR.SQR2, select the channel to be converted of the sequence SQR2 to be converted, as shown in *Figure 20-6*, set ADC_SQR.SQR2 to 0, and the channel to be converted of the sequence SQR2 to AIN0;

Step 13: Configure ADC_SQR.SQR3, select the channel to be converted of the sequence SQR3 to be converted, as shown in *Figure 20-6*, set ADC_SQR.SQR3 to 2, and the channel to be converted of the sequence SQR3 to AIN2;

Step 14: Set ADC_ICR to 0, clear ADC interrupt flag;

Step 15: Set ADC_START.START to 1 to start ADC conversion;

Step 16: Query and wait for ADC_ISR.EOS to become 1, and read the ADC_RESULT0~ADC_RESULT3 registers in sequence to obtain the ADC conversion results of each channel. When ADC_ISR.EOS is 1, it means that a sequence conversion of 4 channels is completed;

Step 17: Set ADC_START.START to 0 to stop ADC conversion;

Step 18: If you need to convert other channels, repeat steps 9 to 18;

Step 19: Set ADC_CR0.EN to 0 to turn off the ADC module.

### 20.5.5 Sequence scan conversion mode (MODE=4)

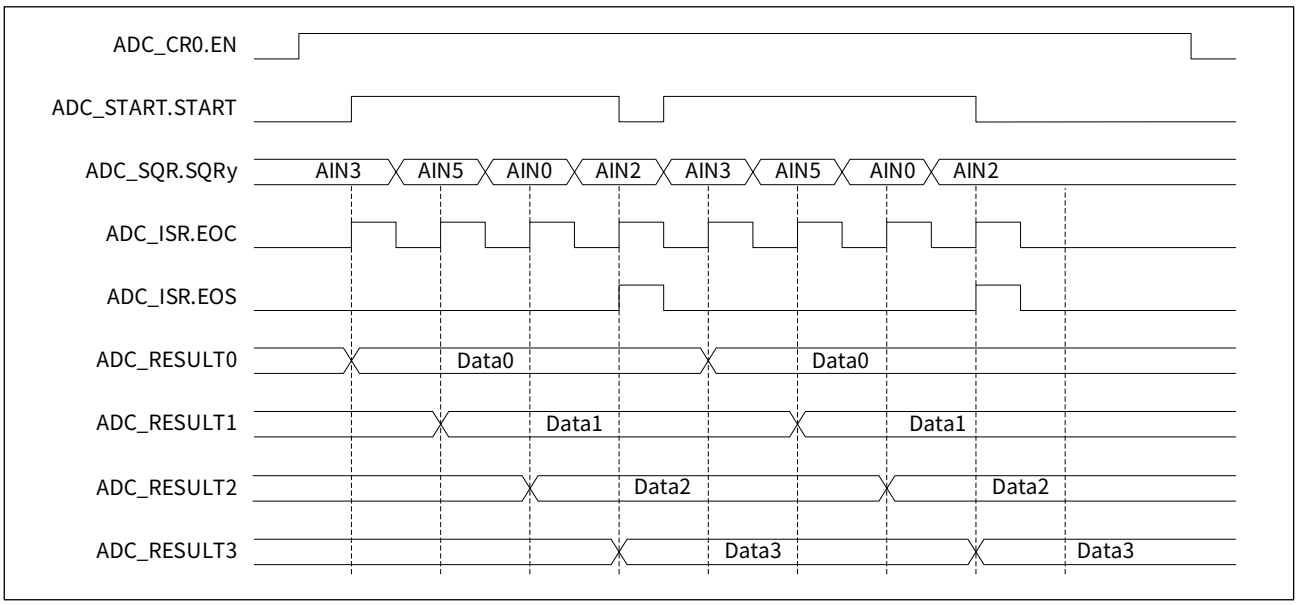
The difference between the sequence scan conversion mode and the sequence continuous conversion is that the sequence scan conversion mode only completes the conversion of the selected sequence once.

In this mode, whether the ADC is started through the software START bit or an external trigger, starting the ADC once will convert all the selected conversion sequences once.

After each ADC conversion is completed, the ADC_ISR.EOC flag is automatically set to 1, and the conversion result is stored in the conversion result registers ADC_RESULT0~ADC_RESULT3 with the same serial number as the sequence SQR0~SQR3. When all conversions of the selected conversion sequence are completed, the ADC_ISR.EOS flag becomes 1, the ADC_START.START bit is automatically cleared to 0, and the ADC stops converting.

The timing of sequence scan conversion mode is shown in the following figure:

Figure 20-7 Sequence scan conversion timing diagram



The ADC sequence scan conversion is started by the START bit, and the reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBEN2.ADC to 1, enable the GPIO clock corresponding to the ADC channel, the GPIO clock corresponding to the external reference voltage pin, and the ADC working clock;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set the GPIO pin corresponding to the external reference voltage to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

*Caution:*

*If the ADC reference voltage does not select the external reference voltage pin, skip this step.*

Step 4: Set ADC_CR0.EN to 1 to enable ADC module;

Step 5: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 6: Set ADC_CR0.MODE to 4, select sequence scan conversion mode;

Step 7: Configure ADC_CR0.REF and select the reference voltage of ADC;

Step 8: Configure ADC_CR0. SAM and ADC_CR0. CLK, set ADC sampling speed and clock selection;

Step 9: Configure ADC_SQR.ENS, select the sequence to be converted, as shown in *Figure 20-7*, set ADC_SQR. ENS to 3, and the conversion sequence to be SQR0~SQR3;

Step 10: Configure ADC_SQR.SQR0, select the channel to be converted for the sequence SQR0 to be converted, as shown in *Figure 20-7*, set ADC_SQR.SQR0 to 3, and the channel to be converted for the sequence SQR0 to AIN3;

Step 11: Configure ADC_SQR.SQR1, select the channel to be converted for the sequence SQR1 to be converted, as shown in *Figure 20-7*, set ADC_SQR.SQR1 to 5, and the channel to be converted for the sequence SQR1 to AIN5;

Step 12: Configure ADC_SQR.SQR2, select the channel to be converted of the sequence SQR2 to be converted, as shown in *Figure 20-7*, set ADC_SQR.SQR2 to 0, and the channel to be converted of the sequence SQR2 to AIN0;

Step 13: Configure ADC_SQR.SQR3, select the channel to be converted for the sequence SQR3 to be converted, as shown in *Figure 20-7*, set ADC_SQR.SQR3 to 2, and the channel to be converted for the sequence SQR3 to AIN2;

Step 14: Set ADC_ICR to 0, clear ADC interrupt flag;

Step 15: Set ADC_START.START to 1 to start ADC conversion;

Step 16: Wait for ADC_ISR.EOS to become 1, and read the ADC_RESULT0~ADC_RESULT3 registers in sequence to obtain the ADC conversion result of the corresponding channel. When ADC_ISR.EOS becomes 1, it means that a sequence conversion of 4 channels is completed, the ADC_START.START bit is automatically cleared to 0, and the ADC conversion stops;

Step 17: If you need to convert other channels, repeat steps 9 to 17;

Step 18: Set ADC_CR0.EN to 0 to turn off the ADC module.

### 20.5.6 Sequence multiple conversion mode (MODE=5)

The difference between the sequence multiple conversion mode and the sequence scan conversion mode is that the sequence scan conversion mode only completes the conversion of the selected sequence once, while the sequence multiple conversion mode is continuously converted multiple times. The sequence conversion times are determined by the ADC_CR2.CNT bit field. value, the default conversion times is 1.

In multiple conversion mode, the accumulation conversion function must be enabled, that is, set ADC_CR2.ACCRST to 1 to clear the ADC_RESULTACC register, and set ADC_CR2.ACCEN to 1 to enable automatic accumulation of ADC conversion results.

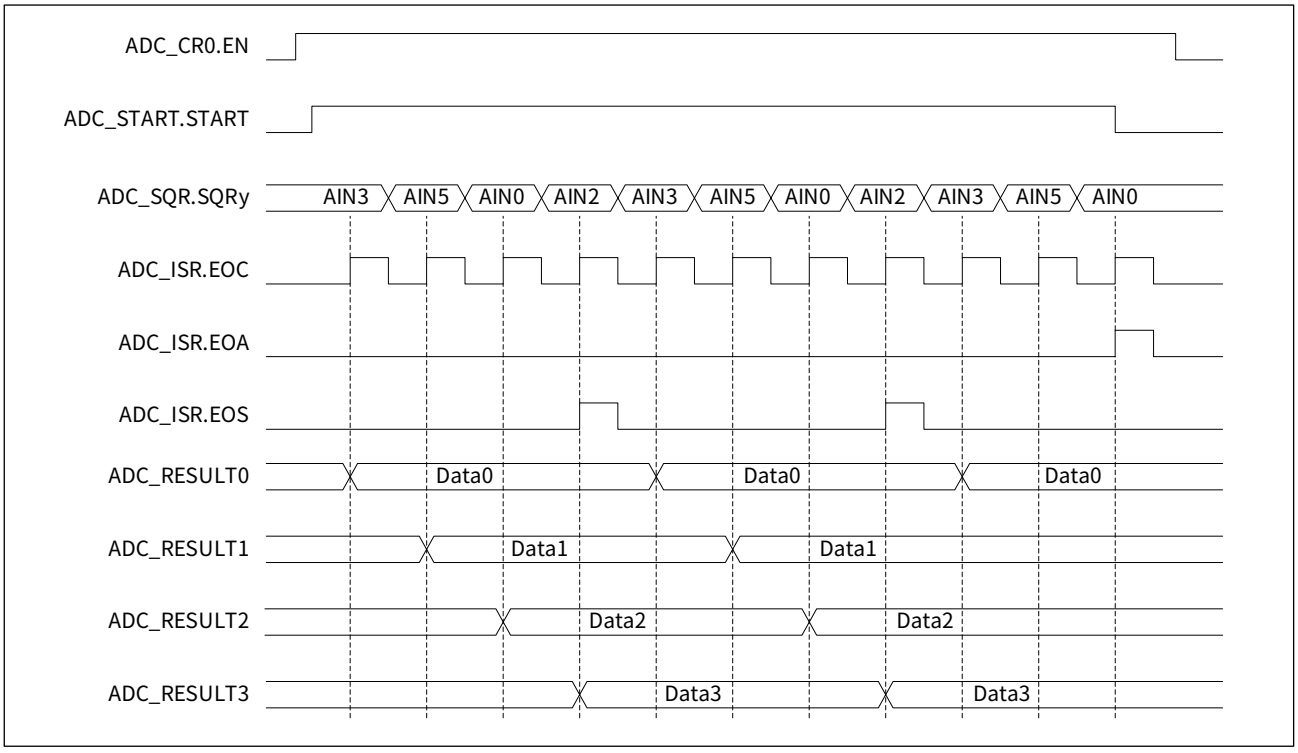
In this mode, whether the ADC is started by the software START bit or an external trigger, once the ADC is started, the selected conversion sequence will continue to be converted in turn, and the conversion will not stop until the sequence conversion times of the ADC_CR2.CNT bit value is reached.

After each ADC conversion is completed, the ADC_ISR.EOC flag is automatically set to 1, and the conversion result is stored in the conversion result registers ADC_RESULT0~ADC_RESULT3 with the same serial number as the sequence SQR0~SQR3. At the same time, the conversion result is automatically accumulated, and the accumulated value is stored in the ADC_RESULTACC register. When all the conversions of the selected conversion sequence are completed, the ADC_ISR.EOS flag becomes 1. If the sequence conversion times set by the ADC_CR2.CNT bit field value is not reached, the conversion will continue. The user should read the conversion result in time to avoid overflow of the conversion result.

When the number of sequence conversions reaches the ADC_CR2.CNT bit value, the ADC_ISR.EOA flag bit is set to 1, the ADC_START.START bit is automatically cleared to 0, and the ADC conversion stops.

The timing of the sequence multiple conversion mode is shown in the following figure, where ADC_CR2.CNT is 10 and the number of conversions is 11.

Figure 20-8 Sequence multiple conversion timing diagram



The ADC sequence multiple conversions are started through the START bit, and the reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBEN2.ADC to 1, enable the GPIO clock corresponding to the ADC channel, the GPIO clock corresponding to the external reference voltage pin, and the ADC working clock;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set the GPIO pin corresponding to the external reference voltage to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

*Caution:*

*If the ADC reference voltage does not select the external reference voltage pin, skip this step.*

Step 4: Set ADC_CR0.EN to 1 to enable ADC module;

Step 5: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 6: Set ADC_CR0.MODE to 5, select sequence multiple conversion mode;

Step 7: Configure ADC_CR0.REF and select the reference voltage of ADC;

Step 8: Configure ADC_CR0. SAM and ADC_CR0. CLK, set ADC sampling speed and clock selection;

Step 9: Configure ADC_SQR.ENS, select the sequence to be converted, as shown in *Figure 20-8*, set ADC_SQR. ENS to 3, and the conversion sequence to be SQR0~SQR3;

Step 10: Configure ADC_SQR.SQR0, select the channel to be converted of the sequence SQR0 to be converted, as shown in *Figure 20-8*, set ADC_SQR.SQR0 to 3, and the channel to be converted of SQR0 to AIN3;

Step 11: Configure ADC_SQR.SQR1, select the channel to be converted of the sequence SQR1 to be converted, as shown in *Figure 20-8*, set ADC_SQR.SQR1 to 5, and the channel to be converted of SQR1 to AIN5;

Step 12: Configure ADC_SQR.SQR2, select the channel to be converted of the sequence SQR2 to be converted, as shown in *Figure 20-8*, set ADC_SQR.SQR2 to 0, and the channel to be converted of SQR2 to AIN0;

Step 13: Configure ADC_SQR.SQR3, select the channel to be converted of the sequence SQR3 to be converted, as shown in *Figure 20-8*, set ADC_SQR.SQR3 to 2, and the channel to be converted of SQR3 to AIN2;

Step 14: Configure ADC_CR2.CNT, as shown in *Figure 20-8*, set ADC_CR2.CNT to 10, then the number of conversions is 11;

Step 15: Set ADC_CR2.ACCRST and ADC_CR2.ACCEN to 1 to enable the conversion result accumulation function;

Step 16: Set ADC_ICR to 0, clear ADC interrupt flag;

Step 17: Set ADC_START.START to 1 to start ADC conversion;

Step 18: Wait for ADC_ISR.EOS to become 1 in a loop, and continuously read the ADC_RESULT0~ADC_RESULT3 registers in turn to obtain the ADC conversion result of the corresponding channel. When ADC_ISR. EOS is 1, it means that a sequence conversion of 4 channels is completed;

Step 19: Wait for ADC_ISR.EOA to become 1 in a loop, and continuously read ADC_RESULT0~ADC_RESULT3 registers in turn to obtain ADC conversion results. When ADC_ISR.EOA is 1, it means that multiple conversions are completed, the ADC_START.START bit is automatically cleared to 0, and the ADC conversion stops;

Step 20: If you need to convert other channels, repeat steps 9 to 19;

Step 21: Set ADC_CR0.EN to 0 to turn off the ADC module.

### 20.5.7 Sequence intermittent conversion mode (MODE=6)

In sequence intermittent conversion mode, each time the ADC is turned on, only the current sequence is converted, not all sequences selected.

In this mode, the ADC can be started via the software START bit, or an external trigger can be selected to start.

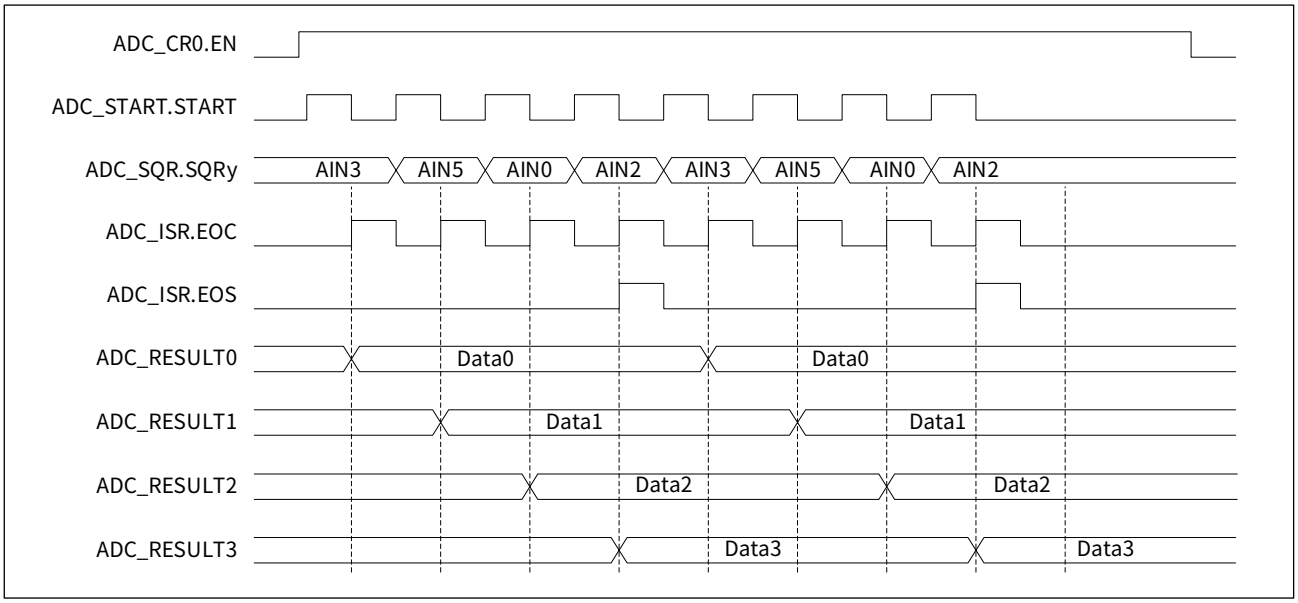
Each time the ADC is started, an ADC conversion is performed for the specified channel in the current ADC conversion sequence SQRy (y=0~3). The ADC conversion is completed, the ADC_ISR.EOC flag is automatically set to 1, and the conversion result is stored in the corresponding ADC_RESULTy (y=0~3). At the same time, the ADC_START.START bit is automatically cleared to 0, and the ADC conversion is completed. Waiting for the ADC conversion to start again.

When the ADC is restarted next time, the channel to be converted is automatically switched to the designated channel of the next conversion sequence SQR(y+1), and the ADC conversion is performed again. If the currently completed conversion sequence is SQR3, the next sequence to be converted is automatically reset to SQR0.

The ADC_ISR.EOS flag becomes 1 when all channels of the conversion sequence SQRy are converted.

The timing of sequence intermittent conversion mode is shown in the following figure:

Figure 20-9 Sequence intermittent conversion timing diagram




The ADC sequence intermittent conversion is started by the START bit, and the reference operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBEN2.ADC to 1, enable the GPIO clock corresponding to the ADC channel, the GPIO clock corresponding to the external reference voltage pin, and the ADC working clock;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set the GPIO pin corresponding to the external reference voltage to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

*Caution:*

*If the ADC reference voltage does not select the external reference voltage pin, skip this step.*

Step 4: Set ADC_CR0.EN to 1 to enable ADC module;

Step 5: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 6: Set ADC_CR0.MODE to 6, and select sequence intermittent conversion mode;

Step 7: Configure ADC_CR0.REF and select the reference voltage of ADC;

Step 8: Configure ADC_CR0. SAM and ADC_CR0. CLK, set ADC sampling speed and clock selection;

Step 9: Configure ADC_SQR.ENS, select the sequence to be converted, as shown in *Figure 20-9*, set ADC_SQR. ENS to 3, and the conversion sequence to be SQR0~SQR3;

Step 10: Configure ADC_SQR.SQR0, select the channel to be converted of the sequence SQR0 to be converted, as shown in *Figure 20-9*, set ADC_SQR.SQR0 to 3, and the channel to be converted of the sequence SQR0 to AIN3;

Step 11: Configure ADC_SQR.SQR1, select the channel to be converted of the sequence SQR1 to be converted, as shown in *Figure 20-9*, set ADC_SQR.SQR1 to 5, and the channel to be converted of the sequence SQR1 to AIN5;

Step 12: Configure ADC_SQR.SQR2, select the channel to be converted of the sequence SQR2 to be converted, as shown in *Figure 20-9*, set ADC_SQR.SQR2 to 0, and the channel to be converted of the sequence SQR2 to AIN0;

Step 13: Configure ADC_SQR.SQR3, select the channel to be converted of the sequence SQR3 to be converted, as shown in *Figure 20-9*, set ADC_SQR.SQR3 to 2, and the channel to be converted of the sequence SQR3 to AIN2;

Step 14: Set ADC_ICR to 0, clear ADC interrupt flag;

Step 15: Set ADC_START.START to 1 to start ADC conversion;

Step 16: Wait for ADC_ISR.EOC to become 1, and choose to read the ADC_RESULT0 register to obtain the ADC conversion result of the channel specified by the conversion sequence SQR0;

Step 17: Set ADC_START.START to 1 to start ADC conversion again;

Step 18: Wait for ADC_ISR.EOC to become 1, and choose to read the ADC_RESULT1 register to obtain the ADC conversion result of the channel specified by the conversion sequence SQR1;

Step 19: Set ADC_START.START to 1 to start ADC conversion again;

Step 20: Wait for ADC_ISR.EOC to become 1, and choose to read the ADC_RESULT2 register to obtain the ADC conversion result of the channel specified by the conversion sequence SQR2;

Step 21: Set ADC_START.START to 1 and start ADC conversion again;

Step 22: Wait for ADC_ISR.EOC to become 1, and choose to read the ADC_RESULT3 register to get the ADC conversion result for the channel specified by conversion sequence SQR3. When ADC_ISR.EOS is 1, it means that a sequence conversion of 4 channels is completed;

Step 23: Repeat steps 15 to 22 to continue ADC conversion; if you need to convert other channels, repeat steps 8 to 22;

Step 24: Set ADC_CR0.EN to 0 to turn off the ADC module.

## 20.6 Accumulation conversion function

The accumulation conversion function can be performed in any working mode of the ADC. And in the multiple conversion mode, the cumulative conversion function must be enabled.

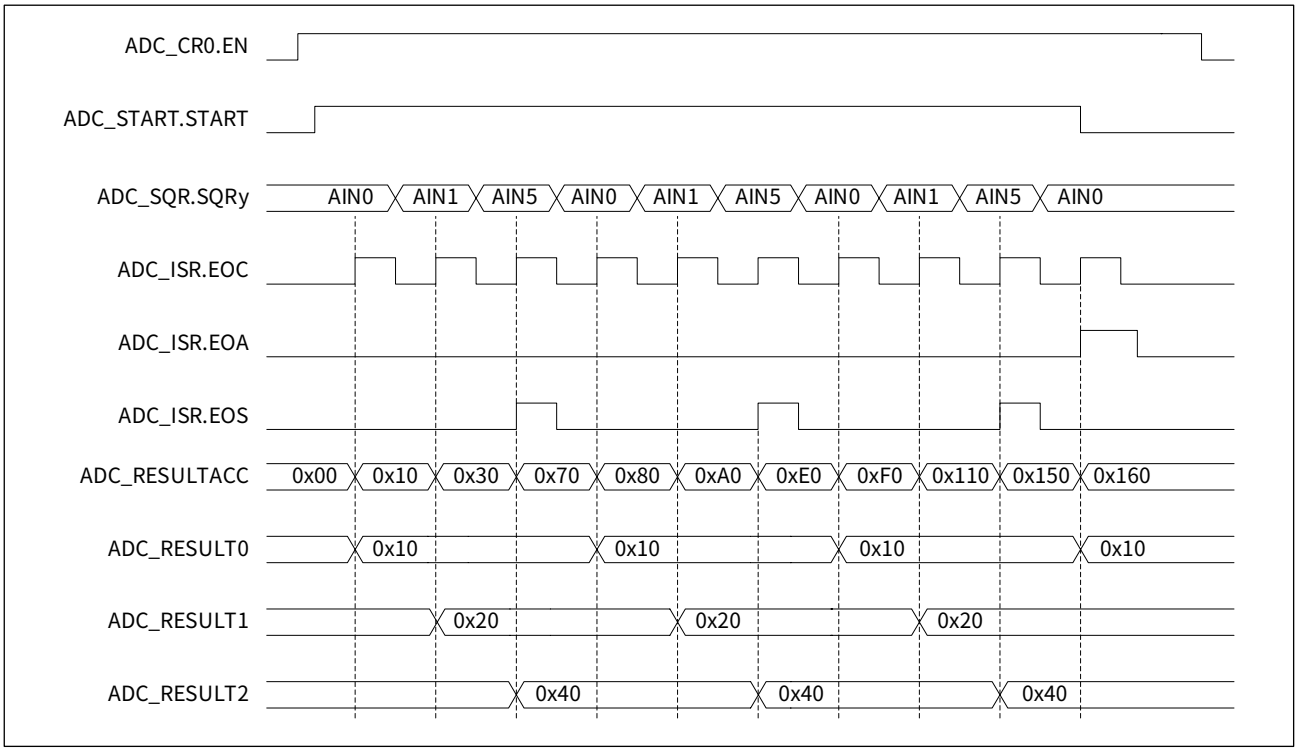
Set ADC_CR2.ACCEN to 1 to enable the automatic accumulation function of ADC conversion results. Each time the ADC completes a channel conversion, it automatically accumulates the conversion results, and the accumulated value is stored in the ADC_RESULTACC register.

Before using the accumulation function, the ADC_RESULTACC register must be cleared first. Setting ADC_CR2. ACCRST to 1 can clear the ADC_RESULTACC register to 0.

*Figure 20-10 Accumulation conversion timing diagram* demonstrates the process of 10 consecutive conversion accumulations for the three channels of AIN0, AIN1, and AIN5. It is assumed that the ADC conversion results of AIN0, AIN1, and AIN5 are 0x010, 0x020, and 0x040 in sequence.

Set ADC_START.START to 1 to start the ADC, and the state machine inside the ADC will convert AIN0, AIN1, and AIN5 in turn, and the ADC conversion will not stop until the total number of conversions reaches 10. Each time a conversion completes, the ADC_RESULTACC register automatically accumulates the conversion result.

Figure 20-10 Accumulation conversion timing diagram




The ADC sequence is started for multiple conversions through the START bit, and the conversion results are accumulated. The operation flow is as follows:

Step 1: Set SYSCTRL_AHBEN.GPIOx to 1, SYSCTRL_APBEN2.ADC to 1, enable the GPIO clock corresponding to the ADC channel, the GPIO clock corresponding to the external reference voltage pin, and the ADC working clock;

Step 2: Set the GPIO pin corresponding to the ADC channel to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

Step 3: Set the GPIO pin corresponding to the external reference voltage to the analog function. For the specific register configuration, please refer to chapter *8 General-purpose input/output (GPIO)*;

*Caution:*

*If the ADC reference voltage does not select the external reference voltage pin, skip this step.*

Step 4: Set ADC_CR0.EN to 1 to enable ADC module;

Step 5: Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

Step 6: Set ADC_CR0.MODE to 5, select sequence multiple conversion mode;

Step 7: Configure ADC_CR0.REF and select the reference voltage of ADC;

Step 8: Configure ADC_CR0. SAM and ADC_CR0. CLK, set ADC sampling speed and clock selection;

Step 9: Configure ADC_CR2.CNT, as shown in *Figure 20-10*, set ADC_CR2.CNT to 9, then the number of conversions is 10;

Step 10: Set ADC_CR2.ACCEN to 1 to enable the automatic accumulation control of ADC conversion results;

Step 11: Set ADC_CR2.ACCRST to 1, and the ADC conversion result accumulation register ADC_RESULTACC is cleared to 0;

Step 12: Configure ADC_SQR.ENS, select the sequence to be converted, as shown in *Figure 20-10*, set ADC_SQR.ENS to 2, and the conversion sequence to be SQR0~SQR2;

Step 13: Configure ADC_SQR.SQR0, select the channel to be converted for the sequence SQR0 to be converted, as shown in *Figure 20-10*, set ADC_SQR.SQR0 to 0, and the channel to be converted for the sequence SQR0 to AIN0;

Step 14: Configure ADC_SQR.SQR1, select the channel to be converted for the sequence SQR1 to be converted, as shown in *Figure 20-10*, set ADC_SQR.SQR1 to 1, and the channel to be converted for the sequence SQR1 to AIN1;

Step 15: Configure ADC_SQR.SQR2, select the channel to be converted of the sequence SQR2 to be converted, as shown in *Figure 20-10*, set ADC_SQR.SQR2 to 5, and the channel to be converted of the sequence SQR2 to AIN5;

Step 16: Set ADC_IER.EOA to 1 to enable multiple conversion completion interrupts;

Step 17: Set ADC_ICR.EOA to 0, clear ADC_ISR.EOA interrupt flag;

Step 18: Set ADC_ICR to 0, clear ADC interrupt flag;

Step 19: Set ADC_START.START to 1 to start ADC conversion;

Step 20: Wait for ADC_ISR.EOC to become 1 in a loop, and continuously read the ADC_RESULTACC register to obtain the accumulated value of ADC conversion result. When ADC_ISR.EOS is 1, it means that a sequence conversion of 3 channels is completed;

Step 21: Wait for ADC_ISR.EOA to become 1, while reading the ADC_RESULTACC register continuously. When ADC_ISR.EOA is 1, it means that multiple conversions are completed, the ADC_START.START bit is automatically cleared to 0, and the ADC conversion stops. In this example, when AIN0 completes the 4th ADC conversion, the ADC conversion stops;

Step 22: Set ADC_CR0.EN to 0 to turn off the ADC module.

## 20.7 Auto-off mode

The user can enable the ADC auto-off function by setting ADC_START.AUTOSTOP to 1. When the specified ADC conversion is completed, the ADC_CR0.EN bit is automatically cleared to 0, and the ADC function is disabled. If the ADC conversion needs to continue, ADC_CR0.EN must be reset to 1 to enable the ADC module.

If the single channel single conversion mode is used, when the conversion is completed, the ADC enable is automatically turned off.

If the sequence scan conversion mode is used, when all sequence conversions are completed, the ADC enable is automatically turned off.

If the multiple conversion mode (single channel multiple conversion mode or sequence multiple conversion mode) is used, when the set number of conversions is reached, the ADC conversion stops and the ADC enable is automatically turned off.

If the continuous conversion mode (single channel continuous conversion mode or sequence continuous conversion mode) is used, the ADC enable will not be automatically turned off after the conversion is completed. Setting ADC_START.START to 0 will turn off continuous conversion and automatically turn off ADC enable.

When using sequence intermittent conversion mode, when all selected channels are converted, the ADC enable is automatically turned off.

## 20.8 External trigger source

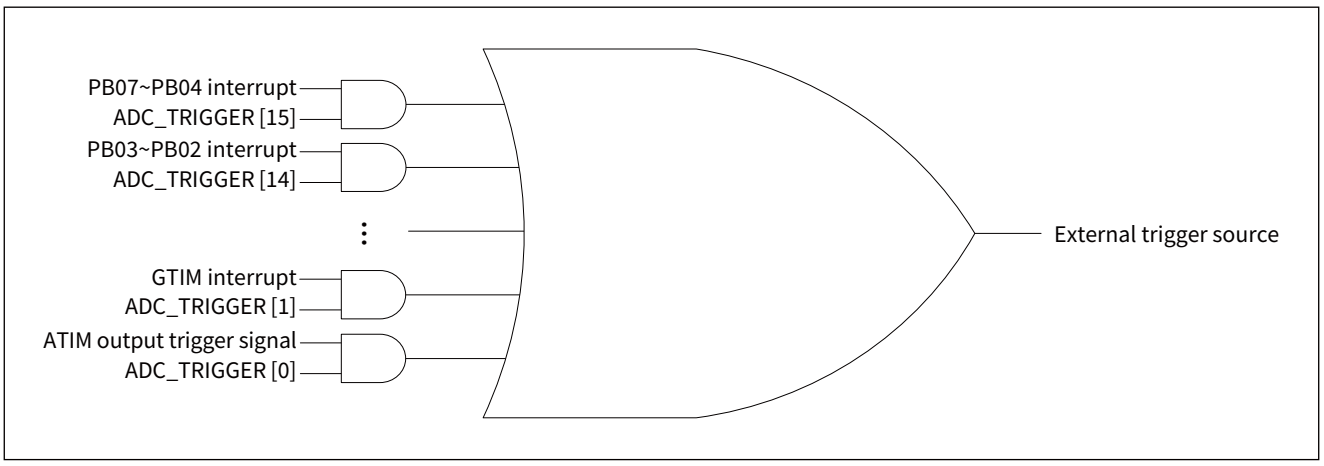
ADC conversion can be started either by software (that is, setting ADC_START.START to 1) or by external trigger. The trigger source is selected by the external trigger register ADC_TRIGGER. There are 16 ways to trigger the ADC, as shown in the following table:

Table 20-7 ADC conversion external trigger source

| ADC_TRIGGER bit field | Bit field name | Functional description |
|---|---|---|
| 15 | PB74 | PB07~PB04 interrupt triggers ADC to start |
| 14 | PB32 | PB03~PB02 interrupt triggers ADC to start |
| 13 | I2C | I2C interrupt triggers ADC to start |
| 12 | PB10 | PB01~PB00 interrupt triggers ADC to start |
| 11 | SPI | SPI interrupt triggers ADC to start |
| 10 | PA76 | PA07~PB06 interrupt triggers ADC to start |
| 9 | UART2 | UART2 interrupt triggers ADC to start |
| 8 | UART1 | UART1 interrupt triggers ADC to start |
| 7 | BTIM3 | BTIM3 interrupt triggers ADC to start |
| 6 | BTIM2 | BTIM2 interrupt triggers ADC to start |
| 5 | BTIM1 | BTIM1 interrupt triggers ADC to start |
| 4 | PA54 | PA05~PA04 interrupt triggers ADC to start |
| 3 | PA32 | PA03~PA02 interrupt triggers ADC to start |
| 2 | PA10 | PA01~PA00 interrupt triggers ADC to start |
| 1 | GTIM | GTIM interrupt triggers ADC to start |
| 0 | ATIM | The trigger signal output by ATIM triggers the ADC to start |

The schematic diagram of the external trigger source for ADC conversion is shown in the following figure:

Figure 20-11 Schematic diagram of external trigger source for ADC conversion



## 20.9 Analog watchdog

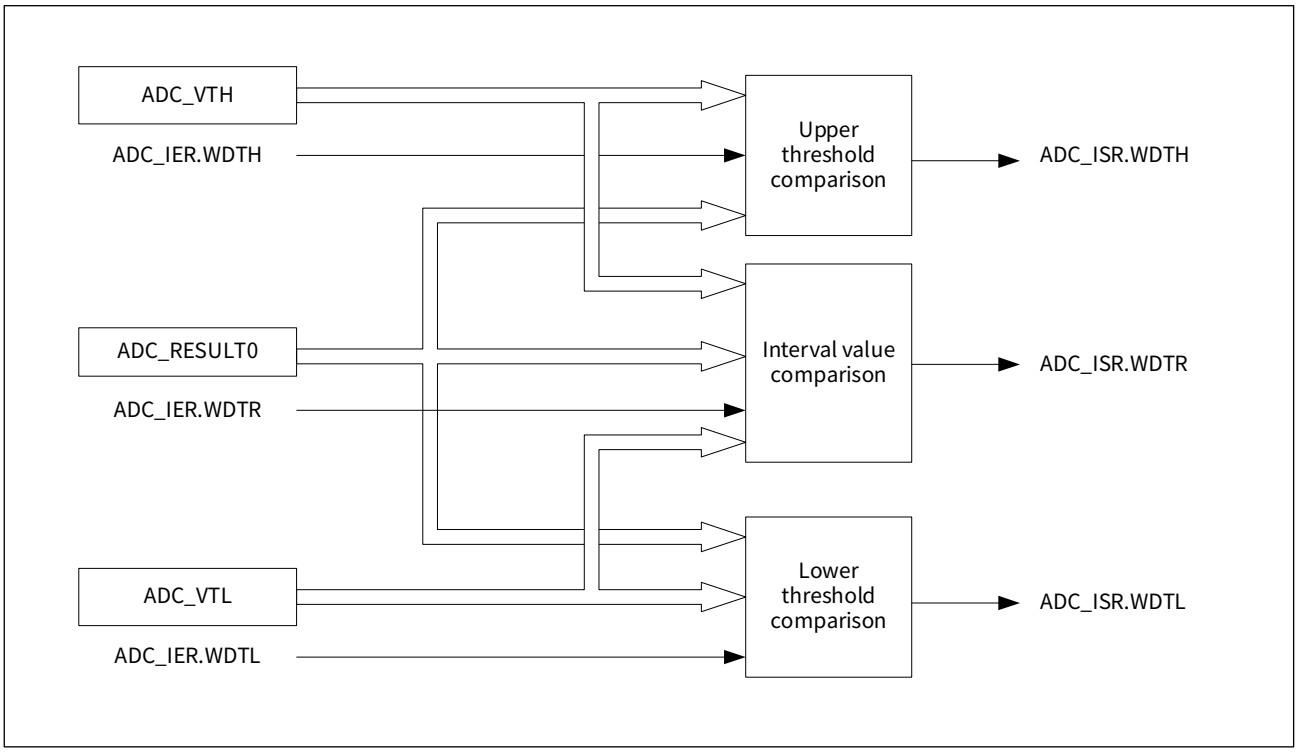
The analog watchdog function supports the comparison of the ADC conversion result with the threshold set by the user, supports the comparison of upper threshold, lower threshold, and interval value, and sets the comparison threshold through the threshold registers ADC_VTH and ADC_VTL.

Set the WDTALL bit field of the control register ADC_CR1 to 1 to enable the analog watchdog, and enable the analog watchdog function of the specified channel through the WDTCH bit field of the ADC_CR1 register.

The analog watchdog function is often used for automatic monitoring of analog quantities. If the corresponding bit fields (WDTR, WDTH, WDTL) of the interrupt enable register ADC_IER are set, an interrupt request will be generated when the ADC conversion result meets the user's expectation.

A schematic diagram of the threshold comparison of the analog watchdog is shown in the following figure:

Figure 20-12 Schematic diagram of ADC threshold comparison




Upper threshold comparison: When the conversion result is within the interval [ADC_VTH, 4095], the ADC_ISR.WDTH flag bit is set to 1.

Lower threshold comparison: When the conversion result is in the [0, ADC_VTL) interval, the ADC_ISR.WDTL flag is set to 1.

Interval value comparison: When the conversion result is in the interval [ADC_VTL, ADC_VTH), the ADC_ISR.WDTR flag is set to 1.

## 20.10 Temperature sensor

CW32F003 has a built-in temperature sensor module. The output voltage of the sensor changes with temperature. Set the sampling channel of the ADC module as an internal temperature sensor, and the current ambient temperature can be calculated through the ADC measurement results.

The temperature sensor is off by default. By setting the TSEN bit field of the control register ADC_CR0 to 1, the temperature sensor is enabled.

The formula for calculating the ambient temperature is as follows:

$$\text{Ambient temperature} = T_0 \times 0.5 + 0.0924 \times V_{ref} \times (\text{AdcValue} - \text{Trim})$$

Where:

$V_{ref}$ is the reference voltage of the current ADC module, which is 1.5V or 2.5V.

$T_0$ is the 8-bit initial calibration temperature value, which is recorded in the FLASH memory of the chip. Its address is 0x0010 07C5, and the unit is 0.5 degrees Celsius. The read value needs to be divided by 2 to be the actual temperature.

AdcValue is the ADC conversion result of the ADC module measuring the output voltage of the temperature sensor, and the value range is 0 ~ 4095.

Trim is a 16-bit calibration value, which needs to be read from the FLASH memory of the chip during calculation, and its storage address is shown in the following table:

Table 20-8 ADC calibration value address

| ADC reference voltage | Calibration value storage address | Calibration value accuracy |
|---|---|---|
| Internal 1.5V | 0x0010 07C6 - 0x0010 07C7 | ±3°C |
| Internal 2.5V | 0x0010 07C8 - 0x0010 07C9 | ±3°C |

A calculation example is as follows:

Condition 1: $V_{ref}$=1.5, AdcValue = 0x8CB, Trim = 0x883, $T_0$ = 0x32

Temperature 1: 0x32×0.5 + 0.0924×1.5×( 0x8CB – 0x883 ) = 35°C

Condition 2: $V_{ref}$ = 2.5, AdcValue = 0x599, Trim = 0x516, $T_0$ = 0x32

Temperature 2: 0x32×0.5 + 0.0924×2.5×( 0x599 – 0x516 ) = 55.3°C

The reference operation flow for measuring the ambient temperature through the ADC is as follows:

**Step 1:** Set SYSCTRL_APBEN2.ADC to 1, enable ADC configuration clock and working clock;

**Step 2:** Set ADC_CR0.EN to 1 to enable the ADC module;

**Step 3:** Query and wait for the ADC_ISR.READY bit to become 1, and wait for the ADC module to start up;

**Step 4:** Set ADC_CR0.MODE to 0, select single channel single conversion mode;

**Step 5:** Configure ADC_CR0.REF, select ADC reference voltage as internal 1.5V or internal 2.5V;

**Step 6:** Configure ADC_CR0. SAM and ADC_CR0. CLK, and set the conversion speed of ADC;

**Step 7:** Set ADC_CR0.TSEN to 1 to enable the temperature sensor;

**Step 8:** Set ADC_CR1.CHMUX to 0x0E, and select the channel to be converted as the voltage output of the temperature sensor;

**Step 9:** Set ADC_CR0.BUF to 1 to enable the built-in follower;

**Step 10:** Set ADC_ICR.EOC to 0, clear ADC_ISR.EOC flag;

**Step 11:** Set ADC_START.START to 1 to start ADC conversion;

Step 12: Wait for ADC_ISR.EOC to become 1, and read the ADC_RESULT0 register to get the ADC conversion result;

Step 13: Set ADC_CR0.EN to 0 and turn off the ADC module;

Step 14: Read $T_0$ and Trim, and calculate the current ambient temperature according to the formula.

## 20.11 ADC interrupts

ADC interrupt request, as shown in the following table:

Table 20-9 ADC interrupt sources and interrupt flags

| Interrupt source | Interrupt flag bit | Interrupt enable | Flag clear |
|---|---|---|---|
| Conversion result overflow | ADC_ISR.OVW | ADC_IER.OVW is set to 1 | ADC_ICR.OVW is cleared to 0 |
| Conversion result >= ADC_VTL, and< ADC_VTH | ADC_ISR.WDTR | ADC_IER.WDTR is set to 1 | ADC_ICR.WDTR is cleared to 0 |
| Conversion result >= ADC_VTH | ADC_ISR.WDTH | ADC_IER.WDTH is set to 1 | ADC_ICR.WDTH is cleared to 0 |
| Conversion result < ADC_VTL | ADC_ISR.WDTL | ADC_IER.WDTL is set to 1 | ADC_ICR.WDTL is cleared to 0 |
| ADC completes multiple conversions | ADC_ISR.EOA | ADC_IER.EOA is set to 1 | ADC_ICR.EOA is cleared to 0 |
| ADC sequence conversion completed | ADC_ISR.EOS | ADC_IER.EOS is set to 1 | ADC_ICR.EOS is cleared to 0 |
| ADC conversion completed | ADC_ISR.EOC | ADC_IER.EOC is set to 1 | ADC_ICR.EOC is cleared to 0 |

## 20.12 List of registers

ADC base address: ADC_BASE = 0x4001 2400

Table 20-10 List of ADC registers

| Register name | Register address | Register description |
|---|---|---|
| ADC_CR0 | ADC_BASE + 0x00 | ADC control register 0 |
| ADC_CR1 | ADC_BASE + 0x04 | ADC control register 1 |
| ADC_START | ADC_BASE + 0x08 | ADC startup register |
| ADC_SQR | ADC_BASE + 0x0C | ADC sequence configuration register |
| ADC_CR2 | ADC_BASE + 0x10 | ADC control register2 |
| ADC_VTH | ADC_BASE + 0x14 | ADC high threshold register |
| ADC_VTL | ADC_BASE + 0x18 | ADC low threshold register |
| ADC_TRIGGER | ADC_BASE + 0x1C | ADC external trigger register |
| ADC_RESULT0 | ADC_BASE + 0x20 | ADC conversion result 0 register |
| ADC_RESULT1 | ADC_BASE + 0x24 | ADC conversion result 1 register |
| ADC_RESULT2 | ADC_BASE + 0x28 | ADC conversion result 2 register |
| ADC_RESULT3 | ADC_BASE + 0x2C | ADC conversion result 3 register |
| ADC_RESULTACC | ADC_BASE + 0x30 | ADC conversion result accumulation value register |
| ADC_IER | ADC_BASE + 0x34 | ADC interrupt enable register |
| ADC_ICR | ADC_BASE + 0x38 | ADC interrupt flag clear register |
| ADC_ISR | ADC_BASE + 0x3C | ADC interrupt flag register |

## 20.13 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 20.13.1 ADC_CR0 control register 0

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:14 | BIAS | RW | Please keep the default value |
| 13 | BUF | RW | Built-in follower enable control<br>0: Turn off the follower, the external input signal is directly connected to the ADC<br>1: Turn on the follower, the external input signal is connected to the ADC after passing through the follower<br>Caution 1: When this function is enabled, the maximum conversion speed is 200K SPS<br>Caution 2: The following conditions must enable this function:<br>The output impedance of the signal to be converted is relatively high; the signal to be converted is VDD/3, the output voltage of the built-in temperature sensor, and the output voltage of the internal reference 1.2V. |
| 12:11 | SAM | RW | ADC sampling cycle selection<br>00: 5 ADCCLK clock cycles<br>01: 6 ADCCLK clock cycles<br>10: 8 ADCCLK clock cycles<br>11: 10 ADCCLK clock cycles |
| 10:8 | CLK | RW | ADC working clock configuration<br>000: PCLK 100: PCLK / 16<br>001: PCLK / 2 101: PCLK / 32<br>010: PCLK / 4 110: PCLK / 64<br>011: PCLK / 8 111: PCLK / 128 |
| 7:6 | REF | RW | ADC reference voltage selection<br>00: Built-in 1.5V reference voltage<br>01: Built-in 2.5V reference voltage<br>10: External reference voltage ExRef<br>11: VDD supply voltage |
| 5 | TSEN | RW | Built-in temperature sensor enable control<br>0: Built-in temperature sensor disabled<br>1: Built-in temperature sensor enabled |
| 4 | BGREN | RW | Built-in BGR reference enable control<br>0: Built-in BGR reference disabled<br>1: Built-in BGR reference enabled |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 3:1 | MODE | RW | ADC operating mode configuration<br>000: Single channel single conversion mode<br>001: Single channel multiple conversion mode, see CR2.CNT for the number of conversions<br>010: Single channel continuous conversion mode<br>011: Sequence continuous conversion mode<br>100: Sequence scan conversion mode<br>101: Sequence multiple conversion mode, see CR2.CNT for the number of conversions<br>110: Sequence intermittent conversion mode |
| 0 | EN | RW | ADC enable control<br>0: ADC disabled<br>1: ADC enabled<br>Caution: After enabling ADC, you should wait for ISR.READY to become 1 before performing ADC conversion. |

### 20.13.2 ADC_CR1 control register 1

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:14 | RFU | - | Reserved bits, please keep the default value |
| 13 | WDTALL | RW | All channels analog watchdog enable control<br>0: Analog watchdog disabled<br>1: Analog watchdog enabled |
| 12 | RFU | - | Reserved bits, please keep the default value |
| 11:8 | WDTCH | RW | Single channel analog watchdog enable configuration<br>0000: AIN0 1000: AIN8<br>0001: AIN1 1001: AIN9<br>0010: AIN2 1010: AIN10<br>0011: AIN3 1011: AIN11<br>0100: AIN4 1100: AIN12<br>0101: AIN5 1101: VDD/3<br>0110: AIN6 1110: TS built-in temperature sensor<br>0111: AIN7 1111: 1.2V core voltage reference source |
| 7 | RFU | - | Reserved bits, please keep the default value |
| 6 | ALIGN | RW | ADC conversion result alignment<br>0: Right justified, the conversion result is stored in [11:0]<br>1: Left justified, conversion result is stored in [15:4] |
| 5 | DISCARD | RW | Single channel ADC conversion result saving strategy configuration<br>0: Overwrite old data that has not been read, keep new data<br>1: Discard the newly converted data, keep the unread old data |
| 4 | RFU | - | Reserved bits, please keep the default value |
| 3:0 | CHMUX | RW | Single channel conversion mode to be converted channel configuration<br>0000: AIN0 1000: AIN8<br>0001: AIN1 1001: AIN9<br>0010: AIN2 1010: AIN10<br>0011: AIN3 1011: AIN11<br>0100: AIN4 1100: AIN12<br>0101: AIN5 1101: VDD/3<br>0110: AIN6 1110: TS built-in temperature sensor<br>0111: AIN7 1111: 1.2V core voltage reference source |

### 20.13.3 ADC_CR2 control register 2

Address offset: 0x10 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | ACCRST | R0W1 | The accumulated value register is cleared to 0<br>0: No function<br>1: The conversion result accumulation value register ADC_RESULTACC is cleared to 0 |
| 8 | ACCEN | RW | Conversion result accumulation enabled<br>0: Accumulation function disabled<br>1: Accumulation function enabled |
| 7:0 | CNT | RW | Configuration of the number of conversions in multiple conversion mode<br>The number of conversions is CNT+1 |

### 20.13.4 ADC_SQR sequence configuration register

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:18 | RFU | - | Reserved bits, please keep the default value |
| 17:16 | ENS | RW | Sequence configuration to convert<br>00: Convert SQR0<br>01: Convert SQR0-SQR1<br>10: Convert SQR0-SQR2<br>11: Convert SQR0-SQR3 |
| 15:12 | SQR3 | RW | Sequence 3 channel configuration to be converted<br>See SQR0 for details |
| 11:8 | SQR2 | RW | Sequence 2 channel configuration to be converted<br>See SQR0 for details |
| 7:4 | SQR1 | RW | Sequence 1 channel configuration to be converted<br>See SQR0 for details |
| 3:0 | SQR0 | RW | Sequence 0 Channel configuration to be converted<br>0000: AIN0 1000: AIN8<br>0001: AIN1 1001: AIN9<br>0010: AIN2 1010: AIN10<br>0011: AIN3 1011: AIN11<br>0100: AIN4 1100: AIN12<br>0101: AIN5 1101: VDD/3<br>0110: AIN6 1110: TS built-in temperature sensor<br>0111: AIN7 1111: 1.2V core voltage reference source |

### 20.13.5 ADC_VTH high threshold register

Address offset: 0x14 Reset value: 0x0000 0FFF

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:12 | RFU | - | Reserved bits, please keep the default value |
| 11:0 | VTH | RW | Analog watchdog detects high threshold |

### 20.13.6 ADC_VTL low threshold register

Address offset: 0x18 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:12 | RFU | - | Reserved bits, please keep the default value |
| 11:0 | VTL | RW | Analog watchdog detects low threshold |

### 20.13.7 ADC_TRIGGER external trigger register

Address offset: 0x1C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15 | PB74 | RW | PB07~PB04 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 14 | PB32 | RW | PB03~PB02 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 13 | I2C | RW | I2C interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 12 | PB10 | RW | PB01~PB00 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 11 | SPI | RW | SPI interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 10 | PA76 | RW | PA07~PB06 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 9 | UART2 | RW | UART2 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 8 | UART1 | RW | UART1 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 7 | BTIM3 | RW | BTIM3 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 6 | BTIM2 | RW | BTIM2 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 5 | BTIM1 | RW | BTIM1 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 4 | PA54 | RW | PA05~PA04 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 3 | PA32 | RW | PA03~PA02 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 2 | PA10 | RW | PA01~PA00 interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 1 | GTIM | RW | GTIM interrupt triggers ADC to start<br>0: Disabled<br>1: Enabled |
| 0 | ATIM | RW | The trigger signal output by ATIM triggers the ADC to start<br>0: Disabled<br>1: Enabled |

### 20.13.8 ADC_START start register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | AUTOSTOP | RW | ADC auto disable configuration<br>0: Continue to keep CR0.EN at 1 after the conversion is completed<br>1: Automatically set CR0.EN to 0 after the conversion is completed |
| 0 | START | RW | ADC start conversion control<br>0: Stop ADC conversion<br>1: Start ADC conversion |

### 20.13.9 ADC_IER interrupt enable register

Address offset: 0x34 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | OVW | RW | Conversion result overflow interrupt enable control<br>0: Disabled<br>1: Enabled |
| 5 | WDTR | RW | Analog watchdog interval interrupt enable control<br>0: Disabled<br>1: Enabled |
| 4 | WDTH | RW | Analog watchdog upper threshold interrupt enable control<br>0: Disabled<br>1: Enabled |
| 3 | WDTL | RW | Analog watchdog lower threshold interrupt enable control<br>0: Disabled<br>1: Enabled |
| 2 | EOA | RW | Multiple conversion complete interrupt enable control<br>0: Disabled<br>1: Enabled |
| 1 | EOS | RW | Sequence conversion complete interrupt enable control<br>0: Disabledd<br>1: enable |
| 0 | EOC | RW | Conversion complete interrupt enable control<br>0: Disabled<br>1: Enabled |

### 20.13.10 ADC_ISR interrupt flag register

Address offset: 0x3C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | READY | RO | Analog circuit initialization complete flag<br>0: Initialization not completed<br>1: Initialization has been completed, and ADC conversion can be started |
| 6 | OVW | RO | Conversion result overflow flag<br>0: The event did not occur<br>1: A new conversion is completed before the data in the result register has been read |
| 5 | WDTR | RO | Analog watchdog interval flag<br>0: The conversion result is outside the range of [ADC_VTL, ADC_VTH)<br>1: The conversion result is in the range of [ADC_VTL, ADC_VTH) |
| 4 | WDTH | RO | Analog watchdog upper threshold flag<br>0: The conversion result is outside the range of [ADC_VTH, 4096)<br>1: The conversion result is in the interval [ADC_VTH, 4096) |
| 3 | WDTL | RO | Analog watchdog lower threshold flag<br>0: The conversion result is outside the range [0, ADC_VTL)<br>1: The conversion result is in the interval [0, ADC_VTL) |
| 2 | EOA | RO | Multiple conversion complete flag<br>0: Multiple conversions are not completed<br>1: Multiple conversions completed |
| 1 | EOS | RO | Sequence conversion complete flag<br>0: Sequence conversion not completed<br>1: Sequence conversion completed |
| 0 | EOC | RO | Conversion complete flag<br>0: One ADC conversion is not completed<br>1: One ADC conversion has been completed |

### 20.13.11 ADC_ICR interrupt flag clear register

Address offset: 0x38 Reset value: 0x0000 007F

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:7 | RFU | - | Reserved bits, please keep the default value |
| 6 | OVW | R1W0 | The conversion result overflow flag is cleared to 0 control<br>W0: The corresponding flag in the ISR register cleared<br>W1: No function |
| 5 | WDTR | R1W0 | Analog watchdog interval flag is cleared to 0 control<br>W0: The corresponding flag in the ISR register cleared<br>W1: No function |
| 4 | WDTH | R1W0 | Analog watchdog upper threshold flag is cleared to 0 control<br>W0: The corresponding flag in the ISR register cleared<br>W1: No function |
| 3 | WDTL | R1W0 | Analog watchdog lower threshold flag is cleared to 0 control<br>W0: The corresponding flag in the ISR register cleared<br>W1: No function |
| 2 | EOA | R1W0 | Multiple conversion completion flag is cleared to 0 control<br>W0: The corresponding flag in the ISR register cleared<br>W1: No function |
| 1 | EOS | R1W0 | Sequence conversion complete flag is cleared to 0 control<br>W0: The corresponding flag in the ISR register cleared<br>W1: No function |
| 0 | EOC | R1W0 | Conversion complet flag is cleared to 0 control<br>W0: The corresponding flag in the ISR register cleared<br>W1: No function |

### 20.13.12 ADC_RESULT0 conversion result 0 register

Address offset: 0x20 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | RESULT | RO | ADC conversion result 0 register |

### 20.13.13 ADC_RESULT1 conversion result 1 register

Address offset: 0x24 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | RESULT | RO | ADC conversion result 1 register |

### 20.13.14 ADC_RESULT2 conversion result 2 register

Address offset: 0x28 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | RESULT | RO | ADC conversion result 2 register |

### 20.13.15 ADC_RESULT3 conversion result 3 register

Address offset: 0x2C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:0 | RESULT | RO | ADC conversion result 3 register |

### 20.13.16 ADC_RESULTACC conversion result accumulated value register

Address offset: 0x30 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:24 | RFU | - | Reserved bits, please keep the default value |
| 23:0 | RESULT | RO | ADC conversion result accumulated value register |

# 21 Analog voltage comparator (VC)

## 21.1 Overview

CW32F003 integrates two analog voltage comparators (VC), which are used to compare two analog input voltages, and output the comparison results from the pins. The positive terminal input of the voltage comparator supports up to 8 external analog inputs, and the negative terminal supports not only 8 external analog inputs, but also internal voltage reference, internal resistance voltage divider, internal temperature sensor and other voltage references. The comparison result output has filtering function, hysteresis window function, and polarity selection. Support compare interrupt, which can be used to wake up MCU in low power mode.

## 21.2 Main features

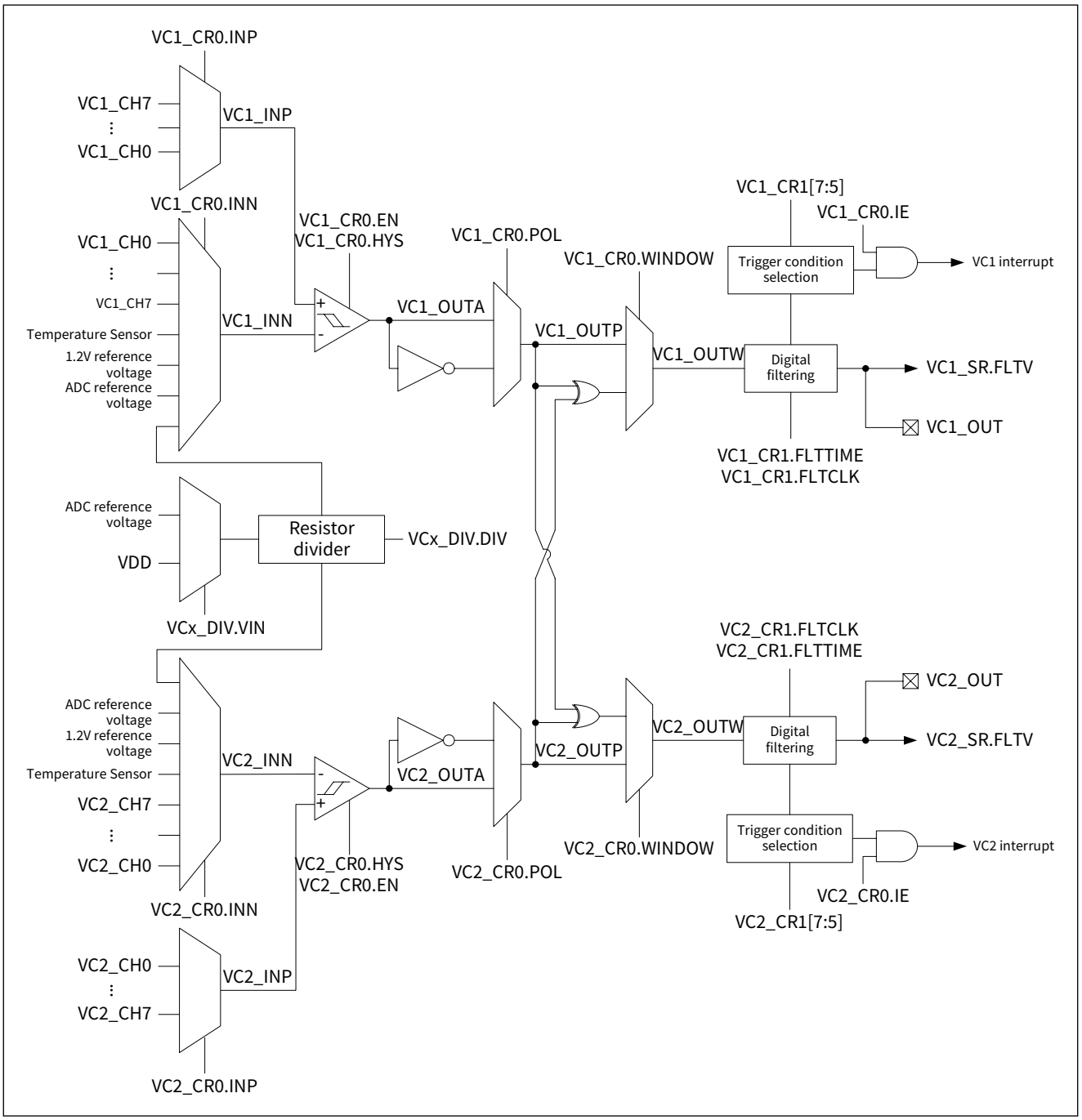
- Dual-channel analog voltage comparators VC1, VC2
- Internal 64-step resistor divider
- Up to 8 external analog signal inputs
- 4 on-chip analog input signals
  - Built-in resistor divider output voltage
  - Built-in temperature sensor output voltage
  - Built-in 1.2V reference voltage
  - ADC reference voltage
- Selectable output polarity
- Supports hysteresis window compare function
- Programmable filters and filter time
- 3 interrupt trigger modes, which can be used in combination
  - High level trigger
  - Rising edge trigger
  - Falling edge trigger
- Supports running in low power mode, interrupt wake-up MCU

## 21.3 Functional description

### 21.3.1 Functional block diagram

The functional block diagram of the analog voltage comparator (VC) is shown in the following figure:

Figure 21-1 VC functional block diagram




The input selection of the positive and negative terminals of VC1 and VC2 is selected by the INP and INN bit fields of the control register VCx_CR0, as shown in the following table:

Table 21-1 VC positive and negative input signal configuration

| VCx_CR0.INP | VCx positive terminal input signal | VCx_CR0.INN | VCx negative terminal input signal |
|---|---|---|---|
| 0000 | VCx_CH0 | 0000 | VCx_CH0 |
| 0001 | VCx_CH1 | 0001 | VCx_CH1 |
| 0010 | VCx_CH2 | 0010 | VCx_CH2 |
| 0011 | VCx_CH3 | 0011 | VCx_CH3 |
| 0100 | VCx_CH4 | 0100 | VCx_CH4 |
| 0101 | VCx_CH5 | 0101 | VCx_CH5 |
| 0110 | VCx_CH6 | 0110 | VCx_CH6 |
| 0111 | VCx_CH7 | 0111 | VCx_CH7 |
| - | - | 1000 | Built-in resistor divider output voltage |
| - | - | 1001 | Reference voltage configured by ADC module<br>Caution: ADC_CR0.EN should be enabled |
| - | - | 1010 | Built-in 1.2V reference voltage<br>Caution: ADC_CR0.BGREN should be enabled |
| - | - | 1011 | Built-in temperature sensor output voltage<br>Caution: ADC_CR0.TSEN and ADC_CR0.BGREN should be enabled |

The built-in resistor divider configuration is controlled by three bit fields in the VCx_DIV register[1]:

- EN bit field, enable resistor divider
- VIN bit field, select the input voltage of the voltage divider
  - VIN is 0, select VDD
  - VIN is 1, select the reference voltage configured by the ADC module (Caution that the ADC must be enabled first)
- DIV bit field, voltage divider proportional coefficient, valid range is 0~63, select the output voltage of the voltage divider, the calculation formula is as follows:

$$\text{Voltage divider output voltage} = \text{input voltage} \times (1 + \text{DIV}) / 64$$

*Caution 1:*

*VC1_DIV and VC2_DIV point to the same entity register, and VC1 and VC2 share the resistor divider circuit.*

When selecting the built-in 1.2V reference voltage or the ADC module reference voltage as the negative input of the comparator, it is necessary to set the BGREN bit field of the ADC control register ADC_CR0 to 1 to enable the BGR module inside the chip. The startup time of the internal BGR is about 20μs, the VC voltage comparator needs to wait for the internal BGR to stabilize before it can work normally.

### 21.3.2 I/O pins

The analog voltage comparator supports 8 external analog signal inputs, and the user must configure the corresponding GPIO port as an analog function (GPIOx_ANALOG.PINy = 1).

The analog voltage comparator supports outputting the comparison result from the pin. The user must configure the corresponding GPIO port as a digital output and select the function multiplexing.

The I/O pins supported by VC1 and VC2 are shown in the following table:

Table 21-2 VC I/O pins configuration

| VC1 I/O | GPIO | Configuration | VC2 I/O | GPIO | Configuration |
|---|---|---|---|---|---|
| VC1_CH0 | PC02 | Analog | VC2_CH0 | PB02 | Analog |
| VC1_CH1 | PB00 | Analog | VC2_CH1 | PA00 | Analog |
| VC1_CH2 | PB01 | Analog | VC2_CH2 | PA01 | Analog |
| VC1_CH3 | PB05 | Analog | VC2_CH3 | PA04 | Analog |
| VC1_CH4 | PB04 | Analog | VC2_CH4 | PA06 | Analog |
| VC1_CH5 | PB03 | Analog | VC2_CH5 | PA07 | Analog |
| VC1_CH6 | PB02 | Analog | VC2_CH6 | PC00 | Analog |
| VC1_CH7 | PA00 | Analog | VC2_CH7 | PC01 | Analog |
| VC1_OUT | PA00 | Digital output (AFR=7) | VC2_OUT | PA01 | Digital output (AFR=2) |
| | PA07 | Digital output (AFR=3) | | PA02 | Digital output (AFR=6) |
| | PC01 | Digital output (AFR=7) | | | |

### 21.3.3 Delay/response time

The time from when VC is enabled or the input voltage at the positive and negative ends of VC changes to the time when the voltage comparator outputs the correct comparison result is defined as the delay/response time of the comparator. The delay/response time is configured by the RESP bit field of the control register VCx_CR0. The response time value is adjustable from 200ns to 20μs. The shorter the response time, the greater the power consumption of the VC module.

### 21.3.4 Polarity selection

The polarity of the output signals of the voltage comparators VC1 and VC2 is set by the POL bit field of the control register VCx_CR0:

- POL is 1, the polarity of VCx_OUTP signal is opposite to that of VCx_OUTA signal, that is, when the positive terminal is greater than the negative terminal, VCx outputs a low level
- POL is 0, and the VCx_OUTP signal has the same polarity as the VCx_OUTA signal, that is, when the positive terminal is greater than the negative terminal, VCx outputs a high level

### 21.3.5 Digital filter

The built-in digital filter of the voltage comparator is used to digitally filter the output signal of the voltage comparator. It is controlled by the FLTEN bit field of the control register VCx_CR1. FLTEN is 1 to enable digital filter, and FLTEN is 0 to disable digital filter. Users can use the filtering function to filter system noise, such as high current noise when the motor is stopped, to avoid system malfunction caused by the noise output of the comparator.

The clock for the digital filter is selected by the FLTCLK bit field in the control register VCx_CR1:

- FLTCLK is 1, use PCLK as filter clock
- FLTCLK is 0, the built-in RC oscillator clock is used as the filter clock, and its frequency is about 150KHz

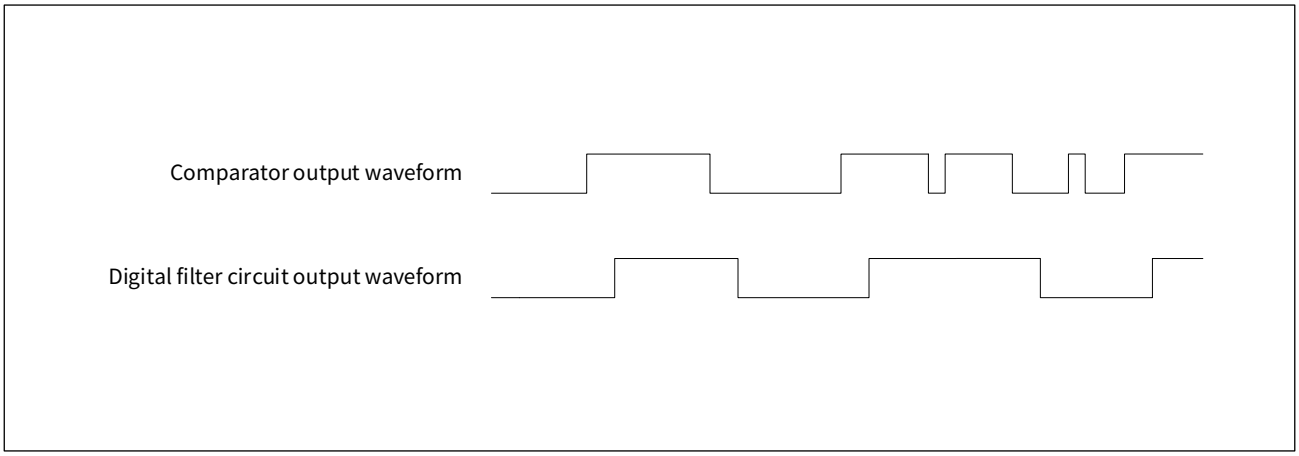
The filter width of the digital filter is selected by the FLTTIME bit field of the control register VCx_CR1, and the signal whose width is less than a certain clock cycle is filtered out. There are 8 filter widths to choose from.

The digitally filtered output level of the voltage comparator can be read out through the FLTV bit field of the status register VCx_SR.

When the user sets the VCx_OUT pin as a digital output and selects the multiplexing function of the VC comparison output, the VCx_OUT pin will output the digitally filtered output level of the voltage comparator.

The filter response waveform of VC is shown in the following figure:

Figure 21-2 VC filter response time



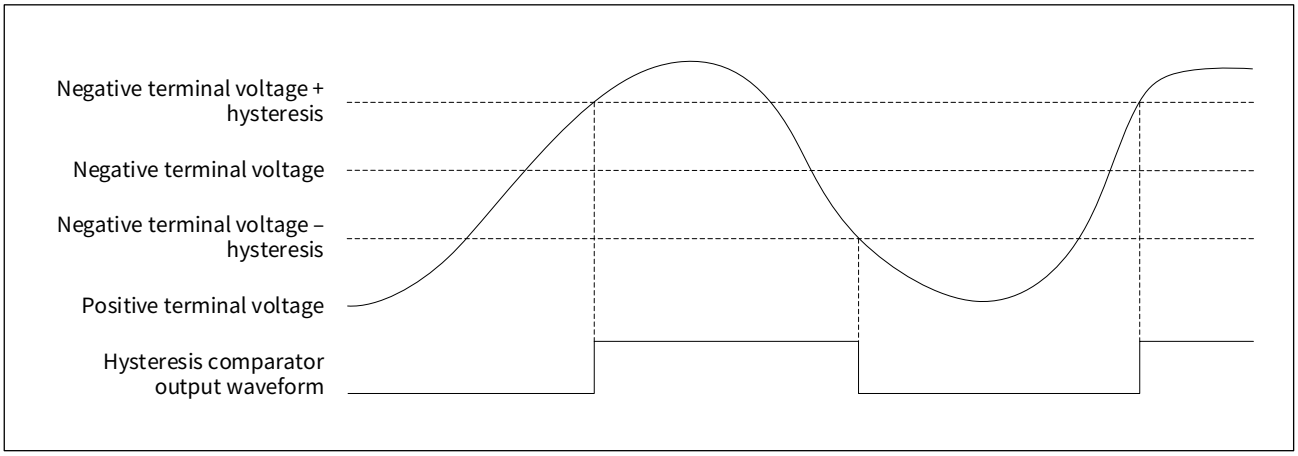

The output of the voltage comparator can be used as the BK brake input or OCREF_CLR input of the advanced timer (ATIM), which is controlled by the ATIMBK and ATIMCLR bit fields of the control register VCx_CR1 respectively. For the detailed introduction of BK and OCREF_CLR, please refer to chapter *13 Advanced-control timer (ATIM)*. This function is usually used in motor protection applications, please refer to the relevant application Cautions for details.

### 21.3.6 Hysteresis function

The analog voltage comparator supports the hysteresis function. After the hysteresis function is used, the output result of the comparator will not be reversed immediately with the change of the input signal, but will reverse only after the offset value of the two input signals is higher or lower than the hysteresis threshold voltage.

The waveform diagram of VC hysteresis function is shown in the following figure:

Figure 21-3 VC hysteresis function




The hysteresis function can effectively enhance the anti-interference ability of the chip, and avoid unnecessary frequent flipping of the output end of the comparator due to the small amplitude jitter of the input signal.

The hysteresis threshold voltage is determined by the HYS bit field of the control register VCx_CR0, as shown in the following table:

Table 21-3 VC hysteresis threshold voltage

| VCx_CR0.HYS | Hysteresis window configuration |
|---|---|
| 00 | No hysteresis |
| 01 | Hysteresis window about 10mV |
| 10 | Hysteresis window about 20mV |
| 11 | Hysteresis window about 30mV |

### 21.3.7 Window compare function

The analog voltage comparator supports the window compare function, which can output the comparison results of VC1 and VC2 after XOR operation, which is enabled by the WINDOW bit field of the control register VCx_CR0.

When WINDOW is 1, the VCx_OUTW signal is the XOR value of the VC1_OUTP signal and the VC2_OUTP signal;

When WINDOW is 0, the VCx_OUTW signal is at the same level as the VCx_OUTP signal.

### 21.3.8 BLANK window function

While keeping the VCx module working, if you want to temporarily stop the voltage comparison function, or in some application systems (such as motor control), the short-term reasonable fluctuation of the monitored signal may cause unnecessary reverse of output levels of voltage comparator, the voltage comparator of this chip has added the BLANK window function, that is, when the specified external trigger condition starts the BLANK window, no voltage comparison is performed within the set BLANK window, and the output level of the voltage comparator remains current. level status. After the BLANK window period, the voltage comparator resumes normal operation.

The BLANK window duration is configured by the BLANKFLT bit field of the control register VCx_CR1. The window duration is

$$(2^{(BLANKFLT+2)} \sim 2^{(BLANKFLT+2)} + 2) \text{ PCLK cycles}$$

As shown in the following table:

Table 21-4 BLANK window duration configuration

| VCx_CR1.BLANKFLT | The number of PCLK clocks in the BLANK window |
|---|---|
| 000 | 4 ~ 6 |
| 001 | 8 ~ 10 |
| 010 | 16 ~ 18 |
| 011 | 32 ~ 34 |
| 100 | 64 ~ 66 |
| 101 | 128 ~ 130 |
| 110 | 256 ~ 258 |
| 111 | 512 ~ 514 |

The trigger start condition of the BLANK window is configured by the BLANKCH1B, BLANKCH2B, and BLANKCH3B bit fields of the control register VCx_CR1, and the BLANK window is triggered by the rising edge of CH1B, CH2B, and CH3B of ATIM respectively.

## 21.4 VC interrupts

The voltage comparator of CW32F003 supports working in low power consumption mode, and the comparison interrupt can wake up the chip from low power consumption mode.

Set the IE bit field of the control register VCx_CR0 to 1 to enable the VCx interrupt. When an interrupt occurs, the interrupt flag bit INTF of the status register VCx_SR will be set to 1 by hardware. The user can write 0 to the INTF bit to clear the interrupt flag.

Set the HIGHIE, RISEIE, FALLIE bit fields of the control register VCx_CR1 to select different interrupt triggering methods:

- HIGHIE is 1, VCx_OUT output signal high level triggers interrupt
- RISEIE is 1, the rising edge of VCx_OUT output signal triggers interrupt
- FALLIE is 1, the falling edge of VCx_OUT output signal triggers interrupt

## 21.5 Programming examples

In this example, the digital filtering function is used, and the compare interrupt is configured as follows:

Step 1: Configure VCx_DIV.EN to 1, enable the voltage divider;

Step 2: Configure VCx_DIV.DIV and set the voltage division coefficient;

Step 3: Configure VCx_CR0.INP and select the voltage source to be monitored at the positive terminal;

Step 4: Configure VCx_CR0.INN and select the voltage source to be monitored at the negative terminal;

Step 5: Configure VCx_CR1.FLTTIME and select the filter time;

Step 6: Configure VCx_CR1.FLTCLK, select filter clock;

Step 7: Configure VCx_CR1.FLTEN to 1 to enable VCx filter;

Step 8: Set the HIGHIE, RISEIE, and FALLIE of the VCx_CR1 register to 1, and select the interrupt trigger mode;

Step 9: Set VCx_CR0.IE to 1 to enable VCx interrupt;

Step 10: Set VCx_CR0.EN to 1 to enable VCx;

Step 11: Wait for the VCx_SR.READY flag to become 1;

Step 12: In the initialization routine of the VC module and the interrupt service routine of the VC module, write 0 to the INTF bit of the VCx_SR register, clear the interrupt flag, and enable the generation of the VCx interrupt.

## 21.6 List of registers

VC1 base address: VC1_BASE = 0x4001 2A00

VC2 base address: VC2_BASE = 0x4001 2A10

Table 21-5 List of VC registers

| Register name | Register address | Register description |
|---|---|---|
| VCx_DIV | VCx_BASE + 0x00 | Resistor divider control register[1] |
| VCx_CR0 | VCx_BASE + 0x04 | Control register 0 |
| VCx_CR1 | VCx_BASE + 0x08 | Control register 1 |
| VCx_SR | VCx_BASE + 0x0C | Status register |

*Caution 1:*

*VC1_DIV and VC2_DIV point to the same entity register, and VC1 and VC2 share the resistor divider circuit.*

## 21.7 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 21.7.1 VCx_DIV resistor divider control register

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | VIN | RW | Resistor divider circuit input voltage setting<br>0: VDD<br>1: The reference voltage configured by the ADC module (the ADC needs to be enabled) |
| 6 | EN | RW | Resistor divider circuit enable control<br>0: Disabled<br>1: Enabled |
| 5:0 | DIV | RW | Resistor divider circuit output voltage configuration<br>Output voltage = input voltage × ( 1 + DIV ) / 64 |

*Caution:*

*VC1_DIV and VC2_DIV point to the same entity register, and VC1 and VC2 share the resistor divider circuit.*

### 21.7.2 VCx_CR0 control register 0

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:12 | INN | RW | VCx negative terminal input signal configuration<br>0000: VCx_CH0<br>0001: VCx_CH1<br>0010: VCx_CH2<br>0011: VCx_CH3<br>0100: VCx_CH4<br>0101: VCx_CH5<br>0110: VCx_CH6<br>0111: VCx_CH7<br>1000: Built-in voltage divider output voltage<br>1001: The reference voltage configured by the ADC module (Caution: ADC_CR0.EN needs to be enabled)<br>1010: Built-in 1.2V reference voltage (Caution: ADC_CR0.BGREN needs to be enabled)<br>1011: Built-in temperature sensor output voltage (Caution: ADC_CR0.TSEN and ADC_CR0.BGREN need to be enabled)<br>Caution: To configure the ADC_CR0 register, you need to enable the ADC peripheral clock first |
| 11:8 | INP | RW | VCx positive terminal input signal configuration<br>0000: VCx_CH0<br>0001: VCx_CH1<br>0010: VCx_CH2<br>0011: VCx_CH3<br>0100: VCx_CH4<br>0101: VCx_CH5<br>0110: VCx_CH6<br>0111: VCx_CH7 |
| 7 | WINDOW | RW | Window compare function configuration<br>0: Window function disabled, VCx_OUTW signal equals VCx_OUTP signal<br>1: Window function enabled, VCx_OUTW signal equals to the XOR of VC1_OUTP and VC2_OUTP |
| 6 | POL | RW | VCx output signal polarity settings<br>0: When the positive terminal is greater than the negative terminal, VCx outputs a high level<br>1: When the positive terminal is greater than the negative terminal, VCx outputs a low level |
| 5 | IE | RW | Interrupt enable configuration<br>0: Disabled<br>1: Enabled |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 4:3 | HYS | RW | VCx hysteresis window configuration<br>00: No hysteresis<br>01: Hysteresis window about 10mV<br>10: Hysteresis window about 20mV<br>11: Hysteresis window about 30mV |
| 2:1 | RESP | RW | VCx response speed configuration<br>00: ultra-low speed<br>01: low speed<br>10: medium speed<br>11: high speed<br>Caution: The faster the response speed, the greater the power consumption. |
| 0 | EN | RW | VCx enable control<br>0: Disabled<br>1: Enabled<br>Caution: After enabling VCx, wait for the VCx_SR.READY flag bit to become 1. |

### 21.7.3 VCx_CR1 control register 1

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:16 | RFU | - | Reserved bits, please keep the default value |
| 15:13 | BLANKFLT | RW | BLANK window duration configuration:<br>( $2^{(BLANKFLT+2)}$ ~ $2^{(BLANKFLT+2)}$+2 ) PCLK cycles<br>Caution: Any rising edge of CH1B, CH2B, and CH3B of ATIM will trigger the start of the BLANK window. During the duration of the BLANK window, no voltage comparison will be performed. |
| 12 | BLANKCH3B | RW | ATIM's CH3B rising edge triggers VCx to start BLANK window configuration<br>0: Disabled<br>1: Enabled |
| 11 | BLANKCH2B | RW | ATIM's CH2B rising edge triggers VCx to start BLANK window configuration<br>0: Disabled<br>1: Enabled |
| 10 | BLANKCH1B | RW | ATIM's CH1B rising edge triggers VCx to start BLANK window configuration<br>0: Disabled<br>1: Enabled |
| 9 | ATIMBK | RW | ATIM's BK brake signal connectivity configuration<br>0: No function<br>1: VCx output connected to BK brake signal of ATIM |
| 8 | ATIMCLR | RW | ATIM's OCREF_CLR signal connectivity configuration<br>0: No function<br>1: VCx output connected to OCREF_CLR signal of ATIM |
| 7 | HIGHIE | RW | VCx output signal high level trigger interrupt enabled<br>0: Disabled<br>1: Enabled |
| 6 | RISEIE | RW | VCx output signal rising edge trigger interrupt enabled<br>0: disable<br>1: Enabled |
| 5 | FALLIE | RW | VCx output signal falling edge trigger interrupt enabled<br>0: Disabled<br>1: Enabled |
| 4 | FLTCLK | RW | Digital filter module filter clock settings<br>0: Built-in RC oscillator clock, its frequency is about 150KHz<br>1: PCLK |

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 3:1 | FLTTIME | RW | Digital filter module filter time configuration<br>000: Filter out signals with width less than 1 clock cycle<br>001: Filter out signals with width less than 3 clock cycle<br>010: Filter out signals with width less than 7 clock cycles<br>011: Filter out signals with width less than 15 clock cycles<br>100: Filter out signals with width less than 63 clock cycles<br>101: Filter out signals with width less than 255 clock cycles<br>110: Filter out signals with width less than 1023 clock cycles<br>111: Filter out signals with width less than 4095 clock cycles |
| 0 | FLTEN | RW | Digital filter module enable configuration<br>0: Disabled<br>1: Enabled |

### 21.7.4 VCx_SR status register

Address offset: 0x0C Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:3 | RFU | - | Reserved bits, please keep the default value |
| 2 | READY | RO | VCx status flag, by reading this bit to determine whether VCx has been stable<br>0: Not yet stable, cannot be used<br>1: Already stable and ready to use<br>Caution: If VCx is enabled, the flag must be polled and set to 1 before entering DeepSleep mode. |
| 1 | FLTV | RO | Level value of digital filter output<br>0: Digital filter output low level<br>1: Digital filter output high level |
| 0 | INTF | RW0 | Interrupt flag<br>R0: No VC interrupt occurred<br>R1: VC interrupt has occurred<br>W0: VC interrupt flag cleared<br>W1: No function |

# 22 Low voltage detector (LVD)

## 22.1 Overview

Low voltage detector (LVD) is used to monitor VDD power supply voltage or external pin input voltage. When the comparison result between the monitored voltage and the LVD threshold meets the trigger condition, an LVD interrupt or reset signal will be generated, which is usually used to handle some urgent tasks .

The interrupt and reset flags generated by the LVD can only be cleared by software; only after the interrupt or reset flag is cleared and the trigger condition is reached again, the LVD can generate an interrupt or reset signal again.

## 22.2 Main features

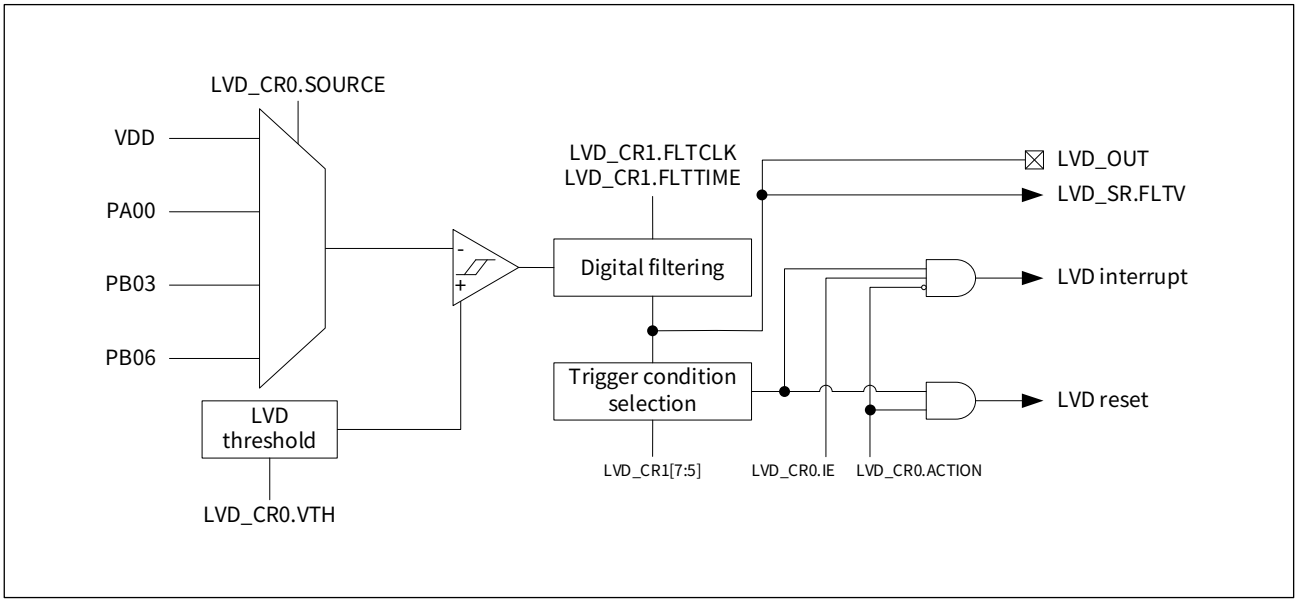
- 4-channel monitoring voltage source: VDD power supply voltage, PA00, PB03, PB06 pin input
- 16-step threshold voltage, range 1.8V ~ 3.3V
- 3 trigger conditions, which can be used in combination
  - Level triggered: voltage below threshold
  - Falling edge trigger: the falling edge when the voltage falls below the threshold
  - Falling edge trigger: the falling edge when the voltage falls below the threshold
- Can trigger to generate interrupt or reset signal, both cannot be generated at the same time
- 8th order filter configurable
- Hysteresis function supported
- Supports running in low power mode, interrupt wake-up MCU

## 22.3 Functional description

### 22.3.1 Functional block diagram

The functional block diagram of the Low voltage detector (LVD) is shown in the following figure:

Figure 22-1 LVD functional block diagram




The LVD can monitor the VDD power supply voltage, as well as the external pin input voltage (PA00, PB03, PB06), which is selected by the SOURCE bit field of the control register LVD_CR0. When using external analog signal input, the user must configure the corresponding GPIO port as analog function (GPIOx_ANALOG.PINy = 1).

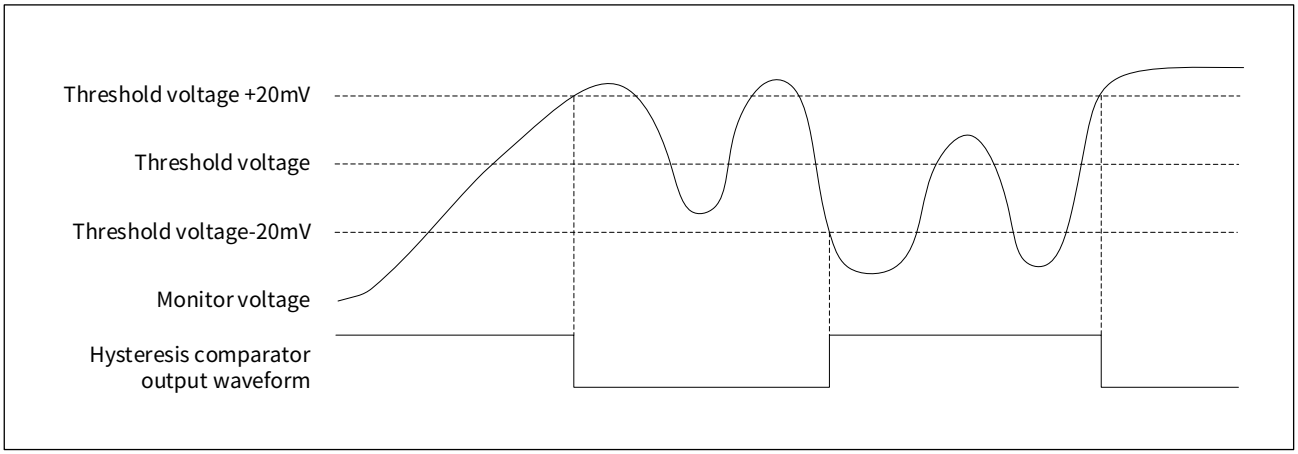
The LVD output result can be output from the PA03/PA06 or PB01 pin, the user must configure the corresponding GPIO port as a digital output, and select function multiplexing at the same time.

### 22.3.2 Hysteresis function

The built-in voltage comparator of the LVD has a hysteresis function, which can avoid frequent flipping of the output result of the voltage comparator when the monitored voltage of the LVD is near the threshold voltage, and enhance the anti-interference ability of the system.

The comparator output signal toggles only when the monitored voltage is 20mV above or below the threshold voltage. The specific waveform is shown in the following figure:

Figure 22-2 LVD hysteretic response




The threshold voltage of LVD is determined by the VTH bit value of the control register LVD_CR0, and the effective value is 0 ~ 15, as shown in the following table:

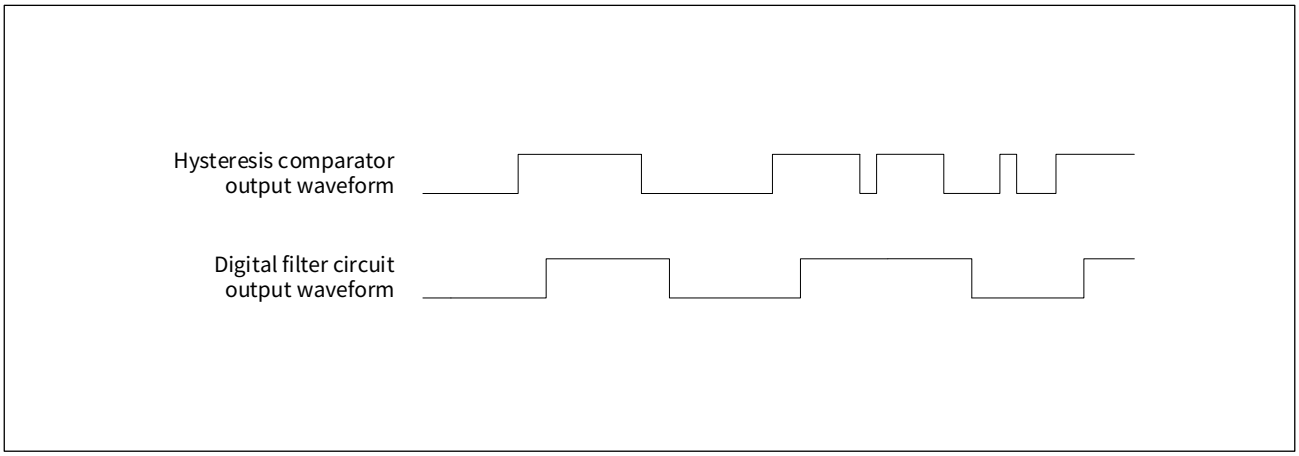
Table 22-1 LVD threshold voltage

| LVD_CR0.VTH | Threshold voltage (unit: V) | LVD_CR0.VTH | Threshold voltage (unit: V) |
|---|---|---|---|
| 0000 | 1.8 | 1000 | 2.6 |
| 0001 | 1.9 | 1001 | 2.7 |
| 0010 | 2.0 | 1010 | 2.8 |
| 0011 | 2.1 | 1011 | 2.9 |
| 0100 | 2.2 | 1100 | 3.0 |
| 0101 | 2.3 | 1101 | 3.1 |
| 0110 | 2.4 | 1110 | 3.2 |
| 0111 | 2.5 | 1111 | 3.3 |

### 22.3.3 Digital filter

In order to enhance the robustness of the system, the LVD supports the digital filtering function, which can digitally filter the output result signal of the LVD voltage comparison, and the signal smaller than the filtering width is filtered out without triggering an interrupt or reset. The specific waveform is shown in the following figure:

Figure 22-3 LVD filter output




Set the FLTEN bit field of the control register LVD_CR1 to 1 to enable the digital filter module;

The FLTCLK bit field of the control register LVD_CR1 is used to select the clock for digital filter:

- FLTCLK bit is 1, select HSIOSC as filter clock
- FLTCLK bit is 0, and the built-in RC oscillator clock is selected as the filter clock, and its frequency is about 150KHz

The FLTTIME bit field of the control register LVD_CR1 is used to select the number of clocks for digital filter, as shown in the following table:

Table 22-2 Number of LVD digital filter clocks

| LVD_CR1.FLTTIME | Number of filter clocks |
|---|---|
| 000 | 1 |
| 001 | 3 |
| 010 | 7 |
| 011 | 15 |
| 100 | 63 |
| 101 | 255 |
| 110 | 1023 |
| 111 | 4095 |

From the FLTV bit field of the LVD status register LVD_SR, the signal level after LVD digital filter can be read out;

When the function of GPIO is multiplexed as LVD_OUT, the digitally filtered signal can be output from GPIO to facilitate observation and measurement.

## 22.4 LVD interrupts

LVD supports working in low power mode, and interrupt output can wake up the chip from low power mode.

When the comparison result of the monitored voltage and the LVD threshold meets the trigger condition, an interrupt or reset signal can be generated. Whether to generate an interrupt or a reset signal is controlled by the ACTION bit field of the control register LVD_CR0:

- ACTION is 1, LVD triggers to generate reset
- ACTION is 0, LVD triggers an interrupt

Set the IE bit field of the control register LVD_CR0 to 1 to enable the LVD interrupt. When the trigger condition is met, an LVD interrupt will be generated. The interrupt flag bit LVD_SR.INTF will be set to 1 by hardware. The user can write 0 to the INTF bit to clear the interrupt flag.

Set the LEVEL, FALL, and RISE bit fields of the control register LVD_CR1 to select different interrupt or reset trigger methods. The three can be used in combination:

- LEVEL is 1, when the monitored voltage is lower than the threshold, an interrupt is triggered or a reset is generated
- FALL is 1, the falling edge of the monitored voltage falling below the threshold triggers an interrupt or generates a reset
- RISE is 1, the rising edge of the monitored voltage rises above the threshold to trigger an interrupt or generate a reset

## 22.5 Programming examples

### 22.5.1 Brown-out reset programming example

In this example, the MCU is reset when the monitored voltage falls below the threshold voltage, using digital filtering.

The configuration method is as follows:

Step 1: Configure LVD_CR0.SOURCE and select the voltage source to be monitored;

Step 2: Configure LVD_CR0.VTH and set the threshold voltage;

Step 3: Configure LVD_CR1.FLTTIME and select LVD filter time;

Step 4: Configure LVD_CR1.FLTCLK and select the filter clock;

Step 5: Configure LVD_CR1.FLTEN to enable LVD filter;

Step 6: Set LVD_CR1.LEVEL to 1, and select to trigger LVD action when the monitored voltage is lower than the threshold;

Step 7: Set LVD_CR0.ACTION to 1, and select LVD trigger action as system reset;

Step 8: Set LVD_CR0.EN to 1 to enable LVD.

### 22.5.2 Interrupt programming example

In this example, an interrupt is generated when the monitored voltage is above or below a threshold voltage, using digital filtering.

The configuration method is as follows:

Step 1: Configure LVD_CR0.SOURCE and select the voltage source to be monitored;

Step 2: Configure LVD_CR0.VTH and select the threshold voltage;

Step 3: Configure LVD_CR1.FLTTIME and select LVD filter time;

Step 4: Configure LVD_CR1.FLTCLK and select the filter clock;

Step 5: Configure LVD_CR1.FLTEN to enable LVD filter;

Step 6: Set both LVD_CR1.RISE and FALL to 1, and select rising and falling edge triggers;

Step 7: Set LVD_CR0.ACTION to 0, and select LVD trigger action as interrupt;

Step 8: Set LVD_CR0.IE to 1 to enable LVD interrupt;

Step 9: Enable the LVD interrupt in the NVIC interrupt vector table;

Step 10: Set LVD_CR0.EN to 1 to enable LVD;

Step 11: In the LVD initialization routine and the LVD interrupt service routine, write 0 to the LVD_SR.INTF bit to clear the interrupt flag and allow a new LVD interrupt to be generated.

## 22.6 List of registers

LVD base address: LVD_BASE = 0x4001 2A80

Table 22-3 List of LVD registers

| Register name | Register address | Register description |
|---|---|---|
| LVD_CR0 | LVD_BASE + 0x00 | Control register 0 |
| LVD_CR1 | LVD_BASE + 0x04 | Control register 1 |
| LVD_SR | LVD_BASE + 0x08 | Status register |

## 22.7 Register descriptions

See section *1 Documentation conventions* for abbreviations used in register descriptions.

### 22.7.1 LVD_CR0 control register 0

Address offset: 0x00 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:10 | RFU | - | Reserved bits, please keep the default value |
| 9 | IE | RW | Interrupt enable control<br>0: LVD interrupt disabled<br>1: LVD interrupt enable |
| 8 | RFU | - | Reserved bits, please keep the default value |
| 7:4 | VTH | RW | Threshold voltage selection<br>0000: 1.8V 1000: 2.6V<br>0001: 1.9V 1001: 2.7V<br>0010: 2.0V 1010: 2.8V<br>0011: 2.1V 1011: 2.9V<br>0100: 2.2V 1100: 3.0V<br>0101: 2.3V 1101: 3.1V<br>0110: 2.4V 1110: 3.2V<br>0111: 2.5V 1111: 3.3V |
| 3:2 | SOURCE | RW | Monitor source configuration<br>00: VDD supply voltage<br>01: PB06 port input voltage<br>10: PB03 port input voltage<br>11: PA00 port input voltage |
| 1 | ACTION | RW | Trigger action configuration<br>0: NVIC interrupts<br>1: System resets |
| 0 | EN | RW | Enable control<br>0: LVD disabled<br>1: LVD enabled |

### 22.7.2 LVD_CR1 control register 1

Address offset: 0x04 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:8 | RFU | - | Reserved bits, please keep the default value |
| 7 | LEVEL | RW | Triggered when the monitored voltage falls below the threshold level<br>(that is, the LVD output signal is triggered by a high level)<br>0: Disabled<br>1: Enabled |
| 6 | FALL | RW | Trigger on falling edge of monitored voltage falling below threshold<br>(that is, the rising edge of the LVD output signal is triggered)<br>0: Disabled<br>1: Enabled |
| 5 | RISE | RW | Triggered on rising edge when the monitored voltage rises back above the threshold<br>(that is, the falling edge of the LVD output signal is triggered)<br>0: Disabled<br>1: Enabled |
| 4 | FLTCLK | RW | Digital filter module filter clock settings<br>0: Built-in RC oscillator clock, its frequency is about 150KHz<br>1: HSIOSC |
| 3:1 | FLTTIME | RW | Digital filter module filter time configuration<br>000: Filter out signals with width less than 1 clock cycle<br>001: Filter out signals with width less than 3 clock cycle<br>010: Filter out signals with width less than 5 clock cycle<br>011: Filter out signals with width less than 15 clock cycle<br>100: Filter out signals with width less than 63 clock cycle<br>101: Filter out signals with width less than 255 clock cycle<br>110: Filter out signals with width less than 1023 clock cycle<br>111: Filter out signals with width less than 4095 clock cycle |
| 0 | FLTEN | RW | Digital filter module enable configuration<br>0: Disabled<br>1: Enabled |

### 22.7.3 LVD_SR status register

Address offset: 0x08 Reset value: 0x0000 0000

| Bit field | Name | Permission | Function description |
|---|---|---|---|
| 31:2 | RFU | - | Reserved bits, please keep the default value |
| 1 | FLTV | RO | Level value of digital filter output<br>0: Digital filter output low level<br>1: Digital filter output high level |
| 0 | INTF | RW0 | Interrupt flags<br>R0: LVD interrupt did not occur<br>R1: LVD interrupt has occurred<br>W0: LVD interrupt flag cleared<br>W1: No function |

# 23 Debug interface (DBG)

## 23.1 Overview

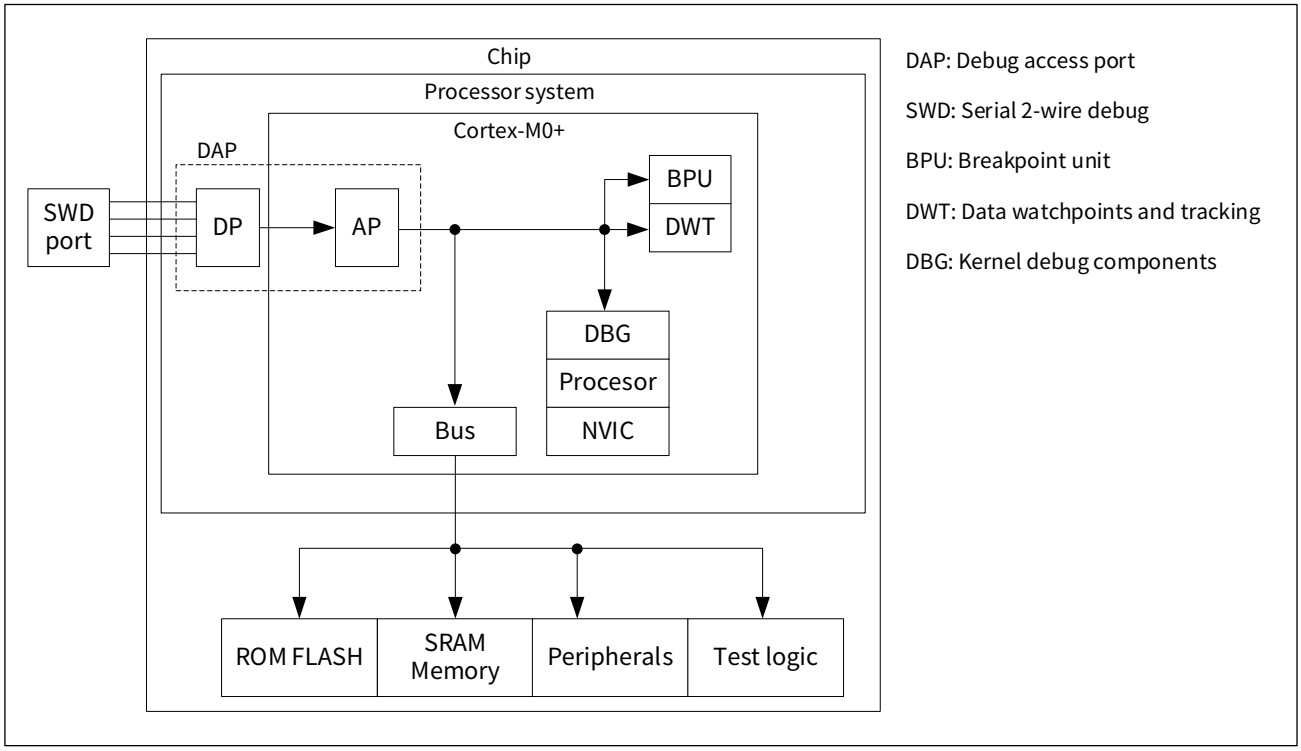
The core of CW32F003 is ARM® Cortex®-M0+, the core has built-in DAP hardware debug module, and supports SWD mode debugging. The hardware debug module can be suspended when fetching instructions (instruction breakpoints) or accessing data (data breakpoints), and the program stops running. query; and the core and peripherals can be restored, and program execution continues.

When using the debugging simulation tool to connect to the CW32F003 through the SWD interface, enter the debugging mode, and perform debugging operations through the DAP hardware debugging module in the chip core.

## 23.2 Serial wire debug port SWD

Use the SWD interface of the CW special debugger or the general debugging simulation tool to connect with the DAP debugging module inside the target chip, and carry out data exchange through the packet transmission protocol to realize the debugging operation. The SWD mode is a 2-wire serial communication, including a clock line SWCLK and a bidirectional data line SWDIO. As shown in the following figure:

Figure 23-1 SWD functional block diagram




The pin assignment of the SWD interface of the CW32F003 series is shown in the following table. The PA02/PA05 pin is the SWD function by default when the chip leaves the factory.

Table 23-1 SWD pin assignment

| SWD port name | Pin function | Pin assignment |
|---|---|---|
| SWCLK | Serial clock input | PA05 |
| SWDIO | Serial data input and output | PA02 |

The PA02/PA05 pins can be configured as SWD function or GPIO/ISP function, and the function configuration is performed by the SWDIO bit field of the system control register SYSCTRL_CR2. When SWDIO is 0, the PA02/PA05 pin is configured as SWD function; when SWDIO is 1, it is configured as GPIO/ISP function.

It is generally recommended to use a 100KΩ pull-up resistor for the SWD pin. When the PA02/PA05 of the CW32F003 is used as the SWD function, there is a built-in pull-up resistor with a resistance value between 50KΩ ~ 200KΩ. Users can add a pull-up resistor externally to improve the anti-interference performance.

The PA02/PA05 pin of the CW32F003 series is the SWD function by default. If the user has set the encryption level, it is necessary to judge whether the SWD function is supported according to the set level. The configuration and function of the SWD pin are shown in the following table:

Table 23-2 SWD pin function

| Encryption level | SYSCTRL_CR2.SWDIO | PA02/PA05 function |
|---|---|---|
| Level0/1 encryption | 0 (default) | SWD function |
| | 1 | GPIO/ISP function |
| Level2 encryption | 0 (default) | NA, no function |
| | 1 | GPIO/ISP function |
| Level3 encryption | 0 (default) | NA, no function |
| | 1 | GPIO function |

*Cautions :*

*1. The encryption level can be set through the ISP communication method. For details, please refer to the ISP communication protocol document;*

*2. After the chip is powered on, both SWDIO and SWCLK are internal pull-ups by default, and users do not need to connect external pull-up resistors;*

*3. When the chip encryption level is set to 2, the SWD function is disabled and can only be programmed by ISP;*

*4. When the chip encryption level is set to 3, the SWD function and the ISP function are disabled, and the chip cannot be programmed with new programs again.*

## 23.3 Debug communication protocol

The debugger and the DAP debug module of the target chip communicate through the SWD packet transmission protocol. The packet transmission protocol is a 2-wire synchronous serial protocol, using the SWCLK clock signal and the SWDIO data signal:

- SWCLK is a one-way clock signal, which is output by the debugger to the target chip
- SWDIO is a bidirectional data signal, which is driven by the debugger and the target chip bidirectionally time-sharing

The protocol defines the conversion time of the transceiver and the transceiver with a length of one SWCLK cycle. During the conversion time of the transceiver and the target chip, neither the debugger nor the target chip drives SWDIO, and SWDIO is pulled up to a high level by a pull-up resistor.

When transmitting data on the SWDIO signal line, follow the principle that the lowest LSB is transmitted first, and the highest MSB is transmitted last.

Through the packet transfer protocol, the debugger can read and write access to the DP register and AP register in the DAP debug module of the target chip, hereinafter referred to as SW-DP and SW-AP.

### 23.3.1 Transmission protocol format

The DAP debug module includes the DP debug port register and the AP access port register. The communication between the debugger and the DAP debug module of the target chip is actually the read and write operations on the DP register and the AP register.

The transmission communication frame usually contains three fields:

- Package request
  The length is 8-bit, the debugger to the target chip;
- Response
  The length is 3-bit; the target chip to the debugger;
- Data transmission
  The length is 33-bit, the target chip to the debugger or the debugger to the target chip, optional field.

The function definition of each request package is as follows:

Table 23-3 Definition of packet request bits

| Bit | Name | Description |
|---|---|---|
| 0 | Start up | Must be 1 |
| 1 | APnDP | 0: DP access; 1: AP access |
| 2 | RnW | 0: Write request; 1: Read request |
| 3:4 | A[3:2] | Address field of DP or AP register, bit3 is A2, bit4 is A3 |
| 5 | Parity check | Bit parity bit of the previous 5-bit |
| 6 | Stop | 0 |
| 7 | Reside | Not driven by the host. Target chip reads as 1 due to pull-up |

The request packet is always followed by the conversion time of the transceiver (default 1bit). At this time, neither the master nor the target drives SWDIO.

The response bits transmitted by the target are defined as follows:

Table 23-4 Definition of target transmit response bits

| Bit | Name | Description |
|---|---|---|
| 0:2 | ACK | 001: FAULT (bit0:0, bit1:0, bit2:1)<br>010: WAIT (bit0:0, bit1:1, bit2:0)<br>100: OK (bit0:1, bit1:0, bit2:0) |

After the ACK response is the conversion time of the transceiver (default 1bit), at this time, neither the master nor the target will drive SWDIO.

Data transmitted by the debugger or target chip, the bits are defined as follows:

Table 23-5 Definition of data transmission bits

| Bit | Name | Description |
|---|---|---|
| 0:31 | WDATA or RDATA | Write or read data |
| 32 | Parity check | Bit parity bit of 32-bit data |

The data transmission phase is not necessary, and data transmission occurs only in the following two cases:

- The response received after the data read or write request is the OK response
- The ORUNDETECT flag bit of the DP-CTRL/STAT register is set to 1. At this time, no matter what kind of response (including WAIT and FAULT) is received, data transmission is required

### 23.3.2 SW-DP state machine

There is an ID CODE register inside SW-DP with a fixed value of 0x0BB11477 (Cortex®-M0+ identification code of ARM company). Before using the SWD packet protocol to read and write the target register, the ID number must be read to activate the SW-DP logic of the target chip, otherwise the SW-DP state machine of the target chip will not work.

The operation sequence is as follows:

Step 1: After power-on reset or after the SWDIO line is at a high level for more than 50 cycles, the SW-DP state machine enters the reset state;

Step 2: After entering the reset state, keep the SWDIO line at a low level for at least 2 cycles, and the SW-DP state machine enters the idle state;

Step 3: After entering the idle state, perform read access to the ID CODE register of DP-SW;

*Caution:*

*If this operation is not performed, the target board will send a FAULT response in the subsequent packet communication response phase.*

Step 4: Read and write access to the registers to be accessed according to the packet protocol.

### 23.3.3 Read and write access for SW-DP and SW-AP

- Write access for SW-DP/SW-DP

  After the debugger receives the ACK, the data must be transmitted immediately after the conversion time of 1 clock cycle.

- Read access for SW-DP
  - When the target chip is ready to transmit data, it transmits an OK response, and then transmits data immediately;
  - When the target chip DAP is not ready for data transmission, it transmits a WAIT response to end the communication;
  - When the DAP status of the target chip is wrong, a FAULT response is transmitted to end the communication.
- Read access for SW-AP

  The read access to the AP is registered, that is, the read AP operation cannot return the required result, but the result of the read operation cannot be returned until the next AP operation. If the next access to be performed is not an AP read access operation, the DP-RDBUFF register must be read to obtain the result of this read operation.
- The READOK flag in the DP-CTRL/STAT register of the M0 core's debug control/status register is updated every time a SW-AP read access or DP-RDBUFF read request is made to indicate whether the AP read access is successful.
- SW-DP has write buffers (for SW-DP or SW-AP writes) that can accept the next write operation while the read and write operations are not complete. If the SW-DP's write buffer is full, the chip SW-DP logic will reply with a WAIT response to notify the host to suspend the operation. The exception is special operations such as IDCODE read, DP-CTRL/STAT read or ABORT write operations, which are also accepted when the write buffer is full.
- Since SWCLK and HCLK are not synchronous clocks and belong to the asynchronous clock domain, two additional SWCLK cycles are required after the write operation (after the parity bit of the data transmission) to ensure that the write operation takes effect. The SWDIO line should be driven low (idle state) while the master is driving these 2 SWCLK cycles. This is especially important when writing power-on request operations to the DP-CTRL/STAT registers, otherwise the next operation (one that is valid after the core is powered on) will be executed immediately and will fail.

### 23.3.4 SW-DP register

When APnDP is 0, access the DP register, which is addressed by A[3:2]. When the address is the same, the meaning of the register may be different due to different read and write operations, as shown in the following table:

Table 23-6 List and definition of DP registers

| A[3:2] | Read/write | R egister name | Register content description |
|---|---|---|---|
| 00 | Read | IDCODE | Fixed at 0x0BB1 1477 (to identify SW-DP),<br>ARM Cortex ® -M0+ identification code |
| | Write | ABORT | See *Table 23-7 Definition of DP ABORT register bits* |
| 01 | Read/write | DP-CTRL/STAT<br>(SELECT.CTRLSEL = 0) | Mainly used for:<br>1. Request system or debug power-on;<br>2. Configure transfer operation for AP access;<br>3. Control comparison and verification operations;<br>4. Read some status flags (overflow and power-up acknowledgment) |
| | | WIRE CONTROL<br>(SELECT.CTRLSEL = 1) | Used to configure the physical serial port protocol (such as the duration of transition times, etc.) |
| 10 | Read | READ RESEND | Allow recovery of read data from corrupted debug transfers without repeating the original AP transfer. |
| | Write | SELECT | Used to select the current access AP port and the 4-word register BANK in the AP port.<br>See *Table 23-8 DP SELECT register* |
| 11 | Read/write | READ BUFFER | Since an AP access has been issued, this read buffer is useful (providing the result of reading this AP request when executing the next AP transaction). This read buffer captures the data in the AP, displayed as the result of the previous read, without starting a new operation. |

Table 23-7 Definition of DP ABORT register bits

| Bit | Name | Description |
|---|---|---|
| 31:5 | RESERVE | |
| 4 | ORUNERRCLR | Write 1 to clear the STICKYORUN error flag |
| 3 | WDATAERR | Write 1 to clear the WDATAERR error flag |
| 2 | STICKYERR | Write 1 to clear the STICKYERR error flag |
| 1 | STICKYCMP | Write 1 to clear the STICKYCMP flag |
| 0 | DAPABORT | Write 1 to generate the DAP ABORT signal and abandon the current access operation; when the target chip responds to the WAIT response, this operation must be performed to abort this operation. |

Table 23-8 DP SELECT register

| Bit | Name | Description |
|---|---|---|
| 31:24 | APSEL | Select the current AP port, fixed as b00000000 |
| 23:8 | RESERVE | |
| 7:4 | APBANKSEL | Select the active 4-word register BANK on the current AP |
| 3:1 | RESERVE | |
| 0 | CTRLSEL | DP port register selection:<br>The default is 0 after reset, select the CTRL/STAT register;<br>Set to 1 to select the WCR register (Wire Control Register) |

### 23.3.5 SW-AP register

When APnDP is 1, access the AP register. Since there are many SW-AP registers, it is necessary to combine A[3:2] and the APBANKSEL bit field value of the SW-DP selection register SELECT to uniquely address the SW-AP register.

This article will not go into details about SW-AP one by one, but only introduces a common IDR public register (address 0xFC), as shown in the following table:

Table 23-9 AP IDR public register

| Bit | Definition |
|---|---|
| 31:28 | AP designed version Rervision. |
| 27:24 | AP designer identification code, this chip is fixed to 0x04. |
| 23:17 | AP designer identification code, this chip is fixed to 0x3B. |
| 16:13 | AP type. For example, the memory access port type identification code is b1000, and the undefined type access port type identification code is b0000. |
| 12:8 | Reserve |
| 7:0 | AP ID. Such as MEM-AP or JTAG-AP and so on. |

## 23.4 Kernel debugging

The core can be debugged by manipulating the core debug registers, and the debugger accesses these registers through DAP debugging.

The kernel debug registers mainly include 4 registers, as shown in the following table:

Table 23-10 Kernel debug registers

| Register | Description |
|---|---|
| DHCSR | Debug suspend control and status register, 32-bit, control processor suspend, single step and restart actions |
| DCRSR | Debug kernel register selector register, 17-bit, controls reading and writing of kernel registers during suspend |
| DCRDR | Debug kernel register data register, 32-bit, data transfer register for reading and writing kernel registers during pause |
| DEMCR | Debug exception monitoring control register, 32-bit, used to enable data watchpoint unit and vector capture feature, using vector capture, the debugger can suspend the processor when the processor is reset or a hardware error occurs |

These registers can only be reset by a power-on reset. See《Cortex®-M0+ Technical Reference Manual》 for more details.

## 23.5 Breakpoint unit BPU

The Cortex®-M0+ BPU breakpoint unit provides four breakpoint registers to implement the PC pointer-based breakpoint function.

For more information on the identification registers and access types of the BPU CoreSight component, see《ARMv6-M Architecture Reference Manual》 and the ARM® CoreSight Component Technical Reference Manual.

## 23.6 Data observation points and tracking DWT

Cortex ®- M0+ DWT provides two observation point register sets. Realize the following functions:

- Set data watchpoints:

  Data or peripheral addresses can be marked as watch variables, and access to this address will generate a debug event and suspend program execution.

- Optional DWT program counter sampling register (DWT_PCSR) feature in ARMv6-M. Allows the debugger to periodically sample the PC pointer, providing a rough analysis without stopping the processor.

See《ARMv6-M Architecture Reference Manual》for more information.

## 23.7 Debug components DBG

The debugger implements clock control support for peripherals such as timers, and watchdogs during breakpoints through the debug components DBG. Through the setting of the debug state timer control register SYSCTRL_DEBUG, you can control the timer, watchdog timer, etc. to run normally or pause during the breakpoint in the debug state, as shown in the following table:

Table 23-11 Debug status timer control register SYSCTRL_DEBUG

| Bit | Name | Vaule | Counter function configuration during breakpoints in debug state |
|---|---|---|---|
| 10 | WWDT | 1 | In debug state, the WWDT counter pauses counting |
| | | 0 | In debug state, the WWDT counter counts normally |
| 9 | IWDT | 1 | In debug state, the IWDT counter pauses counting |
| | | 0 | In debug state, the IWDT counter counts normally |
| 6 | AWT | 1 | In debug state, the AWT counter pauses counting |
| | | 0 | In debug state, the AWT counter counts normally |
| 5 | BTIM123 | 1 | In debug state, the BTIM1, BTIM 2, BTIM 3 counters pause counting |
| | | 0 | In debug state, the BTIM1, BTIM 2, BTIM 3 counters count normally |
| 1 | GTIM | 1 | In debug state, the GTIM counter pauses counting |
| | | 0 | In debug state, the GTIM counter counts normally |
| 0 | ATIM | 1 | In debug state, the ATIM counter pauses counting |
| | | 0 | In debug state, the ATIM counter counts normally |

For example, when the timer outputs PWM for motor control, the timer cannot be stopped, otherwise the motor will be damaged. For another example, the watchdog timer should stop counting during the breakpoint to prevent an undesired reset of the system.

## 23.8 Cautions

CW32F003 needs to use HCLK for debugging connection during debugging. It is not allowed to turn off HCLK during the debugging session. Therefore, executing WFI in the debugging environment will turn off the kernel HCLK, causing the debugging function to fail.

If the target chip has entered the DeepSleep mode, the DAP hardware debugging module does not work at this time, the debugger cannot connect with the DAP hardware debugging module of the target chip, and the debugging function cannot be realized.

The user must ensure that the chip is in Active mode or Sleep mode to use the debugger for debugging.

Therefore, it is recommended that users complete all functional debugging of the device in non-DeepSleep mode before merging the relevant code of deep sleep, and then debug the code related to deep sleep.

# 24 Digital signature

## 24.1 Overview

The digital signature is mainly used to store information such as the unique identification (UID) of the chip, product model, FLASH capacity, SRAM capacity, and the number of pins in the chip package, which can be read by SWD or CPU. The digital signature-related information is programmed at the factory, and user firmware or external devices can verify the legitimacy of the chip by reading the digital signature.

## 24.2 Product Unique Identification (UID) register (80bit)

The UID register stores the unique identifier of the chip, and its address is not continuous, as shown in the following table:

Table 24-1 UID definition and address

| UID composition | Address | Example | UID example |
|---|---|---|---|
| Column address | 0x0010 0858 | 0x0012 | 1200780018E0E2FA03E5 |
| Row address | 0x0010 0854 | 0x0078 | |
| Wafer ID | 0x0010 0850 | 0x18 | |
| Lot ID | 0x0010 084C | 0x03FAE2E0 | |
| Serial ID | 0x0010 085C | 0xE5 | |

The UID is written during chip production and cannot be modified by the user. UID registers support reading in single-byte/half-word/full-word, etc., and then concatenated using custom algorithms.

Typical application scenarios of unique identifiers:

- Used as device serial number
- Device legitimacy verification to prevent piracy

  When the device is produced, the user uses the private key to encrypt the UID, and stores the calculation result in the main FLASH memory or OTP memory. If the result is compared with the previously stored calculation result, it is considered that the device is legal, otherwise the program will not start, which can effectively prevent the user's device from being illegally copied (piracy).
- Used as a security key

  Combined with UID and private algorithm, users can perform security verification before users program FLASH to improve the security of the code in FLASH.
- Activate the secure boot process, etc.

## 24.3 Product model register

The product model register stores the ASCII code of the product model, and its address is 00x0010 07D0 - 0x0010 07E1, with a total of 18 bytes. When the product model is less than 18 bytes, it is filled with 0x00.

For example, the model of the chip is CW32F003E4P7, the corresponding data stored (starting from the low address) is: 0x43 0x57 0x33 0x32 0x46 0x30 0x30 0x33 0x45 0x34 0x50 0x37 0x00 0x00 0x00 0x00 0x00 0x00.

*Caution:*

*If there are letters in the product model, the letters need to be capitalized.*

## 24.4 FLASH capacity register

The FLASH capacity register stores the capacity of the built-in FLASH memory of the chip, and its address is 0x0010 07E4 - 0x0010 07E7, a total of 4 bytes.

The FLASH capacity size read from the FLASH capacity register is in bytes, such as 0x0000 5000 for 20KB, and 0x0000 4000 for 16KB.

## 24.5 SRAM capacity register

The SRAM capacity register stores the capacity of the built-in SRAM memory in the chip, and its address is 0x0010 07E8 - 0x0010 07EB, a total of 4 bytes.

The SRAM capacity size read from the SRAM capacity register is in bytes, such as 0x0000 0C00 for 3KB, and 0x0000 0800 for 2KB.

## 24.6 Pin number register

The pin number register stores the chip pin number, its address is 0x0010 07E2 - 0x0010 07E3, a total of 2 bytes. For example, 0x0018 represents 24Pin, and 0x0014 represents 20Pin.

# 25 Revision history

Table 25-1 Document revision history

| Date | Revision | Changes |
|---|---|---|
| August 23, 2022 | Rev 1.0 | Initial release. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |