BF7006AMXX Series

BF7006AM64-LBTX

BF7006AM48-LBTX

BF7006AM28-TBBX

Datasheet

Rev1.0

2021-05-18

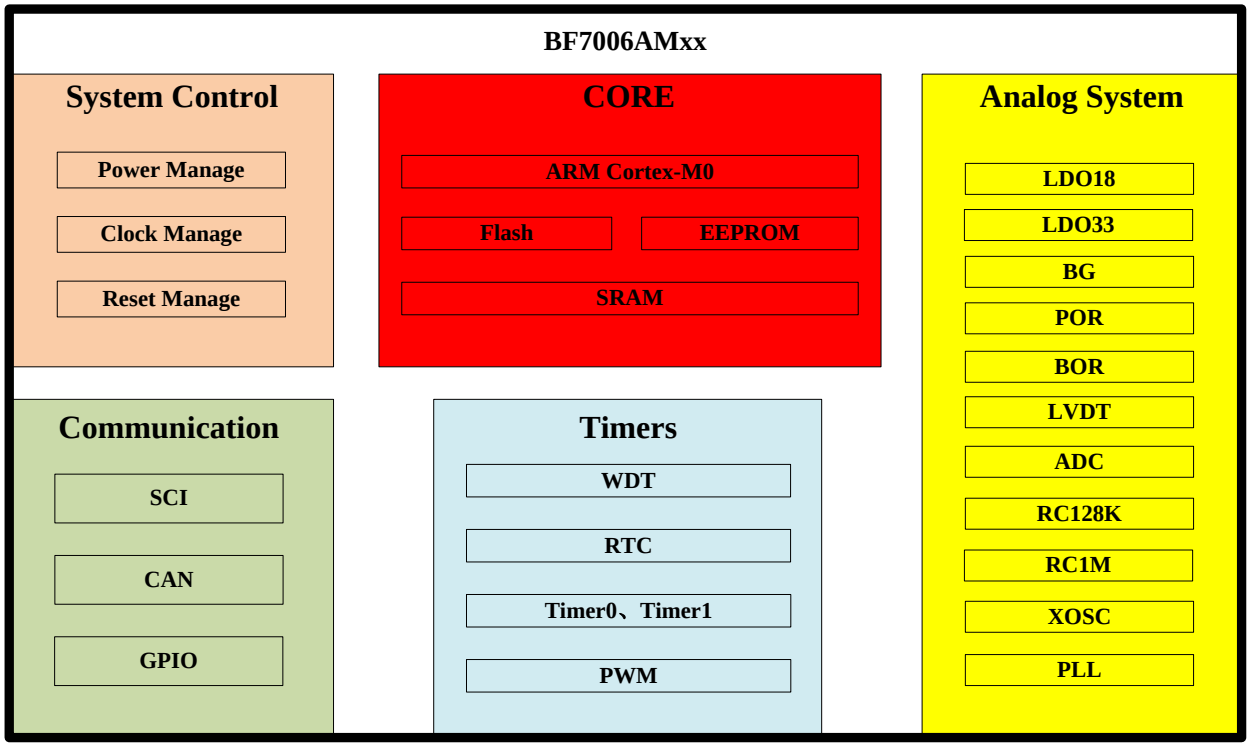
Content

1 Chip Overview

BF7006AMxx is a 32-bit general-purpose MCU that conform the automotive AECQ100 GRADE1 quality level. It is designed in according to the ISO26262 ASIL-B level standard.Besides, it integrates various communication modules such as CAN, LIN, UART, Multiple counter,Timer, and PWM functions, and contains a variety of general-purpose modules such as high-precision ADC, EEPROM instant data storage. The package form contains: LQFP64 , LQFP48 , TSSOP28. And these chips are suitable for a variety of automotive electronic body controls such as Electric window control, Electric seat control, Wipers, LED lamps, Door locks, etc.

1.1 System composition overview

BF7006AMXX chip system composition is as follows：



1.2 The main resources of chip

The main resources of the BF7006AMXX chip are as follows ,and the specific module functions are described in the chapters of each module：

Category	Features	LQFP64	LQFP48	TSSOP28
System	CPU	Cortex_M0	Cortex_M0	Cortex_M0
	Frequency	MAX:32MHz	MAX:32MHz	MAX:32MHz
	EX INT	MAX:24 channels	MAX:24 channels	MAX:14 channels
	IO	MAX:54 channels	MAX:40 channels	MAX:22 channels
	High current driver IO	MAX:6 channels	MAX:6 channels	MAX:6 channels
Clock Source	Internal	RC1MHz、RC128KHz、PLL	RC1MHz、RC128KHz、PLL	RC1MHz、RC128KHz、PLL
	External	XTAL 8MHz/16MHz	XTAL 8MHz/16MHz	XTAL 8MHz/16MHz
Memory	FLASH	MAX:96KB	MAX:96KB	MAX:96KB
	EEPROM	MAX:2KB	MAX:2KB	MAX:2KB
	SRAM	MAX:4KB	MAX:4KB	MAX:4KB
Communication	CAN	MAX:1	MAX:1	MAX:1
	SCI	MAX:2	MAX:2	MAX:2
Counter/ Timer	System Timer	1 of CPU internal	1 of CPU internal	1 of CPU internal
	RTC	MAX:1，32-bit RTC	MAX:1，32-bit RTC	MAX:1，32-bit RTC
	Timer	MAX:2，16-bit timers	MAX:2，16-bit timers	MAX:2，16-bit timers
	PWM	MAX:6 channels，16-bit pwm	MAX:6 channels，16-bit pwm	MAX:6 channels，16-bit pwm
Analog	ADC	Max:24 channels，Max:12-bit resolution,Max:10-bit accuracy，Max:1M sampling rate	Max:16 channels，Max:12-bit resolution,Max:10-bit accuracy，Max:1M sampling rate	Max:8 channels，Max:12-bit resolution,Max:10-bit accuracy，Max:1M sampling rate
Debug	Debug interface	ARM Serial Wire	ARM Serial Wire	ARM Serial Wire
	Number of hardware breakpoints	Max:2	Max:2	Max:2
Function Safety	Low voltage detection	3 levels	3 levels	3 levels
	Power-down reset	support	support	support
	ECC verification	6-bit ECC，FLASH、EEPROM	6-bit ECC，FLASH、EEPROM	6-bit ECC，FLASH、EEPROM
	Memory protection	FLASH、EEPROM	FLASH、EEPROM	FLASH、EEPROM
	CPU crash monitoring	Supports CPU crash reset	Supports CPU crash reset	Supports CPU crash reset
	Clock security	Supports external crystal oscillator failure reset	Supports external crystal oscillator failure reset	Supports external crystal oscillator failure reset
	Hardware WDT	Supports WDT reset	Supports WDT reset	Supports WDT reset
	Module power-on self-check	CAN	CAN	CAN
Main electrical	Operating Voltage	3.3V~5.5V	3.3V~5.5V	3.3V~5.5V
	Operating Temperature	-40℃~+125℃	-40℃~+125℃	-40℃~+125℃

Category	Features	LQFP64	LQFP48	TSSOP28
characte ristics	ESD feature	HBM：≥6kv CDM：≥0.5kv	HBM：≥6kv CDM：≥0.5kv	HBM：≥6kv CDM：≥0.5kv
	Latch up feature	100mA（25℃）	100mA（25℃）	100mA（25℃）

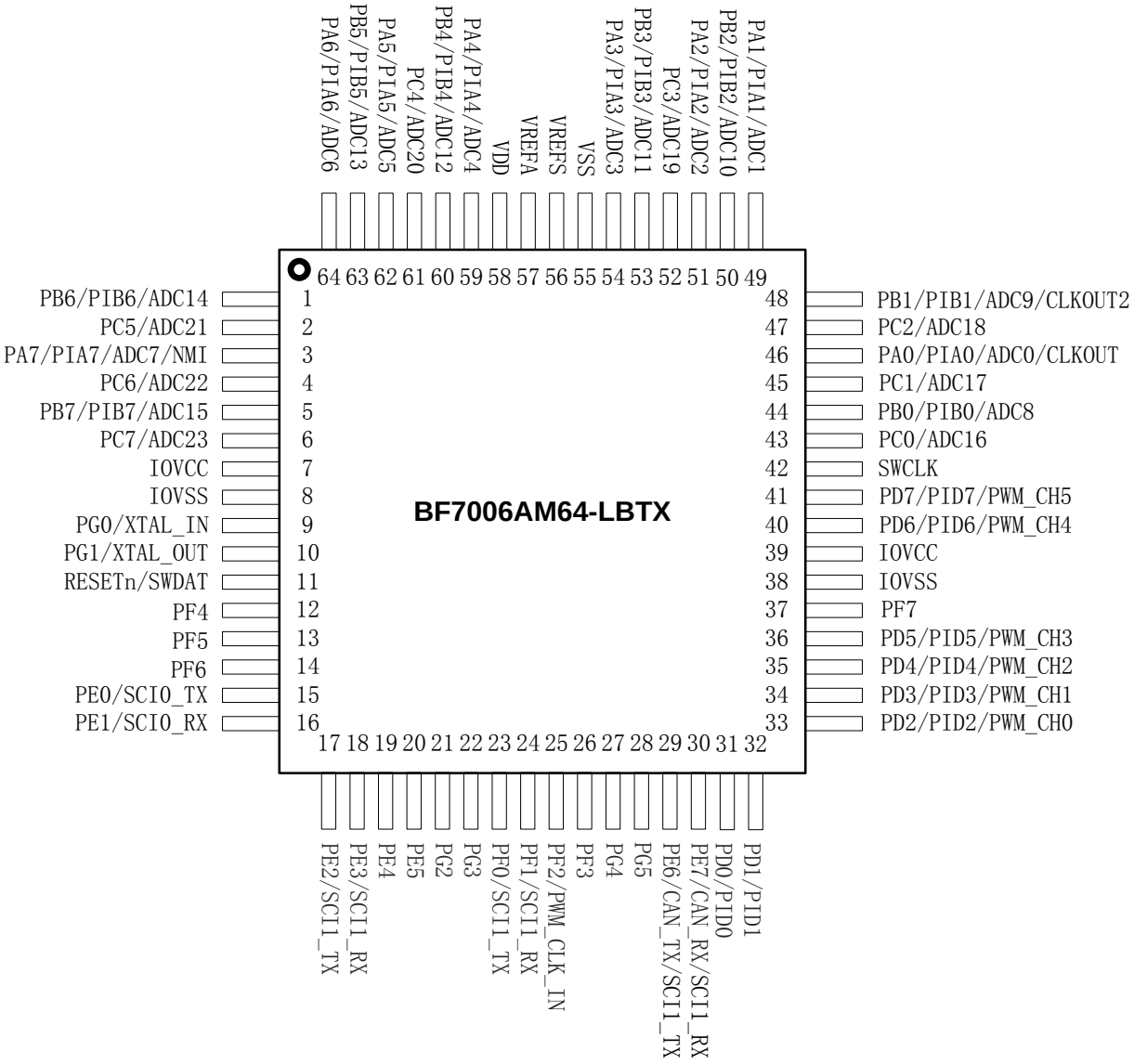
2 Pins and Connections

This section describes the package forms of the BF7006AMXX series and the schematic diagram of the pins. The detailed description of the function of all pins, and the function multiplexing of each pin.

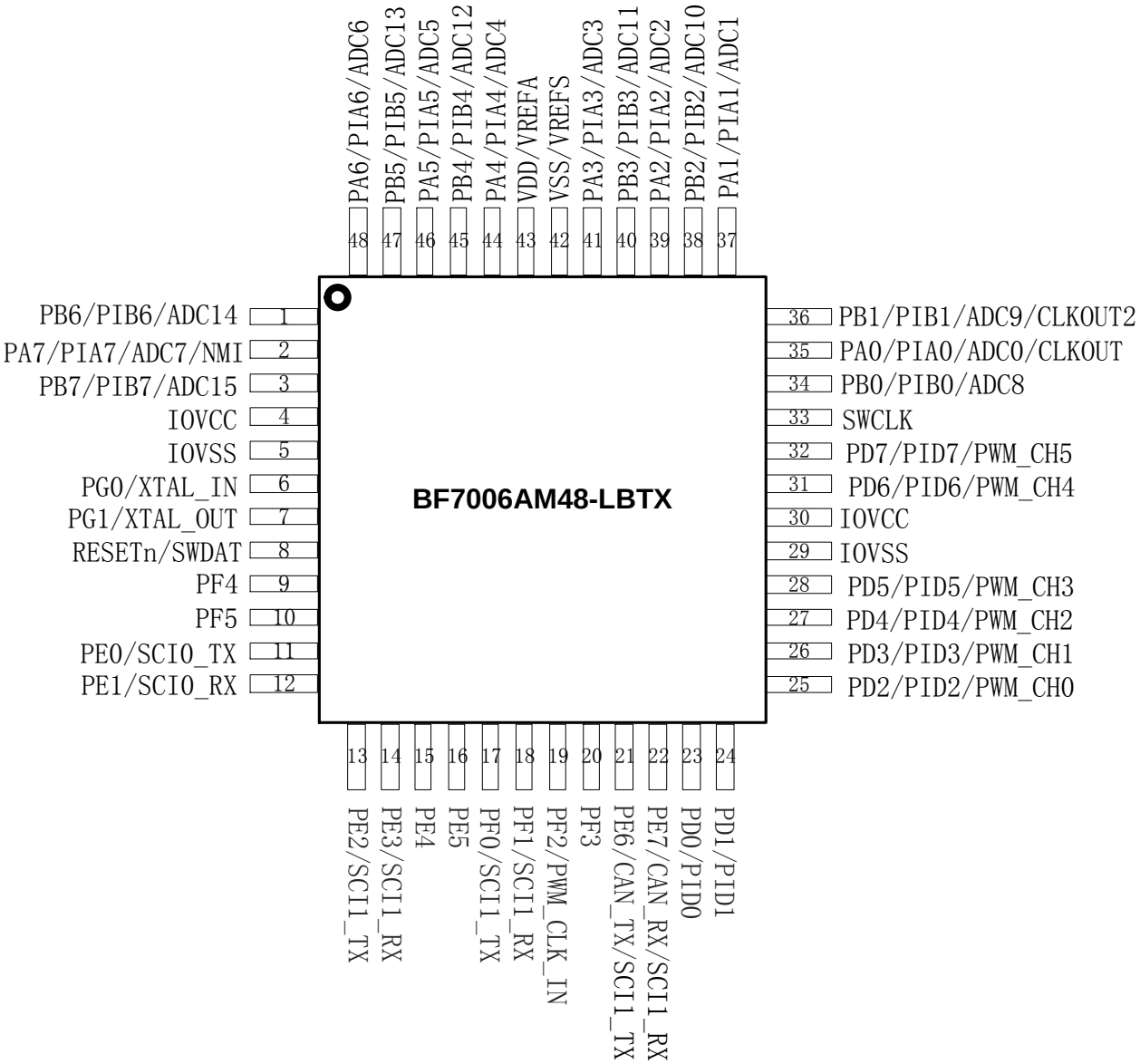
2.1 Device Pin Assignment

This section shows the pin assignments for BF7006AMXX Series MCUs in the available packages.

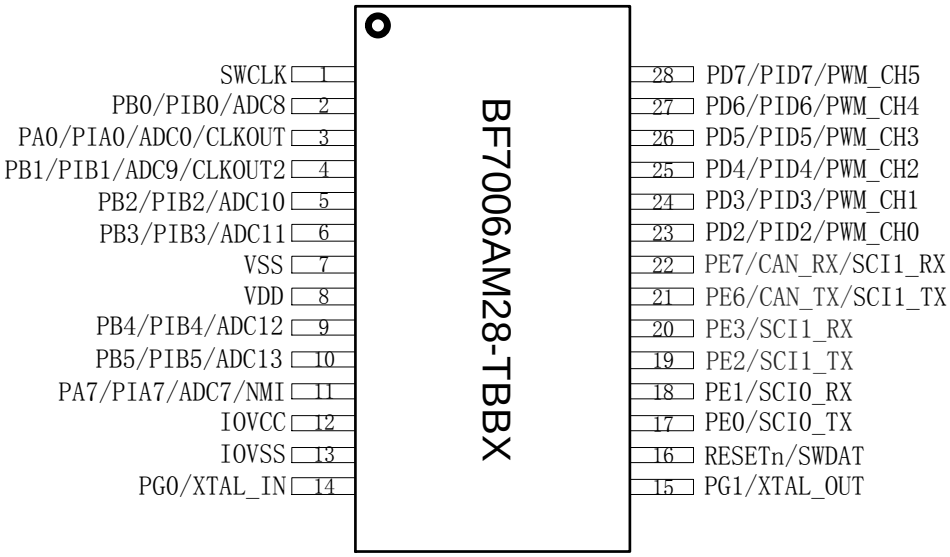
64-pin LQFP64:



48-pin LQFP48：



28-pin TSSOP28：



2.2 Pin Availability by Package Pin-Count

Lowest ← Priority → Highest								
LQFP64	LQFP48	TSSOP 28	First function	Second function	Third function	Forth function	Fifth function	Sixth function
1	1		PB6	PIB6	ADC14	----	----	----
2			PC5	---	ADC21	----	----	----
3	2	11	PA7	PIA7	ADC7	----	NMI	
4			PC6	----	ADC22	----	----	----
5	3		PB7	PIB7	ADC15	----	----	----
6			PC7	----	ADC23	----	----	----
7	4	12	----	----	----	----	----	IOVCC
8	5	13	----	----	----	----	----	IOVSS
9	6	14	PG0	----	----	----	XTAL_IN	----
10	7	15	PG1	----	----	----	XTAL_OUT	----
11	8	16	----	----	----	SWDAT	RESETn	----
12	9		PF4	----	----	----	----	----
13	10		PF5	----	----	----	----	----
14			PF6	----	----	----	----	----
15	11	17	PE0	----	----	----	SCI0_TX	----
16	12	18	PE1	----	----	----	SCI0_RX	----
17	13	19	PE2	----	----	----	SCI1_TX	----
18	14	20	PE3	----	----	----	SCI1_RX	----
19	15		PE4	----	----	----	----	----
20	16		PE5	----	----	----	----	----
21			PG2	----	----	----	----	----
22			PG3	----	----	----	----	----
23	17		PF0	----	----	----	SCI1_TX	----
24	18		PF1	----	----	----	SCI1_RX	----
25	19		PF2	----	----	----	PWM_CLK_IN	----
26	20		PF3	----	----	----	----	----
27			PG4	----	----	----	----	----
28			PG5	----	----	----	----	----
29	21	21	PE6	----	----	SCI1_TX	CAN_TX	----
30	22	22	PE7	----	----	SCI1_RX	CAN_RX	----
31	23		PD0	PID0	----	----	----	----
32	24		PD1	PID1	----	----	----	----
33	25	23	PD2	PID2	----	----	PWM_CH0	----
34	26	24	PD3	PID3	----	----	PWM_CH1	----
35	27	25	PD4	PID4	----	----	PWM_CH2	----
36	28	26	PD5	PID5	----	----	PWM_CH3	----

Lowest ← Priority → Highest								
LQFP64	LQFP48	TSSOP 28	First function	Second function	Third function	Forth function	Fifth function	Sixth function
37			PF7	----	----	----	----	----
38	29		----	----	----	----	----	IOVSS
39	30		----	----	----	----	----	IOVCC
40	31	27	PD6	PID6	----	----	PWM_CH4	----
41	32	28	PD7	PID7	----	----	PWM_CH5	----
42	33	1	----	----	----	----	SWCLK	----
43			PC0	----	ADC16	----	----	----
44	34	2	PB0	PIB0	ADC8	----	----	----
45			PC1	----	ADC17	----	----	----
46	35	3	PA0	PIA0	ADC0	----	CLKOUT	----
47			PC2	----	ADC18	----	----	----
48	36	4	PB1	PIB1	ADC9	----	CLKOUT2	----
49	37		PA1	PIA1	ADC1	----	----	----
50	38	5	PB2	PIB2	ADC10	----	----	----
51	39		PA2	PIA2	ADC2	----	----	----
52			PC3	----	ADC19	----	----	----
53	40	6	PB3	PIB3	ADC11	----	----	----
54	41		PA3	PIA3	ADC3	----	----	
55	42	7	----	----	----	----	----	VSS
56			----	----	----	----	----	VREFS
57			----	----	----	----	----	VREFA
58	43	8	----	----	----	----	----	VDD
59	44		PA4	PIA4	ADC4	----	----	----
60	45	9	PB4	PIB4	ADC12	----	----	----
61			PC4	----	ADC20	----	----	----
62	46		PA5	PIA5	ADC5	----	----	----
63	47	10	PB5	PIB5	ADC13	----	----	----
64	48		PA6	PIA6	ADC6	----	----	----

BF7006AMXX series supports ARM SW two-wire debugging protocol, and provides complete upper and lower computers, connectors and simulation tools. Among them, SWDAT (the debugging data port) and RESETN (the chip external reset function port) are multiplexed for the same pin, which defaults to the RESETN function. When the chip is connected to the debugging tool, the system will automatically recognize and switch the pin to the SWDAT function. When the chip needs to restore the RESETN function, it needs to be powered on again.

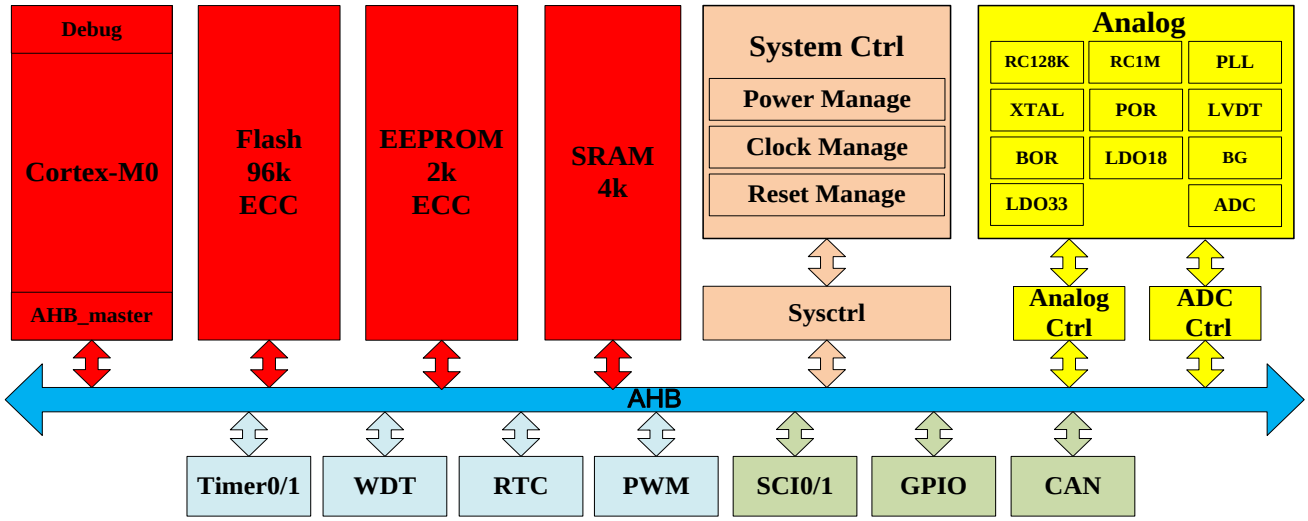
The SCI1 module is mapped to 3 connect, and only one can be selected at the same time.

2.3 Pin function description

Category	Signal	Signal function description
Power pin	IOVCC	IO power supply
	IOVSS	IO power ground
	VDD	Power supply
	VSS	Power ground
	VREFA	ADC reference power supply
	VREFS	ADC reference power ground
System pin	RESETn/SWDAT	Reset input/debug data port（default internal pull-up）
	NMI	External Advanced Interrupt
	SWCLK	Debug port（default internal pull-up）
	CLKOUT	Clock output, which can output external crystal clock divided by 8,internal 250KHz clock (1MHz divided), internal 32KHz (128KHz divided) clock and PLL clock divided by 16
	CLKOUT2	Clock output 2, which can output external crystal clock divided by 16 and internal 1MHz clock divided by 4
	XTAL_IN	Crystal input
	XTAL_OUT	Crystal output
Communication Interface	SCI0_RX	SCI0（UART / LIN）input
	SCI0_TX	SCI0（UART / LIN）output
	SCI1_RX	SCI1（UART / LIN）input
	SCI1_TX	SCI1（UART / LIN）output
	CAN_RX	CAN receive data
	CAN_TX	CAN transmit data
Counter	PWM_CH0	Advanced counter channel 0
	PWM _CH1	Advanced counter channel 1
	PWM _CH2	Advanced counter channel 2
	PWM _CH3	Advanced counter channel 3
	PWM _CH4	Advanced counter channel 4
	PWM _CH5	Advanced counter channel 5
	PWM _CLK_IN	Advanced counter external count clock
ADC	ADC0~ADC23	24 channels of ADC
GPIO	PAx~PGx	General input/output
	PIAx、PIBx、PIDx	External interrupt

3 System

BF7006AMXX is a 32-bit Cortex_M0 core MCU, its design architecture is based on AMBA AHB2.0 bus integration, which simplifies the system to a certain extent. The system mainly includes system functional components such as Cortex_M0 CPU, clock management unit, reset management unit, power management unit, interrupt management unit, functional safety management unit, etc.



3.1 CPU

The CPU is a standard 32-bit Cortex_M0, please refer to the ARM Cortex_M0 user manual for details, including the internal timer（System Tick）, which is not repeated here.

3.2 Address map

The start address and end address of each module device of BF7006AMXX are allocated as follows:

Device name		Start address	End address
1	FLASH main program memory	0x0000_0000	0x0001_7FFF
2	FLASH_NVR memory	0x0001_8000	0x0001_8FFF
3	SRAM memory	0x2000_0000	0x2000_0FFF
4	EEPROM flash	0x4000_0000	0x4000_07FF
5	EEPROM_NVR memory	0x4000_0800	0x4000_08FF
6	FLASH/EEPROM controller	0x5000_0000	0x5000_FFFF
7	System controller	0x5001_0000	0x5001_FFFF
8	SCI0/1	0x5004_0000	0x5004_FFFF
9	CAN	0x5005_0000	0x5005_FFFF
10	PWM	0x5006_0000	0x5006_FFFF

11	RTC	0x5007_0000	0x5007_FFFF
12	WDT	0x5008_0000	0x5008_FFFF
13	ADC controller	0x5009_0000	0x5009_FFFF
14	GPIO	0x500A_0000	0x500A_FFFF
15	TIMER0/1	0x500B_0000	0x500B_FFFF

3.3 System management

The BF7006AMXX system contains four parts: interrupt management, clock and power consumption management, reset management and safety management. Each part can be configured to realize various control functions and management of the system in order to provide different application needs.

3.3.1 Interrupt management

BF7006AMXX interrupt is based on Cortex_M0 interrupt control, and the corresponding function extension is added externally. The main features are:

1. Each module peripheral corresponds to an independent interrupt source；
2. A total of 24 external interrupts are available, of which 24 interrupts on A, B, and D ports are multiplexed with 1 interrupt source and distinguished by interrupt flag bits；
3. The system interrupt SYS_INTR has 9 interrupts, which are crystal startup initialization error, crystal detection error, FLASH_NVR check error, FLASH protection error operation, EEPROM protection error operation, FLASH ECC 1 bit error, FLASH ECC 2 bit error, EEPROM ECC 1 bit error, EEPROM ECC 2 bit error, each interrupt has a corresponding interrupt flag signal；
4. Support LVDT low voltage step-up and step-down interrupt；
5. Support NMI non-maskable interrupt；
6. Interrupt priority can be configured；
7. Interrupt can be enabled or masked；

3.3.1.1 Interrupt allocation

Cortex_M0 internal exceptions and interrupts are not repeated here, the system interrupt numbers are allocated as follows, to ensure that the program is written corresponding to the correct interrupt number：

CPU interrupt list	Allocation	Corresponding status register bit
IRQ0	---	
IRQ1	SYS_INTR system interrupt	SYS_INTEN_ETER SYS_INTEN_EOER SYS_INTEN_FTER SYS_INTEN_FOER SYS_INTEN_EPOT SYS_INTEN_FPOT SYS_INTEN_ADJ SYS_INTEN_XTALCHK SYS_INTEN_XTALINIT

IRQ2	---	
IRQ3	---	
IRQ4	LVDT interrupt	SYS_LVDT_IE_H SYS_LVDT_IE_L
IRQ5	PWM_CH0 interrupt	PWM_C0SC_IF
IRQ6	PWM_CH1 interrupt	PWM_C1SC_IF
IRQ7	PWM_CH2 interrupt	PWM_C2SC_IF
IRQ8	PWM_CH3 interrupt	PWM_C3SC_IF
IRQ9	PWM_CH4 interrupt	PWM_C4SC_IF
IRQ10	PWM_CH5 interrupt	PWM_C5SC_IF
IRQ11	PWM overflow interrupt	PWM_SC_TOF
IRQ12	---	
IRQ13	---	
IRQ14	---	
IRQ15	---	
IRQ16	SCI0 ERR interrupt	SCI_S1_RX_OVERFLOW_FLAG SCI_S1_NOSIE_ERR_FLAG
IRQ17	SCI0 RX interrupt	SCI_S1_RX_FULL_FLAG SCI_S2_BREAK_CHECK_FLAG SCI_S2_RX_EDGE_FLAG
IRQ18	SCI0 TX interrupt	SCI_S1_TX_EMPTY_FLAG SCI_S1_TX_COMP_FLAG
IRQ19	SCI1 ERR interrupt	SCI_S1_RX_OVERFLOW_FLAG SCI_S1_NOSIE_ERR_FLAG
IRQ20	SCI1 RX interrupt	SCI_S1_RX_FULL_FLAG SCI_S2_BREAK_CHECK_FLAG SCI_S2_RX_EDGE_FLAG
IRQ21	SCI1 TX interrupt	SCI_S1_TX_EMPTY_FLAG SCI_S1_TX_COMP_FLAG
IRQ22	GPIO external interrupt	GPIO_PTSC_IF GPIO_ INTST（GPIOA） GPIO_ INTST（GPIOB） GPIO_ INTST（GPIOD）
IRQ23	ADC interrupt	ADC_SC1_COCO
IRQ24	---	
IRQ25	RTC interrupt	RTC_SC_IF
IRQ26	CAN WAKEUP interrupt	CAN_CLRISR_WUPI CAN_IF_WUPI SYS_CAN_SPWKFLAG
IRQ27	CAN ERR interrupt	CAN_IF_BEI CAN_IF_ALI CAN_IF_EPI CAN_IF_DOI

		CAN_IF_EI CAN_CLRISR_BEI CAN_CLRISR_ALI CAN_CLRISR_EPI CAN_CLRISR_DOI CAN_CLRISR_EI
IRQ28	CAN RX interrupt	CAN_IF_RI
IRQ29	CAN TX interrupt	CAN_IF_TI CAN_CLRISR_TI
IRQ30	Timer0 interrupt	TIMER_CFG_IF
IRQ31	Timer1 interrupt	TIMER_CFG_IF

3.3.1.2 Interrupt enable and priority

The interrupt enable signal is divided into two types：

1. The processor CPU has its own enable register (ISER), which is used to control whether the interrupt signal sent to the CPU is valid;
2. The work mode enable register, based on the application configuration, sends external interrupt and internally generated interrupt to the CPU；
3. The external NMI interrupt signal can be sent to CPU according to the working status；
4. About the configuration of interrupt priority，eight 32-bit registers can be configured in CPU，and each interrupt has an 8-bit register to select the priority configuration value，the default interrupt priority is 0 to 31. Please refer to the ARM Cortex_M0 user manual for details, which is not repeated here.

3.3.1.3 Interrupt wakeup

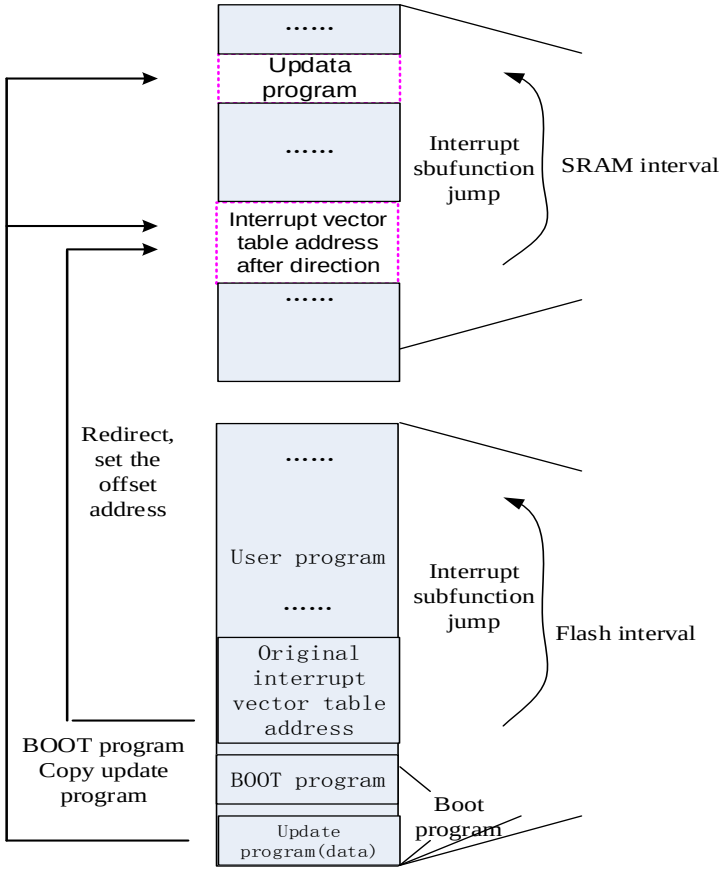
In the system low-power mode，turning off the clock and some other analog modules can save power consumption.At the sametime, the system can be waked up by LVDT interrupt、NMI interrupt、all external interrupt、can interrupt、SCI0/1 interrupt、RTC interrupt、WDT reset in low-power mode. All wakeup interrupts are asynchronous wakeup. According to the different low-power modes of the system and the choice of clock source, the wakeup process and the required time are different. For more details, please refer to the clock and power management chapter.

3.3.1.4 Interrupt vector redirection and online upgrade

BF7006AMXX supports the system vector redirection function. After updating the user program, the system can map the interrupt list address to the latest user program and execute the interrupt service function of the user program.

BF7006AMXX provides two online upgrade methods：

1. Run BOOT program loading program to update data，in the process of writing to the chip, CPU is automatically in a pause state, until the user program update data is successfully written in the FLASH, loop in this way；
2. The vector redirection function can also be used for online upgrade and providing application development BOOT.



During the online upgrade process, the user programs stored in FLASH except for the BOOT program, the user's function program and interrupt service program need to be updated and overwritten. Therefore, during the online upgrade process, if interrupt is needed (online upgrade by CAN and CAN interrupts are needed), the interrupt vector table and interrupt service routine need to be redirected. Since FLASH needs to be updated and cannot be read and written at the same time, it is generally directed(mapped) to SRAM. For the same reason, since the upgrade program cannot be read and executed by the CPU at the same time when the FLASH is being updated, the upgrade program also needs to be copied to RAM for execution. The BOOT program consists of a boot program and an upgrade program, and the upgrade program includes an interrupt vector table and an interrupt service function. Since the upgrade program cannot be executed in FLASH, the upgrade program in FLASH can be regarded as the data to be moved by the boot program. Therefore, it can be called an upgrade program only when it is moved to SRAM and executed.

Process guidance for BOOT program operation：

1. After the CPU is reset, BOOT program starts to run;
2. If there is no upgrade requirement, jump to the user function program. If there is an upgrade requirement, start to execute the BOOT program;
3. Update the user program according to different online upgrade methods;

The offset address of the interrupt vector table must be configured before the upgrade program starts the interrupt. Through two sets of 32-bit original address registers and two sets of 32-bit mapped address registers, the range of the interrupt vector table and the range of the mapped interrupt vector table are configured. When the CPU access address falls in the address range of the original interrupt vector table, it is mapped to the mapped address range through the mapping logic, so that the interrupt response can be performed correctly.

3.3.2 Reset management

BF7006AMXX uses a variety of reset sources to reset the chip system, and different reset sources enter different reset states, which are mainly divided into two types: power-on reset and functional global reset. After the power-on initialization reset, all units and registers of the chip are reset. And other functional resets can reset all functional units except for the key configuration and status registers of the system.

Features：

1. Support multiple reset sources;
2. Asynchronous reset and synchronous release to ensure reliability;
3. Each reset source has a corresponding reset state flag;
4. External reset supports reset signal filtering function;

3.3.2.1 Reset source

1. Power-on reset: After the system is powered on, the analog module sends a reset signal to reset all units;
2. Power-down reset: When the power-down reset module is enabled, the system is powered down. And when the voltage reaches the reset threshold, all units are reset;
3. Watchdog reset: A reset signal is generated after the watchdog timer overflows to reset the system;
4. Software reset: Reset the system by writing the software reset register in the CPU;
5. CPU crash reset: The CPU program is disordered to make the internal LOCKUP signal enable, and the system is reset;
6. External pin reset: reset the system by pulling down the external reset function pin, the reset can be filtered to prevent false reset caused by external interference;
7. Address overflow reset: The FLASH address exceeds the normal range when the CPU executes the addressing instruction, and the system resets when the address overflow occurs;
8. Oscillator clock start-up failure reset: external crystal oscillator start-up failure (this function is usually used when the chip uses an external crystal oscillator as the system clock);
9. Oscillator clock failure reset: When the system clock failure monitoring unit detects the failure of the external crystal oscillator, the system is reset (this function is usually used when the chip uses an external crystal oscillator as the system clock);

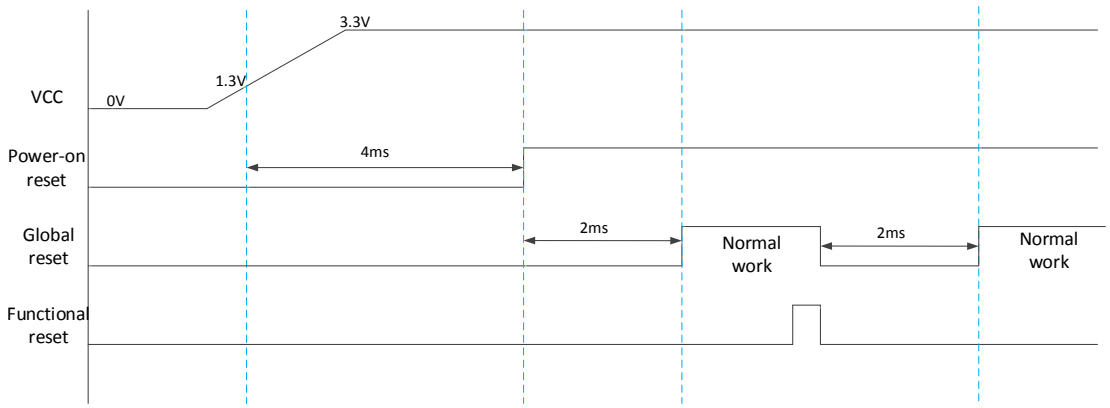
3.3.2.2 Reset filter

For the external reset function, in order to avoid interference signals from causing the chip system to enter the reset state, the system supports the filtering function of the external reset signal, and the filtering clock is RC32KHz.

When the external port reset function is enable and the external reset filter function is required, the RC32K clock source needs to be ensured in working state and the register SYS_EXFLT=1. The filter time is a valid RC_32KHz clock, and the filter time is 30us~60us. If the RC32KHz clock source is not turned on and the configuration filter function is enabled, the external reset function will be invalid and the reset source cannot be sent to the system, this condition needs to pay attention.

For other resets, an asynchronous reset is designed inside the chip, and the registers are released synchronously to ensure that the reset signal is reliable and effective.

3.3.2.3 Reset sequence



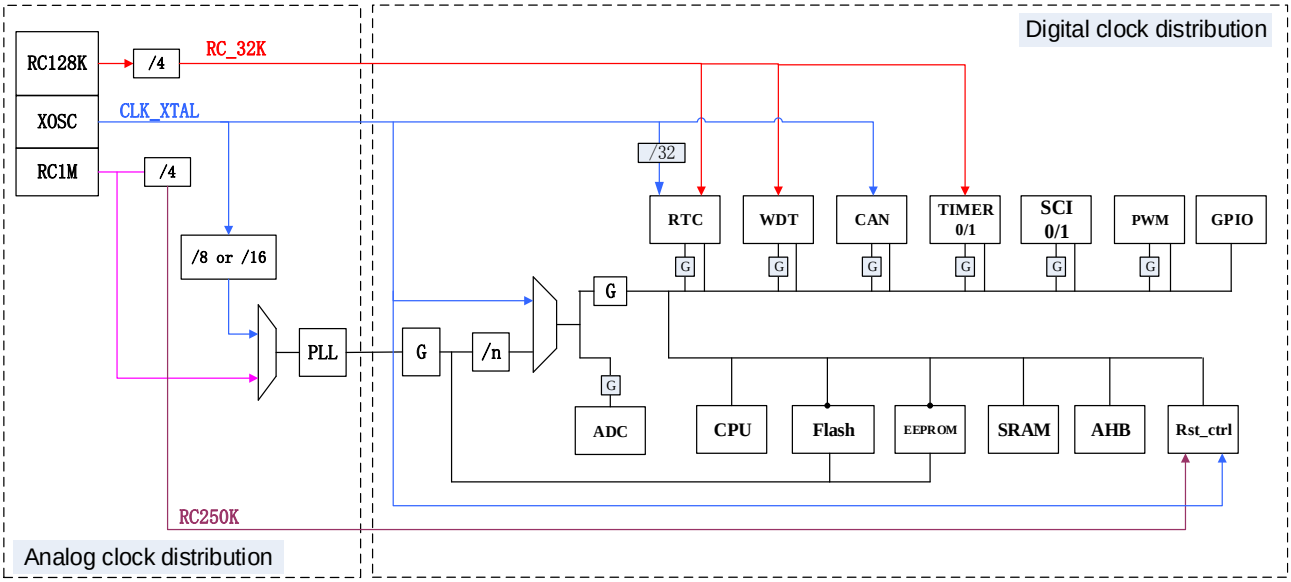
1. The chip is powered on, VCC reaches 1.3V, and the power-on reset signal is valid for 4ms;
2. After that, the system starts to perform a global reset. After 2ms, the chip global reset is completed and the system starts to work normally;
3. When other functions (power failure, watchdog, address overflow, soft reset, CPU crash, crystal oscillator start-up failure, crystal oscillator failure, external pin) reset during operation, a valid system reset occurs and a global reset occurs. After 2ms, the chip global reset is completed and the system starts to work normally.

Note that the VCC power-on time from 0V to 3.3V needs to be completed within 4ms.

3.3.3 Clock and power consumption management

BF7006AMXX supports multiple clock sources, and the system working clock supports multiple frequency divisions. Each peripheral module supports separate control of the clock for better control power consumption and improving system efficiency. At the same time, the system supports two low-power modes to reduce system operating current as much as possible, and supports multiple functional wake-up mechanisms.

3.3.3.1 Clock architecture



Clock structure description:

1. The system contains 3 analog clock sources: RC128KHz, RC1MHz and external crystal oscillator. RC128KHz is divided by 4 on the analog side to form an RC32KHz clock source and sent to the digital system; among them, RC1MHz is divided by 4 on the analog side to form a RC250KHz clock source for using in digital systems; And external crystal oscillators typically support 8MHz and 16MHz;
2. The analog PLL phase-locked loop access clock source supports internal RC1MHz and external crystal oscillator clock, but the access clock only supports 1MHz. Therefore, when the PLL access clock source selects the external crystal oscillator, it must be divided by 8 or 16 to 1MHZ according to the different external crystal frequency.
3. The system clock is the internal PLL output clock; the maximum PLL output clock frequency is 32MHz, and SYS_CLK_SEL can be configured for frequency division according to different application requirements; when the external crystal oscillator clock is selected as the system clock or the crystal oscillator clock is used as the PLL access clock source. When waking up from power consumption, due to the influence of the crystal oscillator start-up time, the waiting time of the system will be much longer than the case of selecting the internal RC1MHz as the PLL access clock source and as the system clock;
4. The CAN module working clock supports external crystal oscillator and internal system clock. Through program configuration, it should be noted that when the crystal oscillator is selected as the working clock, there is a cross-time domain problem with the system running clock. The SYS_CAN_DOMAIN register needs to be properly configured; When the module clock is not supplied to the CAN module, any register of the CAN module cannot be accessed, and the access action may cause the system to fault;
5. The WDT count clock can be selected by sofeware including RC32KHz and its 32 frequency division;
6. RTC count clock can be selected by software including RC32KHz, its 32 frequency division and 32 frequency division of external crystal oscillator;
7. All peripherals and system clocks can be gated to reduce power consumption;

3.3.3.2 Clock selection and switch

By default, the system selects the internal RC1MHz clock as the access clock source of the PLL, and the PLL output frequency divided by 2 (16MHz) is used as the system clock. The external crystal oscillator is turned off by default. Therefore, the working clock before system power-on initialization is based on the internal RC1MHz clock.

If the system requires a high-precision clock source in a certain application, you can configure to turn on the crystal clock and switch the system clock to an external crystal clock source in the initialization program, or switch the PLL access clock source to an external crystal clock. Note that since the crystal oscillator needs a certain stabilization time to turn on and successfully start-up, it is necessary to ensure that the crystal oscillator clock is in a stable state before the software configuration is switched, otherwise the system will fail.

Similarly, if the external crystal oscillator clock is turned off to reduce power consumption when the system enters the low-power sleep mode, and the system working clock is based on the external crystal oscillator clock, the wake-up time will include the crystal oscillator start-up stabilization time and have long wake-up time (please refer to the electrical characteristics of crystal oscillator stabilization time for details).

3.3.3.3 Working modes

BF7006AMXX has 4 working modes, which can be selected according to different situations:

1. Active mode：The module keeps working normally, and the function of each module is controlled by the software configuration;
2. Sleeping mode：The CPU and system modules stop working, and the functions of other peripheral modules are controlled by software configuration and support interrupt wake-up;
3. Sleepdeep mode: In this mode, all digital modules stop working, and some analog modules work selectively (configured according to specific wake-up conditions), and support condition-triggered wake-up;
4. Debug mode：Used for testing, debugging, and programming. Enter directly through the external ARM SW debug interface;

Different working modes control the clock source:

Work mode	Conditions for accessing mode	State of clock source	
Active	——	XTAL	Configured by software
		RC128K	Configured by software
		RC1MHz	Work
		PLL	work
Sleeping	Configure CPU sleep mode as normal sleep and execute WFI instruction	XTAL	Configured by software
		RC128K	Configured by software
		RC1MHz	work
		PLL	work

Sleepdeep	Configure CPU sleep mode as deep sleep and execute WFI instruction	XTAL	Configured by software
		RC128K	Configured by software
		RC1MHz	close
		PLL	close
Debug	External interface input	XTAL	Configured by software
		RC128K	Configured by software
		RC1MHz	work
		PLL	work

Different working modes control the various modules of the system:

Module	Active	Sleeping	Sleepdeep
CPU	Work	PD	PD
AHB	Work	PD	PD
FLASH	Work	PD	PD
EEPROM	Work	PD	PD
SRAM	Work	PD	PD
SYS_CTRL	Work	PD	PD
TIMER0/1	Software control	Software control	PD
PWM	Software control	Software control	PD
RTC	Software control	Software control	Software control
WDT	Software control	Software control	Software control
CAN	Software control	Software control	PD
SCI0/1	Software control	Software control	PD
GPIO	Software control	Software control	PD
ADC	Software control	Software control	Software control
LVDT	Software control	Software control	Software control
BOR	Software control	Software control	Software control

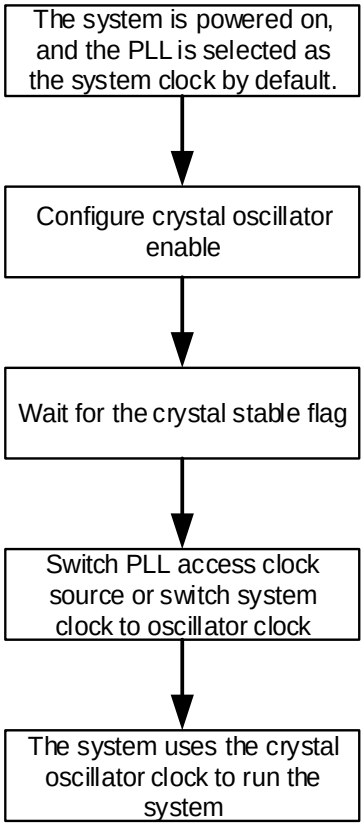
3.3.3.4 Low-power wakeup

1. Idle mode wake-up: Any peripheral interrupt, watchdog reset and external reset can wake up the system. Interrupt wake-up will make the system continue to run from the last sleep point, reset wake-up will reset the system from idle mode and restart operation;
2. Sleep mode wake-up: External interrupts (including NMI), RTC interrupts, LVDT low-voltage interrupts, CAN interrupts, SCI interrupts, watchdog resets and external resets can wake up the system. Interrupt wake-up will make the system continue to run from the last sleep point. Reset wake-up makes the system reset from idle mode and restart operation;
3. CAN wake-up in sleep mode: support external CAN communication dominant level wake-up, and can choose whether to filter (for details, see the chapter waking up the system through CAN);
4. SCI wake-up in sleep mode: support external SCI communication pin pulse wake-up (see the SCI interrupt wake-up chapter for details);
5. ADC wake-up in sleep mode: see the ADC sleep wake-up chapter for details;

Note: Since the sleep mode system shuts down most of the analog modules and all digital modules, when waking up, the system first needs to ensure that the power supply and clock are stable before waking up the CPU and digital system normally. Therefore, it takes a long time to wake up, including the power supply stabilization time 60us, PLL stabilization time 200us~500us (software configurable) and external crystal oscillator stabilization time (if the system uses crystal oscillator as the system working clock, 4096 or 8192 software-selectable crystal oscillator stabilization period and crystal oscillator start-up stabilization time are required).

3.3.3.5 Oscillator clock initialization procedure

BF7006AMXX supports the external oscillator clock directly as the system running clock, or the oscillator clock as the PLL access clock source, and then uses the PLL clock as the system clock, which has reached higher application accuracy requirements.When the external crystal oscillator is directly used as the system operating clock, the maximum frequency does not exceed 16MHz; when the oscillator clock is used as the PLL access clock source, and then the PLL clock is used as the system clock, the maximum frequency of the system does not exceed 32MHz. Before the system initialization program switches the crystal oscillator clock, ensure that the crystal oscillator starts to oscillate stably.



3.3.4 Functional safety management

In order to better make the system meet the ASILB safety level, BF7006AMXX supports a variety of functional safety management mechanisms to realize the safety protection of logic functions from the hardware level.

3.3.4.1 External oscillator failure monitoring

There are two situations for monitoring the failure of the external crystal oscillator:

1. Crystal oscillator initialization start-up failure monitoring:

   After this function is enabled, when the crystal oscillator initialization is started, the internal RC1MHz 4 divider clock RC250KHz will automatically time for 100ms. After 100ms, the crystal oscillator startup success flag will be monitored. If the correct crystal oscillator startup success flag is monitored, the crystal oscillator has been successfully started and the system can continue to run; if the correct crystal oscillator startup success flag is not monitored, the corresponding error interrupt will be issued and the system will be reset according to the register SYS_XTAL_INIT configuration (reset is typically used when the crystal oscillator is used as the system clock source);

2. Failure monitoring when crystal oscillator is working;

   When the crystal oscillator is working normally, there is a risk of failure. When this function is enabled, the system uses the internal RC1MHz divide-by-4 clock RC250KHz to monitor the crystal oscillator clock, which means that the crystal oscillator clock (fast clock) is used to count 2 cycles of RC250KHz (slow clock). Considering the deviation characteristics of the clock by the environment, the monitoring function sets a reasonable count value within a certain interval (register SYS_XTAL_CHKCNT configuration). When the count value is within this interval, the crystal oscillator

is considered normal; when the count value exceeds the counting interval, the crystal oscillator is considered to has an error and an error interrupt is issued; when this error occurs 8 times or more, whether to reset the system according to the SYS_XTAL_CHK configuration (reset is typically used when the crystal oscillator is used as the system clock source).

When the crystal oscillator module is not enabled, the monitoring function will be unavailable. The application should be aware that this function will not work normally when the crystal oscillator is turned off or the system enters the sleep mode.

3.3.4.2 Low-Voltage and power down monitoring

BF7006AMXX supports system power supply monitoring, including low-voltage monitoring LVDT and power-down monitoring BOR. The system provides three low-voltage points of 4.5V, 4.0V and 3.5V, and a power-down reset point of 3.1V. When the system power supply voltage is less than the above value, in order to ensure the safety of the system, the system will issue a low-voltage interrupt or directly reset the system.

LVDT provides two interrupt flag bits, namely the low-voltage interrupt flag and the recovery interrupt flag, which can better indicate the current voltage status. However, only low-voltage interrupts can wake the system from sleep mode.

3.3.4.3 Memory protection and check

The FLASH and EEPROM of BF7006AMXX support erasing and writing protection to prevent the program, especially the BOOT program, from erasing by mistake. The size of the protection space is controlled by the configuration of the corresponding registers FLASH_LOCK_SIZE and EEPROM_LOCK_SIZE, and the configuration of this register requires EFLASH_UNLOCK to be unlocked before the protection is unlocked.

When the chip is connected to the debugging function, the protection function is automatically unlocked.

At the same time, FLASH and EEPROM provide a 32-bit + 6-bit ECC check mechanism, which can correct 1-bit or 2-bit error detection, which improves the reliability of the memory at the system level.

FLASH address protection：

FLASH_LOCK_SIZE	Address protection range	Size (Kbytes)
0x00	Flash_NVR	0
0x01	Flash_NVR + 2K	2
0x02	Flash_NVR + 4K	4
0x03	Flash_NVR + 6K	6
……	……	
0x30	Flash_NVR + 96K	96

EEPROM address protection:

EEPROM_LOCK_SIZE	Address protection range	Size (bytes)
0x00	unprotected	0
0x01	EEP_NVR + 64Bytes	64
0x02	EEP _NVR + 128Bytes	128
0x03	EEP _NVR + 196Bytes	196
.....	.....	
0x20	EEP _NVR + 2048Bytes	2048

3.3.4.4 Functional safety reset

BF7006AMXX supports multiple functional safety resets, including program address access overflow reset, CPU locked protection reset, and hardware watchdog reset.

1. Program address access overflow reset: When the program instruction access exceeds the limited address range, the system will consider this operation illegal and will issue an address overflow reset to prevent the program from running away;
2. CPU locked protection reset: reset based on Cortex_M0's own functional signal LOCKUP design;
3. Hardware watchdog reset: When the program does not clear the watchdog timer within the specified time, the system is considered to be in an infinite loop or paralyzed state. At this time, the hardware watchdog issues a system reset directly; the system uses an independent watchdog count clock to avoid system disorder to affect the watchdog operation.

3.3.4.5 ADC temperature detection channel

BF7006AMXX contains 1 internal channel, which supports the detection of chip temperature changes. The ADC conversion data fed back by this channel is different at different temperatures Users can evaluate the current chip temperature based on the quasi-linear relationship between this data and temperature to ensure system safety. See the ADC chapter for specific configuration.

3.3.5 Register map

System controller module address range：0x5001_0000~0x5001_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	SYS_PTSEL	R/W	Port function seclection register	0x00
0x04	SYS_XTAL_CTRL	R/W	Xtal control register	0x00
0x08	SYS_PLL_SOURCE_SEL	R/W	Pll source clock select register	0xACB3
0x0C	SYS_CLK_SEL	R/W	System clock select register	0x3CA2

0x10	SYS_VECTOR_OFFSET	R/W	Vector redirection offset register	0x0000_0000
0x14	SYS_CLK_PD	R/W	Analog work enable register	0x06
0x18	SYS_CAN_DOMAIN	R/W	CAN time domain counting time register	0x04
0x1C	SYS_CLK_OUT	R/W	Analog clock output register	0x0000
0x20	SYS_IO_LOCK	R/W	IO function lock register	0x01
0x24	SYS_LVDT_CRL	R/W	Voltage monitoring register	0x07
0x28	SYS_EXRST	R/W	External reset function enable register	0x01
0x2C	SYS_EXFLT	R/W	External reset filter register	0x01
0x34	SYS_PERH_HALT	R/W	Debug count stop control register	0x01
0x38	SYS_PLL_T	R/W	PLL stabilization time set register	0x00
0x40	SYS_CAN_CLKSEL	R/W	CAN clock select register	0x00
0x44	SYS_CAN_RST	R/W	CAN module reset register	0x01
0x48	SYS_CAN_SPWKFLAG	R/W	System wakeup CAN flag register	0x00
0x100	SYS_RSTSTAT	R/W	Reset status register	0x0001
0x104	SYS_INTEN	R/W	System interrupt enable register	0x0000
0x108	SYS_INTFLG	R/W	System interrupt status register	0x0000
0x10C	SYS_XTAL_CHK	R/W	Xtal work detection failure register	0x00
0x110	SYS_XTAL_CHKCNT	R/W	Xtal work failure monitoring interval set register	0x0002_0008
0x114	SYS_XTAL_INIT	R/W	Xtal initialization failure monitoring register	0x00
0x118	SYS_LVDT_IE	R/W	lVDT interrupt enable register	0x00
0x11C	SYS_LVDT_IF	R/W	lVDT interrupt flag register	0x00

3.3.6 Register description

3.3.6.1 SYS_PTSEL register

Address	Bit	Name	R/W	Reset value	Description
0x00	4:0	Reserved	——	——	Reserved
	1:0	SYS_PTSEL	R/W	0x0	SCI1 port function relocation selection 0x3: Select PF0 and PF1 as function port of SCI1 0x2: Select PE6 and PE7 as function port of SCI1 0x1: Select PE2 and PE3 as function port of SCI1 0x0: Select PF0 and PF1 as function port of SCI1

3.3.6.2 SYS_XTAL_CTRL register

Address	Bit	Name	R/W	Reset	Description

				value	
0x04	7	SYS_XTAL_CTRL_PD	RO	0x0	Crystal oscillator working status flag 0x1 ：The crystal oscillator is in normal working state 0x0 ：The crystal oscillator is off Because clock switching and crystal oscillator shutdown require multiple cycles across the time domain to complete the action, in the application, when the system configuration SYS_ANACLK_SEL or SYS_CLK_SEL switches from the external crystal oscillator clock source back to the internal clock source, configure this bit. After turning off the external crystal clock, the instruction to enter sleep cannot be executed immediately. This signal needs to be checked, and the operation of entering sleep can be performed when it is 0, otherwise the crystal cannot closed shut down normally in sleep mode.
	6:4	SYS_XTAL_CTRL_CNT	R/W	0x0	Crystal initialization stable period selection 0x0：4096 crystal oscillator periods Other：8192 crystal oscillator periods
	3	SYS_XTAL_CTRL_INIT	RO	0x0	Crystal initialization stable flag 0x1：Stable vibration 0x0：Vibration is not stable yet When switching clocks, you must ensure that the crystal oscillator starts to oscillate stably
	2	Reserved	——	——	Reserved
	1	SYS_XTAL_CTRL_SLEEPPD	R/W	0x0	Configure whether the crystal oscillator is turned off in sleep mode 0x1：Closed 0x0：Keep the previous configuration
	0	SYS_XTAL_CTRL_EN	R/W	0x0	Crystal oscillator enable 0x1：Enable 0x0：Disable

3.3.6.3 SYS_PLL_SOURCE_SEL register

Address	Bit	Name	R/W	Reset value	Description
0x08	15:0	SYS_PLL_SOURCE_SEL	R/W	0xACB3	PLL access clock source selection and configuration 0xACB3：PLL access clock source is RC1MHz 0xCCD2：The PLL access clock source is an external crystal oscillator, which divides by 16

					0xBD5A: The PLL access clock source is an external crystal oscillator, which divides by 12 0x8D9C: The PLL access clock source is an external crystal oscillator, which divides by 8 The PLL access clock source must be 1MHz. When the access clock source is an external crystal oscillator, you should pay attention to the correct frequency division.

3.3.6.4 SYS_CLK_SEL register

Address	Bit	Name	R/W	Reset value	Description
0x0C	15:0	SYS_CLK_SEL	R/W	0x3CA2	System clock source selection and configuration 0x9ABD: System clock source selects PLL, PLL frequency is 32MHz 0x3CA2: System clock source selects PLL, PLL frequency is 16MHz 0xE78C: System clock source selects PLL, PLL frequency is 8MHz 0x7C6B: System clock source selects external crystal oscillator

3.3.6.5 SYS_VECTOR_OFFSET register

Address	Bit	Name	R/W	Reset value	Description
0x10	31:0	SYS_VECTOR_OFFSET	R/W	0x0000_0000	Vector redirection offset address

3.3.6.6 SYS_CLK_PD register

Address	Bit	Name	R/W	Reset value	Description
0x14	7:3	Reserved	——	——	Reserved
	2	SYS_CLK_PDLVDT	R/W	0x1	Configure whether to close the LVDT 0x1: Close

					0x0: Don't close
	1	SYS_CLK_PDBOR	R/W	0x1	Configure whether to close BOR 0x1: Close 0x0: Not close
	0	SYS_CLK_PD128K	R/W	0x0	Configure whether to close RC128K 0x1: Close 0x0: Not close

3.3.6.7 SYS_CAN_DOMAIN register

Address	Bit	Name	R/W	Reset value	Description
0x18	7:3	Reserved	——	——	Reserved
	2:0	SYS_CAN_DOMAIN	R/W	0x4	CAN cross-time domain counting time configuration 0x4: Suitable for system clock 32MHz, external crystal oscillator 8MHz 0x2: Suitable for system clock 32MHz, external crystal oscillator 16MHz, or suitable for system clock 16MHz, external crystal oscillator 8MHz 0x1: Suitable for system clock 16MHz, external crystal oscillator 16MHz,or suitable for system clock 8MHz, external crystal oscillator 8MHz, Suitable for system clock 8MHz, external crystal oscillator 16MHz When the clock configuration is completed, select the above register value according to the specific configuration.

3.3.6.8 SYS_CLK_OUT register

Address	位	名称	R/W	Reset value	Description
0x1C	15:6	SYS_CLK_OUTDUMMY	R/W	0x000	Must be configured 0
	5	SYS_CLK_OUT_CLK2XTAL	R/W	0x0	CLKOUT2 port output oscillator clock 0x1: Output oscillator clock divided by 16 0x0: No output, port CLKOUT2 function disabled
	4	SYS_CLK_OUT_CLK2RC250	R/W	0x0	CLKOUT2 port outputs RC250KHz clock

					0x1: Output RC250KHz clock 0x0: No output, port CLKOUT2 function disabled
	3	SYS_CLK_OUTDUMMY2	R/W	0x0	Must be configured 0
	2	SYS_CLK_OUT_CLKEN	R/W	0x0	CLKOUT output enable 0x1: Enable 0x0: Disable, port CLKOUT function is disabled
	1:0	SYS_CLK_OUT_CLKC	R/W	0x0	CLKOUT output clock selection 0x3: External crystal oscillator divided by 8 0x2: PLL clock divided by 16 0x1: Output RC32KHz 0x0: Output RC250KHz

3.3.6.9 SYS_IO_LOCK register

Address	Bit	Name	R/W	Reset value	Description
0x20	7:1	Reserved	——	——	Reserved
	0	SYS_IO_LOCK	R/W	0x1	IO port function lock 0x1: No effect 0x0: Lock IO function The output data and direction control before the configuration is locked; when the input direction is before the lock, the input signal of the external port has no effect.

3.3.6.10 SYS_LVDT_CRL register

Address	Bit	Name	R/W	Reset value	Description
0x24	7:3	Reserved	——	——	Reserved
	2	SYS_LVDT_CRL_BORDELAY	R/W	0x1	BOR power-down delay selection 0x1: Delay 100us 0x0: Not delay
	1:0	SYS_LVDT_CRL_C	R/W	0x3	LVDT gear selection 0x3: 4.5V gear

					0x2: 4.5V gear 0x1: 4.0V gear 0x0: 3.5V gear

3.3.6.11 SYS_EXRST register

Address	Bit	Name	R/W	Reset value	Description
0x28	7:1	Reserved	——	——	Reserved
	0	SYS_EXRST	R/W	0x1	External reset port enable 0x1: Enable 0x0: Dsiable

3.3.6.12 SYS_EXFLT register

Address	Bit	Name	R/W	Reset value	Description
0x2C	7:1	Reserved	——	——	Reserved
	0	SYS_EXFLT	R/W	0x1	External reset port filter enable 0x1: Enable, filter the 1~2 RC32K (128K clock divided by 4) clock time 0x0: Disbale When the chip turns off the RC128KHz clock, the filtering function cannot be realized. At this time, the reset signal cannot be sent to the system after filtering is enabled.

3.3.6.13 SYS_PERH_HALT register

Address	Bit	Name	R/W	Reset value	Description
0x34	7:1	Reserved	——	——	Reserved
	0	SYS_PERH_HALT	R/W	0x1	Whether to pause the watchdog WDT and RTC counter during debugging 0x1: Pause 0x0: Not pause

3.3.6.14 SYS_PLL_T register

Address	Bit	Name	R/W	Reset value	Description
0x38	7:2	Reserved	——	——	Reserved
	1:0	SYS_PLL_T	R/W	0x0	PLL stabilization time selection 0x3: 200us 0x2: 300us 0x1: 400us 0x0: 500us When the system exits the idle mode or sleep mode, the PLL needs stabilization time due to the power re-supply. This register provides the selection of the PLL stabilization time.

3.3.6.15 SYS_CAN_CLKSEL register

Address	Bit	Name	R/W	Reset value	Description
0x40	7:1	Reserved	——	——	Reserved
	0	SYS_CAN_CLKSEL	R/W	0x0	CAN module clock selection 0x1: System clock 0x0: External oscillator clock (recommended)

3.3.6.16 SYS_CAN_RST register

Address	Bit	Name	R/W	Reset value	Description
0x44	7:1	Reserved	——	——	Reserved
	0	SYS_CAN_RST	R/W	0x1	CAN module reset control 0x1: No operation 0x0: Reset CAN module This bit will not be reset after the CAN module is reset. It needs to be manually configured to 1 again, otherwise the CAN module is always in the reset state.

3.3.6.17 SYS_CAN_SPWKFLAG register

Address	Bit	Name	R/W	Reset value	Description
0x48	7:1	Reserved	——	——	Reserved
	0	SYS_CAN_SPWKFLAG	R/W	0x0	Flag whether the system is awakened by CAN in sleep mode 0x1: CAN causes the system to wakeup from sleep 0x0: No CAN system wakeup occurs Write this bit to clear the flag.

3.3.6.18 SYS_RSTSTAT register

Address	Bit	Name	R/W	Reset value	Description
0x100	16:9	reserved	——	——	reserved
	8	SYS_RSTSTAT_XTAL	R/W	0x0	Crystal oscillator failure reset flag 0x1: Reset 0x0: Not reset When the reset enable is turned off, this flag will also be set when the crystal oscillator fails. Write 1 to clear the flag.
	7	SYS_RSTSTAT_XTALINI	R/W	0x0	Crystal oscillator initialization failure reset flag 0x1: Reset 0x0: Not reset When the reset enable is turned off, this flag will also be set when the crystal oscillator fails to initialize. Write 1 to clear the flag.
	6	SYS_RSTSTAT_ADDR	R/W	0x0	Address overflow reset flag 0x1: Reset 0x0: Not reset Write 1 to clear the flag
	5	SYS_RSTSTAT_EXT	R/W	0x0	External reset flag 0x1: Reset 0x0: Not reset Write 1 to clear the flag
	4	SYS_RSTSTAT_LOCKUP	R/W	0x0	CPU deadlock reset flag 0x1: Reset 0x0: Not reset

					Write 1 to clear the flag
	3	SYS_RSTSTAT_SOFT	R/W	0x0	Software request reset flag 0x1: Reset 0x0: Not reset Write 1 to clear the flag
	2	SYS_RSTSTAT_WDT	R/W	0X0	Wtchdog reset 0X1: Reset 0X0: Not reset Write 1 to clear the flag
	1	SYS_RSTSTAT_BOR	R/W	0X0	Power-down reset flag 0x1: Reset 0x0: Not reset Write 1 to clear the flag
	0	SYS_RSTSTAT_POR	R/W	0x1	Power-on reset flag 0x1: Reset 0x0: Not reset Write 1 to clear the flag

3.3.6.19 SYS_INTEN register

Address	Bit	Name	R/W	Reset value	Description
0x104	16:9	Reserved	——	——	Reserved
	8	SYS_INTEN_ETER	R/W	0x0	EEPROM error 2 bits and above interrupt enable 0x1: Enable 0x0: Disable
	7	SYS_INTEN_EOER	R/W	0x0	EEPROM error 1 bit interrupt enable 0x1: Enable 0x0: Disable
	6	SYS_INTEN_FTER	R/W	0x0	FLASH error 2 bits and above interrupt enable 0x1: Enable 0x0: Disable
	5	SYS_INTEN_FOER	R/W	0x0	FLASH error 1 bit interrupt enable 0x1: Enable 0x0: Disable
	4	SYS_INTEN_EPOT	R/W	0x0	EEPROM protected address illegal erase and write interrupt enable 0x1: Enable 0x0: Disable
	3	SYS_INTEN_FPOT	R/W	0x0	FLASH protected address illegal erase and write interrupt

					enable 0x1：Enable 0x0：Disable
	2	SYS_INTEN_ADJ	R/W	0x0	Power-on configuration word check error interrupt enable 0x1：Enable 0x0：Disable
	1	SYS_INTEN_XTALCHK	R/W	0x0	External crystal oscillator monitoring error 1 interrupt enable 0x1：Enable 0x0：Disable
	0	SYS_INTEN_XTALINIT	R/W	0x0	External crystal oscillator initialization failure interrupt enable 0x1：Enable 0x0：Disable

3.3.6.20 SYS_INTFLG register

Address	Bit	Name	R/W	Reset value	Description
0x108	16:9	Reserved	——	——	Reserved
	8	SYS_INTFLG_ETER	R/W	0x0	EEPROM error 2 bits or more interrupt flag 0x1：Interrupt 0x0：Not interrupt Write 1 to clear the flag.
	7	SYS_INTFLG_EOER	R/W	0x0	EEPROM error 1 bit interrupt flag 0x1：Interrupt 0x0：Not interrupt Write 1 to clear the flag.
	6	SYS_INTFLG_FTER	R/W	0x0	FLASH error 2 bits or more interrupt flag 0x1：Interrupt 0x0：Not interrupt Write 1 to clear the flag.
	5	SYS_INTFLG_FOER	R/W	0x0	FLASH error 1 bit interrupt flag 0x1：Interrupt 0x0：Not interrupt Write 1 to clear the flag.
	4	SYS_INTFLG_EPOT	R/W	0x0	EEPROM protected address illegally erase and write interrupt flag 0x1：Interrupt 0x0：Not interrupt Write 1 to clear the flag.

	3	SYS_INTFLG_FPOT	R/W	0x0	FLASH protected address illegally erase and write interrupt flag 0x1: Interrupt 0x0: Not interrupt Write 1 to clear the flag.
	2	SYS_INTFLG_ADJ	R/W	0x0	Power-on configuration word check error interrupt flag 0x1: Interrupt 0x0: Not interrupt Write 1 to clear the flag.
	1	SYS_INTFLG_XTALCHK	R/W	0x0	External crystal oscillator monitoring error 1 interrupt flag 0x1: Interrupt 0x0: Not interrupt In this case, the failure of the crystal oscillator can only judge the failure reset flag of the crystal oscillator. Write 1 to clear the flag.
	0	SYS_INTFLG_XTALINIT	R/W	0x0	External crystal oscillator initialization failure interrupt flag 0x1: Interrupt 0x0: Not interrupt When the system clock selects the crystal oscillator when the crystal oscillator fails, this flag will not be set 1; in this case, the crystal oscillator failure can only judge the reset flag of the crystal oscillator failure. Write 1 to clear the flag.

3.3.6.21 SYS_XTAL_CHK register

Address	Bit	Name	R/W	Reset value	Description
0x10C	7:2	Reserved	——	——	Reserved
	1	SYS_XTAL_CHK_RSTEN	R/W	0x0	Reset enable after crystal oscillator fails 0x1: Enable 0x0: Disable
	0	SYS_XTAL_CHK_EN	R/W	0x0	Crystal failure monitoring module enable 0x1: Enable 0x0: Disable

3.3.6.22 SYS_XTAL_CHKCNT register

Address	Bit	Name	R/W	Reset value	Description
0x110	31:18	Reserved	——	——	Reserved
	17:9	SYS_XTAL_CHKCNT_H	R/W	0x020	The period value of the left interval of the crystal oscillator failure monitoring (external crystal oscillator period)
	8:0	SYS_XTAL_CHKCNT_L	R/W	0x008	The period value of the right interval of the crystal oscillator failure monitoring (external crystal oscillator period)

3.3.6.23 SYS_XTAL_INIT register

Address	Bit	Name	R/W	Reset value	Description
0x114	7:2	Reserved	——	——	Reserved
	1	SYS_XTAL_INIT_RSTEN	R/W	0x0	Reset enable after crystal initialization fails 0x1: Enable 0x0: Disable
	0	SYS_XTAL_INIT_EN	R/W	0x0	Crystal oscillator initialization failure monitoring module enable 0x1: Enable 0x0: Disable

3.3.6.24 SYS_LVDT_IE register

Address	Bit	Name	R/W	Reset value	Description
0x118	7:1	Reserved	——	——	Reserved
	0	SYS_LVDT_IE	R/W	0x0	LVDT Enable 0x1: Enable 0x0: Disable

3.3.6.25 SYS_LVDT_IF register

Address	Bit	Name	R/W	Reset value	Description
0x11C	7:2	Reserved	——	——	Reserved
	1	SYS_LVDT_IE_H	R/W	0x0	LVDT boost recovery interrupt flag 0x1: Undervoltage event recovery 0x0: Not happen Write 1 to clear the flag.
	0	SYS_LVDT_IE_L	R/W	0x0	LVDT buck interrupt flag 0x1: Undervoltage event occurred 0x0: Not happen Write 1 to clear the flag. The LVDT low-voltage interrupt can wakeup the system from sleep mode and enter the LVDT interrupt program after wakeup. Only this process will not generate an undervoltage flag.

3.3.7 Example program

System clock selection:

```
    void SystemInit (void)
    {
        uint32_t clk_sel;
        if(__SYSTEM_CLOCK == PLL_32M)
            clk_sel = SYS_CLK_PLL_32M;  //select 32M clock
        else if(__SYSTEM_CLOCK == PLL_16M)
            clk_sel = SYS_CLK_PLL_16M;  //select 16M clock
        else if(__SYSTEM_CLOCK == PLL_8M)
            clk_sel = SYS_CLK_PLL_8M;   //select 8M clock
        else if(__SYSTEM_CLOCK == XTAL_16M || __SYSTEM_CLOCK == XTAL_8M)
            clk_sel = SYS_CLK_XTAL;     //select external oscillator clock
        system_clk_sel(clk_sel);
    }
```

System reset:

```
    void Reset_Flag()
    {
        //Crystal oscillator failure reset
        if(SYS_RSTSTAT & SYS_RSTSTAT_XTAL)
        {
            //Clear crystal oscillator failure reset flag
            SYS_RSTSTAT |= SYS_RSTSTAT_XTAL;
```

```
    }
    //Crystal oscillator initialization failed reset
    if(SYS_RSTSTAT & SYS_RSTSTAT_XTALINI)
    {
        //Clear the reset flag of crystal oscillator initialization failure
        SYS_RSTSTAT |= SYS_RSTSTAT_XTALINI;
    }
    //Address overflow reset
    if(SYS_RSTSTAT & SYS_RSTSTAT_ADDR)
    {
        //Clear address overflow reset flag
        SYS_RSTSTAT |= SYS_RSTSTAT_ADDR;
    }
    //External reset
    if(SYS_RSTSTAT & SYS_RSTSTAT_EXT)
    {
        //Clear external reset flag
        SYS_RSTSTAT |= SYS_RSTSTAT_EXT;
    }
    //CPU deadlock reset
    if(SYS_RSTSTAT & SYS_RSTSTAT_LOCKUP)
    {
        //Clear CPU deadlock reset flag
        SYS_RSTSTAT |= SYS_RSTSTAT_LOCKUP;
    }
    //Software reset
    if(SYS_RSTSTAT & SYS_RSTSTAT_SOFT)
    {
        //Clear software reset flag
        SYS_RSTSTAT |= SYS_RSTSTAT_SOFT;
    }
    //WDT reset
    if(SYS_RSTSTAT & SYS_RSTSTAT_WDT)
    {
        //Clear WDT reset flag
        SYS_RSTSTAT |= SYS_RSTSTAT_WDT;
    }
    //Power-on reset
    if(SYS_RSTSTAT & SYS_RSTSTAT_POR)
    {
        //Clear power-on reset
        SYS_RSTSTAT |= SYS_RSTSTAT_POR;
    }
}
```

```
System interrupt:
    void System_Intstat()
    {
        //EEPROM error 2 bits or more interrupt
        if(SYS_IF & SYS_INTFLG_ETER
        {
            //Clear flag of EEPROM error 2 bits or more interrupt
            SYS_IF |= SYS_INTFLG_ETER;
        }
        // EEPROM error 1 bit interrupt
        if(SYS_IF & SYS_INTFLG_EOER)
        {
            //Clear flag of EEPROM error 1 bit interrupt
            SYS_IF |= SYS_INTFLG_EOER;
        }
        //FLASH error 2 bits or more interrupt
        if(SYS_IF & SYS_INTFLG_FTER)
        {
            //Clear flag of FLASH error 2 bits or more interrupt
            SYS_IF |= SYS_INTFLG_FTER;
        }
        // FLASH error 1 bit interrupt
        if(SYS_IF & SYS_INTFLG_FOER)
        {
            //Clear flag of FLASH error 1 bit interrupt
            SYS_IF |= SYS_INTFLG_FOER;
        }
        //EEPROM protected address illegal erase and write interrupt
        if(SYS_IF & SYS_INTFLG_EPOT)
        {
            //Clear flag of EEPROM protected address illegal erase and write interrupt

            SYS_IF |= SYS_INTFLG_EPOT;
        }
        //FLASH protected address illegal erase and write interrupt
        if(SYS_IF & SYS_INTFLG_FPOT
        {
            //Clear flag of FLASH protected address illegal erase and write interrupt
            SYS_IF |= SYS_INTFLG_FPOT;
        }
        //Power-on configuration word check error interrupt
        if(SYS_IF & SYS_INTFLG_ADJ)
        {
            //Clear power-on configuration word check error interrupt
```

```
        SYS_IF |= SYS_INTFLG_ADJ;
    }
    //External crystal oscillator detects an error interrupt
    if(SYS_IF & SYS_INTFLG_XTALCHK)
    {
        //Clear flag of external crystal oscillator detects an error interrupt
        SYS_IF |= SYS_INTFLG_XTALCHK;
    }
    //External crystal oscillator initialization failed interrupt
    if(SYS_IF & SYS_INTFLG_XTALINIT)
    {
        //Clear flag of external crystal oscillator initialization failed interrupt
        SYS_IF |= SYS_INTFLG_XTALINIT;
    }
}
```

4 Memory

BF7006AMXX contains high reliability 96KB FLASH (excluding ECC), 2KB EEPROM (excluding ECC) and 4KB SRAM. Both FLASH and EEPROM support 6-bit ECC checking mechanism and memory bank protection function, and are automatically calculated and checked by hardware without software, extremely improves the safety and reliability of the program code.

BF7006AMXX system design FLASH and EEPROM as independent interfaces, the program can operate FLASH and EEPROM at the same time(except for the synchronous erasing and writing of FLASH and EEPROM).

4.1 FLASH

FLASH is the main space for program storage in the system, which consists of the FLASH_NVR special storage area and the main program storage area. The FLASH_NVR special storage area stores important key trimming data of the chip. Once it is overwritten by mistake, the chip will not work. Therefore, the storage space is always in a hardware protection state. The main program space is operated by the user and is used to store software programs and application data.

4.1.1 Features

Main features：

1. 4Mbytes address space；
2. Each WORD address space supports ECC checking；
3. A single page space size is 2K address space;
4. Data retention at a temperature of 85℃ for at least 10 years;
5. Because the memory bank has ECC calculation and verification, writing FLASH only supports WORD address jump read and write operations.

4.1.2 FLASH_NVR

The FLASH_NVR address space is: 0x0001_8000~0x0001_8FFF, which stores the key trimming data of the chip, and the change may cause the chip to fail to work normally.

4.2 EEPROM

EEPROM is the data storage space in the system, which is composed of the EEPROM_NVR special storage area and the main data storage area. The EEPROM_NVR special storage area is provided for users to use. The user can save the key data processed by the program read by the chip initialization. At the same time, the main data space can be used by the user to store the updated application data to expand the application.

4.2.1 Features

Main features：

1. Storage space 512 x 38bits (including ECC);
2. The size of a single page space is 16 x 38bits (64Bytes address space size, including ECC);
3. The maximum time of reading data is 35ns;
4. Single line writing time is 16~24us;
5. Single page erasing time 16~24ms;
6. The whole block erasing time is 16~24ms;
7. Data retention at a temperature of 85℃ for at least 10 years;

   Because the memory body has ECC calculation and checking, writing FLASH only supports word address jump read and write operations.

4.2.2 EEPROM_NVR

The EEPROM_NVR address space is: 0x4000_0800~0x4000_08FF, which can be used by the user, and the specific data stored is determined by the user.

4.3 SRAM

Main features：

1. 1K x 32 Bits SRAM, 4K address space;
2. Support Byte, Halfword, Word fast access;

4.4 Register map

FLASH/EEPROM controller module address range：0x5000_0000~0x5000_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	EFLASH_SEL	R/W	Erase and wirte control enable register	0x0000
0x04	EFLASH_MODE	R/W	Erase and write control mode register	0x00
0x08	EFLASH_EBCFG	R/W	Erase and write control function register	0x00
0x0C	FLASH_STATE	RO	FLASH working status register	0x01
0x10	EEPROM_STATE	RO	EEPROM working status register	0x01
0x200	EFLASH_ECC_CTRL	R/W	ECC control function register	0x01
0x204	EFLASH_UNLOCK	R/W	Unprotected key register	0x0000
0x208	FLASH_LOCK_SIZE	R/W	FLASH address range protection register	0x00
0x20C	EEEPROM_LOCK_SIZE	R/W	EEPROM address range protection register	0x00

4.5 Register description

4.5.1 EFLASH_SEL register

Address	Bit	Name	R/W	Reset value	Description
0x00	15:0	EFLASH_SEL	R/W	0x0000	Select to erase FLASH or EEPROM 0xAA55: FLASH 0xCD78: EEPROM Other values configured in this register are invalid. It is automatically reset to 0 after the completion of the erasing and writing operation status.

4.5.2 EFLASH_MODE register

Address	Bit	Name	R/W	Reset value	Description
0x04	7:0	EFLASH_MODE	R/W	0x00	FLASH/EEPROM erasing and writing control working mode 0xA5: Perform FLASH erasing, CPU can execute the program only after erasing is completed 0x5A: Perform FLASH erasing and writing, at this time the CPU can execute the program (available when the program executes in SRAM) 0x3C: Perform EEPROM erasing and writing, at this time the CPU executes the program This register can be configured only after EFLASH_SEL is selected. In other cases, this register cannot be written and the reset value is maintained. This register is reset by hardware after the completion of the erasing and writing operation and returns to the initial value. When the program is running in FLASH, it is impossible to erase all the FLASH. Therefore, when this register is configured with 00A5, the entire block erase command cannot be performed.

4.5.3 EFLASH_EBCFG register

Address	Bit	Name	R/W	Reset value	Description
0x08	7:0	EFLASH_EBCFG	R/W	0x00	FlASH/EEPROM erase and write control function register 0x55: Page erase operation 0xAA: Block erase operation

					0x33: Word write operation 0xCC: Continously write operation, the number of written data must not exceed 32 words for FLASH, or 8 words for EEPROM. This register can be configured only after EFLASH_SEL is selected. In other cases, this register cannot be written and the reset value is maintained. This register is reset by hardware after the completion of the erasing and writing operation and returns to the initial value. In EFLASH_MODE =0x00A5 mode, 0xAA cannot be configured. For whole chip erasing, when the address points to the NVR, the NVR space and the main storage space are all erased (EEPROM provides the user-available NVR space, which can be cleared by this configuration), when the address points to the main storage space, only the main storage space is erased (Any configuration of FLASH NVR cannot be erased).

4.5.4 FLASH_STATE register

Address	Bit	Name	R/W	Reset value	Description
0x0C	7:0	FLASH_STATE	RO	0x00	FLASH erase and write complete flag 0x1: Erase and write complete Other values are meaningless.

4.5.5 EEPROM_STATE register

Address	Bit	Name	R/W	Reset value	Description
0x10	7:0	EEPROM_STATE	RO	0x00	EEPROM erase and write complete flag 0x1: Erase and write complete Other values are meaningless

4.5.6 EFLASH_ECC_CTRL register

Address	Bit	Name	R/W	Reset value	Description
0x200	7:1	保留	——	——	Reserved
	0	EFLASH_ECC_CTRL	R/W	0x1	ECC enable 0x1: Enable 0x0: Disable

4.5.7 EFLASH_UNLOCK register

Address	Bit	Name	R/W	Reset value	Description
0x204	31:0	EFLASH_UNLOCK	R/W	0x00	Unlock the write protection key of the memory bank The FLASH_LOCK_SIZE and EEEPROM_LOCK_SIZE registers can be configured only when the configuration is 0xAB23DC54 to release the write protection register control. Other values cannot be configured. After the configuration is completed, this register is cleared.

4.5.8 FLASH_LOCK_SIZE register

Address	Bit	Name	R/W	Reset value	Description
0x208	7:0	FLASH_LOCK_SIZE	R/W	0x00	FLASH address protection range 0x00: Protect FLASH_NVR area 0x01: Protect FLASH_NVR +2K main program address space (start from little endian address) 0x02: Protect FLASH_NVR +4K main program address space (start from little endian address) 0x30: Protect FLASH_NVR +96K main program address space (start from little endian address)

4.5.9 EEPROM_LOCK_SIZE register

Address	Bit	Name	R/W	Reset value	Description
0x20C	7:0	EEPROM_LOCK_SIZE	R/W	0x00	EEPROM address protection range 0x00: Without any protection 0x01: Protect EEPROM_NVR +64Bytes data address space (start from little endian address) 0x02: Protect EEPROM_NVR +128Bytes data address space (start from little endian address) …… 0x20: Protect EEPROM_NVR +2048Bytes data address space (start from little endian address)

5 Peripherals

BF7006AMXX contains a wealth of peripheral resources, which supports multiple communication functions, including CAN communication, UART communication and LIN communication. BF7006AMXX also has multiple counters/timers, including RTC, WDT, Timer and Advance Timer; supports 54 general-purpose GPIOs and high precision and high speed ADC. This chapter introduces the characteristics, functions and detailed register description of each peripheral unit of BF7006AMXX in detail.

5.1 SCI

5.1.1 Features

BF7006AMXX supports up to 2 SCI modules, one of which has an independent functional port, and the other multiplexes external ports. Only one group of external ports is allowed to be used as SCI at the same time. The SCI module supports UART and LIN2.0 protocol communication.

1. Support full-duplex, half-duplex serial communication;
2. Double-buffered transmitter and receiver with independent enable;
3. Programmable baud rate (13-bit modulus frequency divider);
4. Interrupt-driven or polling operation:
   — The transmit data register is empty and the transmit is complete
   — Receive data register is full
   — Receive overflow, parity error, frame error and noise error
   — Idle receiver detection
   — Active edge detection on the receiving pin
5. Support LIN synchronization interval detection;
6. Support hardware parity generation and check;
7. Programmable 8-bit or 9-bit character length;
8. Programmable 1-bit or 2-bit STOP bit length;
9. Support idle line or address mark wakeup;
10. Optional 13-bit abort character detection / 11-bit abort character detection;
11. Optional transmitter output polarity and receiver input polarity;
12. Support baud rate adaptive function;

5.1.2 Functional description

This section introduces the main functions of BF7006AMXX SCI in detail.

5.1.2.1 Baud rate generation

Baud rate generation modulus bandrate={SCI_BDH[4:0],SCI_BDL}.

Baud rate calculation formula: when bandrate=0, no baud rate clock is generated, when bandrate=1~8191, SCI baud rate=BUSCLK/(16xbandrate). BUSCLK is the system clock and also the SCI

working clock.

Each time the baud rate register is configured, the internal counter will be cleared to regenerate the baud rate signal.

Communication requires the transmitter and receiver to use the same baud rate.

The calculation error generated by the register itself: 20KHz/2MHz=1% (baud rate 20KHz, system clock 32MHz), the smaller the baud rate, the smaller the error, the smaller the system clock frequency, the greater the error, and the greater the system clock error is the greater the calculation error.

The allowable baud rate deviation range for communication: 8/11*16=4.5%.

This system supports automatic baud rate matching. In the LIN protocol, the sync segment character is 0x55. When detecting baud rate, the measurement starts from the falling edge of the received START bit until the falling edge of the 8th data bit stops. Baud rate will be automatically updated after the communication is completed, and it can be read through the register SCI_BDH/SCI_BDL. Note that receiving the synchronization segment and automatically matching the baud rate is performed simultaneously with the receiving function. After the character is received, a receiving full interrupt will be issued. The maximum deviation before the baud rate is matched is not allowed to exceed 40%, otherwise the calibration will be invalid.

5.1.2.2 Transmitter function

The idle state of the transmitter output pin Tx is logic high by default (SCI_C3[4]=0 after reset). If SCI_C3[4]=1, the transmitter output is reversed.

The transmitter can send three kinds of characters: leading idle character, stop character, and data character. Three types of characters are queued to be sent. Writing 0 and then 1 in the SCI_C2[4] bit will queue leading idle characters. Writing 1 and then 0 in the SCI_C2[0] bit will queue an abort character. Writing to the data register SCI_D will queue a data character.

By setting SCI_C2[3], the transmitter is enabled. This will queue a leading idle character, which is a complete character frame in the idle state, and send 12-bit or 11-bit or 10-bit idle characters (logic high) according to SCI_C1[4] and SCI_C1[6]. In the normal application process, sending idle characters is necessary. The program will wait for SCI_S1[7] to be valid before setting indicating the last character has been moved to the transmission shifter. And then write a ‘0‘following an ‘1’ into the SCI_C2[3]. After that, once the shifter is available, the operation immediately queues the idle characters sent. Note: When SCI_C2[3] =0, as long as the characters in the shifter (including three characters) are not completed, the SCI transmitter will not stop sending.

By writing data to SCI_D, the program saves the data to the send data buffer, and a data character will be queued. The central element of the SCI transmitter is the transmit shift register with a length of 10 or 11 or 12 bits (the settings in the SCI_C1[4] and SCI_C1[6] control bits). Assuming that SCI_C1[4]=0, select the normal 8-bit data mode. In 8-bit data mode, there are 1 start bit, 8 data bits and 1/2 stop bit in the shift register. When the transmit shift register can be used for a new SCI character, the value waiting in the transmit data register is transferred to the shift register, and the transmit data register empty SCI_S1[7] flag is set at the same time, showing that another character can be written into the data transmission buffer of SCI_D.

By writing the SCI_C2[0] bit to 1 and then writing 0, an abort character will be queued. The abort character is a full character time of logic 0 (10-bit time, including start and stop bits). The 13-bit time longer

interrupt character can be enabled by setting SCI_S2[2]=1. Similarly, SCI_C1[4] and SCI_C1[6] can choose to add one bit time respectively. Generally speaking, the program should wait for SCI_S1[7] to be valid before setting, to show that the last character of the message has been moved to the transmit shifter, and then write 1 and 0 into the SCI_C2[0] bit in turn. After that, once the shifter is available, the operation immediately queues an abort character to be sent. If SCI_C2[0] is still 1 when the abort character that has entered the queue enters the shifter, the additional abort character will be queued.

If there is no new character (including three characters) waiting in the sending data buffer after the stop shifting out of the Tx pin, the transmitter sets the sending completion flag and enters the idle mode, Tx is in the high state, waiting to send more characters.

The conditions for generating an empty interrupt for sending data include: configuring the transmitter to enable 0 to 1 to generate an empty interrupt, and generating an empty interrupt when the transmit FIFO transmits to the shift register. Disable the transmitter during transmission will stop sending after the current character has been sent, and clear the previously queued characters. Sending completion interrupt generation condition: After the characters in the queue are all sent, a completion interrupt is generated.

5.1.2.3 Receiver function

By setting SCI_S2[4]=1, the receiver input is inverted. The receiver is enabled by setting the SCI_C2[2] bit.

There are three types of received characters: data characters, stop characters and idle characters.

The data character consists of a logic 0 start bit, 8 (or 9) data bits (LSB first) and a logic 1 stop bit. After receiving the stop bit into the receiving shifter, if the receiving data register is not full, the data character is transferred to the receiving data register, and the receiving data register is full status flag is set. If the SCI_S1[5] that the receive data register is full has been set at this time, the overflow SCI_S1[3] status flag is set, and the new data is lost. Because the SCI receiver is double-buffered, the program has a full character time after setting SCI_S1[5] and before reading the data in the receive data buffer to avoid receiver overflow.

When the program detects that the receive data register is full, it obtains data from the receive data register by reading SCI_D.

The stop character counts from the 0 character of the start until the stop bit detects the 0 character. The SCI_S2[1] bit selects whether to enable the 11-bit stop character detection. Once the rising edge on the pin is detected during the process, the count is cleared. If enough 0 characters (11/12/13 bits) are detected, set the abort character detection flag SCI_S2[7].

Idle characters start from the idle character bit count after the stop/start bit according to the SCI_C1[2] bit selection, and the detection starts after the receiver is active for a period of time (SCI_S1[5] is effectively set once). Once a 0 character is detected, the count is cleared, and enough 1 characters (10/11/12 bits) are detected, and the idle character detection flag SCI_S1[4] is set.

Note: After the abort character detection is enabled, only the abort character is detected, regardless of data reception, which is used for LIN protocol flow control; after the abort character detection is turned off, only the data is received, and the abort character detection is ignored.

5.1.2.4 Receiver sampling method

The SCI receiver uses a 16 times baud rate clock for sampling. The receiver searches for the falling edge on the Rx serial data input pin by extracting logic level samples at 16 times the baud rate. The falling edge is defined as logic 0 samples after 3 consecutive logic 1 samples. The 16 times baud rate clock is used to divide the bit time into 16 segments, labeled RT1 to RT16 respectively. When the falling edge is located, three samples should be taken from RT3, RT5 and RT7 to ensure that this is the real start bit, not just noise. If at least two of these three samples are 0, receive The device assumes that it is synchronized with the receiver character and starts to shift and receive the following data. If the above conditions are not met, it exits the state machine and returns to the state of waiting for the falling edge.

The receiver then samples each bit time of RT8, RT9 and RT10, including the start bit and stop bit, to determine the logic level of the bit. The logic level is the logic level of the vast majority of samples taken during the bit time. In the start bit, if at least two of the samples on RT3, RT5, and RT7 are 0, then the bit is assumed to be 0, even if one or all samples taken on RT8, RT9, and RT10 are 1. If any sample in any bit time of the character frame (8 samples RT3~RT10 of the start bit, 3 samples RT8~RT10 of the other bits) cannot be consistent with the logic level of this bit, when the received character is transmitted When the data buffer is received, the noise error flag SCI_S1[2] is set.

The falling edge detection logic keeps looking for a falling edge. If an edge is detected, the sample clock resynchronizes the bit time. In this way, when noise or baud rate is not matched, the reliability of the receiver can be improved.

5.1.2.5 Receiver wakeup

Receiver sleep wakeup is a hardware mechanism. The function of this mechanism is to use hardware detection to eliminate the software overhead of processing unimportant information characters. Allows SCI receivers to ignore characters in messages used for different SCI receivers.

In this application system, all receivers estimate the first character of each message, and once they determine that the information is intended for different receivers, they immediately write logic 1 to the SCI_C2[1] control bit. When this bit is set, it is forbidden to set the status flags related to the receiver (when the SCI_S2[3] bit is set, the idle bit IDLE will be set and an interrupt will be generated).

In the receiver sleep state (software sets the SCI_C2[1] bit to enter sleep), the wake-up mode can be selected through the SCI_C1[3] bit (that is, the hardware automatically clears the SCI_C2[1] bit), including idle character wake-up and address mark wake-up.

Idle character detection is described above. Once the receiver detects a complete idle character, SCI_C2[1] is automatically cleared. After waking up, the receiver will set the corresponding status flag when the next character is received.

Address mark wake-up is when the receiver detects the logic 1 in the highest bit of the received character (in SCI_C1[4]=0 mode, it is the 8th bit; in SCI_C1[4]=1 mode, it is the 9th bit R8)， SCI_C2[1] is automatically cleared. After waking up, the relevant status flags and interrupts of the receiver can be set in the current character.

5.1.2.6 Pin connection modes

When SCI_C1[7] = 1, SCI_C1[5] selects loop mode or single-wire mode.

Cyclic mode：

The loop mode is independent of external system connections and is sometimes used to check software to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input, and the SCI does not use the Rx pin.

Single-wire mode：

In single-wire mode, the SCI_C3[5] bit controls the direction of the serial data on the Tx pin. When SCI_C3[5]=0, the Tx pin is the input of the SCI receiver and connects to the receiver input. When SCI_C3[5]=1, the Tx pin is an output driven by the transmitter.

5.1.2.7 Interrupt wakeup

In the low power consumption mode of the system, this module can be configured to enable the receive edge interrupt function to wake up the system.

When the configuration module enables SCI_EN=1 and the Rx pin active edge interrupt enables SCI_BDH[6]=1, the module will asynchronously detect the edge flip of the Rx pin. When the Rx pin is low (SCI_S2[4]=0),asynchronous interrupt signal is generated.

5.1.3 Register map

There are 2 SCI modules in BF7006AMXX, and the module registers are distinguished according to the following address range:

SCI0 module address range：0x5004_0000~0x5004_3FFF

SCI1 module address range：0x5004_4000~0x5004_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	SCI_BDH	R/W	SCI(0/1) baud rate control register: High	0x00
0x04	SCI_BDL	R/W	SCI(0/1) baud rate control register: Low	0x00
0x08	SCI_C1	R/W	SCI(0/1) control register 1	0x00
0x0C	SCI_C2	R/W	SCI(0/1) control register 2	0x00
0x10	SCI_S1	RO	SCI(0/1) interrupt status flag register	0x00
0x14	SCI_S2	RO/RW	SCI(0/1) synchronization interval control register	0x00
0x18	SCI_C3	RO/RW	SCI(0/1) control register 3	0x00
0x1C	SCI_D	R/W	SCI(0/1) data register 3	0x00
0x20	SCI_EN	R/W	SCI(0/1) module enable register	0x00

5.1.4 Register description

5.1.4.1 SCI_BDH register

Address	Bit	Name	R/W	Reset value	Description
0x00	7	SCI_BDH_BREAK_CHECK_EN	R/W	0x0	Interval detection interrupt enable 0x1: Interrupt enable 0x0: Interrupt disable
	6	SCI_BDH_RX_EDGE_INT_EN	R	0x0	Rx pin active edge interrupt enable 0x1: Interrupt enable 0x0: Interrupt disable
	5	SCI_BDH_RATE_AUTOMATCH_EN		0x0	Baud rate automatic matching enable 0x1: Adaptive baud rate 0x0: Fixed baud rate Synchronous segment baud rate automatic matching is only applicable to 8-bit data 0x55, which is the next communication must be 0x55, and the baud rate will be automatically updated after the communication is completed, and it can be read through the register SCI_BDH/SCI_BDL, After enabling, it can only be synchronized once. If you need to synchronize again, you need to pull the enable bit low and then high.
	4:0	SCI_BDH_BPR_H		0x0	The upper 5 bits of the baud rate modulus divisor register Baud rate bandrate={SCI_BDH[4:0],SCI_BDL}, when bandrate=0, no baud rate clock is generated, when bandrate=1~8191, SCI baud rate=system bus clock BUSCLK/(16xbandrate).

5.1.4.2 SCI_BDL register

Address	Bit	Name	R/W	Reset value	Description
0x04	7:0	SCI_BDL_BPR_L	R/W	0x0	The lower 8 bits of the baud rate modulus divisor register

5.1.4.3 SCI_C1 register

Address	Bit	Name	R/W	Reset value	Description
0x08	7	SCI_C1_CYCLE_MODE	R/W	0x0	Cycle mode enable 0x1: Cyclic mode or single-wire mode, Tx connects to Rx 0x0: Normal two-wire mode
	6	SCI_C1_STOP_WIDTH	R/W	0x0	Stop bit selection 0x1: 2 bits 0x0: 1 bit
	5	SCI_C1_SINGLE_TXD	R/W	0x0	Single-wire mode enable 0x1: Single wire mode is selected when SCI_C1_CYCLE_MODE =1, Tx pin is valid 0x0: Internal cycle mode, Tx pin is invalid
	4	SCI_C1_DATA_WIDTH	R/W	0x0	Transmission data mode selection 0x1: 9-bit mode (the 9th bit is the parity bit) 0x0: 8-bit mode
	3	SCI_C1_WAKE_SEL	R/W	0x0	Receiver wakeup mode selection 0x1: Address flag wake up 0x0: Idle line wakeup
	2	SCI_C1_IDLE_SEL	R/W	0x0	Idle line type selection 0x1: Idle character bit count starts after stop bit 0x0: After the start bit, the idle character bit count starts and counts 10 bit time (if SCI_C1_STOP_WIDTH =1 or SCI_C1_DATA_WIDTH =1, add 1 bit time respectively)
	1	SCI_C1_PARITY_EN	R/W	0x0	Parity check enable 0x1: Parity check enable 0x0: Parity check disable Parity enable is used for receiver parity check enable and transmitter parity generation enable, applicable in 8-bit/9-bit mode. In 8-bit mode, the 8th bit sent by the transmitter is the bit generated by the parity check of the first 7 bits, and the value obtained by the receiver checking the first 7 bits is compared with the 8th bit; In 9-bit mode, the t8 sent by the transmitter is a bit generated by 8-bit data calculation, and the value obtained by the receiver checking the first 8-bit data is compared with r8.
	0	SCI_C1_PARITY_ODD	R/W	0x0	Parity selection 0x1: Odd parity 0x0: Even parity Parity check selection, odd check means that the total

					number of 1s in the data character (including the parity bit) is odd. An even number indicates that the total number of 1s in the data character (including the parity bit) is an even number.

5.1.4.4 SCI_C2 register

Address	Bit	Name	R/W	Reset value	Description
0x0C	7	SCI_C2_TX_EMPTY_INT_EN	R/W	0x0	Transmit buffer empty interrupt enable 0x1: interrupt enable 0x0: interrupt disabled Interrupt enable does not affect the setting of the interrupt flag in the register, so it can be used in polling mode if the interrupt is not enabled.
	6	SCI_C2_TX_COMP_INT_EN	R/W	0x0	Transmit complete interrupt enable 0x1: interrupt enable 0x0: interrupt disabled Interrupt enable does not affect the setting of the interrupt flag in the register, so it can be used in polling mode if the interrupt is not turned on
	5	SCI_C2_RX_FULL_INT_EN	R/W	0x0	Receive full interrupt enable 0x1: interrupt enable 0x0: interrupt disabled Interrupt enable does not affect the setting of the interrupt flag in the register, so it can be used in polling mode if the interrupt is not enabled.
	4	SCI_C2_IDLE_INT_EN	R/W	0x0	Idle line interrupt enable 0x1: Interrupt enable 0x0: Interrupt disabled Interrupt enable does not affect the setting of the interrupt flag in the register, so it can be used in polling mode if the interrupt is not enabled.
	3	SCI_C2_TX_EN	R/W	0x0	Transmitter enable 0x1: The transmitter is turned on 0x0: Transmitter is off By clearing this bit to 0 and then setting it to 1, an idle character can be queued to be sent.
	2	SCI_C2_RX_EN	R/W	0x0	Receiver enable 0x1: The receiver is enabled 0x0: The receiver is disabled

	1	SCI_C2_RWU	R/W	0x0	Receiver wakeup control 0x1: The receiver is in the standby state, waiting for a wakeup condition 0x0: The receiver is working normally Writing 1 to this bit puts the SCI receiver into a sleep state, waiting for the automatic hardware detection of the selected wakeup condition. The wakeup condition can be an idle line between messages, or a logic 1 in the highest data bit in a character. The application software sets this bit, and it can also be cleared by software. The general application hardware wakes up and automatically clears.
	0	SCI_C2_BREAK_TX	R/W	0x0	To send an interval segment, write 1 and 0 into this bit successively, which means that an interval segment is placed in the transmission data stream, and a set of abort characters will be sent once the shifter is available.

5.1.4.5 SCI_S1 register

Address	Bit	Name	R/W	Reset value	Description
0x10	7	SCI_S1_TX_EMPTY_FLAG	RO	0x0	Transmit buffer empty interrupt flag 0x1: The sending buffer is empty 0x0: The send buffer is full Write SCI_D to clear this flag.
	6	SCI_S1_TX_COMP_FLAG	RO	0x0	Transmit complete interrupt flag 0x1: transmission is complete, the transmitter is idle 0x0: The transmitter is working Transmitting new data, reconfiguring the transmit enable, and configuring SCI_C2_BREAK_TX all clear this flag.
	5	SCI_S1_RX_FULL_FLAG	RO	0x0	Receive full interrupt flag 0x1: The receive buffer is full 0x0: The receive buffer is empty This flag can be cleared by reading SCI_D.
	4	SCI_S1_IDLE_FLAG	RO	0x0	Idle line interruption flag 0x1: Idle line detected 0x0: No idle line detected After a period of activity, when the SCI receiving line has been idle for a full character time, IDLE is set. When SCI_C1_IDLE_SEL =0, the receiver starts to count the idle bit time after the start bit. Therefore, if the received

					characters are all 1, these bit times and stop bit times are counted into the full character time(10/11/12 bit time depends on the SCI_C1_STOP_WIDTH and SCI_C1_DATA_WIDTH control bits) of the logic high required by the receiver to detect an idle line. When SCI_C1_IDLE_SEL =1, the receiver does not start counting the idle bit time until after the stop bit. Therefore, the stop bit and any logic high time at the end of the previous character will not count into the full character time required by the receiver to detect an idle line. This flag can be cleared by reading SCI_D.
	3	SCI_S1_RX_OVERFLOW_FLAG	RO	0x0	Receive overflow interrupt flag 0x1: Receive overflow (new data is lost) 0x0: No overflow When the new serial character is received and needs to be transmitted to the receiving data register, the original received character has not been read from SCI_D, and this flag is set. In this case, the new characters are lost (all related error flags are lost). This flag can be cleared by reading SCI_D.
	2	SCI_S1_NOSIE_ERR_FLAG	RO	0x0	Noise interrupt flag 0x1: Noise detected 0x0: No noise detected The receiver samples 16 samples per bit. If the samples are inconsistent, set this flag (8 samples RT3~RT10 for the start bit, 3 samples RT8~RT10 for the other bits, including the stop bit). This flag can be cleared by reading SCI_D. This bit only records the error condition of the data currently stored in the data register, and the data update status will be automatically updated.
	1	SCI_S1_FRAME_ERR_FLAG	RO	0x0	Frame error interrupt flag 0x1: Frame error detected 0x0: No frame error detected When the receiver detects a logic 0 when it should be a stop bit, the flag is set, and the receive full interrupt flag is also set. This flag can be cleared by reading SCI_D. This bit only records the error condition of the data currently stored in the data register, and the data update status will be automatically updated.
	0	SCI_S1_PARITY_ERR_FLAG	RO	0x0	Parity error interrupt flag 0x1: Receiver parity error 0x0: The parity check is correct This flag can be cleared by reading SCI_D. This bit only records the error condition of the data currently stored

					in the data register, and the data update status will be automatically updated.

5.1.4.6 SCI_S2 register

Address	Bit	Name	R/W	Reset value	Description
0x14	7	SCI_S2_BREAK_CHECK_FLAG	RW	0x0	Interval detection interrupt flag 0x1: Interval segment detected 0x0: Interval segment not detected When SCI_S2_BREAK_CHECK_EN =1 and the LIN stop character is detected, this flag is set. ——Detect the character 0 of 11/12/13 bits. The software directly writes 1 to the register bit to clear the flag.
	6	SCI_S2_RX_EDGE_FLAG	RW	0x0	Rx pin active edge interrupt flag 0x1: An active edge appears on the receiving pin 0x0: There is no active edge on the receiving pin The software directly writes 1 to the register bit to clear the flag.
	5	Reserved	——	——	Reserved
	4	SCI_S2_RX_INVERSION	RW	0x0	Rx data inversion 0x1: Reversed data received 0x0: The received data is not inverted
	3	SCI_S2_RWU_IDLESEL	RW	0x0	Receive wake-up idle detection 0x1: During the receiving standby state, the IDLE bit is set when an idle character is detected, 0x0: During the receiving standby state, the IDLE bit is not set when an idle character is detected It is used to control whether the IDLE state and interrupt are set during the standby period of the receiver.
	2	SCI_S2_BREAK_TX_SIZE	RW	0x0	Interval segment generation bit length 0x1: Send with 13 bit time (if SCI_C1_STOP_WIDTH =1 or SCI_C1_DATA_WIDTH =1, add 1 bit time respectively) 0x0: Send with 10-bit time (if SCI_C1_STOP_WIDTH =1 or SCI_C1_DATA_WIDTH =1, add 1 bit time respectively)
	1	SCI_S2_BREAK_CHECK_EN	RW	0x0	Interval detection enable 0x1: Detect on the length of 11 bit time (if SCI_C1_STOP_WIDTH =1 or SCI_C1_DATA_WIDTH =1, add 1 bit time respectively) 0x0: Do not detect Set this bit, after the 0x00 character is detected, if the

					STOP bit still detects the character 0, it will prevent the setting of the frame error and the receive data register full flag, and the next character 0 will be detected. After the interval detection is enabled, if normal data is detected, it will be received normally, including frame errors and full reception will be set normally. Once the 0x00 character is detected and STOP=0, no data will be received and only the interval segment will be detected. If this bit is not enabled, all data is received normally, frame error and receive full are set normally.
	0	SCI_S2_RX_ACTIVE_FLAG	RO	0x0	Receiver active flag 0x1: Receiver is active 0x0: Receiver is idle When the SCI receiver detects the beginning of the valid start bit, it is set; when the receiver detects an idle line, it is cleared; this status flag can be used to tell the system whether it is currently receiving SCI characters.

Description：

1. The relationship between SCI_S2_BREAK_TX_SIZE, SCI_C1_STOP_WIDTH and SCI_C1_DATA_WIDTH corresponding character length is as follows:

SCI_S2_BREAK_TX_SIZE	SCI_C1_STOP_WIDTH	SCI_C1_DATA_WIDTH	Character length
0	0	0	10 bit times
0	0	1	11 bit times
0	1	0	11 bit times
0	1	1	12 bit times
1	0	0	13 bit times
1	0	1	14 bit times
1	1	0	14 bit times
1	1	1	15 bit times

5.1.4.7 SCI_C3 register

Address	Bit	Name	R/W	Reset value	Description
0x18	7	SCI_C3_R8	RO	0x0	The 9th data of the receiver, read only This bit is the 9th data bit of the receiver. When configured for 9-bit data, R8 can be regarded as the 9th received data on the left side of the MSB of the data register. It is necessary to read R8 before reading the data register, because reading the data register can complete the automatic receiving full flag clearing, allowing R8 and SCI_D to be overwritten by new data.
	6	SCI_C3_T8	R/W	0x0	The 9th data of the transmitter The 9th data of the transmitter, when the SCI is configured for 9-bit data, T8 can be regarded as the 9th transmitted data bit to the left of the MSB of the buffered data in the data register. T8 should be written before the data register is written (if it needs to be modified from its original value). If T8 does not need to be modified in the new value (for example, when it is used to generate a flag or parity check), it does not need to be written before each write to the data register. When the parity check is enabled, this bit is used as the parity bit, which is automatically generated by the hardware, not the written value.
	5	SCI_C3_TX_DIR	R/W	0x0	Tx pin direction selection in single-wire mode 0x1: Tx pin is the output in single-wire mode 0x0: Tx pin is the input in single-wire mode
	4	SCI_C3_TX_INV	R/W	0x0	Tx data inversion 0x1: Send data reverse 0x0: Send data is not reversed
	3	SCI_C3_RX_OF_INT_EN	R/W	0x0	Receive overflow interrupt enable 0x1: interrupt enable 0x0: interrupt disabled
	2	SCI_C3_NOISE_ERR_INT_EN	R/W	0x0	Noise interrupt enable 0x1: interrupt enable 0x0: interrupt disabled
	1	SCI_C3_FRAME_ERR_INT_EN	R/W	0x0	Frame error interrupt enable 0x1: interrupt enable 0x0: interrupt disabled
	0	SCI_C3_PARITY_ERR_INT_EN	R/W	0x0	Parity error interrupt enable 0x1: interrupt enable

					0x0: interrupt disabled

5.1.4.8 SCI_D register

Address	Bit	Name	R/W	Reset value	Description
0x1C	7:0	SCI_DATA	R/W	0xFF	SCI data register Reading returns the contents of the read-only receive data buffer, and writing is the write sending data buffer.

5.1.4.9 SCI_EN register

Address	Bit	Name	R/W	Reset value	Description
0x20	7:1	Reserved	——	——	Reserved
	0	SCI_ENABLE	R/W	0x0	SCI module enable register Module working clock gating enable 0x1: Indicates that the enable is valid, and the module working clock is turned on 0x0: Disable the working clock of the module and reset the functional module When the module enable is turned off, the module can only perform write and read registers. In the system low power consumption mode, the module Rx edge interrupt wake-up function must also be enabled when the configuration module is enabled.

5.1.5 Example program

UART function:

```
//Data structure initialization:
sci_parameter_struct sci0_parameter_init =
{
    SCI_BAUD_RATE_9600,             //Set the baud rate
    SCI_NORMAL_MODE,                //Normal working mode
    SCI_STOP_BIT_1BIT,                  //1 bit for stop
    SCI_DATA_BIT_8BITS,             //8 bits for data transmission mode
    SCI_PARITY_DISABLE,             //Parity check disable
```

```
    SCI_PARITY_ODD,                 //Select odd parity
    SCI_BREAK_TX_SIZE_13BITS,           //Break transmission with 13 bit times
    SCI_BREAK_CHECK_ENABLE,      //Interval detection enable
    SCI_RATE_AUTOMATCH_DISABLE, //Fixed baud rate
    SCI_TX_ENABLE,                  //Tx enable
    SCI_RX_ENABLE,                  //Rx enable
    SCI_IDLE_SEL_STOPBIT,               //Idle character bit count starts after stop bit
    SCI_WAKE_SEL_ADDRRECE,          //Address mark wakeup
    SCI_TX_INV_DISABLE,             //Send data is not reversed
    SCI_RWU_IDLESEL_DISABLE,            //During the receiving standby state, the IDLE bit
is not set when an idle character is detected
    SCI_IE_TX_EMPTY | SCI_IE_TX_COMP | SCI_IE_RX_FULL | SCI_IE_BREAK_CHECK |
SCI_IE_RX_EDGE,/*Transmit buffer empty interrupt enable、Transmit complete interrupt
enable、Receive full interrupt enable、Interval detection interrupt enable、Rx pin active
edge interrupt enable*/
    SCI_NVIC_ERR_ENABLE | SCI_NVIC_TX_DISABLE | SCI_NVIC_RX_ENABLE//Error interrupt enable、
Transmit interrupt disable、Receive interrupt enable
};

sci_parameter_struct sci1_parameter_init =
{
    SCI_BAUD_RATE_9600,             // Set the baud rate
    SCI_NORMAL_MODE,                // Normal working mode
    SCI_STOP_BIT_1BIT,                  //1 bit for stop
    SCI_DATA_BIT_8BITS,                 //8 bits for data transmission mode
    SCI_PARITY_DISABLE,                 // Parity check disable
    SCI_PARITY_ODD,                     // Select odd parity
    SCI_BREAK_TX_SIZE_13BITS,           // Break transmission with 13 bit times
    SCI_BREAK_CHECK_ENABLE,      // Interval detection enable
    SCI_RATE_AUTOMATCH_DISABLE,     // Fixed baud rate
    SCI_TX_ENABLE,                      // Tx enable
    SCI_RX_ENABLE,                      // Rx enable
    SCI_IDLE_SEL_STOPBIT,               // Idle character bit count starts after stop bit
    SCI_WAKE_SEL_ADDRRECE,          // Address mark wakeup
    SCI_TX_INV_DISABLE,                 // Send data is not reversed
    SCI_RWU_IDLESEL_DISABLE,            // During the receiving standby state, the IDLE bit
is not set when an idle character is detected
    SCI_IE_TX_EMPTY | SCI_IE_TX_COMP | SCI_IE_RX_FULL | SCI_IE_BREAK_CHECK |
SCI_IE_RX_EDGE,/* Transmit buffer empty interrupt enable、Transmit complete interrupt
enable、Receive full interrupt enable、Interval detection interrupt enable、Rx pin active
edge interrupt enable */
    SCI_NVIC_ERR_ENABLE | SCI_NVIC_TX_ENABLE | SCI_NVIC_RX_DISABLE// Error interrupt
enable、Transmit interrupt enable、Receive interrupt disable
};
```

```
//Main function:
int main(void)
{
    //UART initialization
    sci1_port_sel(SCI1_PF0_PF1);            //Select PF0 and PF1 as functional port of SCI1
    sci_init(SCI1,sci1_parameter_init);
    sci_init(SCI0,sci0_parameter_init);     //Select SCI0 or SCI1 with corresponding data
structure
    sci_send_byte(SCI1,0x45);               //Write 0x45 to SCI1 register
}

//Subfunction:
/***************Select SCI1 functional port******************/
void sci1_port_sel(uint8_t port_sel)
{
    SYS_PTSEL = port_sel;                   //Write the selected port group to the
corresponding register
}
/******************SCI initialization*********************/
ErrorStatus sci_init(uint32_t scix,sci_parameter_struct sci_parameter_init)
{
    uint16_t tmp;
    uint8_t temp;
    tmp = ((SystemCoreClock/sci_parameter_init.baud_rate) >> 4);
    /*Write and shift the value in the SCI initialization data structure to the baud rate
control register and status register*/
    SCI_BDH(scix) = (tmp >> 8);
    SCI_BDL(scix) = tmp;
    SCI_C1(scix) = (sci_parameter_init.work_mode | sci_parameter_init.data_bit_width |
sci_parameter_init.idle_sel | \
                    sci_parameter_init.parity_en | sci_parameter_init.parity_sel);
    SCI_BDH(scix) |= (sci_parameter_init.int_enable >> 8 |
sci_parameter_init.rate_automatch_en);
    SCI_C2(scix) = ((sci_parameter_init.int_enable & 0xf0) | sci_parameter_init.tx_en |
sci_parameter_init.rx_en);
    SCI_S2(scix) = (sci_parameter_init.rwu_idlesel | sci_parameter_init.break_tx_size |
sci_parameter_init.break_check_en);
    SCI_C3(scix) = (sci_parameter_init.tx_inversion_en | (sci_parameter_init.int_enable &
0x0f));
    temp = sci_parameter_init.nvic_enable;
    if(scix == SCI0){
        if(temp & SCI_NVIC_ERR_ENABLE){
            NVIC_EnableIRQ(SCI0_ERR_IRQn);
```

```
        }
        if(temp & SCI_NVIC_TX_ENABLE){
            NVIC_EnableIRQ(SCI0_TX_IRQn);
        }
        if(temp & SCI_NVIC_RX_ENABLE){
            NVIC_EnableIRQ(SCI0_RX_IRQn);
        }
    }else{
        if(temp & SCI_NVIC_ERR_ENABLE){
            NVIC_EnableIRQ(SCI1_ERR_IRQn);
        }
        if(temp & SCI_NVIC_TX_ENABLE){
            NVIC_EnableIRQ(SCI1_TX_IRQn);
        }
        if(temp & SCI_NVIC_RX_ENABLE){
            NVIC_EnableIRQ(SCI1_RX_IRQn);
        }
    }                       //Enable SCI error interrupt, transmit and receive interrupt
    sci_enable(scix);
    return SUCCESS;
}
/*****************SCI transmits data*********************/
void sci_send_byte(uint32_t scix,uint8_t dat)
{
    while(!(SCI_S1(scix) & SCI_S1_TX_EMPTY_FLAG));
    SCI_D(scix) = dat;                      //Wait for the transmit buffer to be empty and
write data to the data register
}
/*************SCI1 transmit interrupt service function****************/
void SCI1_TX_IRQHandler(void)
{
    uint8_t state = sci_tx_int_flag_get(SCI1);
    uint8_t tmp;
    if(state & SCI_IF_TX_EMPTY){
        SCI_D(SCI1) = tmp;
    }                                   //Clear the interrupt flag if the transmit buffer is
empty
}
/*************SCI0 receive interrupt service function****************/
void SCI0_RX_IRQHandler(void)
{
    uint8_t state = sci_rx_int_flag_get(SCI0);
    uint8_t tmp;
    if(state & SCI_IF_BREAK_CHECK){
```

```
        sci_break_int_flag_clr(SCI0);           //Clear interval detection interrupt flag
    }else if(state & SCI_IF_RX_FULL){
        tmp = SCI_D(SCI0);              //Clear receive full interrupt flag
    }else if(state & SCI_IF_RX_EDGE){
        sci_rx_edge_int_flag_clr(SCI0);     //Clear Rx pin active edge interrupt flag
    }
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);
    gpio_bit_set(GPIOE,GPIO_PIN_5);     //Turn on the light
}
```

LIN function:

```
//LIN sends data array and cyclic increment initialization definition
volatile uint8_t lin0_send_dat[11];
uint8_t i=0;
//Data structure initialization
sci_parameter_struct sci_parameter_init =
{
    SCI_BAUD_RATE_9600,             // Set the baud rate
    SCI_NORMAL_MODE,                // Normal working mode
    SCI_STOP_BIT_1BIT,                  //1 bit for stop
    SCI_DATA_BIT_8BITS,             //8 bits for data transmission mode
    SCI_PARITY_DISABLE,             // Parity check disable
    SCI_PARITY_ODD,                 // Select odd parity
    SCI_BREAK_TX_SIZE_13BITS,           // Break transmission with 13 bit times
    SCI_BREAK_CHECK_ENABLE,     // Interval detection enable
    SCI_RATE_AUTOMATCH_DISABLE, // Fixed baud rate
    SCI_TX_ENABLE,                  //Tx enable
    SCI_RX_ENABLE,                  //Rx enable
    SCI_IDLE_SEL_STOPBIT,               // Idle character bit count starts after stop bit
    SCI_WAKE_SEL_ADDRRECE,          // Address mark wakeup
    SCI_TX_INV_DISABLE,             // Send data is not reversed
    SCI_RWU_IDLESEL_DISABLE,            // During the receiving standby state, the IDLE bit
is not set when an idle character is detected
    SCI_IE_TX_EMPTY | SCI_IE_TX_COMP | SCI_IE_RX_FULL | SCI_IE_BREAK_CHECK |
SCI_IE_RX_EDGE,/*Transmit buffer empty interrupt enable、Transmit complete interrupt
enable、Receive full interrupt enable、Interval detection interrupt enable、Rx pin active
edge interrupt enable*/
    SCI_NVIC_ERR_ENABLE | SCI_NVIC_TX_ENABLE | SCI_NVIC_RX_ENABLE//Error interrupt enable、
Transmit interrupt enable、Receive interrupt enable
};
//Main function:
int main(void)
{
    sci1_port_sel(SCI1_PF0_PF1);        //Select PF0 and PF1 as functional port of SCI1
```

```
    sci_init(SCI0,sci_parameter_init);
    sci_init(SCI1,sci_parameter_init);      //Select SCI0 or SCI1 with corresponding
structure
    gpio_init(GPIOF,GPIO_MODE_OUT,GPIO_PIN_3);
    gpio_bit_set(GPIOF,GPIO_PIN_3); //Pull up LIN0_FLT
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_2);
    gpio_bit_set(GPIOG,GPIO_PIN_2); //Pull up LIN1_FLT
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_3);
    gpio_bit_set(GPIOG,GPIO_PIN_3); //Pull up LIN1_CS
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_4);
    gpio_bit_set(GPIOG,GPIO_PIN_4); //Pull up LIN0_CS

    lin0_send_dat[0] = 0x55;
    lin0_send_dat[1] = 0x32;
    lin0_send_dat[2] = 0x32;
    lin0_send_dat[3] = 0x32;
    lin0_send_dat[4] = 0x32;
    lin0_send_dat[5] = 0x32;
    lin0_send_dat[6] = 0x32;
    lin0_send_dat[7] = 0x32;
    lin0_send_dat[8] = 0x32;
    lin0_send_dat[9] = 0x32;
    lin0_send_dat[10] = 0x00;
    for(i=0;i<10;i++)
    {
        lin0_send_dat[10] += lin0_send_dat[i];
    }
    sci_lin_sci0tx_sci1rx((uint8_t*)lin0_send_dat,11);//SCI0 transmits 11 datas to SCI1
}

//Subfunction:
/***************Select SCI1 functional port******************/
void sci1_port_sel(uint8_t port_sel)
{
    SYS_PTSEL = port_sel;                   ////Write the selected port group to the
corresponding register
}
/*******************SCI initialization**********************/
ErrorStatus sci_init(uint32_t scix,sci_parameter_struct sci_parameter_init)
{
    uint16_t tmp;
    uint8_t temp;
    tmp = ((SystemCoreClock/sci_parameter_init.baud_rate) >> 4);
```

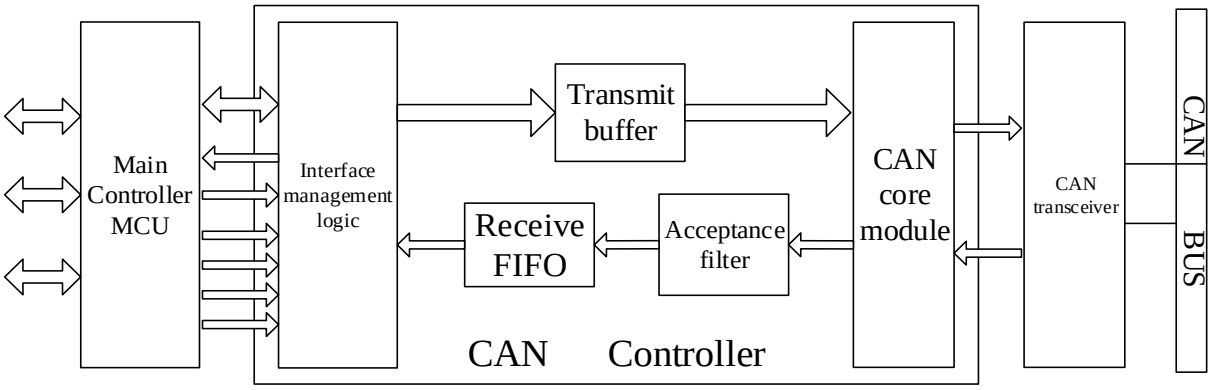
```
    /* Write and shift the value in the SCI initialization data structure to the baud rate
control register and status register */
    SCI_BDH(scix) = (tmp >> 8);
    SCI_BDL(scix) = tmp;
    SCI_C1(scix) = (sci_parameter_init.work_mode | sci_parameter_init.data_bit_width |
sci_parameter_init.idle_sel | \
                    sci_parameter_init.parity_en | sci_parameter_init.parity_sel);
    SCI_BDH(scix) |= (sci_parameter_init.int_enable >> 8 |
sci_parameter_init.rate_automatch_en);
    SCI_C2(scix) = ((sci_parameter_init.int_enable & 0xf0) | sci_parameter_init.tx_en |
sci_parameter_init.rx_en);
    SCI_S2(scix) = (sci_parameter_init.rwu_idlesel | sci_parameter_init.break_tx_size |
sci_parameter_init.break_check_en);
    SCI_C3(scix) = (sci_parameter_init.tx_inversion_en | (sci_parameter_init.int_enable &
0x0f));
    temp = sci_parameter_init.nvic_enable;
    if(scix == SCI0){
        if(temp & SCI_NVIC_ERR_ENABLE){
            NVIC_EnableIRQ(SCI0_ERR_IRQn);
        }
        if(temp & SCI_NVIC_TX_ENABLE){
            NVIC_EnableIRQ(SCI0_TX_IRQn);
        }
        if(temp & SCI_NVIC_RX_ENABLE){
            NVIC_EnableIRQ(SCI0_RX_IRQn);
        }
    }else{
        if(temp & SCI_NVIC_ERR_ENABLE){
            NVIC_EnableIRQ(SCI1_ERR_IRQn);
        }
        if(temp & SCI_NVIC_TX_ENABLE){
            NVIC_EnableIRQ(SCI1_TX_IRQn);
        }
        if(temp & SCI_NVIC_RX_ENABLE){
            NVIC_EnableIRQ(SCI1_RX_IRQn);
        }
    }                           //Enable SCI error interrupt, transmit and receive interrupt
    sci_enable(scix);
    return SUCCESS;
}
/*****************LIN0 transmit and LIN1 receive*******************/
void sci_lin_sci0tx_sci1rx(uint8_t *dat,uint8_t num)
{
    lin_dat = dat;
```

```
    lin_data_num = num;
    sci_lin_break_check_enable(SCI1);       //Enable SCI1 interval detection
    sci_lin_break_trans(SCI0);              //SCI0 transmits interval
    NVIC_EnableIRQ(SCI0_TX_IRQn);       //Enable SCI0 transmit interrupt
 }
/****************LIN0 transmit interrupt service function*****************/
void SCI0_TX_IRQHandler(void)
{
    uint8_t state = sci_tx_int_flag_get(SCI0);
    if(state & SCI_IF_TX_EMPTY){
        lin_data_num --;
        if(lin_data_num == 0){
            NVIC_DisableIRQ(SCI0_TX_IRQn);//Disable the transmit interrupt after data string
transmission completes
        }
        SCI_D(SCI0) = *lin_dat ++;
    }
}
/****************LIN1 receive interrupt service function*****************/
void SCI1_RX_IRQHandler(void)
{
    uint8_t state = sci_rx_int_flag_get(SCI1);
    uint8_t tmp;
    if(state & SCI_IF_BREAK_CHECK){
        sci_break_int_flag_clr(SCI1);           // Clear interval detection interrupt flag
    }else if(state & SCI_IF_RX_FULL){
        tmp = SCI_D(SCI1);              // Clear receive full interrupt flag
    }else if(state & SCI_IF_RX_EDGE){
        sci_rx_edge_int_flag_clr(SCI1);     // Clear Rx pin active edge interrupt flag
    }
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);
    gpio_bit_set(GPIOE,GPIO_PIN_5);     //Turn on the light
}
```

5.2 CAN

This chapter provides the internal block diagram, functional description and register introduction of the CAN controller, as well as application reference guidance.

CAN is a multi-master serial communication bus. The basic design specification requires a high bit rate, high resistance to electromagnetic interference, and the ability to detect any errors generated.



5.2.1 Features

CAN controller is a widely used field bus, it has the following characteristics:

1. Non-destructive arbitration;
2. High bus utilization, as long as the bus is free, any unit can start sending messages;
3. The data transmission rate supports up to 500Kbit/s;
4. Decide to receive or block the message according to the ID of the message;
5. Error handling and error detection mechanism;
6. After the sent information is damaged, it can be resent automatically;
7. The node has the function of automatically exiting the bus in the case of serious errors;
8. Support low-power wakeup function;
9. Reception and transmission of standard frame and extended frame information (CAN2.0B);
10. There are single/double acceptance filters in standard and extended formats, including shielding and code registers (CAN2.0B);
11. Programmable error limit alarm (CAN2.0B);
12. Add negative error interrupt, arbitration loss interrupt and bus error interrupt (CAN2.0B);
13. Add listening-only mode and loopback self-test mode (CAN2.0B).
14. Arbitration lost interrupt and detailed bit position (CAN2.0B).
15. Module clock can be configured as system clock (see 3.3.6.15), but PLL or RC module deviation may be introduced.

The arbitration mechanism ensures that neither information nor time is lost. When two or more units start to send messages at the same time, each bit of each message ID is arbitrated and compared one by one. The unit that wins the arbitration (the unit that continuously outputs the most dominant level is judged to be the highest priority) can continue to send the message, and the unit that loses the arbitration immediately stops sending and starts receiving.

5.2.2 Functional description

5.2.2.1 FIFO

FIFO is the storage area where the data sent on the CAN bus is loaded into the CAN controller and is divided into sending buffer FIFO and receiving buffer FIFO. These FIFOs contain 5 bytes of identifier and frame information and can contain up to 8 data bytes.

—— 1 byte frame information

—— 2 or 4 identifier byte standard frame or extended frame

——Up to 8 data bytes

CAN address	Name	Composition and label
0x40	Frame information	1 bit indicates if the message includes a standard frame or extended frame 1 remote transmission request bit 4-bit data length code indicates the number of data bytes
0x44 0x48	Identifier byte 1 Identifier byte 2	Standard frame: 11-bit identifier Extended frame: 16-bit identifier
0x4C 0x50	Identifier byte 3 Identifier byte 4	Extended frame only: 13 identifiers
Standard frame: 0x4C ~ 0x68 Extended frame: 0x54 ~ 0x70	Data bytes 1-8	Up to 8 data bytes indicated by the data length code

In the receiving buffer, the CAN controller has a multicast function, that is, the main controller sends a message, and all CAN controllers receive the message, and they are temporarily stored in the receiving buffer after receiving. The filter automatically checks the identifier and data byte. When the received identifier and data byte completely match the filter setting of the filter, the CAN controller will store the message identifier and data byte in the receiving buffer. Since the identifier of the CAN node is unique, only the unique node receives it.

5.2.2.2 Receive buffer

There are two buffers inside the CAN controller, which store the received data and the number of bytes of the received data respectively, which can prevent temporary storage when it is too late to process the CAN received data. 128*8 bits stores data, 32*8 bits stores the length of each message. When the data is not read when receiving data and the buffered data reaches 96, an overflow interrupt will be issued (an overflow interrupt is generated when the interrupt is enabled).

5.2.2.3 Interrupt

The following interrupts can cause the CAN controller to act on certain states immediately. Once CAN generates an interrupt, the CAN controller sets the interrupt output to a high level until the main controller reads the interrupt register and then clears the interrupt.

1. Receive interrupt.
2. Transmit interrupt.
3. Error alarm interrupt;
4. Data overflow interrupt;
5. Wakeup interrupt;
6. Negative error interrupt;
7. Arbitration lost interrupt;
8. Bus error interrupt;

The specific interrupt description is as follows:

1. Error alarm: the bus is closed, the transmission and reception error count exceeds the error alarm limit (96 in the basic mode, and the register error alarm limit register CAN_EMLR configuration value in the extended mode);

2. Data overflow: send out data overflow interrupt when the receive FIFO is greater than 96;

3. Negative error: A negative error interrupt is issued when the error count value is greater than or equal to 128;

Table 1. Error status and count value

Unit error status	Transmit error count value (TEC)		Receive error count value (REC)
Active error status	0～127	And	0～127
Passive error status	128～255	Or	128～255
Bus off status	256～		—

4. Arbitration lost: When arbitration is lost, a corresponding arbitration lost interrupt (interrupt enable) will be generated. At the same time, the current bit position of the bit stream processor is captured and sent to the arbitration loss capture register.

5. Bus error: When a bus error occurs, it is forced to generate a corresponding error interrupt (when the interrupt is enabled). At the same time, the current position of the bit stream processor is captured and sent to the error code capture register. These include bit errors, padding errors, CRC errors, format errors, and ACK errors.

Table 9. Type of errors

Type of error	Wrong content	Error detection frame(segment)	Detection unit
Bit error	Compare the output level and the bus level (without filling bits), the error detected when the two levels are not the same.	Data frame(SOF- EOF) Remote control frame(SOF-EOF) Error frame Overload frame	Transmit unit Receive unit
Fill error	Error detected when 6 bits of the same level are continuously detected in the segment that needs bit stuffing	Data frame(SOF-CRC sequence) Remote control frame(SOF-CRC sequence)	Transmit unit Receive unit
CRC error	Error detected when the CRC result calculated from the received data is different from the received CRC order	Data frame(CRC sequence) Remote control frame(CRC sequence)	Receive unit
Format error	Error detected when the format opposite to the fixed format bit field is detected	Data frame(CRC delimiter, ACK delimiter, EOF) Remote control frame(CRC delimiter, ACK delimiter, EOF) Error delimiter Overload delimiter	Receive unit
ACK error	The error detected when the transmitting unit detects the invisible level in the ACK slot(error detected when the ACK is not transmitted)	Data frame(ACK slot) Remote control frame(ACK slot)	Transmit unit

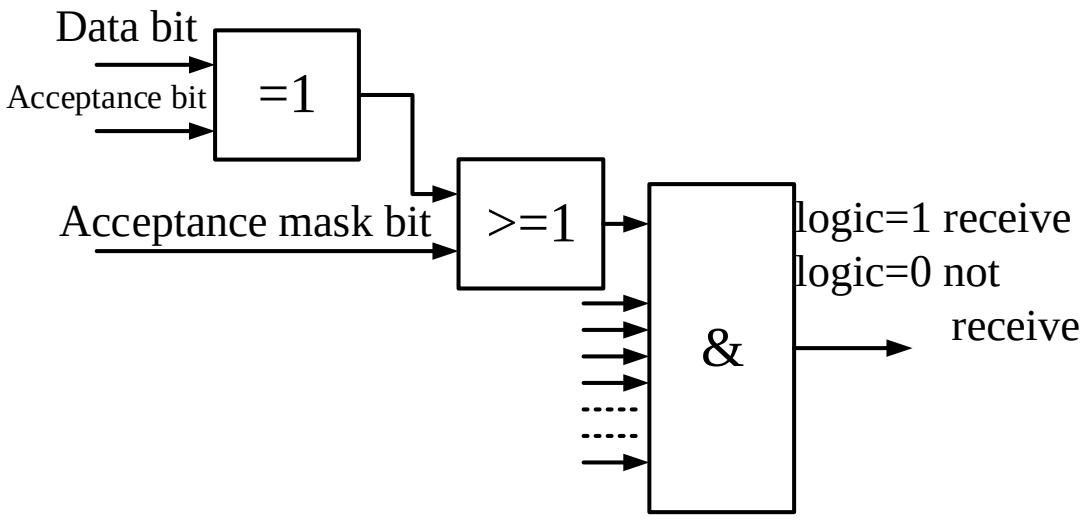
5.2.2.4 Acceptance filtering

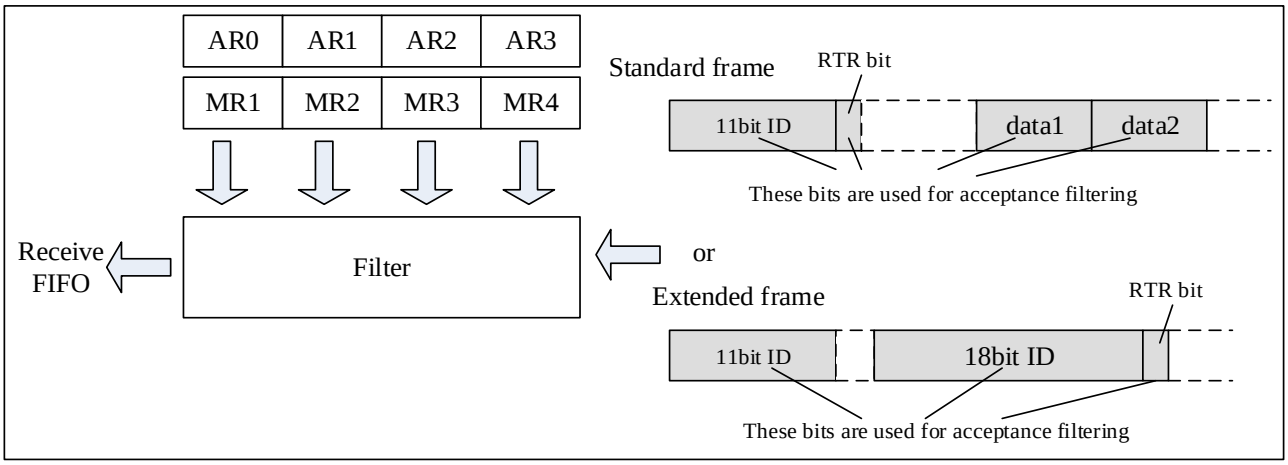
The acceptance filter consists of 4 8-bit acceptance code registers acceptance mask registers (CAN_IDAR0 / CAN_IDAR1 / CAN_IDAR2 / CAN_IDAR3) (CAN_IDMR0 / CAN_IDMR1 / CAN_IDMR2 / CAN_IDMR3). These registers can be used to control one long filter or two short filters. Which bits of the message are used for acceptance filtering depends on the received frame (standard frame or extended frame) and the selected filter mode (single filter or double filter). Acceptance filtering of standard frames can include RTR bits or even data bytes. For message bits that do not need to undergo acceptance filtering, the acceptance mask register must be set to 1 in the corresponding bit position.

If the message does not include data bytes (a remote frame or the data length code is zero) but the acceptance filter includes data bytes, the message will be received if the identifier up to the RTR bit is valid.

Single filter：

This filter configuration can define a long filter (4 bytes). The bit correspondence between the filter byte and the information byte depends on the current received frame format.





If the information received is the standard frame format, only the first two data bytes are used in the acceptance filter to store the complete identification code including the RTR bit. If there is no data byte due to the setting of the RTR bit, or there is no or only one data byte due to the setting of the corresponding data length code, the information will also be received. For a successfully received message, an acceptance signal must be sent after the comparison of all individual bits.

Note: The lower four bits of CAN_IDMR1 and CAN_IDAR1 are not used. In order to be compatible with future products, these bits can be set to 1 to be "not affected".

As shown in the figure below (standard frame filtered by single filter):

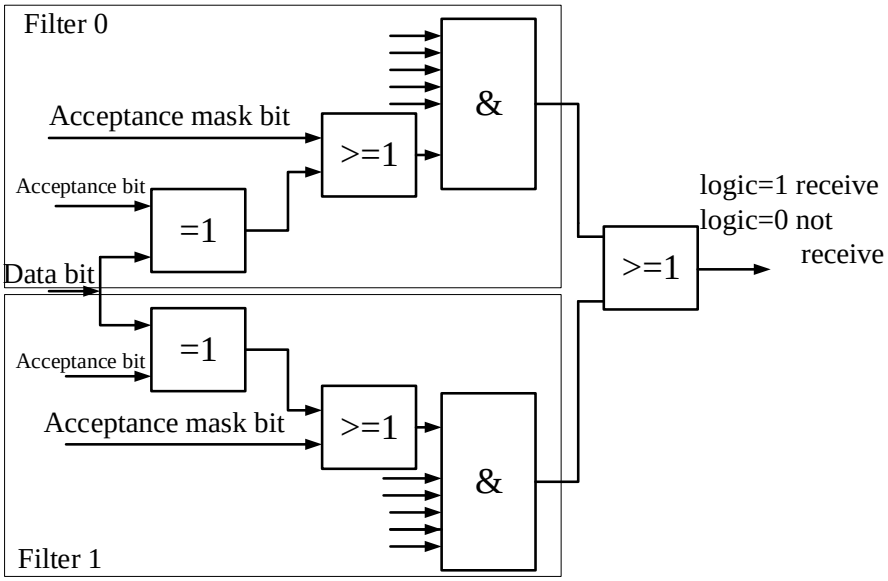
n	0	1(upper 4 bits)	2	3
CAN_IDARn	01XX X010	XXXX	XXXX XXXX	XXXX XXXX
CAN_IDMRn	0111 1010	1111	1111 1111	1111 1111
Receive message(ID[28:18], RTR)	01xx x010	XXXX		

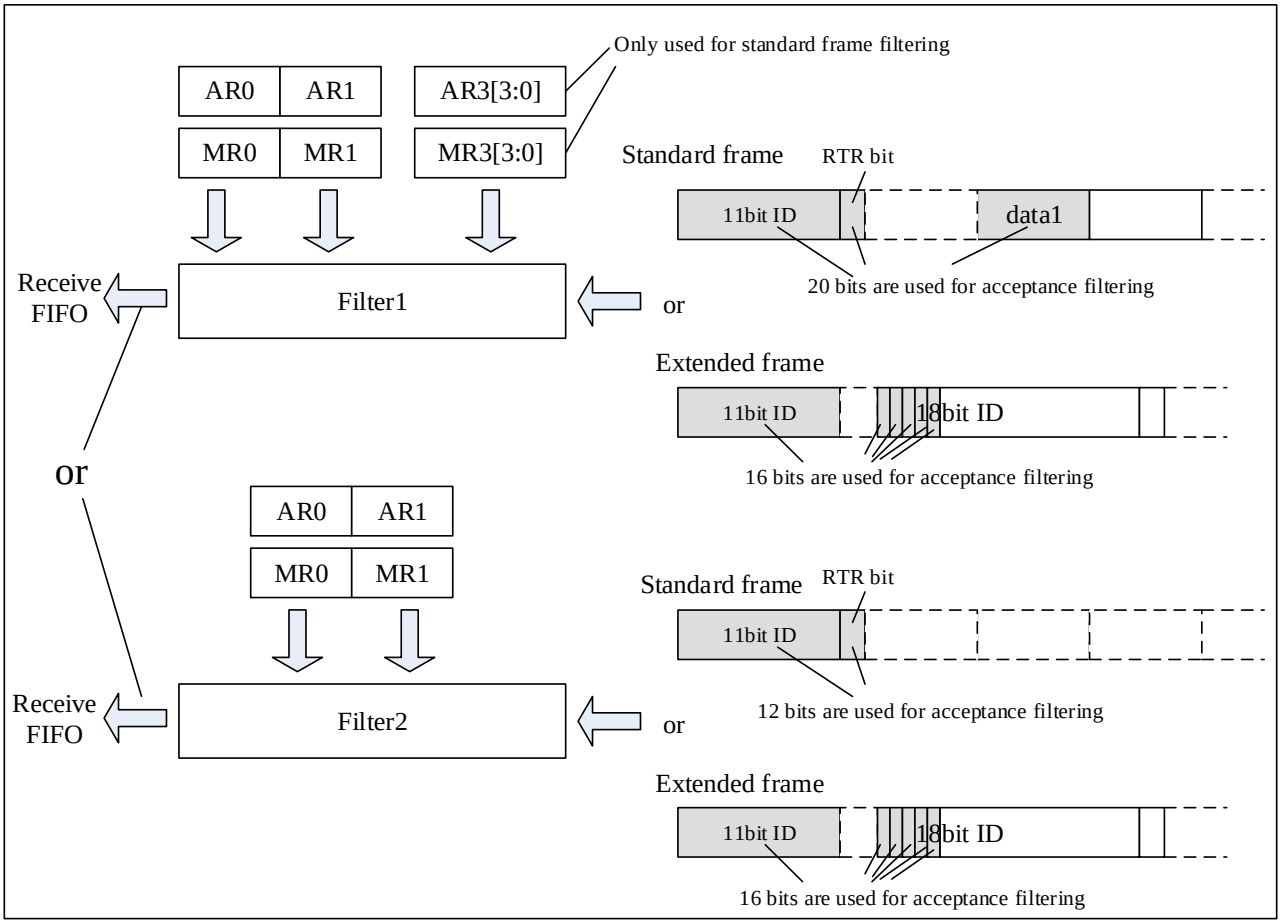
In the "1" position of the acceptance mask register, the corresponding bit of the identifier can be any value, such as the remote transmission request bit and the bits of data bytes 1 and 2.

Extended frame: If the received information is in extended frame format, all identification codes including RTR bits will be accepted and filtered. In order to successfully receive information, an acceptance signal must be sent after each bit is compared. The lowest two bits of CAN_IDMR3 and CAN_IDAR3 are not used, and these bits should be set to "do not affect".

Double filters:

This configuration can define two short filters. A piece of received information is compared with two filters to determine whether to put it in the receiving buffer. At least one filter sends out the receiving signal, and the received information is valid. The bit correspondence between the filter byte and the information byte depends on the currently received frame format.





Standard frame: If standard frame information is being received, the two defined filters are different. The first filter compares the entire standard identification code including the RTR bit and the first data byte of the message. The second filter only compares the entire standard identification code including the RTR bit.

In order to successfully receive the information, there should be at least one filter to indicate acceptance when comparing all individual bits. When the RTR bit or data length code is 0, it means that there is no data byte. No matter what, as long as the part from the start to the RTR bit is indicated to be received, the information can pass through filter 1.

If no data byte filtering is requested from the filter, the lower four bits of CAN_IDMR1 (AMR1) and CAN_IDMR3 (AMR3) must be set to 1 (no effect). When using the entire standard identification code including RTR bits, both filters work in same way.

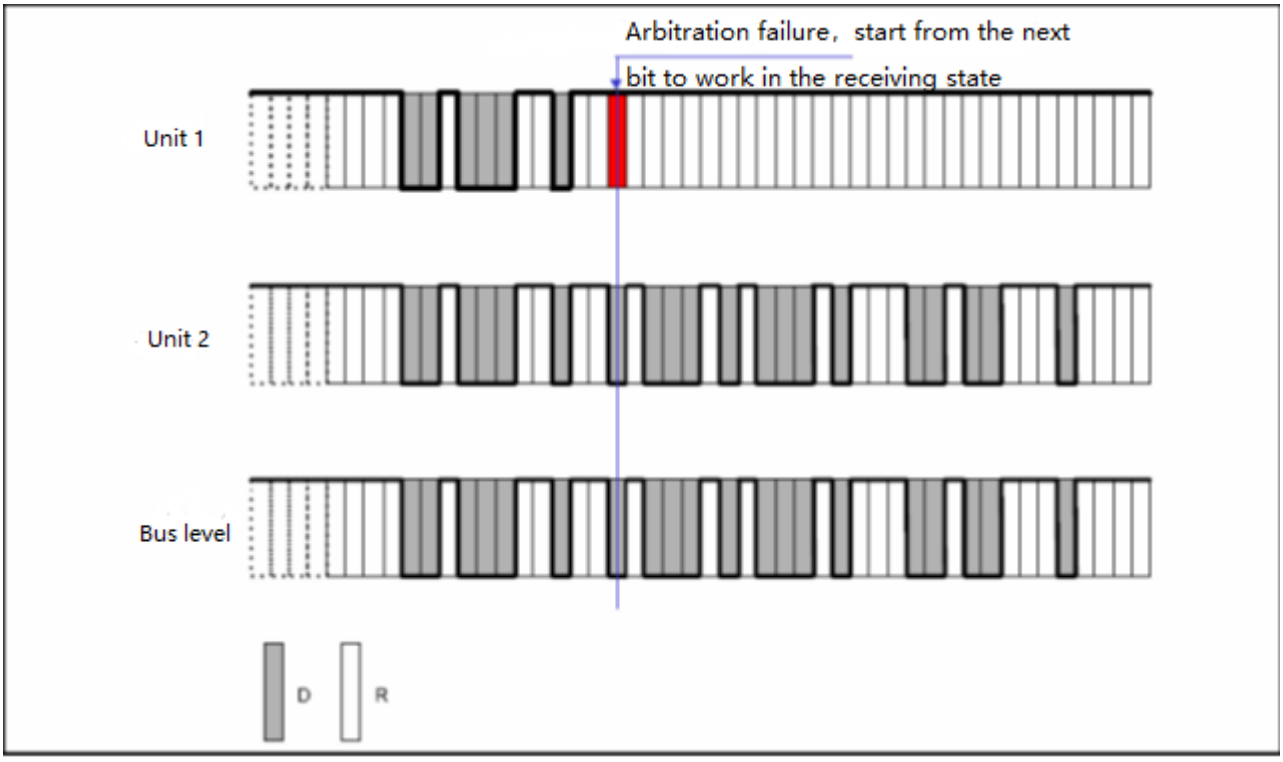
Extended frame: If the extended frame information is received, the two defined filters are the same. Both filters only compare the first two bytes of the extended identification code.

In order to receive information successfully, at least one filter indicates the reception when comparing all individual bits.

Frame	Single-filter	Double-filter

type			
Standard	Accepted message bits: - 11-bit identifier - RTR bit - The first data byte is 8 bits - The second data byte is 8 bits Acceptance code register and mask register used - CAN_IDAR0/ CAN_IDAR1 (high 4 bits) / CAN_IDAR2/ CAN_IDAR3 - CAN_IDMR0/ CAN_IDMR1 (high 4 bits) / CAN_IDMR2/ CAN_IDMR3 (The unused bits of the receive mask register should be set to 1)	Filter 1 Accepted message bits: - 11-bit identifier - RTR bit - The first data byte is 8 bits Acceptance code register and mask register used: - CAN_IDAR0/ CAN_IDAR1 Or CAN_IDAR3 low 4 bits - CAN_IDMR0/ CAN_IDMR1 Or CAN_IDMR3 low 4 bits	Filter 2 Message bits used for acceptance testing - 11-bit identifier - RTR bit Acceptance code register and acceptance mask register used: - CAN_IDAR2 or CAN_IDAR3 high 4 bits - CAN_IDMR2 or CAN_IDMR3 high 4 bits
Extended	Message bits used for acceptance: -11-bit basic identifier -18-bit extended identifier -RTR bit Acceptance code and acceptance mask used: - CAN_IDAR0/ CAN_IDAR1/ CAN_IDAR2 or CAN_IDAR3 high 6 bits - CAN_IDMR0/ CAN_IDMR1/ CAN_IDMR2 or CAN_IDMR3 high 6 bits (The unused bits of the acceptance mask register should be set to 1)	Filter 1: Message bits used for acceptance -11-bit basic identifier - The high 5 bits of the extended identifier : Acceptance code and acceptance mask used - CAN_IDAR0/ CAN_IDAR1 and CAN_IDMR0/ CAN_IDMR1	Filter 2: Message bit used to test acceptance - 11-bit basic identifier - The high 5 bits of the extended identifier Acceptance code and acceptance mask used - CAN_IDAR2/ CAN_IDAR3/ and CAN_IDMR2/ CAN_IDMR3

5.2.2.5 Arbitration



In the bus idle state, the unit that first starts to send a message gets the right to send. When multiple units start sending at the same time, each sending unit starts arbitration from the first bit of the arbitration section. The unit that continuously outputs the most dominant level can continue to send.

CAN uses lossless arbitration. When two masters start sending a new frame of information at the same time, arbitration is only performed in the arbitration domain. When the signal sampled by the sending host to the bus is a dominant level and what it sends is a recessive level, it will lose arbitration, stop sending information and switche to a receiving state.

5.2.2.6 Loopback self test

The loopback self-test mode is independent of the connection of the external system, and is sometimes used to check software to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input. The Rx input pin is ignored and the Tx output enters a recessive state (logic 1). When sending, CAN regards the message it sends itself as the message received from the remote node. In this state, CAN will ignore the bit sent in the ACK gap in the CAN frame response field to ensure that its own message is received correctly. Both send and receive interrupts are generated at the same time.

5.2.2.7 CAN self wakeup

When CAN itself is idle, the module does not receive any messages and all transmit buffers are empty, you can request to enter sleep mode by setting the CAN_CMR[5] bit to 1. In sleep mode, power consumption can be reduced by stopping all CAN clocks (except the clocks that access registers on the CPU side).

Only when CAN is in sleep mode (CAN_CMR[5] = 1, CAN_SLPAK = 1), wake-up function is enabled (CAN_WUP[0] = 1) and wake-up interrupt is enabled (CAN_IE[4]=1), wake-up interrupt of CAN may happen.

The time for the CAN controller to enter sleep mode depends on the fixed synchronization delay and its current state:

1. If there is a message buffer waiting to be sent, CAN will continue to send until all the sent message buffers are empty (successfully sent or aborted), and then enter the sleep mode;

2. If CAN is receiving, it continues to receive, and once the CAN bus is idle, it immediately enters sleep mode;

3. If CAN is neither transmitting nor receiving, it will immediately enter sleep mode;

If the wake-up interrupt enable CAN_WUP[0] bit is not yet set, CAN will mask any signals it detects on CAN. The Rx pin is therefore set to a recessive state, which will lock CAN in sleep mode. CAN_WUP[0] must be set before entering sleep mode in order to function. When any dominant signal on the CAN bus is detected, CAN exits sleep mode, the clock is turned on and the wake-up interrupt status CAN_IF[4] is set.

CAN can exit sleep mode (wake up) only when the following situations occur:

1. If the CAN bus is valid and CAN_WUP[0] = 1;

2. Clear CAN_CMR[5] bit;

After exiting sleep mode, configure the wakeup function to enable CAN_WUP[0] = 0.

When the CAN controller is in sleep mode, the length of the wakeup pulse on the CAN bus is set by configuring CAN_WUP[1]. When CAN_WUP[1]=0, CAN is waked up by any dominant signal on the CAN bus, and when CAN_WUP[1]=1, CAN only wakes up when the dominant pulse length on the CAN bus is greater than or equal to 2us. 2us is filtered by external crystal oscillator clock count.

5.2.2.8 CAN wakeup system

If the system enters sleep mode (sleepdeep), it can only be waked up by external dominant pulse wakeup and exit sleep mode. At this time, the CAN wakeup interrupt enable must be configured(CAN_IE[4]=1), otherwise the system cannot be waked up.

Considering that when the system enters sleep mode, it may be accidentally waked up by external noise signals, the software can select whether to perform hardware filtering by configuring the CAN_SWU_FILT register, and the filtering time is 1.5us. This filter is only used as the CAN pulse filter when the system wakes up, and it is invalid in other cases.

5.2.3 Register map

CAN module address range：0x5005_0000~0x5005_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	CAN_MOD	R/W	CAN mode register	0x01
0x04	CAN_CMR	R/W	CAN command register	0x00
0x08	CAN_SR	RO	CAN status register	0x3C
0x0C	CAN_IF	RO	CAN interupt register	0x00
0x10	CAN_IE	R/W	CAN interupt enable register	0x00
0x18	CAN_BTR0	R/W	CAN bus timing register 0	0x00
0x1C	CAN_BTR1	R/W	CAN bus timing register 1	0x00
0x20	CAN_SLPAK	RO	CAN sleep mode acknowledge register	0x00
0x24	CAN_WUP	R/W	CAN self-wakeup register	0x00
0x2C	CAN_ALC	RO	CAN arbitration lost capture register	0x00
0x30	CAN_ECC	RO	CAN error code capture register	0x00
0x34	CAN_EMLR	R/W	CAN error alarm limit register	0x60
0x38	CAN_RXERR	R/W	CAN rx error count register	0x00
0x3C	CAN_TXERR	R/W	CAN tx error count register	0x00
0x40	CAN_IDAR0	R/W	CAN acceptance code register0	0x00
0x40	CAN_FRCTL	R/W	CAN data information register	0x00
0x44	CAN_IDAR1	R/W	CAN acceptance code register1	0x00
0x44	CAN_ID0	R/W	CAN identifier register0	0x00
0x48	CAN_IDAR2	R/W	CAN acceptance code register2	0x00
0x48	CAN_ID1	R/W	CAN identifier register1	0x00
0x4C	CAN_IDAR3	R/W	CAN acceptance code register3	0x00
0x4C	CAN_ID2	R/W	CAN identifier register2	0x00
0x50	CAN_IDMR0	R/W	CAN acceptance mask register0	0x00
0x50	CAN_ID3	R/W	CAN identifier register3	0x00
0x54	CAN_IDMR1	R/W	CAN acceptance mask register1	0x00
0x54	CAN_DATA0	R/W	CAN data register0	0x00
0x58	CAN_IDMR2	R/W	CAN acceptance mask register2	0x00
0x58	CAN_DATA1	R/W	CAN data register1	0x00
0x5C	CAN_IDMR3	R/W	CAN acceptance mask register3	0x00
0x5C	CAN_DATA2	R/W	CAN data register2	0x00
0x60	CAN_DATA3	R/W	CAN data register3	0x00
0x64	CAN_DATA4	R/W	CAN data register4	0x00
0x68	CAN_DATA5	R/W	CAN data register5	0x00
0x6C	CAN_DATA6	R/W	CAN data register6	0x00
0x70	CAN_DATA7	R/W	CAN data register7	0x00
0x74	CAN_RMC	RO	CAN rx fifo data count register	0x00

Address offset	Register name	R/W	Functional description	Initial value
0x78	CAN_ENABLE	R/W	CAN enable register	0x00
0x80	CAN_CLRISR	R/W	CAN interrupt status clear register	0x00
0x84	CAN_CLRECC	R/W	CAN error capture code clear register	0x00
0x88	CAN_SWU_FILT	R/W	CAN system wakeup filter register	0x00

5.2.4 Register description

5.2.4.1 CAN_MOD register

Address	Bit	Name	R/W	Reset value	Description
0x00	7:4	Reserved	——	——	Reserved
	3	CAN_MOD_AFM	R/W	0x0	Acceptance filter mode: 0x1: Select a single acceptance filter (32-bit length) 0x0: Select two acceptance filters (each with 16 bits activated)
	2	CAN_MOD_STM	R/W	0x0	Loopback self-test mode: 0x1 : Enable loopback self-test 0x0 : Loopback self-test is prohibited When this bit is set, CAN performs an internal loopback that can be used for self-test operations. The bit stream output of the transmitter flows internally back to the receiver. The loopback self-test mode is independent of the connection of the external system, and is sometimes used to check software to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input. The Rx input pin is ignored and the Tx output enters a recessive state (logic 1). When transmitting, CAN operates as usual, taking the message it transmits itself as the message received from the remote node. In this state, CAN will ignore the bit sent in the ACK gap in the CAN frame response field to ensure that its own message is received correctly, and at the same time generate transmission and reception interrupts.
	1	CAN_MOD_LOM	R/W	0x0	Listen only mode 0x1:Even if the message is successfully received, the CAN controller will not send a response signal to the bus; the error counter stops at the current value

					0x0：normal working mode Listen-only mode will not transmit a response signal to the bus, data will not be written, and no receive interrupt will be generated, but a bus error (response delimiter error) will occur.
	0	CAN_MOD_RM	R/W	0x1	Set mode: 0x1:After setting, stop the current receiving/transmitting information and enter the setting mode. The software can configure the relevant registers in the setting mode 0x0：Normal working mode

5.2.4.2 CAN_CMR register

Address	Bit	Name	R/W	Reset value	Description
0x04	7:6	Reserved	——	——	Reserved
	5	CAN_CMR_SPM	R/W	0x0	Self sleep mode request 0x1 ： Sleep mode request, when CAN bus is idle, CAN enters sleep mode 0x0 ： During operation, CAN works normally This bit requests CAN to enter sleep mode, which is an internal power saving mode. When the CAN bus is idle,the module does not receive any messages and all the sending buffers are empty, the sleep mode request is accepted The sleep mode remains valid until the bit is cleared by the CPU or according to the setting of CAN_WUP_ENABLE, CAN detects that it is valid on the bus and clears itself
	4	CAN_CMR_SRR	R/W	0x0	Self-accepting request 0x1：Transmitting information can be received at the same time 0x0：No action Acceptance filter settings, transmitting interrupts, and receiving interrupts are also valid in this mode
	3	CAN_CMR_CDO	R/W	0x0	Clear data overflow 0x1：Clear data overflow status bit CAN_SR_DOS 0x0：No action The overflow state only occurs once during the overflow process. If the overflow state is still in the overflow state when the overflow state is cleared, the overflow state can no longer occur. This bit will be automatically cleared after configuration.

	2	CAN_CMR_RRB	R/W	0x0	Clear data buffer status and receive interrupt 0x1: Clear data buffer status and receive interrupt 0x0: No action Every time you read a piece of data in the receive buffer, you must configure this bit to 1, otherwise you cannot continue to read the next piece of data. This bit is automatically cleared.
	1	CAN_CMR_AT	R/W	0x0	Abort transmitting 0x1: Abort the next data after the current transmission, used to stop the automatic retransmission action 0x0: No action During the suspension process, if you want to know whether the message was successfully transmitted, you can check it through the transmitting complete status bit. This should be after the transmit buffer status bit is set or the transmit interrupt is generated. It should be noted that even if the message is aborted because the transmit buffer status bit becomes 0, a transmit interrupt will be generated. After this bit is configured, it will be automatically cleared when CAN_CMR_TR is set.
	0	CAN_CMR_TR	R/W	0x0	Transmit request 0x1:Transmit message 0x0: No action This bit is automatically cleared after transmitting the configuration, and needs to be reset to 1 for the next transmitting.

5.2.4.3 CAN_SR register

Address	Bit	Name	R/W	Reset value	Description
0x08	7	CAN_SR_BS	RO	0x0	Bus state 0x1: Bus is closed, CAN exits bus activity 0x0: The bus is turned on, CAN joins the bus activity When the transmission error counter exceeds the limit (255), the bus status bit is set to 1 (bus off), the CAN controller will set the bit mode to 1, and an error alarm interrupt is generated.At this time, the transmission error counter is set to 127, and the reception error counter is cleared. This mode will remain until the CPU clears the set mode. After completing these, the CAN controller will wait for the

					minimum time specified in the protocol (128 bus idle signals) by transmitting a count of the error counter minus one. After that, the bus status bit is cleared (the bus is turned on), the error status bit is set to 0, the error counter is reset and an error alarm interrupt is generated During this period, the TX error counter can be read to know the status information about the bus close repair..
	6	CAN_SR_ES	RO	0x0	Error status 0x1: Error; at least one error counter is full or exceeds the error alarm limit value 0x0: Two error counters are below the alarm limit value According to the CAN2.0B protocol, errors detected during reception and transmission will affect the error counter. The error status bit is set when at least one error counter is full or exceeds the CPU alarm limit CAN_EMLR. When the interrupt is enabled, an error alarm interrupt will be generated. The default value after CAN_EMLR hardware reset is 96
	5	CAN_SR_TS	RO	0x0	Transmit status 0x1: Transmit, CAN controller is transmitting information 0x0: Idle, there is no message to send
	4	CAN_SR_RS	RO	0x0	Receiving status 0x1: Receive, CAN controller is receiving information 0x0: Idle, no messages being received If the receiving status bit and the transmitting status bit are both 0, the CAN bus is idle. If these two bits are both 1, the controller is waiting for the next idle. After the hardware is started, 11 consecutive recessive bits must be detected until the idle state comes. After the bus is closed, 128 consecutive recessive bits of 11 bits will be generated.
	3	CAN_SR_TCS	RO	0x1	Transmitting completed status 0x1 : Finished, the last transmitting request was successfully processed 0x0: Unfinished, the current transmitting request has not been processed Once the transmitting request bit or the self-receiving request bit is set to 1, the transmitting completion status bit will be set to 0. The transmission completed status bit will remain at 0 until the transmission is successful.
	2	CAN_SR_TBS	RO	0x1	Transmit buffer status 0x1: Release; CPU can write information to transmit buffer 0x0: Locked; the CPU cannot access the transmitting buffer, and there is information waiting to be sent or

					being sent If the CPU tries to write to the transmit buffer when the transmit buffer status bit is 0, the written byte will not be accepted and will be lost without any prompt.
	1	CAN_SR_DOS	RO	0x0	Data overflow status 0x1: About to overflow, RXFIFO has more than 96Bytes of data 0x0: No overflow When the information to be received has successfully passed the acceptance filter, the CAN controller needs to have enough space in the RXFIFO to store the information descriptor and each received data byte. If there is not enough space to store the information, the information will be lost.The size of CANRXFIFO is 128Bytes, and the data overflow status will be sent out at 96Bytes.If the message is not successfully received (for example, due to an error), there is no indication of a data overflow condition.
	0	CAN_SR_RBS	RO	0x0	Data buffer status 0x1: RXFIFO has available information 0x0: Empty, no information available After reading all the information in the RXFIFO, this bit is cleared.

5.2.4.4 CAN_IF register

Address	Bit	Name	R/W	Reset value	Description
0x0C	7	CAN_IF_BEI	RO	0x0	Bus error interrupt status: 0x1: This bit is set when the CAN controller detects a bus error and the interrupt is enabled 0x0: No bus error interrupt
	6	CAN_IF_ALI	RO	0x0	Arbitration lost interrupt status: 0x1: This bit is set when the CAN controller loses arbitration and the interrupt is enabled 0x0: No arbitration lost interrupt
	5	CAN_IF_EPI	RO	0x0	Negative error interrupt status: 0x1: When at least one error counter of the CAN controller is full or exceeds the error alarm limit value, or the passive error count value returns to the active error count value, this bit is set when the interrupt is enabled 0x0: No information error interrupt
	4	CAN_IF_WUPI	RO	0x0	Wakeup interrupt status

					0x1: When CAN is in sleep mode and wake-up enable is turned on, it is set when the CAN bus is detected as valid 0x0: No wakeup detected when in sleep mode
	3	CAN_IF_DOI	RO	0x0	Data is about to overflow interrupt status 0x1: This bit is set when the data FIFO reaches more than 96Byte and the interrupt is enabled 0x0: No data is about to overflow
	2	CAN_IF_EI	RO	0x0	Error interrupt status 0x1: Set when any error occurs and the interrupt is enabled 0x0: No error occurs
	1	CAN_IF_TI	RO	0x0	Transmit interrupt status 0x1: The transmit buffer can be filled with the next transmit data and set when the interrupt is enabled 0x0: The transmit buffer is working
	0	CAN_IF_RI	RO	0x0	Receive interrupt status 0x1: Set when the receive buffer is not empty and the interrupt is enabled 0x0: Receive buffer empty The function of this bit is equivalent to CAN_SR_RBS. So CAN_CMR_RRB can temporarily clear this bit. If there is still available information in the data buffer after the release command is executed, this bit will be reset again until the buffer is empty.

5.2.4.5 CAN_IE register

Address	Bit	Name	R/W	Reset value	Description
0x10	7	CAN_IF_BEIE	R/W	0x0	Bus error interrupt enable 0x1: Enable 0x0: Disable
	6	CAN_IF_ALIE	R/W	0x0	Arbitration lost interrupt enable 0x1: Enable 0x0: Disable
	5	CAN_IF_EPIE	R/W	0x0	Negative error interrupt enable 0x1: Enable 0x0: Disable
	4	CAN_IF_WUIE	R/W	0x0	Wakeup interrupt enable 0x1: Enable 0x0: Disable
	3	CAN_IF_DOIE	R/W	0x0	Data is about to overflow interrupt enable

					0x1：Enable 0x0：Disable
	2	CAN_IF_EIE	R/W	0x0	Error interrupt enable 0x1：Enable 0x0：Disable
	1	CAN_IF_TIE	R/W	0x0	Transmit interrupt enable 0x1：Enable 0x0：Disable
	0	CAN_IF_RIE	R/W	0x0	Receive interrupt enable 0x1：Enable 0x0：Disable

5.2.4.6 CAN_BTR0 register

Address	Bit	Name	R/W	Reset value	Description
0x18	7:6	CAN_BTR0_SJW	R/W	0x0	Sync jump width In order to compensate for the phase shift between the clock oscillators of different bus controllers, any bus controller must resynchronize on the edge of the relevant signal currently being transmitted. The synchronization jump width defines the maximum number of clock cycles that can be shortened or extended by resynchronization per bit cycle. Synchronization jump width time = clock preset value* (2 *CAN_BTR0_SJW [1]+ CAN_BTR0_SJW [0]+1).
	5:0	CAN_BTR0_BRP	R/W	0x0	Clock preset value CAN communication clock cycle = 2*CAN module clock cycle* (CAN_BTR0_BRP+1).

5.2.4.7 CAN_BTR1 register

Address	Bit	Name	R/W	Reset value	Description
0x1C	7	CAN_BTR1_SAM	R/W	0x0	Internal sampling times 0x1：Sampling 3 times, taking the middle one as the real data to improve reliability 0x0：Sample 1 time

	6:4	CAN_BTR1_TSEG2	R/W	0x0	Time period 2, used to configure the baud rate
	3:0	CAN_BTR1_TSEG1	R/W	0x0	Time period 1, used to configure the baud rate CAN baud rate = 1/(2*CAN module clock period*(1+CAN_BTR0_BRP)*(3+ CAN_BTR1_TSEG1+ CAN_BTR1_TSEG2))

5.2.4.8 CAN_SLPAK register

Address	Bit	Name	R/W	Reset value	Description
0x20	7:1	Reserved	——	——	Reserved
	0	CAN_SLPAK	RO	0x0	Sleep mode confirmation status 0x1 : In sleep mode 0x0 : Normal working This flag shows whether the CAN module has entered sleep mode. It is used as the handshake flag for CAN_CMR_SPM sleep mode request. When CAN_CMR_SPM = 1, CAN_SLPAK = 1, the sleep mode is valid. According to the CAN_WUP_ENABLE setting, if a signal on the CAN bus is detected in sleep mode, CAN will clear the flag. Clearing the CAN_CMR_SPM bit will also reset the CAN_SLPAK bit.

5.2.4.9 CAN_WUP register

Address	Bit	Name	R/W	Reset value	Description
0x24	7:2	Reserved	——	——	Reserved
	1	CAN_WUP_MODE	R/W	0x0	Self wakeup filter 0x1: Wakeup only when the dominant pulse length on the CAN bus is greater than 2us. 0x0: CAN is awakened by any dominant signal on the bus
	0	CAN_WUP_ENABLE	R/W	0x0	Self wakeup enable 0x1: Enable 0x0: Disable

5.2.4.10 CAN_ALC register

Address	Bit	Name	R/W	Reset value	Description
0x2C	7:5	Reserved	——	——	Reserved
	4:0	CAN_ALC_CODE	RO	0x0	Arbitration for lost information When the arbitration is lost, the corresponding arbitration lost interrupt will be generated. At the same time, the current bit position of the bit stream is captured and sent to the arbitration loss capture register. Until the software reads this value, the contents of the register will not change. The new arbitration lost interrupt is valid until the arbitration lost capture register is read once.

CAN_ALC_CODE[4] ~ CAN_ALC_CODE[0]					说明
0	0	0	0	0	Arbitration is lost in bit1 of the identification code
0	0	0	0	1	Arbitration is lost in bit2 of the identification code
0	0	0	1	0	Arbitration is lost in bit3 of the identification code
0	0	0	1	1	Arbitration is lost in bit4 of the identification code
0	0	1	0	0	Arbitration is lost in bit5 of the identification code
0	0	1	0	1	Arbitration is lost in bit6 of the identification code
0	0	1	1	0	Arbitration is lost in bit7 of the identification code
0	0	1	1	1	Arbitration is lost in bit8 of the identification code
0	1	0	0	0	Arbitration is lost in bit9 of the identification code
0	1	0	0	1	Arbitration is lost in bit10 of the identification code
0	1	0	1	0	Arbitration is lost in bit11 of the identification code
0	1	0	1	1	Arbitration is lost in the SRTR bit

0	1	1	0	0	Arbitration is lost in the IDE bit
0	1	1	0	1	Arbitration is lost in bit12 of the identification code
0	1	1	1	0	Arbitration is lost in bit13 of the identification code
0	1	1	1	1	Arbitration is lost in bit14 of the identification code
1	0	0	0	0	Arbitration is lost in bit15 of the identification code
1	0	0	0	1	Arbitration is lost in bit16 of the identification code
1	0	0	1	0	Arbitration is lost in bit17 of the identification code
1	0	0	1	1	Arbitration is lost in bit18 of the identification code
1	0	1	0	0	Arbitration is lost in bit19 of the identification code
1	0	1	0	1	Arbitration is lost in bit20 of the identification code
1	0	1	1	0	Arbitration is lost in bit21 of the identification code
1	0	1	1	1	Arbitration is lost in bit22 of the identification code
1	1	0	0	0	Arbitration is lost in bit23 of the identification code
1	1	0	0	1	Arbitration is lost in bit24 of the identification code
1	1	0	1	0	Arbitration is lost in bit25 of the identification code
1	1	0	1	1	Arbitration is lost in bit26 of the identification code
1	1	1	0	0	Arbitration is lost in bit27 of the identification code
1	1	1	0	1	Arbitration is lost in bit28 of the identification code
1	1	1	1	0	Arbitration is lost in bit29 of the identification code
1	1	1	1	1	Arbitration is lost in the RTR bit

5.2.4.11 CAN_ECC register

Address	Bit	Name	R/W	Reset value	Description
0x30	7:6	CAN_ECC_MODE	RO	0x0	Error 0x3: Other errors 0x2: Fill error 0x1: Error fromat error 0x0: Bit error
	5	CAN_ECC_DIR	RO	0x0	Error direction 0x1: An error occurred while receiving 0x0: An error occurred while transmitting
	4:0	CAN_ECC_CODE	RO	0x0	Error code See the table below for details.

CAN_ECC_CODE[4] ~ CAN_ECC_CODE[0]					Error
0	0	0	1	1	Frame start
0	0	0	1	0	ID28-ID21
0	0	1	1	0	ID20-ID18
0	0	1	0	0	SRTR bit
0	0	1	0	1	IDE bit
0	0	1	1	1	ID17~ID13
0	1	1	1	1	ID12-ID5
0	1	1	1	0	ID4-ID0
0	1	1	0	0	RTR bit

CAN_ECC_CODE[4] ~ CAN_ECC_CODE[0]					Error
0	1	1	0	1	Reserved bit 1
0	1	0	0	1	Reserved bit 0
0	1	0	1	1	Data length code
0	1	0	1	0	Data area
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	Response channel
1	1	0	1	1	Response delimiter
1	1	0	1	0	Frame end
1	0	0	1	0	Suspend
1	0	0	0	1	Active error flag
1	0	1	1	0	Negative error flag
1	0	0	1	1	Domination control bit error
1	0	1	1	1	Error delimiter
1	1	1	0	0	Overload flag

5.2.4.12 CAN_EMLR register

Address	Bit	Name	R/W	Reset value	Description
0x34	7:0	CAN_EMLR	R/W	0x60	False alarm limit value The count value reached when the error alarm is issued. This register can only be written in CAN set mode.

5.2.4.13 CAN_RXERR register

Address	Bit	Name	R/W	Reset value	Description
0x38	7:0	CAN_RXERR	R/W	0x00	Receive error count value Normal working mode is read-only, this register can only be written in CAN set mode. If bus is closed, this register is initialized. When the bus is off, writing to this register is invalid.

5.2.4.14 CAN_TXERR register

Address	Bit	Name	R/W	Reset value	Description
0x3C	7:0	CAN_TXERR	R/W	0x00	Transmit error count value Normal working mode is read-only, this register can only be written in CAN set mode. If a bus is closed, this register is initialized to 127 for calculating the minimum time defined by the bus (128 bus idle signals). During this period of time, reading the counter will reflect the status information of the bus close recovery. When the bus is off, writing to this register is invalid. During the bus close period, write access to this register 0-254 will clear the bus close flag. After the reset mode is cleared, the controller will wait for an 11-bit continuous recessive bit (bus is idle). Writing 255 to this register will cause a bus close event. If the reset mode is entered before the bus close is restored, the bus close remains valid and the technical value is locked.

5.2.4.15 CAN_IDAR0/CAN_IDAR1/CAN_IDAR2/CAN_IDAR3 register

Address	Bit	Name	R/W	Reset value	Description
0x40 0x44 0x48 0x4C	7:0	CAN_IDAR0 CAN_IDAR1 CAN_IDAR2 CAN_IDAR3	R/W	0x00	Acceptance code register, see the description of acceptance filtering chapter for details. This register can only be written in CAN set mode.

5.2.4.16 CAN_IDMR0/CAN_IDMR1/CAN_IDMR2/CAN_IDMR3 register

Address	Bit	Name	R/W	Reset value	Description
0x50 0x54	7:0	CAN_IDMR0 CAN_IDMR1	R/W	0x00	Acceptance mask register, see the description of acceptance filtering chapter for details. This register can only be

0x58 0x5C		CAN_IDMR2 CAN_IDMR3			written in CAN set mode.

5.2.4.17 CAN_RMC register

Address	Bit	Name	R/W	Reset value	Description
0x74	7:6	Reserved	——	——	Reserved
	5:0	CAN_RMC	RO	0x00	Receive message counter Reflects the number of information available in the receiving buffer, its value is increased by 1 each time it is received, and 1 is decreased each time the receiving buffer is released. After each reset, this register is cleared to 0.

5.2.4.18 CAN_ENABLE register

Address	Bit	Name	R/W	Reset value	Description
0x78	7:1	Reserved	——	——	Reserved
	0	CAN_ENABLE	R/W	0x00	CAN module enable 0x1: Enable 0x0: Disable When CAN is enabled, select the external IO port to automatically switch to the CAN function.Before configuring this bit, configure CAN to enter the set mode. After configuring this bit again, configure CAN to exit the set mode, and CAN can start working normally.

5.2.4.19 CAN_CLRISR register

Address	Bit	Name	R/W	Reset value	Description

0x80	7	CAN_CLRISR_BEI	R/W	0x00	Write 1 to clear the bus interrupt status BEI, the value read is always 0
	6	CAN_CLRISR_ALI	R/W	0x00	Write 1 to clear the arbitration loss interrupt status ALI, the read value is always 0
	5	CAN_CLRISR_EPI	R/W	0x00	Write 1 to clear the negative error interrupt state EPI, the value read is always 0
	4	CAN_CLRISR_WUPI	R/W	0x00	Write 1 to clear the wake-up interrupt status WUPI, the value read is always 0
	3	CAN_CLRISR_DOI	R/W	0x00	Write 1 to clear the data overflow interrupt status DOI, the read value is always 0
	2	CAN_CLRISR_EI	R/W	0x00	Write 1 to clear the error interrupt status EI, the read value is always 0
	1	CAN_CLRISR_TI	R/W	0x00	Write 1 to clear the transmit interrupt status TI, the read value is always 0
	0	Reserved	— —	——	Reserved

5.2.4.20 CAN_CLRECC register

Address	Bit	Name	R/W	Reset value	Description
0x84	7:1	Reserved	— —	——	Reserved
	0	CAN_CLRECC	R/W	0x00	Write 1 to clear the error code capture register CAN_ECC, the value read is always 0

5.2.4.21 CAN_FRCTL register

Address	Bit	Name	R/W	Reset value	Description
0x40	7	CAN_FRCTL_FF	R/W	——	Frame format 0x1:Extended frame 0x0: Standard frame
	6	CAN_FRCTL_RTR	R/W	0x00	Remote frame 0x1: Remote frame 0x0: Data frame
	5:4	Reserved	— —	——	Reserved

	3:0	CAN_FRCTL_DLC	R/W		Data length Since the RTR bit is set at the beginning of the remote frame transmission (remote), the data length code is not considered. This makes the number of received/sent data bytes 0. If two CAN controllers use the same identification code to start remote frame transmission at the same time, the data length code must be correctly stated to avoid bus errors. For compatibility, data length codes greater than 8 are not available. If it is greater than 8, it will be counted as 8 bytes. Write this register to write the data to be sent, and read this register to read the received data.

5.2.4.22 CAN_ID0/CAN_ID1/CAN_ID2/CAN_ID3 register

Address	Bit	Name	R/W	Reset value	Description
0x44 0x48 0x4C 0x50	7:0	CAN_ID0 CAN_ID1 CAN_ID2 CAN_ID3	R/W	0x00	CAN ID Write this register to write the data to be sent, and read this register to read the received data.

5.2.4.23 CAN_DATA0~ CAN_DATA7 register

Address	Bit	Name	R/W	Reset value	Description
0x54 0x58 0x5C 0x60 0x64 0x68 0x6C 0x70	7:0	CAN_DATA0 CAN_DATA1 CAN_DATA2 CAN_DATA3 CAN_DATA4 CAN_DATA5 CAN_DATA6 CAN_DATA7	R/W	0x00	CAN ID data Write this register to write the data to be sent, and read this register to read the received data.

5.2.4.24 CAN_SWU_FILT register

Address	Bit	Name	R/W	Reset value	Description
0x88	7:1	Reserved	——	——	Reserved
	0	CAN_SWU_FILT	R/W	0x00	In system sleep mode, CAN Rx port filtering is enabled 0x1: Filter 0x0: Not filter The filter is directly filtered through the analog port, and the maximum is 1.5us.

5.2.5 Example program

```
//Main function:
int main(void)
{
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_5);
    gpio_bit_reset(GPIOG,GPIO_PIN_5);        //Pulldown CAN_STB
    can_boot_config();//CAN initialization configuration, including baud rate configuration, frame format selection and initialization of data frame id,length,data content,etc.
    can_send_frame();                        //CAN transmit frame

    can_sleep_config(1,0,ENABLE);            //Configure CAN sleep
    can_int_enable(CAN_IE_WUIE);             //Enable wakeup interrupt

    if(can_sleep_request())                  //CAN sleep request
    {
        gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);
        gpio_bit_set(GPIOE,GPIO_PIN_5); //Turn on the light
    }

    sleep_deep();                            //System enters sleep mode
    CAN_FILTER_EN = 0x01;                 //Set Rx port filter wakeup

//  rtc_init(RTC_ENABLE | RTC_CLK_SEL_32K | RTC_INT_ENABLE | RTC_TRG_CLK_DIV(0),0x7D00);
    //RTC initialiazation
}
//Subfunction:
/***************CAN initialization parameter configuration*****************/
void can_boot_config()
{
```

```
    /*can parameter initialization*/
  can_parameter_struct can_parameter_init =
  {
    CAN_NOMARL,                 //Work mode
    CAN_BTR0_SJW_SET(0),            //Sync jump width
    CAN_BTR0_BRP_SET(1),            //CAN baud rate = CAN module clock frequency /
(2*(1+CAN_BTR0_BRP_SET(x))*(3+CAN_BTR1_TSEG1_SET(n)+CAN_BTR1_TSEG2_SET(m)))
    CAN_BTR1_TSEG1_SET(3),      //Time period, used to configure the baud rate
  //CAN_BTR0_BRP_SET(x),x=0~63;CAN_BTR1_TSEG1_SET(n),n=0~15;CAN_BTR1_TSEG2_SET(m),m=0~7
    CAN_BTR1_TSEG2_SET(2),      //Time period, used to configure the baud rate
    CAN_BTR1_SAM_SET(1),            //Internal sampling times
    100,                            //Error alarm limit value
    CAN_WUIE_ENABLE | CAN_ERR_ALL_ENABLE | CAN_RIE_ENABLE | CAN_TIE_ENABLE,
                  // Interrupt enable
    CAN_WU_NVIC_ENABLE | CAN_ERR_NVIC_ENABLE | CAN_RX_NVIC_ENABLE | CAN_TX_NVIC_ENABLE
        // External interrupt enable
  };
    /*CAN filter parameter initialization*/
  can_filter_parameter_struct can_filter_parameter =
  {
    CAN_DOUBLE_FILTER,      //Select two acceptance filters
    CAN_EXTENDED_FRAME, //Standard frame
    CAN_DATA_FRAME,         //Data frame
    ENABLE,                 //RTR bit mask enable
    /* Single filter setting*/
    0x12345678,                 //Single filter id,Standard frame filtering:11 bits
valid; Extended frame filtering:29 bits valid
    0xff,               //Filter acceptance data 0, used for standard frame filtering
    0xff,               // Filter acceptance data 1, used for standard frame filtering
    0xffff,                     //Single filter mask acceptance id, standard frame
filtering: 11 bits valid; Extended frame filtering: 29 bits valid
    0xff,           //Filter mask acceptance data 0, used for standard frame filtering
    0xff,           // Filter mask acceptance data 1, used for standard frame filtering
    /* double filter setting */
    0x12345678,                 //Double filters id0, standard frame filtering: 11 bits
valid; Extended frame filtering: 29 bits ID valid acceptance bits ID28~13
    0x98765432,                 // Double filters id0, standard frame filtering: 11 bits
valid; Extended frame filtering: 29 bits ID valid acceptance bits ID28~13
    0xff,               //Filter acceptance data 0, used for standard frame filtering
    0xffff,           //Double filters mask acceptance id0, standard frame filtering: 11
bits valid; Extended frame filtering: 16 bits valid
    0xffff,           //Double filters mask acceptance id0, standard frame filtering: 11
bits valid; Extended frame filtering: 16 bits valid
    0xff            //Filter mask acceptance data 0, used for standard frame filtering
```

```
    };
    can_clock_sel(CAN_XTAL_CLOCK);//Select external crystal oscillator as clock source
    can_init(can_parameter_init,can_filter_parameter)       ;//CAN register initialization
configuration
    can_transmit_data.tx_ff = CAN_STANDARD_FRAME;   //Standard frame, Extended frame
    can_transmit_data.tx_ft = CAN_DATA_FRAME;           //Data frame
    can_transmit_data.tx_sfid = 0x7df;
    can_transmit_data.tx_efid = 0x12345678;
    can_transmit_data.tx_dlen = 8;
    can_transmit_data.tx_data[0] = 0x11;
    can_transmit_data.tx_data[1] = 0x22;
    can_transmit_data.tx_data[2] = 0x33;
    can_transmit_data.tx_data[3] = 0x44;
    can_transmit_data.tx_data[4] = 0x55;
    can_transmit_data.tx_data[5] = 0x66;
    can_transmit_data.tx_data[6] = 0x77;
    can_transmit_data.tx_data[7] = 0x88;
    can_transmit_message(&can_transmit_data);           //CAN transmits data
}
/***************CAN transmits frame*****************/
void can_send_frame()
{
    can_transmit_data.tx_ff = can_receive_data.rx_ff;       //Standard frame, Extended frame
    can_transmit_data.tx_ft = can_receive_data.rx_ft;           //Data frame
    if(gpio_bit_get(GPIOC,GPIO_PIN_5))
    {
        can_transmit_data.tx_sfid = 0x555;
        can_transmit_data.tx_data[0] = 0x11;
        can_transmit_data.tx_data[1] = 0x22;
        can_transmit_data.tx_data[2] = 0x33;
        can_transmit_data.tx_data[3] = 0x44;
        can_transmit_data.tx_data[4] = 0x55;
        can_transmit_data.tx_data[5] = 0x66;
        can_transmit_data.tx_data[6] = 0x77;
        can_transmit_data.tx_data[7] = 0x88;
    }
    else
    {
        can_transmit_data.tx_sfid = 0x56a;
        can_transmit_data.tx_data[0] = 0x12;
        can_transmit_data.tx_data[1] = 0x34;
        can_transmit_data.tx_data[2] = 0x56;
        can_transmit_data.tx_data[3] = 0x78;
        can_transmit_data.tx_data[4] = 0x90;
```

```
        can_transmit_data.tx_data[5] = 0x12;
        can_transmit_data.tx_data[6] = 0x34;
        can_transmit_data.tx_data[7] = 0x56;
    }
    can_transmit_message(&can_transmit_data);           //CAN transmits data
    can_int_flag_clr(CAN_IF_TI);
    can_transmit_request();                             //CAN data transmission request
    while((can_interrupt_flag_get() & CAN_IF_TI) == 0x00);  //Wait for transmission complete
    can_int_flag_clr(CAN_IF_TI);                        //Clear the transmission complete
interrupt flag bit
}
/***************CAN self sleep wakeup configuration*****************/
void can_sleep_config(uint8_t wakeup_mode,uint8_t filter_en,FunctionalState state)
{
    if(state == ENABLE){
        CAN_WUP |= CAN_WUP_ENABLE;
    }else{
        CAN_WUP &= ~CAN_WUP_ENABLE;
    }                                                   //Enable self-wakeup
    if(wakeup_mode){
        CAN_WUP |= CAN_WUP_MODE;
    }else{
        CAN_WUP &= ~CAN_WUP_MODE;
    }                                                   //Set self-wakeup filtering
    if(filter_en)
        CAN_FILTER_EN = 0x01;
    else
        CAN_FILTER_EN = 0x00;                           //Enable Rx port filtering in sleep mode
}
/***************CAN interrupt enable*****************/
void can_int_enable(uint8_t intstate)
{
    CAN_IE |= intstate;                                 //Set CAN interrupt enable
}
/***************CAN self-sleep request*****************/
ErrorStatus can_sleep_request()
{
    uint32_t timeout = 400;
    CAN_CMR |= CAN_CMR_SPM;                     //Self-sleep request
    while(!CAN_SLPAK){                          //Wait for entering sleep mode
        timeout --;
        if(timeout == 0){
            return ERROR;
        }
```

```
    }
    return SUCCESS;
}
/***************System sleep*****************/
void sleep_deep()
{
    SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
    __WFI();                                    //Make system enter sleep-deep mode
}
```

5.3 PWM

This module mainly realizes the PWM waveform function with adjustable duty cycle of the output frequency and input capture function， and can also be used as a counter at the same time.

5.3.1 Features

1. 16-bit up or down counter;
2. Support up to 6 PWM channels;
3. Each channel supports output comparison or edge-aligned PWM mode waveform output, supports setting, clearing, and switching output comparison operations, and the polarity of the PWM output is optional;
4. Each channel supports rising, falling or any edge input capture trigger;
5. Support center aligned pulse width modulation on all channels;
6. System clock, crystal oscillator clock or external clock input can be selected as timer clock source and support clock 1/2/4/8/16/32/64/128 frequency division;
7. Support one function interrupt and counter overflow interrupt for each channel;
8. Each channel port and external clock port are individually enabled and strobed, and the channel port or external clock port is not enabled when PWM counting is working. They can be individually configured as GPIO to use;
9. Support PWM counting to trigger ADC sampling.

5.3.2 Functional description

This section introduces the main functions of BF7006AMXX PWM in detail.

5.3.2.1 Counter mode

The module contains a 16-bit counter. The counter clock source can select external input clock or system clock or crystal oscillator clock. The system clock or crystal oscillator clock is configured by the system. This module only configures to select external input clock or system clock. See the description of the register PWM_SC_CLKS_SEL. The clock can be configured with prescaler (1/2/4/8/16/32/64/128).

The module work control clock is always the system clock. When the counting clock is selected as the external input clock, the system clock actually samples the input clock signal to control the counting. The interrupt signal is generated by the system clock and is not affected by the external input clock.

The counter has two counting modes. When center-aligned PWM is selected, the counter runs in

up/down counting mode. Otherwise, the counter operates as a up counter. As an up counter, the timer counter counts from 0x0000 to its terminal count, and then restarts from 0x0000. The terminal count is the modulus value in PWM_MOD or 0xFFFF.

When the center-aligned PWM mode is selected, the counter counts up from 0x0000 to its terminal count, and then counts down to 0x0000, where it starts to count up again. Both 0x0000 and the terminal count value are normal length counts (the length of a timer clock cycle). In this mode, the timer overflow flag PWM_SC_TOF is set at the end of the terminal counting period (when the counter changes to the next lower count value).

Counter overflow will generate overflow interrupt PWM_SC_TOF. When there is no modulus limit, nor in PWM_SC_CPWMS=1 mode, the 16-bit timer counter counts from 0x0000 to 0xFFFF, and then overflows to 0x0000 in the next counting clock cycle. When transitioning from 0xFFFF to 0x0000, PWM_SC_TOF is set. When the modulus limit is set, PWM_SC_TOF is set when the value set in the modulus register transitions to 0x0000. When in the center-aligned PWM mode (PWM_SC_CPWMS=1), the PWM_SC_TOF flag is set when the counter reaches the count value set in the modulus register and ends when it changes direction (counts from the value set in the modulus register transit to the next lower count value). This corresponds to the end of the PWM period (the 0x0000 count value corresponds to the center of the period).

The counter can be manually reset at any time by writing any value to the PWM_CNT register.

No matter which of the following channel working modes, the counter will automatically reload and count, and will not stop by itself until it is stopped by configuring PWM_SC_CLK_SEL=0. If the value of the modulus register is 0x0000, the counter enters the idle mode and will count to 0xffff. In addition, they are all modulus counting methods, which are counting up to the modulus register value and restarting counting.

5.3.2.2 Input capture

In the non-center alignment mode, each channel working mode can be individually configured. In center-aligned mode, all channels default to output PWM waveform function.

Use the input capture function to capture the time when an external event occurs. When an active edge occurs on the pin of the input capture channel, the contents of the counter can be locked into the channel value register. The rising edge, falling edge or any edge can be selected as the enable edge of the trigger input capture.

The input captured signal is synchronized (two beats of the system clock across the time domain) and then judged to achieve the capture function.

In the input capture mode, the channel value register is read-only.

The flag bit is set after the input capture event (the selected edge is detected). This bit can select to generate a CPU interrupt request.

5.3.2.3 Output compare

With the output comparison function, timing pulses with programmable position, polarity, duration and frequency can be generated. When the counter reaches the value in the channel value register of the output comparison channel, the channel pins can be set, cleared or switched, and can be selected to force

the pins to be set to 0 or 1, invert the pin level.

When the output comparison switching mode is just selected, the previous value on the pin has been driven until the next output comparison event occurs, and then the pin is switched.

The interrupt flag is set every time the counter matches the 16-bit value in the channel value register.

5.3.2.4 Edge aligned

This type of PWM output uses the up-counting mode of the counter (PWM_SC_CPWMS=0); it can also be used when other channels are configured for input capture or output compare functions. The period of this PWM signal is determined by the value of the modulus register PWM_MOD plus 1. The duty cycle is determined by the setting in the channel register PWM_CxV. The polarity of this PWM signal is determined by the setting in the PWM_CxSC_ELS control bit. Both 0% and 100% duty cycles are possible.

Pulse width = PWM_CxV

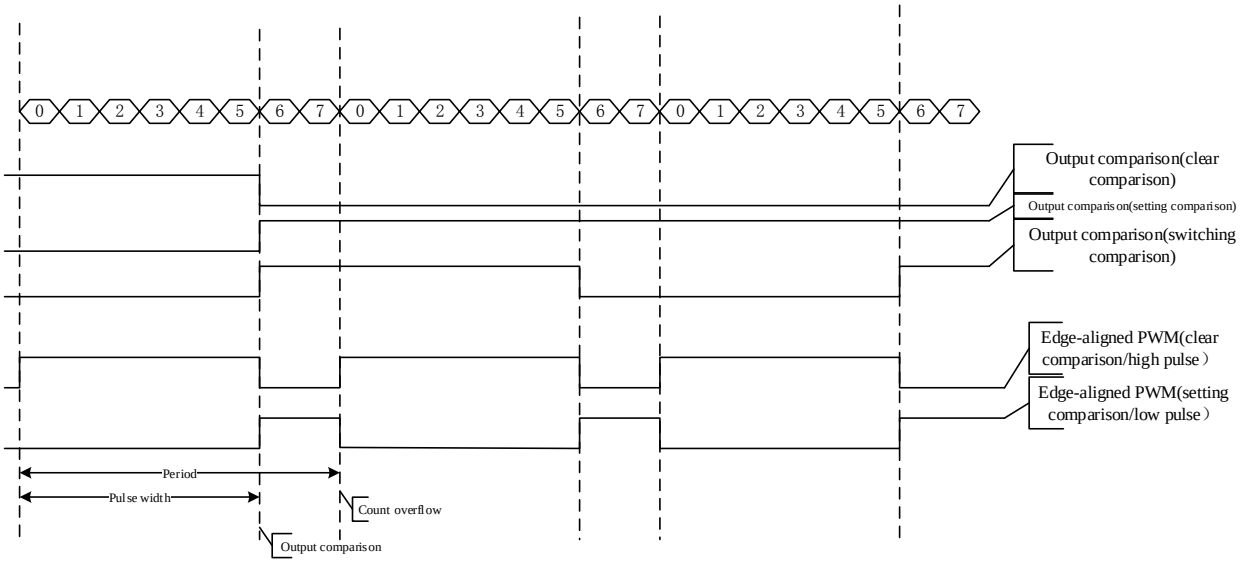
Period = PWM_MOD + 1

The output comparison value in the channel register determines the pulse width (duty cycle) of the PWM signal. The time interval between modulo overflow and output comparison is the pulse width. If PWM_CxSC_ELS=0, the counter overflow forces the PWM signal to enter the high state; and the output comparison forces the PWM signal to enter the low state. If PWM_CxSC_ELS=1, the counter overflows and forces the PWM signal to enter a low state; and the output comparison forces the PWM signal to enter a high state.

When the channel value register is set to 0x0000, the duty cycle is 0%. A 100% duty cycle can be achieved by setting the channel counter PWM_CxV to a value greater than the modulus setting.

The interrupt flag is set when the counter matches the channel value register that marks the end of the duty cycle period.

The detailed counting and waveform output are shown in the following diagram:



5.3.2.5 Center aligned

Double the value of the 16-bit modulus register sets the PWM output period, while the channel value

register sets half of the duty cycle duration. The timer counter counts up until it reaches the modulus value, and then counts down until it reaches 0. In the case of down counting, when the count matches the channel value register, the PWM output enters the valid state. In the case of up counting, when the count matches the channel value register, the PWM output enters an invalid state. This type of PWM signal is called center-aligned because the center of the active duty cycle of all channels is aligned with the count value 0.

This type of PWM output uses the up/down counting mode of the counter. The output comparison value in PWM_CxV determines the pulse width (duty cycle) of the PWM signal, and the value in MOD determines the period. PWM_CxSC_ELS will determine the polarity of PWM_SC_CPWMS output.

Pulse width = 2 x PWM_CxV

Period = 2 x PWM_MOD；PWM_MOD = 0x0001 ~ 0xFFFF

If the channel value register PWM_CxV is 0, the duty cycle will be 0%. If PWM_CxV is a positive value greater than the modulus setting, the duty cycle will be 100% because the duty cycle comparison will not occur.

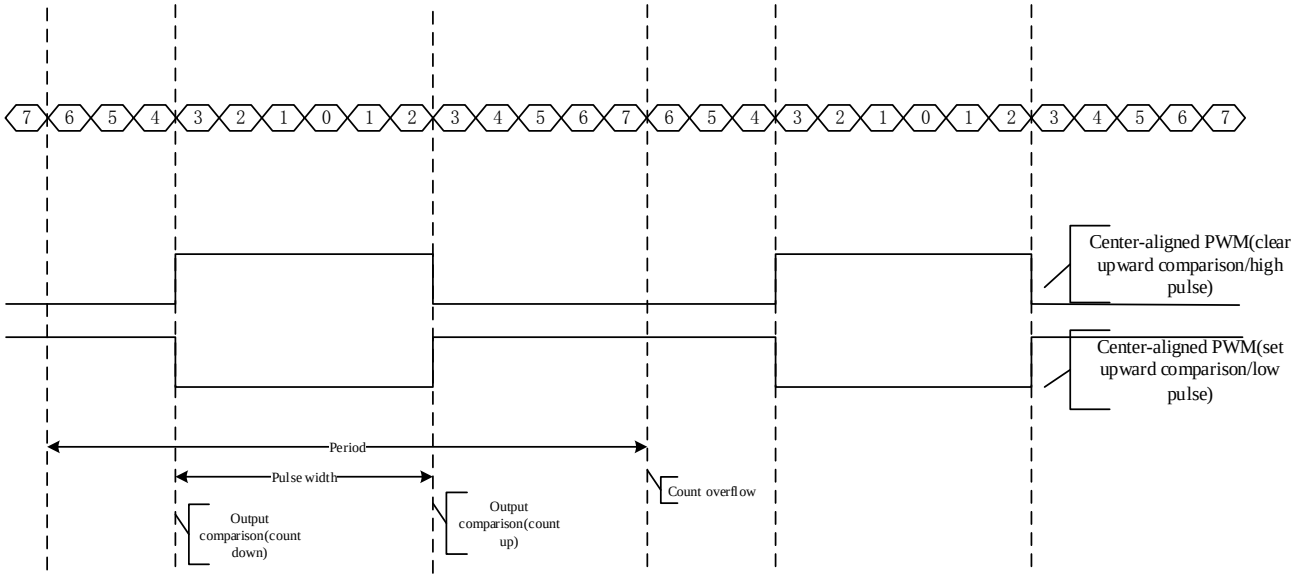
The output comparison value in the channel register determines the pulse width (duty cycle) of the PWM_SC_CPWMS signal. If PWM_CxSC_ELS=0, when counting up, a numerical comparison will force the PWM_SC_CPWMS output signal to enter a low state; when counting down, a numerical comparison will force the output to enter a high state. After the counter reaches the modulus setting in PWM_MOD, it starts counting down to 0, and then starts counting up to the modulus setting. This sets the period to twice the PWM_MOD.

When the counter is running in up/down counting mode, the input capture, output comparison, and edge-aligned PWM functions are meaningless. So this means that when PWM_SC_CPWMS=1, all enabled channels must be used in PWM_SC_CPWMS mode.

The channel output compare interrupt flag is set at the beginning and end of the duty cycle time (when the timer counter matches the channel value register). PWM_SC_TOF is set when the counter direction changes from counting up to counting down at the end of terminal counting (value in the modulus register). In this case, PWM_SC_TOF corresponds to the end of the PWM period.

Note: In the center-aligned mode, updating the counting period during the counting process will cause the length of the invalid level to be inconsistent. The length of the valid level remains unchanged, which means that the length of the level aligned with 0 as the center remains unchanged.

The detailed counting and waveform output are shown in the following diagram：



5.3.3 Register map

PWM module address range：0x5006_0000~0x5006_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	PWM_SC	R/W	pwm counter control register	0x00
0x04	PWM_CNT	R/W	pwm count value register	0x0000
0x08	PWM_MOD	R/W	pwm counter modulus register	0xFFFF
0x0c	PWM_C0SC	R/W	pwm channel 0 control register	0x00
0x10	PWM_C0V	R/W	pwm channel 0 count value register	0x0000
0x14	PWM_C1SC	R/W	pwm channel 1 control register	0x00
0x18	PWM_C1V	R/W	pwm channel 1 count value register	0x0000
0x1c	PWM_C2SC	R/W	pwm channel 2 control register	0x00
0x20	PWM_C2V	R/W	pwm channel 2 count value register	0x0000
0x24	PWM_C3SC	R/W	pwm channel 3 control register	0x00
0x28	PWM_C3V	R/W	pwm channel 3 count value register	0x0000
0x2c	PWM_C4SC	R/W	pwm channel 4 control register	0x00
0x30	PWM_C4V	R/W	pwm channel 4 count value register	0x0000
0x34	PWM_C5SC	R/W	pwm channel 5 control register	0x00
0x38	PWM_C5V	R/W	pwm channel 5 count value register	0x0000
0x3C	PWM_ADCV	R/W	ADC trigger count value register	0x0000

5.3.4 Register description

5.3.4.1 PWM_SC register

Address	Bit	Name	R/W	Reset value	Description
0x00	7	PWM_SC_TOF	R/W	0x0	Timer overflow flag 0x1: Overflow 0x0: Not overflow Write 0 to clear this bit.
	6	PWM_SC_TOIE	R/W	0x0	Timer overflow interrupt enable 0x1: Enable interrupt 0x0: Disable (can be used for polling)
	5	PWM_SC_CPWMS	R/W	0x0	Counting method selection 0x1: Count up and then count down (center aligned) 0x0: Count up
	4:3	PWM_SC_CLK_SEL	R/W	0x0	Counter clock source for prescaler input 0x3: Select external clock input 0x2: Select system clock counter 0x1: Select system clock counter 0x0: Disable the counter
	2:0	PWM_SC_CLK_DIV	R/W	0x0	Count clock divided by 2 to the power of PWM_SC_CLK_DIV

5.3.4.2 PWM_CNT register

Address	Bit	Name	R/W	Reset value	Description
0x04	15:0	PWM_CNT	R/W	0x0000	Count value register Record the current count value, system reset or writing this register would clear the counter

5.3.4.3 PWM_MOD register

Address	Bit	Name	R/W	Reset value	Description

0x08	15:0	PWM_MOD	R/W	0xFFFF	Counter modulus register Configuring PWM_MOD, system reset or writing PWM_CNT would clear the counter. It is necessary to disable the counter or reset the counter before writing to the modulus register to avoid confusion about the time when the first counter overflow occurs.

5.3.4.4 PWM_CxSC（x=0~5）register

Address	Bit	Name	R/W	Reset value	Description
0x0C 0x14 0x1C 0x24 0x2C 0x34	7	PWM_CxSC_IF	R/W	0x0	Channel x interrupt status 0x1: An input capture or output compare event occurred 0x0: No interrupt When channel x is used as an input capture channel, this bit will be set when a valid trigger edge occurs on the channel x pin. When channel x is an output comparison or edge-aligned/center-aligned PWM channel, this bit will be set when the value in the counter register matches the value in the channel x count value register. When channel x is used as an edge-aligned/center-aligned PWM channel and the duty cycle is set to 0% or 100%, this bit will not be set during matching.
	6	PWM_CxSC_IE	R/W	0x0	Channel x interrupt enable 0x1: Enable 0x0: Disable (for polling)
	5:4	PWM_CxSC_MS	R/W	0x0	Channel x mode selection (see the table below)
	3:2	PWM_CxSC_ELS	R/W	0x0	Channel x edge/level polarity selection (see the table below)
	1:0	保留	——	——	Reserved

Mode and polarity selection				
PWM_SC_CPWMS	PWM_CxSC_MS	PWM_CxSC_ELS	Mode	Function
x	x	00	Invalid	
1	xx	10	Center aligned PWM	High-true pulse (clear the upward comparison output)
		x1		Low-true pulse (set the upward comparison output)
0	00	01	Input capture mode	Capture only on rising edge
		10		Capture only on falling edge

Mode and polarity selection				
PWM_SC_CPWMS	PWM_CxSC_MS	PWM_CxSC_ELS	Mode	Function
		11		Capture on rising or falling edge
	01	01	Output compare mode	Switch comparison output
		10		Clear comparison output
		11		Set comparison output
	1x	10	Edge aligned PWM	High-true pulse (clear comparison output)
		x1		Low-true pulse (set comparison output)

5.3.4.5 PWM_CxV（x=0~5）register

Address	Bit	Name	R/W	Reset value	Description
0x10 0x18 0x20 0x28 0x30 0x38	15:0	PWM_CxV	R/W	0x0	Channel count value register Record the count value captured by the input or configure the count value for output comparison. In the input capture mode, this register records the counter value when the valid edge is captured. Any write operation to the channel value register will be ignored in the input capture mode. In output compare or PWM mode, this register is used to configure the count value of each channel's compare output. When the counter is not in working mode, the register is updated directly after the write operation. Otherwise (the counter is in the working state), the register will be updated after the write operation until the counter changes from PWM_MOD-1 to PWM_MOD.

5.3.4.6 PWM_ADCV register

Address	Bit	Name	R/W	Reset value	Description

0x3C	15:0	PWM_ADCV	R/W	0x0000	ADC trigger count value register Configure the count value that the trigger condition meets in the PWM trigger mode of the ADC. When the ADC selects the internal PWM hardware trigger mode, this register is meaningful. It is used to configure the count value of the trigger condition. When the count value reaches this value, the flag bit is sent to the ADC for sampling. The flag bit only can be cleared by clearing the overflow interrupt operation (write PWM_SC_TOF=0). When PWM_SC_CLK_SEL =0 (the counter is not working), the register is updated directly after the write operation. Otherwise (the counter is in the working state), the register will be updated after the write operation until the counter changes from PWM_MOD-1 to PWM_MOD. According to the update coherency mechanism, the current update will be cleared when the configuration counter control register is configured with different values.

5.3.5 Example program

```
uint8_t aaa,bbb;//PWM mode selection function
//Main function:
int main(void)
{
    /*PWM initialization*/
    aaa = 2;bbb = 2;
    /*PWM initialization, including PWM_INT_ENABLE (PWM interrupt initialization)、
PWM_CPWMS_UNIDIR (Select counting up)、PWM_CLK_SEL_SYS (Select the system clock as the clock
source)、PWM_CLK_DIV_32 (Clock divide by 32)、2000 (Period is 2ms) */
    pwm_init(PWM_INT_ENABLE | PWM_CPWMS_UNIDIR | PWM_CLK_SEL_SYS | PWM_CLK_DIV_32,2000);
    /*PWM channel intialization, including PWM_CHn_INT_ENABLE (PWM channel interrupt
enable)、PWM_CHn_MS(aaa) (Set edge-alignment)、PWM_CHn_ELS(bbb) (Clear comparison output)、
200 (Duty cycle is 10%, which is 200/2000, 500、800、1100...etc) */
    pwm_ch0_init(PWM_CHn_INT_ENABLE | PWM_CHn_MS(aaa) | PWM_CHn_ELS(bbb),200);
    pwm_ch1_init(PWM_CHn_INT_ENABLE | PWM_CHn_MS(aaa) | PWM_CHn_ELS(bbb),500);
    pwm_ch2_init(PWM_CHn_INT_ENABLE | PWM_CHn_MS(aaa) | PWM_CHn_ELS(bbb),800);
    pwm_ch3_init(PWM_CHn_INT_ENABLE | PWM_CHn_MS(aaa) | PWM_CHn_ELS(bbb),1100);
    pwm_ch4_init(PWM_CHn_INT_ENABLE | PWM_CHn_MS(aaa) | PWM_CHn_ELS(bbb),1400);
    pwm_ch5_init(PWM_CHn_INT_ENABLE | PWM_CHn_MS(aaa) | PWM_CHn_ELS(bbb),1700);
}
//Subfunction:
/*****************PWM initialization*********************/
void pwm_init(uint8_t pwm_sc,uint16_t pwm_mod)
```

```
{
    PWM_SC = pwm_sc;                                    //PWM control register configuration
    PWM_MOD = pwm_mod;                                 //PWM modulus register configuration
    if(pwm_sc & PWM_SC_TOIE){
        NVIC_EnableIRQ(PWM_TOF_IRQN);
    }else{
        NVIC_DisableIRQ(PWM_TOF_IRQN);
    }                                                   //Timer overflow interrupt configuration
}
/***************PWM channel intialization*****************/
void pwm_ch0_init(uint8_t pwm_ch_sc,uint16_t pwm_ch_cnt)
{
    PWM_COSC = pwm_ch_sc;                              //Channel 0 control register configuration
    PWM_COV = pwm_ch_cnt;                               //Channel 0 count value configuration
    if(pwm_ch_sc & PWM_CnSC_IE){
        NVIC_EnableIRQ(PWM_CH0_IRQn);
    }else{
        NVIC_DisableIRQ(PWM_CH0_IRQn);
    }                                                   //Channel 0 interrupt configuration
}
/*************PWM count overflow interrupt service function***************/
void PWM_TOF_IRQHandler(void)
{
    pwm_tof_clr();                             //Clear the counter overflow interrupt flag
}
/***************PWM channel interrupt service function*****************/
void PWM_CH0_IRQHandler(void)
{
    pwm_ch0_if_clr();                                    //Clear channel 0 interrupt flag bit
}
```

5.4 TIMER

BF7006AMXX includes 2 basic timers Timer0 and Timer1, which provide basic timing counting requirements for applications.

5.4.1 Features

1. 16-bit up counter；
2. Count clock can select system clock frequency division and internal 32KHz clock；
3. Support auto-reload mode;

5.4.2 Functional description

The TIMER module has timing function. Its internal main structure is a 16-bit counter and it achieves the timing function by counting the input clock. The counting principle of TIMER is cumulative counting. When the counter counts to the set value, an interrupt is generated；

The counting clock of TIMER can select system clock or internal RC clock; TIMER has two working modes: single timing mode and auto-reload mode, no matter which mode, an interrupt will be generated when the timing is completed.

5.4.3 Register map

TIMER0 module address range：0x500B_0000~0x500B_3FFF

TIMER1 module address range：0x500B_4000~0x500B_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	TIMER_CFG	R/W	Timer0/1 counter control register	0x00
0x04	TIMER_MOD	R/W	Timer0/1 counter modulus register	0x0000
0x08	TIMER_CNT	R/W	Timer0/1 count value register	0x0000

5.4.4 Register description

5.4.4.1 TIMER_CFG register

Address	Bit	Name	R/W	Reset	Description

				value	
0x00	7	RESERVED	——	——	Reserved
	6	TIMER_CFG_IF	R/W	0x0	Interrupt flag bit, write 1 to clear Disabling TIMER cannot clear this bit.
	5	TIMER_CFG_IE	R/W	0x0	Interrupt enable 0x1: Enable TIMER interrupt 0x0: Disable TIMER interrupt
	4:3	TIMER_CFG_CLK_DIV	R/W	0x0	TIMER frequency division (only used when TIMER selects sys_clk) 0x3: System clock divided by 8 0x2: System clock divided by 4 0x1: System clock divided by 2 0x0: System clock
	2	TIMER_CFG_CLK_SEL	R/W	0x0	TIMER clock selection register 0x1: Select RC32KHz 0x0: Select system clock
	1	TIMER_CFG_RLD	R/W	0x0	TIMER auto reload enable register 0x1: Auto reload mode 0x0: Manual mode
	0	TIMER_CFG_EN	R/W	0x0	TIMER counting enable register 0x1: Enable 0x0: Disable In manual reload mode, the hardware will automatically clear this register after the timing is completed, and the register will count again when configured during the scanning process.

5.4.4.2 TIMER_MOD register

Address	Bit	Name	R/W	Reset value	Description
0x04	15:0	TIMER_MOD	R/W	0x0000	TIMER overflow value configuration register, writing to this register during scanning would trigger re-counting

5.4.4.3 TIMER_CNT register

Address	Bit	Name	R/W	Reset value	Description
0x08	15:0	TIMER_CNT	RO	0x0000	TIMER counting register

5.4.5 Example program

```
//Main function:
int main(void)
{
    gpio_init(GPIOB,GPIO_MODE_OUT,GPIO_PIN_6);  //PB6 is set as output
    gpio_bit_set(GPIOB,GPIO_PIN_6);             //PB6 outputs high level
    gpio_bit_reset(GPIOB,GPIO_PIN_6);               //PB6 outputs low level
    /*Timer initialization, including TIMERx (Timer selection, timer 0 or timer 1 can be selected)、IMER_INT_ENABLE (Timer interrupt enable)、IMER_CLK_SYS_DIV2 (Clock divided by 2)、TIMER_AUTO_RLD_ENABLE (Atuo-reload enable)、TIMER_ENABLE (Timer enable)、0x3E80 (Timer overflow value, 0x3E80 is 16000. Every time the counter reaches 16000, an overflow interrupt will be generated. According to the clock frequency, an interrupt will be generated every 1ms.) */
    timer_init( TIMER0, \
        TIMER_INT_ENABLE | TIMER_CLK_SYS_DIV2 | TIMER_AUTO_RLD_ENABLE | TIMER_ENABLE, \
        0x3E80  \
        );

    timer_init( TIMER1, \
        TIMER_INT_ENABLE | TIMER_CLK_SYS_DIV2 | TIMER_AUTO_RLD_ENABLE | TIMER_ENABLE, \
        0x3E80  \
        );
}
//Subfunction:
/*****************Timer initialization*******************/
```

```
void timer_init(uint32_t timerx,uint8_t timer_cfg,uint16_t timer_mod)
{
    TIMER_CFG(timerx) = timer_cfg;              //Configure the timer control register
    TIMER_MOD(timerx) = timer_mod;              //Configure the timer modulus register
    if(timerx == TIMER0)
        if(timer_cfg & TIMER_CFG_IE)
            NVIC_EnableIRQ(TIMER0_IRQn);
        else
            NVIC_DisableIRQ(TIMER0_IRQn);
    }else{
        if(timer_cfg & TIMER_CFG_IE)
            NVIC_EnableIRQ(TIMER1_IRQn);
        else
            NVIC_DisableIRQ(TIMER1_IRQn);
    }                                           //Timer interrupt enable configuration
}
uint8_t timer0_f = 0;   //Level flip status bit
uint16_t timer0_d = 0;  //Level flip delay parameters
/*****************Timer0 interrupt service function*******************/
void TIMER0_IRQHandler(void)
{
    timer_intflag_clr(TIMER0);                      //Clear interrupt flag
    timer0_d++;
    if(timer0_d == 500)
    {
        timer0_d = 0;
        timer0_f = ~timer0_f;                       //Level delay flip
    }
    if(timer0_f)
        gpio_bit_reset(GPIOB,GPIO_PIN_6);
    else
        gpio_bit_set(GPIOB,GPIO_PIN_6);         //LED flashing
}
/*****************Timer1 interrupt service function*******************/
void TIMER1_IRQHandler(void)
{
    timer_intflag_clr(TIMER1);                      //Clear interrupt flag
        timer0_f = ~timer0_f;
    if(timer0_f)
        gpio_bit_reset(GPIOB,GPIO_PIN_6);
    else
        gpio_bit_set(GPIOB,GPIO_PIN_6);         //Level flip
}
```

5.5 RTC

RTC is a 32-bit counter. This module can be used for timing counting or task scheduling functions. In addition, it can provide periodic wake-up services without external components.

5.5.1 Features

1. 32-bit up counter;
2. The counting clock can be selected from 1KHz and 32KHz and the external oscillator clock divided by 32;
3. Support waking up system from sleep mode;
4. Support RTC hardware timing to trigger ADC sampling;

5.5.2 Functional description

This section introduces the main functions of BF7006AMXX RTC in detail.

5.5.2.1 General description

RTC supports 32-bit up counting, and supports trigger ADC frame mode wakeup and RTC wakeup under low power mode.

After the MCU is reset, the counter is set to 0x00, and the RTC is disable. The 1KHz internal clock is selected as the default clock source.

Three kinds of clock sources can be selected by software: 1K internal clock, external clock and 32K internal clock.

5.5.2.2 Frame wakeup

The RTC interrupt can be used to wake up the ADC frame work mode, a fixed period to wake up the ADC for external sampling to wake up the system. See the ADC chapter for details.

5.5.3 Register map

RTC module address range：0x5007_0000~0x5008_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	RTC_SC	R/W	RTC status control register	0x0000
0x04	RTC_CNT	R/W	RTC count register	0x0000_0000
0x08	RTC_MOD	R/W	RTC modulus register	0x0000_0000

5.5.4 Register description

5.5.4.1 RTC_SC register

Address	Bit	Name	R/W	Reset value	Description
0x00	15:10	RESERVED	— —	——	Reserved
	9	RTC_SC_EN	R/W	0x0	RTC enable 0x1: Enable RTC 0x0: Disable RTC
	8	RESERVED	— —	——	Reserved
	7	RTC_SC_IF	R/W	0x1	Real-time interrupt flag 0x1: The value specified in the RTC overflow value register has been reached 0x0: The value specified in the RTC overflow value register is not reached Write logic 1 to clear this bit and interrupt request.
	6:5	RTC_SC_CLK_SEL	R/W	0x0	Counting clock source selection 0x3: Real-time clock source is internal 32KHz clock 0x2: Real-time clock source is internal 32KHz clock 0x1: The real-time clock source is an external oscillator clock divided by 32 0x0: Real-time clock source is 1KHz clock Changing this bit dynamically during counting can clear the RTC_CNT counter. When selecting the clock source, ensure that the clock source is normally enabled (if applicable) to ensure the correct operation of the RTC. Due to the requirement of clock switching stability, RTC clock switching selects 1KHz clock, and it will work normally after waiting 2ms. Therefore, it is recommended that when you need to configure the RTC again, you should disable the RTC first, and then configure it.
	4	RTC_SC_IE	R/W	0x0	Interrupt enable 0x1: Enable 0x0: Disable
	3:0	RESERVED	— —	——	Reserved

5.5.4.2 RTC_CNT register

Address	Bit	Name	R/W	Reset value	Description
0x04	31:0	RTC_CNT	R/W	0x0	The current value of the counter.

5.5.4.3 RTC_MOD register

Address	Bit	Name	R/W	Reset value	Description
0x08	31:0	RTC_MOD	R/W	0x0	The comparison value of the counter that generates the interrupt. Write to this register to clear RTC_CNT. In actual applications, it is forbidden to configure less than 3, otherwise the timing will be wrong and frequently interrupted.

5.5.5 Example program

```
//Main function:
int main(void)
{
    gpio_init(GPIOB,GPIO_MODE_OUT,GPIO_PIN_6);  //PB6 is set as output
    gpio_bit_set(GPIOB,GPIO_PIN_6);             //PB6 outputs high level
    gpio_bit_reset(GPIOB,GPIO_PIN_6);               //PB6 outputs low level
    /*RTC initialization, including RTC_ENABLE (RTC enable)、RTC_CLK_SEL_32K (select 32KHz
internal clock source)、RTC_INT_ENABLE (RTC interrupt enable)、RTC_TRG_CLK_DIV(0) (does not
trigger ADC sampling)、0x7D00 (RTC interrupt compare value, 0x7D00 is 32000, an interrupt is
generated when the counter reaches 32000. And an interrupt is generated in 1s according to
the clock frequency.) */
    rtc_init(RTC_ENABLE | RTC_CLK_SEL_32K | RTC_INT_ENABLE | RTC_TRG_CLK_DIV(0),0x7D00);
}
//Subfunction:
/*****************RTC initialization*******************/
void rtc_init(uint16_t rtc_sc,uint32_t rtc_mod)
{
    ErrorStatus rval;
    if((rtc_sc & RTC_SC_CLK_SEL) == RTC_CLK_SEL_XTAL_DIV32)      //Select external crystal
oscillator as clock source
    {
```

```
        if(!(SYS_XTAL_CTRL & SYS_XTAL_CTRL_INIT)){  // The crystal oscillator is not stable
            rval = xtal_init();                     //Crystal oscillator initialization
            if(rval == ERROR){
                rtc_sc &= ~RTC_SC_CLK_SEL;
                rtc_sc |= RTC_CLK_SEL_1K;               // If initialization fails, select 1K
internal clock as the clock source
            }
        }
    }
    RTC_SC = rtc_sc;            //RTC status control register configuration
    RTC_MOD = rtc_mod;                          //RTC modulus register configuration
    if(rtc_sc & RTC_SC_IE){
        NVIC_EnableIRQ(RTC_IRQn);
    }else{
        NVIC_DisableIRQ(RTC_IRQn);
    }                                           //RTC interrupt enable configuration
}
/*****************RTC interrupt service function******************/
void RTC_IRQHandler(void)
{
    rtc_int_flag_clr();                             //Clear flag of RTC interrupt
    timer0_f = ~timer0_f;
    if(timer0_f)
        gpio_bit_reset(GPIOB,GPIO_PIN_6);
    else
        gpio_bit_set(GPIOB,GPIO_PIN_6);                         //LED flashing
}
```

5.6 WDT

WDT is a 16-bit up counter. When the system software cannot be executed normally, the WDT watchdog will force a system reset. In order to prevent a system reset from the WDT timer, the application software must reset the WDT counter periodically. If the application fails to reset the WDT counter before the timeout, a system reset will be generated at this time, forcing the system to return to a known starting point.

The WDT count clock is an internal 32KHz clock or a divided 1KHz clock. After the system is powered on, the WDT is turned on by default. It is recommended that the user first disables the watchdog, configures it according to the application needs, and then enables the WDT.

5.6.1 Features

1. Programmable function of overflow period (16 bits);
2. Counting clock can select 32KHz and its 32 frequency division;

3. The only instruction for refreshing watchdog. Illegal instruction for refreshing watchdog will cause the watchdog to reset to prevent misoperation;
4. Support window mode;
5. Register configuration protection unlock function can avoid misconfiguration of watchdog registers;
6. In the low power mode of the system, it supports watchdog reset wakeup;
7. The watchdog can be configured to halt in debug mode.

5.6.2 Functional description

This section introduces the main functions of BF7006AMXX WDT in detail.

5.6.2.1 General description

After the system is reset, the WDT is turned on by default, and the WDT is configured as a 32KHz internal clock by default, and the overflow register value of the WDT is 0xFFFF. If the watchdog is not refreshed regularly, a system will be reset. In order to update the WDT counter, the software must perform sequential writes before the counter overflows (less than or equal to the overflow value), otherwise it will force a reset.

5.6.2.2 Window mode

When the software completes its main control loop program faster than expected, problems may occur in certain applications. Therefore, the window mode is adopted according to the application requirements. If refreshing watchdog is too early in the window mode, the WDT will cause the system to reset.

When the window mode is valid, the WDT can be refreshed only after the counter reaches a minimum value (The maximum value is 75% of the timer overflow value. When the user-configured WDT_WINVAL value is greater than 75% of the counter overflow value, the system will default to 75% of the timer overflow value). Write in advance, WDT will reset the system. The minimum value is determined by the WDT_WINVAL register.

This function is usually also used for code debugging.

5.6.2.3 WDT configuration protection

Refreshing watchdog must write 0x55AA to WDT_CNT, otherwise writing a value other than 0x55AA will immediately cause a reset.

The watchdog's control register, overflow value register, and window register are all "written only once" after the system is reset. If the watchdog configuration needs to be updated, it must be after the next system reset. This regulation provides a protection mechanism to ensure that runaway programs cannot randomly configure watchdogs. The watchdog configuration sequence requires: first configure the window register, overflow value register, and then configure other control registers, and ensure that

WDT_CS_UPDATE=0. Even if the application uses the default reset settings of all control status registers of the WDT, the user must write to all control status registers during the reset initialization process to lock in this setting.

In some special cases, the user wants to reconfigure the WDT or diable the WDT enable without resetting. The WDT provides a secondary configuration mechanism. By configuring WDT_CS_UPDATE=1, the user can execute the "sequential unlock" command at any time to reconfigure the WDT within 256 system clocks without resetting the system.

When WDT_CS_UPDATE=1, write 0x6BC3 to WDT_CNT, and then complete the new configuration of WDT within 256 system clocks, and the new configuration will take effect after 256 system clocks. When the unlock counter count does not reach 256, write 0x6BC3 to WDT_CNT again, and the unlock counter will count from 0 again to prevent 256 system clocks from being insufficient.

5.6.2.4 Low-power wakeup

In low power mode, WDT reset can wake up the system and start running from the starting position. But reconfiguring the WDT requires 256 system clocks. After reconfiguring the WDT, if you need to enter the low-power mode, you need to insert a delay before this. This can ensure that the new WDT configuration has taken effect before the MCU enters the low-power mode, otherwise the MCU may not be able to wake up from the low-power mode.

5.6.2.5 Debug mode

When the system is in the debugging mode, you can configure whether the watchdog is suspended or not, which is used to debug the watchdog or avoid the influence of the watchdog during debugging.

5.6.3 Register map

WDT module address range：0x5008_0000~0x5008_FFFF

Address offset	Rigister name	R/W	Functional description	Initial value
0x00	WDT_CS	R/W	WDT control status register	0x0180
0x04	WDT_CNT	R/W	WDT counter register	0x0000
0x08	WDT_TOVAL	R/W	WDT overflow value register	0xFFFF
0x0C	WDT_WINVAL	R/W	WDT window register	0x0000

5.6.4 Register description

5.6.4.1 WDT_CS register

Address	Bit	Name	R/W	Reset value	Description
0x00	15	WDT_CS_WINEN	R/W	0x0	Watchdog window mode enable 0x1: Enable window mode 0x0: Disable window mode
	14:9	RESERVED	——	——	Reserved
	8	WDT_CS_CLKSEL	R/W	0x1	Watchdog clock selection 0x1: 32KHz internal oscillator clock 0x0: 1KHz internal oscillator clock
	7	WDT_CS_EN	R/W	0x1	Watchdog enable 0x1: Enable 0x0: Disable
	6	RESERVED	——	——	Reserved
	5	WDT_CS_UPDATE	R/W	0x0	Allow to update WDT configuration 0x1: Allow to update the WDT configuration, the software can re-update the WDT configuration within 256 system clocks after executing the unlock sequence write operation 0x0: Allow to update the WDT configuration, initialize the WDT configuration after reset, and cannot update the WDT configuration subsequently
	4:2	RESERVED	——	——	Reserved
	1	WDT_CS_SLEEP	R/W	0x0	Idle mode watchdog enable 0x1: Enable 0x0: Disable
	0	WDT_CS_DEEPSLEEP	R/W	0x0	Sleep mode watchdog enable 0x1: Enable 0x0: Disable

5.6.4.2 WDT_CNT register

Address	Bit	Name	R/W	Reset value	Description
0x04	15:0	WDT_CNT	R/W	0x0	The 16-bit WDT counter can read the value of this register

					through software at any time. It is not possible to directly change the value of this register through a write operation, but the write operation can achieve the following functions: Write 0x55AA to clear the WDT counter. If the written value is not equal to 0x55AA, the watchdog cannot be refreshed and reset the system. When WDT_CS_UPDATE=1, write 0x6BC3 to WDT_CNT, and then complete the new configuration of WDT within 256 system clocks.

5.6.4.3 WDT_TOVAL register

Address	Bit	Name	R/W	Reset value	Description
0x08	15:0	WDT_TOVAL	R/W	0xFFFF	Overflow compare value register. It is strictly forbidden to configure the value of too short counting period to this register, otherwise the system will frequently enter the reset.

5.6.4.4 WDT_WINVAL register

Address	Bit	Name	R/W	Reset value	Description
0x0C	15:0	WDT_WINVAL	R/W	0x0	Window overflow value The maximum value is 75% of the timer overflow value. When the user configuration value is greater than 75% of the counter overflow value, the system will default to 75% of the timer overflow value. When this register is configured as 0, it is equivalent to working in non-window mode. This register should not be configured too small in actual applications, otherwise the watchdog will frequently enter the reset and cannot refresh watchdog.

5.6.5 Example program

```
//Main function:
int main()
```

```
{
    wdt_config(WDT_WIN_ENABLE | WDT_CLOCK_32KHZ | \
               WDT_UPDATA_ENABLE | WDT_SLEEP_DISABLE | \
               WDT_DEEPSLEEP_DISABLE | WDT_ENABLE);/*Watchdog initialization, including
WDT_WIN_ENABLE (Window mode enable)、WDT_CLOCK_32KHZ (Select 32KHz clock source)、
WDT_UPDATA_ENABLE (Allow to update WDT configuration)、WDT_SLEEP_DISABLE (Low power mode
disable)、WDT_DEEPSLEEP_DISABLE (Sleep mode disable)、WDT_ENABLE (Watchdog enable) */
    wdt_overflow_count(50000);              //Configure overflow comparison value
    wdt_overflow_count_win(30000);          //Configure window overflow value
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);//PE5 is set as output

    gpio_bit_set(GPIOE,GPIO_PIN_5);     //PE5 outputs high level
    delay(1000);                            //delay
    gpio_bit_reset(GPIOE,GPIO_PIN_5);       //PE5 outputs low level
    while(1)
    {
        wdt_clear();            //Refresh watchdog regularly to prevent the program from
frequently being reset
    }
}
//Subfunction:
/****************Delay function******************/
void delay(uint16_t time)
{
    uint16_t i=0;
    while(time--)
    {
        for(i = 0;i < 1000;i++);
    }
}
/***************Watchdog initialization*****************/
void wdt_config(uint16_t wdt_cs)
{
    WDT_CS = wdt_cs;                    //WDT control status register configuration

/************Watchdog overflow value configuration ***************/
void wdt_overflow_count(uint16_t wdt_cnt)
{
    WDT_TOVAL = wdt_cnt;                //WDT overflow value register configuration
}
/************Window overflow value configuration ***************/
void wdt_overflow_count_win(uint16_t wdt_cnt)
{
    WDT_WINVAL = wdt_cnt;               //WDT window register configuration
```

```
}
/***************Refresh watchdog function*****************/
void wdt_clear()
{
    WDT_CNT = 0x55AA;   //Write 0x55AA to the WDT count register to clear the WDT counter
}
```

5.7 GPIO

BF7006AMXX has abundant external GPIO resources and supports up to 54 general-purpose GPIOs corresponding to A0~A7, B0~B7, C0~C7, D0~D7, E0~E7, F0~F7, G0~G5.

5.7.1 Features

1. Data direction control, and support input high impedance state;
2. Support internal pull-up resistor function;
3. Support external GPIO interrupt function, including rising edge, falling edge, high level, low level interrupt;
4. Support NMI external interrupt, interrupt priority is the highest;

5.7.2 Functional description

This section introduces the main functions of BF7006AMXX GPIO in detail.

5.7.2.1 General description

BF7006AMXX GPIO is a general-purpose GPIO port that supports general IO output and input functions. Ports A, B, D support external interrupt function, a total of 24 external interrupts. Four interrupt types can be selected, including rising edge, falling edge, high level, and low level. At the same time, all GPIOs support internal pull-up functions, with a pull-up resistance of 20K.

BF7106AMXX VCC is powered on from 0V-3.1V, the GPIO port is in an indeterminate state (the output follows the VCC voltage value or the output is low or the input is high impedance), pay attention to the protection of the peripheral components of the GPIO port; after the VCC is powered on higher than 3.3V, the chip will complete the power-on reset, and the GPIO port status will be executed according to the program settings;

BF7106AMXX VCC voltage drops to 3.1V, after generating a power-down reset, the GPIO port defaults to input high impedance state.

5.7.2.2 NMI interrupt

As the highest priority non-maskable interrupt, NMI is multiplexed on the PA7 port.（For NMI interrupt description, please refer to ARM Cortex_M0 related information）. The system expands the NMI control function as follows:

1. NMI can be enabled or disabled;
2. NMI supports four interrupt types: rising edge, falling edge, high level, low level interrupt;

3. The NMI function has the highest priority;
4. Support internal pull-up resistor function;

5.7.3 Register map

GPIO module address range：0x500A_0000~0x500A_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	GPIO_PTD (GPIOA)	R/W	GPIOA data register	0x00
0x04	GPIO_PTDD (GPIOA)	R/W	GPIOA direction register	0x00
0x08	GPIO_PTPE (GPIOA)	R/W	GPIOA pull up enable register	0x00
0x10	GPIO_PTSC (GPIOA)	R/W	GPIOA interrupt control and status register	0x00
0x14	GPIO_PTPS (GPIOA)	R/W	GPIOA interrupt enable register	0x00
0x18	GPIO_PTES (GPIOA)	R/W	GPIOA interrupt select register	0x00
0x1C	GPIO_PTD (GPIOB)	R/W	GPIOB data register	0x00
0x20	GPIO_PTDD (GPIOB)	R/W	GPIOB direction register	0x00
0x24	GPIO_PTPE (GPIOB)	R/W	GPIOB pull up enable register	0x00
0x2C	GPIO_PTSC (GPIOB)	R/W	GPIOB interrupt control and status register	0x00
0x30	GPIO_PTPS (GPIOB)	R/W	GPIOB interrupt enable register	0x00
0x34	GPIO_PTES (GPIOB)	R/W	GPIOB interrupt select register	0x00
0x38	GPIO_PTD (GPIOC)	R/W	GPIOC data register	0x00
0x3C	GPIO_PTDD (GPIOC)	R/W	GPIOC direction register	0x00
0x40	GPIO_PTPE (GPIOC)	R/W	GPIOC pull up enable register	0x00
0x48	GPIO_PTD (GPIOD)	R/W	GPIOD data register	0x00
0x4C	GPIO_PTDD (GPIOD)	R/W	GPIOD direction register	0x00
0x50	GPIO_PTPE (GPIOD)	R/W	GPIOD pull up enable register	0x00
0x58	GPIO_PTSC (GPIOD)	R/W	GPIOD interrupt control and status register	0x00
0x5C	GPIO_PTPS (GPIOD)	R/W	GPIOD interrupt enable register	0x00
0x60	GPIO_PTES (GPIOD)	R/W	GPIOD interrupt select register	0x00
0x64	GPIO_PTD (GPIOE)	R/W	GPIOE data register	0x00
0x68	GPIO_PTDD (GPIOE)	R/W	GPIOE direction register	0x00
0x6C	GPIO_PTPE (GPIOE)	R/W	GPIOE pull up enable register	0x00
0x74	GPIO_PTD (GPIOF)	R/W	GPIOF data register	0x00
0x78	GPIO_PTDD (GPIOF)	R/W	GPIOF direction register	0x00
0x7C	GPIO_PTPE (GPIOF)	R/W	GPIOF pull up enable register	0x00
0x84	GPIO_PTD (GPIOG)	R/W	GPIOG data register	0x00
0x88	GPIO_PTDD (GPIOG)	R/W	GPIOG direction register	0x00
0x8C	GPIO_PTPE (GPIOG)	R/W	GPIOG pull up enable register	0x00
0x94	GPIO_INTST (GPIOA)	R/W	GPIOA interrupt status register	0x00
0x98	GPIO_INTST (GPIOB)	R/W	GPIOB interrupt status register	0x00
0x9C	GPIO_INTST (GPIOD)	R/W	GPIOD interrupt status register	0x00
0xA0	GPIO_NMISC	R/W	NMI interrupt control and status register	0x00

5.7.4 Register description

5.7.4.1 GPIO_PTDD（GPIOx）register

Address	Bit	Name	R/W	Reset value	Description
0x04 0x20 0x3C 0x4C 0x68 0x78 0x88	7: 0	GPIO_PTDD (GPIOA) GPIO_PTDD (GPIOB) GPIO_PTDD (GPIOC) GPIO_PTDD (GPIOD) GPIO_PTDD (GPIOE) GPIO_PTDD (GPIOF) GPIO_PTDD (GPIOG)	R/W	0x00	Data direction of the port 0x1: Output 0x0: Input

5.7.4.2 GPIO_PTD（GPIOx）register

Address	Bit	Name	R/W	Reset value	Description
0x00 0x1C 0x38 0x48 0x64 0x74 0x84	7: 0	GPIO_PTD (GPIOA) GPIO_PTD (GPIOB) GPIO_PTD (GPIOC) GPIO_PTD (GPIOD) GPIO_PTD (GPIOE) GPIO_PTD (GPIOF) GPIO_PTD (GPIOG)	R/W	0x00	Port data For port pins configured as inputs, the reading returns to the logic level on the pin. For port pins configured as outputs, the reading returns the last value written to the register. The written value is locked in all bits of this register. For port pins configured as outputs, the logic level drives the corresponding MCU pins. The reset default is high impedance input.

5.7.4.3 GPIO_PTPE（GPIOx）register

Address	Bit	Name	R/W	Reset value	Description
0x08 0x24	7: 0	GPIO_PTPE (GPIOA) GPIO_PTPE (GPIOB)	R/W	0x00	Pull-up enable 0x1: Enable

0x40 0x50 0x6C 0x7C 0x8C		GPIO_PTPE（GPIOC） GPIO_PTPE（GPIOD） GPIO_PTPE（GPIOE） GPIO_PTPE（GPIOF） GPIO_PTPE（GPIOG）			0x0：Disable For port pins configured as outputs, these bits have no effect, and internal pull-up devices are disabled.

5.7.4.4 GPIO_PTSC（GPIOx）register

GPIO_PTSC（GPIOA）、GPIO_PTSC（GPIOB）、GPIO_PTSC（GPIOD）					
Address	Bit	Name	R/W	Reset value	Description
0x10 0x2C 0x58	7:4	RESERVED	——	——	Reserved
	3	GPIO_PTSC_IF	R/W	0x0	Port interrupt flag 0x1：Port interrupt detected 0x0：No port interrupt detected Shows whether a certain type of port interrupt is detected. Writing has no effect on it, please distinguish it from the interrupt flag of each port.
	2	GPIO_PTSC_ACK	R/W	0x0	Port type interrupt clear 0x1：Clear interrupt 0x0：No effect The value read is always 0
	1	GPIO_PTSC_IE	R/W	0x0	Port interrupt enable 0x1：Port interrupt enable 0x0：Port interrupt disable
	0	GPIO_PTSC_TRGMOD	R/W	0x0	Port interrupt detection mode 0x1：The port detects edge and level. 0x0：The port only detects edges When the GPIO selects the edge, the IO port inputs a long level, and the interrupt flag pulls high flag to only identify the edge. After the flag bit is cleared, it waits for the next edge and then the interrupt pulls high flag bit.

5.7.4.5 GPIO_PTPS（GPIOx）register

GPIO_PTPS（GPIOA）、GPIO_PTPS（GPIOB）、GPIO_PTPS（GPIOD）

Address	Bit	Name	R/W	Reset value	Description
0x14 0x30 0x5C	7: 0	GPIO_PTPS (GPIOA) GPIO_PTPS (GPIOB) GPIO_PTPS (GPIOD)	R/W	0x00	8-bit interrupt enable for each port type 0x1: Enable 0x0: Disable This register is the interrupt enable for each bit of each port.

5.7.4.6 GPIO_PTES（GPIOx）register

GPIO_PTES (GPIOA)、GPIO_PTES (GPIOB)、GPIO_PTES (GPIOD)					
Address	Bit	Name	R/W	Reset value	Description
0x18 0x34 0x60	7: 0	GPIO_PTES (GPIOA) GPIO_PTES (GPIOB) GPIO_PTES (GPIOD)	R/W	0x00	Port interrupt type selection 0x1: Detect rising edge/high level generated by interrupt 0x0: Detect falling edge/low level generated by interrupt

5.7.4.7 GPIO_INTST（GPIOx）register

GPIO_INTST (GPIOA)、GPIO_ INTST (GPIOB)、GPIO_ INTST (GPIOD)					
Address	Bit	Name	R/W	Reset value	Description
0x94 0x98 0x9C	7: 0	GPIO_ INTST (GPIOA) GPIO_ INTST (GPIOB) GPIO_ INTST (GPIOD)	R/W	0x00	Each port bit interrupt flag 0x1: Interrupt 0x0: No interrupt When an interrupt occurs, the corresponding bit is pulled high to 1, and write 1 to the bit to clear it.

5.7.4.8 GPIO_NMISC register

Address	Bit	Name	R/W	Reset	Description

				value	
0xA0	7	RESERVED	——	——	Reserved
	6	GPIO_NMISC_NMPDD	R/W	0x0	Interrupt request (NMI) pull-up device disabled 0x1: NMI interrupt pull-up function disabled 0x0: NMI interrupt pull-up device enable; This bit only controls the pull-up function under the NMI function.
	5	GPIO_NMISC_EDG	R/W	0x0	NMI interrupt type selection. 0x1: Rising edge, high level 0x0: Falling edge, low level
	4	GPIO_NMISC_PE	R/W	0x0	NMI interrupt port enable 0x1: External port configuration function is NMI interrupt 0x0: The external port is not configured to function as NMI interrupt The application should clear the NMI interrupt flag before the configuration is enabled to prevent entering the interrupt by mistake.
	3	GPIO_NMISC_IF	RO	0x0	NMI interrupt flag 0x1: Interrupt 0x0: No interrupt
	2	GPIO_NMISC_ACK	R/W	0x0	Used to confirm interrupt request event. Writing 0 has no meaning or effect. Reading always returns 0. Write 1 to clear the interrupt flag. The level interrupt cannot clear the flag until the level interrupt fails.
	1	GPIO_NMISC_IE	R/W	0x0	NMI interrupt enable 0x1: Generate an interrupt request when NMI is valid 0x0: No interrupt request is generated when NMI is valid (using polling)
	0	GPIO_NMISC_MOD	R/W	0x0	NMI interrupt mode selection 0x1: All modes of interrupt 0x0: Interrupt on falling edge or rising edge

5.7.5 Example program

```
//Main function:
int main(void)
{
    gpio_init(GPIOA,GPIO_MODE_IPU,GPIO_PIN_5);              //PA5 is set as input
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);              //PE5 is set as output
    gpio_init(GPIOA,GPIO_MODE_OUT,GPIO_PIN_2);              //PA2 is set as output

    gpio_trigge_mode(GPIOA,GPIO_TRG_FALLING,GPIO_PIN_5);    // PA5 is set to falling edge
trigger interrupt
```

```
    gpio_interrupt_set(GPIOA,GPIO_PIN_5,ENABLE);                //PA5 interrupt enable

    while(1)
    {
        /*On the hardware circuit, PA2 is connected to PA5. When the level of PA2 flips, it
will change the PA5 input level state, trigger an external interrupt, and enter the GPIO
interrupt service function */
        gpio_bit_set(GPIOA,GPIO_PIN_2);
        gpio_bit_reset(GPIOA,GPIO_PIN_2);
    }
}
//Subfunction:
/*****************GPIO initialization function*******************/
void gpio_init(uint32_t gpio_periph,GPIO_MODE mode,uint8_t pin)
{
    switch(mode){
        case GPIO_MODE_IN_FLOATING:
            GPIO_PTDD(gpio_periph) &= ~pin;
            GPIO_PTPE(gpio_periph) &= ~pin;     //Configure the port as floating input mode
            break;
        case GPIO_MODE_IPU:
            GPIO_PTDD(gpio_periph) &= ~pin;
            GPIO_PTPE(gpio_periph) |= pin;  //Configure the port to be in pull-up input mode
            break;
        case GPIO_MODE_OUT:
            GPIO_PTDD(gpio_periph) |= pin;  //Configure the port to output mode
            break;
        default:
            break;
    }
}
/*********GPIO Interrupt trigger mode selection function**********/
ErrorStatus gpio_trigge_mode(uint32_t gpio_periph,GPIO_TRG_MODE trg_mode,uint8_t pin)
{
    if(!((gpio_periph == GPIOA) || (gpio_periph == GPIOB) || (gpio_periph == GPIOD))){
        return ERROR;
    }                                   //Ports other than Group A, B, D return ERROR
    /* GPIO interrput trigge mode configuration */
    switch(trg_mode){
        case GPIO_TRG_HIGH:
            GPIO_PTSC(gpio_periph) |= GPIO_PTSC_TRGMOD;
            GPIO_PTES(gpio_periph) |= pin;              //High level triggers interrput
            break;
        case GPIO_TRG_LOW:
```

```
            GPIO_PTSC(gpio_periph) |= GPIO_PTSC_TRGMOD;
            GPIO_PTES(gpio_periph) &= ~pin;          //Low level triggers interrupt
            break;
        case GPIO_TRG_RISING:
            GPIO_PTSC(gpio_periph) &= ~GPIO_PTSC_TRGMOD;
            GPIO_PTES(gpio_periph) |= pin;           //Rising edge triggers interrupt
            break;
        case GPIO_TRG_FALLING:
            GPIO_PTSC(gpio_periph) &= ~GPIO_PTSC_TRGMOD;
            GPIO_PTES(gpio_periph) &= ~pin;          //Falling edge triggers interrupt
            break;
        default:
            return ERROR;
    }
    /* gpio trigge pin enable */
    GPIO_PTPS(gpio_periph) |= pin;                   //Port triggers interrupt enable
    return SUCCESS;
}
/************GPIO port interrupt enable************/
void gpio_interrupt_set(uint32_t gpio_periph,uint8_t pin,FunctionalState value)
{
        /* gpio trigger interrupt enable */
    if(ENABLE == value){
        GPIO_PTSC(gpio_periph) |= GPIO_PTSC_IE;
        NVIC_EnableIRQ(GPIO_IRQn);
    }else{
        GPIO_PTSC(gpio_periph) &= ~GPIO_PTSC_IE;
        NVIC_DisableIRQ(GPIO_IRQn);
    }                                                //Port interrupt configuration
}
/************GPIO port outputs high level************/
void gpio_bit_set(uint32_t gpio_periph,uint8_t pin)
{
    GPIO_PTD(gpio_periph) |= pin;                                  //Port outputs high level
}
/************GPIO port outputs low level************/
void gpio_bit_reset(uint32_t gpio_periph,uint8_t pin)
{
    GPIO_PTD(gpio_periph) &= ~pin;                               //Ports outputs low level
}
/************GPIO interrupt service function************/
uint8_t time_flag = 0;
void GPIO_IRQHandler(void)
{
```

```
    time_flag = ~time_flag;
    if(time_flag)
    {
        gpio_bit_set(GPIOE,GPIO_PIN_5);
    }
    else
    {
        gpio_bit_reset(GPIOE,GPIO_PIN_5);                    //Level flip
    }
    uint8_t stateA = GPIOA_INTSTA;
    uint8_t stateB = GPIOB_INTSTA;
    uint8_t stateD = GPIOD_INTSTA;
    clr_gpio_interrupt_state(GPIOA,stateA);
    clr_gpio_interrupt_state(GPIOB,stateB);
    clr_gpio_interrupt_state(GPIOD,stateD);             //Clear interrupt flag
}
```

5.8 ADC

The analog-to-digital converter ADC can convert continuous analog signals into discrete digital signals. The ADC is a 12-digit successive approximation ADC. The configurable register makes the timing control easier and can be conveniently applied to communicate with peripherals on automotive MCU.

5.8.1 Features

1. Linear SAR ADC with 12-bit resolution;
2. A total of 28 analog input ports, of which 24 are external;
3. Output formatted in 12-bit and 8-bit right-justified unsigned format
4. Single or continuous conversion;
5. Configurable sampling and conversion time;
6. Support hardware trigger ADC sampling function;
7. Automatic compare with interrupt for less-than, or greater-than or equal-to, programmable value;
8. Support for triggering ADC effective sampling system when the system is in sleep mode.

5.8.2 Functional description

This section introduces the main functions of BF7006AMXX ADC in detail.

5.8.2.1 General description

During reset or when ADC_SC1_ADCH=0x11111, the ADC module is disabled. When the current conversion has been completed and the next conversion has not been initiated, the module enters the lowest power consumption state.

ADC can implement analog-to-digital conversion on any channel selected by software. After the conversion is completed, the result is saved in the data register ADC_DATA. Then set the conversion

completion flag ADC_SC1_COCO, if the conversion completion interrupt ADC_SC1_AIEN =1 has been enabled, the interrupt will be triggered.

The ADC module can automatically compare the conversion result with the contents of the compare register. The comparison function is enabled by setting the ADC_SC2_ACFE0 or ADC_SC2_ACFE1 bit and running with any conversion mode and configuration.

5.8.2.2 Channel selection

There are 28 analog inputs, of which 24 are external ADC channels, with temperature sensors and internal gap band channels inside. The pin control register ADC_APCTL is used to disable the I/O control function of the pin as an analog input. If the reserved channel is selected, it is fixed as pin VREFS input.

ADC_SC1_ADCH	Channels	Input
00000~10111	AD0~AD23	ADC0~ADC23
11000~11001	AD24~AD25	Reserved
11010	AD26	Temperature sensor
11011	AD27	BG（1.267V）
11100	Reserved	Reserved
11101	VREFA	VREFA
11110	VREFS	VREFS
11111	Module disabled	Module disabled

5.8.2.3 Trigger methods

The ADC controller has two trigger types: software trigger and hardware trigger. ADC_SC2_ADTRG is used to select the trigger type of conversion. When software trigger is selected, the conversion can be initiated after configuring the channel. When the hardware trigger is selected, the conversion can be initiated after the following hardware triggers.

There are three types of hardware triggers, which can be selected by register bits ADC_SC2_ADHTS and ADC_IWK:

- Internal interrupt trigger——When ADC hardware trigger is enabled, RTC counter overflow triggers ADC conversion;
- Internal interrupt trigger——PWM count trigger ADC conversion（5.3.4.6）. This trigger does not support system low-power sleep mode;
- External interrupt NMI triggers ADC conversion;

When the system provides the hardware trigger source and the hardware trigger has been enabled, the conversion is initiated on the rising edge of the hardware trigger. If a rising edge occurs again while the conversion is in progress, the rising edge is ignored. In continuous conversion mode, only the first rising edge that triggers continuous conversion is valid. The hardware trigger function can be combined with any conversion mode and configuration to run together.

5.8.2.4 ADC conversion

ADC conversion can be performed in 12-bit or 8-bit mode, depending on the ADC_CFG_MODE bit. The conversion can be triggered by software or hardware. In addition, the ADC module sampling time can be set, and continuous conversion can also be configured, and the method of automatically comparing the conversion result with the comparison value set by the software can be configured

Conversion mode:

1. During normal working, select whether to perform continuous conversion and continuously convert the same ADC channel;
2. In the system low power consumption mode, only the hardware trigger single conversion mode is supported. The ADC is in the low power consumption mode before the trigger. After the trigger event, the ADC conversion sequence is issued after waiting for 100us, and continuously convert is to convert the same channel continuously.

Initiate a conversion：

1. Select software trigger or hardware trigger
2. If continuous conversion is enabled, a new conversion will be initiated automatically after the current conversion is completed. In software trigger operation, continuous conversion starts after the channel is selected, and continues until it is aborted. In hardware trigger operation, continuous conversion starts after the hardware trigger event and continues until it is aborted.

It is not allowed to switch modes during ADC conversion. If necessary, please disable ADC_SC1_ADCH first, and then enable ADC_SC1_ADCH after completing the mode switch configuration.

Complete the conversion：

When the conversion result is transferred to the data result register ADC_DATA, the conversion is complete.

When configuring the data comparison function, if the comparison condition is not met, ADC_SC1_COCO will not be set; in other cases, ADC_SC1_COCO will be set. By setting the ADC_SC1_COCO flag, if ADC_SC1_AIEN =1 when ADC_SC1_COCO is set, an interrupt will be triggered.

Abort conversion：

Any conversion in progress will be aborted when the following conditions occur:

1. Reconfigure channel selection;
2. ADC_SC2, ADC_CFG, ADC_CV0, ADC_CV1 control and configuration register write. This indicates that the operating mode has changed and the current conversion is invalid;
3. MCU reset; when the conversion is aborted, the content of the data register ADC_DATA remains unchanged, which is the value transferred after the last successful conversion. If due to chip reset, ADC_DATA is the reset value.

5.8.2.5 Automatic compare

The ADC can be configured with a comparison function to check the upper or lower limit. After the input is sampled and converted, the result is compared with the comparison value ADC_CV0 or ADC_CV1.

The comparison condition is set by ADC_SC2_ACFGT0 or ADC_SC2_ACFGT1. After the comparison condition is met, ADC_SC1_COCO is set, and the conversion result is directly transmitted to ADC_DATA.

At the same time, the ADC also supports the double condition comparison function, which is valid only when the comparison function is valid(ADC_SC2_ACFE=1). ADC_SC2_ ACFGT1 determines the second

comparison condition, and can also be configured with upper and lower comparison limits. ADC_CV1 corresponds to the second comparison value. When the two comparisons are enabled, the ADC_SC1_COCO will be set only if both comparison results are satisfied. The conversion result is directly transmitted to ADC_DATA.

The comparison function can be used to monitor the voltage of the channel, and the MCU may be in a low-power mode at this time.When the comparison condition is met, the ADC interrupt will wakeup the MCU system.

5.8.2.6 Sleep wakeup

In system sleep mode, only hardware trigger single conversion mode is supported. ADC is closed before triggering, ADC is enabled after triggering event, ADC sampling and conversion timing is sent out after 100us, and conversion is triggered once in a cycle.

The hardware triggers to initiate the conversion. If the ADC interrupt is enabled, the conversion completion event (when the comparison function is set, a comparison match event occurs) will set ADC_SC1_COCO, generate an ADC interrupt, and wakeup the MCU system from sleep mode.

According to whether automatic comparison is configured, it can be applied to the following situations:

1. Set the automatic comparison function, configure the ADC to disable before entering the sleep mode. When the trigger condition is met, the system first enables the system clock source, and enables the ADC clock after 500us. The ADC controller enables the ADC first, waits 100us and then enables an ADC conversion. After that, the hardware automatically sets the ADC to close and waits for the next hardware trigger condition to be met. During the cycle, ADC_SC1_COCO is set to generate an ADC interrupt and wakeup the MCU system when the conversion result reaches the comparison condition;
2. Disable the automatic comparison function, configure the ADC to disable before entering the sleep mode. When the trigger condition is met, the system first enables the system clock source, and enables the ADC clock after 500us. The ADC controller enables the ADC first, and waits for 100us to enable an ADC conversion. Then directly set ADC_SC1_COCO to generate ADC interrupt and wakeup the MCU system.
3. In other configuration situations, ADC scanning will not be started after entering sleep mode.

If you want to use the ADC wakeup mode, before MCU enters sleep mode, you need to configure ADC_SC2_ADTRG=1 to select the hardware trigger type, ADC_SCSC1_ADCO=0 to select the single conversion mode, ADC_SC2_ADHTS to select the hardware trigger source, configure ADC_PD =1, and enable the ADC channel ADC_SCSC1_ADCH, and finally configure the system to enter sleep mode in sequence. Similarly, the ADC also supports the ability to configure the RTC timing overflow to trigger ADC sampling, or configure an external NMI interrupt to trigger ADC sampling. After exiting from the system sleep mode, the ADC enable will be disabled, and a software or hardware trigger is required to restart the conversion.

5.8.2.7 Time sequence description

ADC timing time complies with the following requirements:
Timer1=（ADC_CKC_SAMDEL+2）* Single ADC clock time；
Sample_Timer= （ADC_SPT+1）* Single ADC clock time；

Time2=（ADC_CKC_WNUM +3）* Single ADC clock time；
Time3=（12+2）* Single ADC clock time（12-bit mode）
Time3=（8+2）* Single ADC clock time（8-bit mode）
Timing requirements:（ADC_CKC_WNUM +3）*Single ADC clock time> 4* ADC sampling trigger clock
ADC conversion time：T= Timer1+ Sample_Time+ Time2+ Time3

5.8.2.8 Self test function

Capacitance self-test：This test can be completed by configuring and executing the calibration process.Then check whether the data is consistent with the calibration code.

Short circuit self-test：Test short circuit. This test function can be completed by configuring different channels to perform normal conversion. You can configure whether to calibrate, configure the data bit width, and check whether the converted data meets the standard.
Test process：
1. Configure channel "11101" (VREFA), normal conversion is completed, and data 1 is obtained；
2. Configure channel "11110" (VREFS), normal conversion is completed, and data 2 is obtained; Check whether the data 1/2 meets the requirements：data 1==0xfff（12-bit data mode）/0xff（8-bit data mode），data 2==0x0。

Open circuit self-test：It needs to be configured to enter the self-test mode to test the connection of each ADC channel (each channel includes two conversions of VREFA/VREFS sampling in advance). You can configure whether to calibrate, configure the data bit width, and check whether the converted data meets the standard.
Test process：
1. Configure to enter the self-test mode, configure the first conversion (VREFA) of channel "00000", and get data 1 after the conversion is completed;
2. Configure the second conversion (VREFS) of the channel, and the conversion is completed to get data 2;
3. Configure the first conversion (VREFA) of channel "00001", data 1 is obtained after the conversion is completed;
4. Configure the second conversion (VREFS) of the channel, and the conversion is completed to get data 2;

Perform two self-test conversions on all channels in turn to check whether the data 1 and data 2 of each channel are consistent. If the data is consistent, the test is normal.

The automatic comparison function does not distinguish which mode, no matter whether there is calibration or not, whether it is a test mode, it can be equipped with an automatic comparison function. Several test modes are mutually exclusive, which means that self-test, calibration enable, and capacitance test enable cannot be configured at the same time. When configuring calibration enable, regardless of other configurations, calibration operations are required, which means that calibration enable has the highest priority.

5.8.3 Register map

ADC module address range：0x5009_0000~0x5009_FFFF

Address offset	Register name	R/W	Functional description	Initial value
0x00	ADC_SC1	R/W	ADC status and control register1	0x1F
0x04	ADC_SC2	R/W	ADC status and control register2	0x01
0x08	ADC_DATA	R	ADC data result register	0x000
0x0C	ADC_CV0	R/W	ADC compare value register0	0x000
0x10	ADC_CV1	R/W	ADC compare value register1	0x000
0x14	ADC_CFG	R/W	ADC configuration register	0x00
0x18	ADC_APCTL	R/W	ADC pin enable register	0x000000
0x1C	ADC_SPT	R/W	ADC sampling time configuration register	0x002
0x78	ADC_CKC	R/W	ADC analog clock and interval control register	0x00
0x80	ADC_PD	R/W	ADC analog shutdown register	0x1
0x84	ADC_TEST	R/W	ADC self-test mode register	0x0
0x8C	ADC_IWK	R/W	ADC hardware internal trigger selection register	0x0
0x90	ADC_FRSEL	R/W	ADC input signal filter selection register	0x0

5.8.4 Register description

5.8.4.1 ADC_SC1 register

Address	Bit	Name	R/W	Reset value	Description
0x00	7	ADC_SC1_COCO	RO	0x0	Conversion complete flag 0x1: Conversion complete 0x0: Incomplete conversion or no conversion The COCO flag is a read-only bit. It is set every time the conversion is completed. The bit is cleared when ADC_DATA is read.
	6	ADC_SC1_AIEN	R/W	0x0	Interrupt enable 0x1: Conversion complete interrupt enable 0x0: Conversion complete interrupt disabled This bit is used to enable the conversion complete interrupt. When ADC_SC1_COCO is set and this bit is 1, the interrupt will be triggered.

	5	ADC_SC1_ADCO	R/W	0x0	Continuous conversion enable 0x1: Continuous conversion 0x0: Single conversion
	4:0	ADC_SC1_ADCH	R/W	0x1F	Input channel selection When the channel selection bits are all set to 1, the channels are all closed. This function can completely disable ADC and isolate all input channels. Terminating the continuous conversion in this way can prevent an unnecessary single conversion. When continuous conversion is prohibited, there is no need to set the ADC in a low-power state by setting the channel selection bits to 1, because the module will automatically enter a low-power state when the conversion is completed.

5.8.4.2 ADC_SC2 register

Address	Bit	Name	R/W	Reset value	Description
0x04	7	ADC_SC2_ADACT	RO	0x0	Conversion status: 0x1: Conversion in progress 0x0: Conversion not in progress This bit indicates that a conversion is in progress. ADC_SC2_ADACT is set when the conversion is started, and cleared automatically when the conversion is completed or aborted
	6	ADC_SC2_ADTRG	R/W	0x0	Conversion trigger type selection: 0x1: Select hardware trigger 0x0: Select software trigger
	5	ADC_SC2_ACFE0	R/W	0x0	Comparator 0 compare function enable: 0x1: Compare function enable 0x0: Compare function disable
	4	ADC_SC2_ACFGT0	R/W	0x0	Comparator 0 comparison condition selection: 0x1: Trigger when the input is greater than or equal to the comparison value 0x0: Trigger when the input is less than the comparison value
	3	ADC_SC2_ACFE1	R/W	0x0	Comparator 1 compare function enable: 0x1: Compare function enable 0x0: Compare function disable
	2	ADC_SC2_ACFGT1	R/W	0x0	Comparator 1 comparison condition selection: 0x1: Trigger when the input is greater than or equal to the

					comparison value 0x0: Trigger when the input is less than the comparison value
	1	ADC_SC2_ADHTS	R/W	0x0	ADC hardware trigger selection 0x1: External interrupt request (NMI) pin 0x0: Internal interrupt request (including RTC overflow or PWM timer overflow interrupt)
	0	ADC_SC2_COREN	R/W	0x1	12-bit data calibration enable 0x1: Enable 0x0: Disable It only controls whether the data undergoes calibration calculation in 12-bit conversion mode.

5.8.4.3 ADC_DATA register

Address	Bit	Name	R/W	Reset value	Description
0x08	11:0	ADC_DATA	RO	0x0	ADC converted data

5.8.4.4 ADC_CV0 register

Address	Bit	Name	R/W	Reset value	Description
0x0C	11:0	ADC_CV0	R/W	0x0	Comparison value of comparator 0

5.8.4.5 ADC_CV1 register

Address	Bit	Name	R/W	Reset value	Description
0x10	11:0	ADC_CV1	R/W	0x0	Comparison value of comparator 1

5.8.4.6 ADC_CFG register

Address	Bit	Name	R/W	Reset value	Description
0x14	7	RESERVED	——	——	Reserved
	6:4	ADC_CFG_ADIV	R/W	0x0	Clock division selection Select the frequency division used to generate the ADC clock.
	3	RESERVED	——	——	Reserved
	2	ADC_CFG_MODE	R/W	0x0	Conversion mode selection 0x1: 12-bit conversion 0x0: 8-bit conversion
	1:0	RESERVED	——	——	Reserved

ADC_CFG_ADIV	Frequency division multiple	System clock frequency 32MHz	System clock frequency 16MHz	System clock frequency 8MHz
000	1	--	16	8
001	2	16	8	4
010	4	8	4	2
011	6	5.33	2.67	1.33
100	8	4	2	1
101	10	3.2	1.6	0.8
110	12	2.67	1.33	0.67
111	3	10.67	5.33	2.67

5.8.4.7 ADC_APCTL register

Address	Bit	Name	R/W	Reset value	Description
0x18	23:0	ADC_APCTL0 ~ ADC_APCTL23	R/W	0x0	ADC0~ADC23 port function enable 0x1: ADC input function of this port is enabled 0x0: ADC input function of this port is disabled

5.8.4.8 ADC_SPT register

Address	Bit	Name	R/W	Reset value	Description
0x1C	9:0	ADC_SPT	R/W	0x2	ADC sampling time configuration Sampling time = (ADC_SPT + 1) ADC clock Note: When configuring, ensure that ADC_CKC_WNUM *ADC clock cycle>5* (ADC_CKC_CKV configuration cycle of the system clock)

5.8.4.9 ADC_CKC register

Address	Bit	Name	R/W	Reset value	Description
0x78	7	ADC_CKC_SAMBG	R/W	0x0	Sampling timing and comparison timing interval selection 0x1: 1 ADC clock interval 0x0: No interval
	6:4	ADC_CKC_SAMDEL	R/W	0x0	Sampling delay time selection 0x7: 16 ADC clocks 0x6: 14 ADC clocks 0x5: 12 ADC clocks 0x4: 10 ADC clocks 0x3: 8 ADC clocks 0x2: 4 ADC clocks 0x1: 2 ADC clocks 0x0: 0 ADC clocks
	3:2	ADC_CKC_WNUM	R/W	0x0	Selection of distance conversion interval after sampling 0x3: 6 ADC clocks 0x2: 5 ADC clocks 0x1: 4 ADC clocks 0x0: 3 ADC clocks
	1:0	ADC_CKC_CKV	R/W	0x0	Analog input clock signal frequency division 0x3: System working clock divided by 8 0x2: System working clock divided by 4 0x1: System working clock divided by 2 0x0: System working clock divided by 1

5.8.4.10 ADC_PD register

Address	Bit	Name	R/W	Reset value	Description
0x80	7:1	RESERVED	——	——	Reserved
	0	ADC_PD	R/W	0x0	Disable the analog ADC to reduce power consumption 0x1: Disable 0x0: Enable

5.8.4.11 ADC_TEST register

Address	Bit	Name	R/W	Reset value	Description
0x84	7:2	RESERVED	——	——	Reserved
	1	ADC_TEST_NUM	R/W	0x0	Self-test mode count 0x1: Second conversion 0x0: First conversion
	0	ADC_TRST_EN	R/W	0x0	Self-test mode enabled 0x1: Enable 0x0: Disable

5.8.4.12 ADC_IWK register

Address	Bit	Name	R/W	Reset value	Description
0x8C	7:1	RESERVED	——	——	Reserved
	0	ADC_IKW	R/W	0x0	ADC hardware trigger internal source selection 0x1: Select PWM count trigger 0x0: Select real-time counter (RTC) overflow trigger

5.8.5 ADC reference application process

ADC reference application process:

1. ADC initialization

   a) Configure ADC scan mode, ADC data bit width, ADC clock configuration, sampling clock

number;

b) ADC interrupt trigger scan enable and interrupt trigger source configuration;

c) ADC comparator enable and parameter configuration;

d) ADC channel is selected as the analog input configuration;

e) ADC interrupt enable;

2. Configure ADC conversion channel and turn on ADC conversion

3. Read ADC conversion data

a) Interrupt mode: read the conversion data in the interrupt service function;

b) Query mode: After the ADC conversion is completed, read the ADC data result register;

4. Perform step 2 to switch to the next channel

Note: In continuous conversion mode, only the continuous conversion of the current channel is performed, and the next conversion is automatically performed after the conversion is completed, without the need to turn on the ADC conversion.

6 Electrical characteristics

6.1 Thermal Characteristics

Indicator	Minimum	Typical	Maximum	Unit	Remarks
Operating temperature	-40	——	+125	℃	
storage temperature	-55	——	+150	℃	
Reflow soldering temperature	——	——	+260	℃	

6.2 Moisture sensitivity index

Indicator	Minimum	Typical	Maximum	Unit	Remarks
MSL	——	3	——	——	

6.3 ESD index

Indicator	Minimum	Typical	Maximum	Unit	Remarks
ESD HBM	-6000	——	+6000	V	
ESD CDM	-500	——	+500	V	
LATCH UP	-50	——	+50	mA	Full temperature

6.4 Power Characteristics

Indicator	Minimum	Typical	Maximum	Unit	Remarks
VDD	3.3	——	5.5	V	
IDD	——	——	120	mA	

VDDA	3.3	——	5.5	V	
VREFA	——	VDDA	——	V	

Note: VREFA voltage must be equal to VDDA

6.5 Input and Output

Indicator	Minimum	Typical	Maximum	Unit	Remarks
VOH	90%VDD	——	——	V	
VOL	——	——	10%VDD	V	
Output delay	——	——	70	ns	
VIH	70%VDD	——	——	V	
VIL	——	——	30%VDD	V	
Input delay	——	——	5	ns	
The sum of the total current of all IO drives	——	——	120	mA	
Internal pull-up resistor	——	20K	——	Ω	
Input hysteresis voltage	10%VDD	——	——	V	
Internal resistance	——	——	80	Ω	
Ordinary driving ability	——	——	5	mA	
Large driving capacity PB0~PB5	——	——	20	mA	
IO leakage current	——	0.1	15	uA	

6.6 Power-on, power-down, low-voltage detection

Indicator	Minimum	Typical	Maximum	Unit	Remarks
POR (recovery) POR (power down)	0.96	1.30	1.99	V	Engineering test data
	0.84	1.10	1.81	V	Engineering test data
Power-up delay	4	6	8	ms	Engineering test data
BOR (recovery) BOR (power down)	——	3.14	——	V	Engineering test data
	——	3.08	——	V	Engineering test data
Power-down delay	——	85	——	us	
Low detection voltage LVDT1	——	4.5	——	V	
Low detection voltage LVDT2	——	4.0	——	V	
Low detection voltageLVDT3	——	3.5	——	V	
Low voltage detection delay	——	10	——	us	

6.7 Clock source

Indicator		Minimum	Typical	Maximum	Unit	Remarks
RC1M cycle deviation (before trimming)		-15%	——	+15%	——	-40~+125℃
RC1M cycle deviation (after trimming)		-3%	——	+3%	——	-40~+125℃
RC128K cycle deviation (before trimming)		-30%	——	+30%	——	-40~+125℃
RC128K cycle deviation (after trimming		-15%	——	-15%	——	-40~+125℃
External oscillator XOSC		——	8 or 16	——	MHz	-40~+125℃
External crystal oscillator start-up time		——	3.1（8M） 3.1（16M）	——	ms	Engineering test data
PLL frequency deviation		——	——	——	——	-40~+125℃
PLL lock time		——	100	——	us	-40~+125℃

6.8 Analog-to-Digital converter ADC

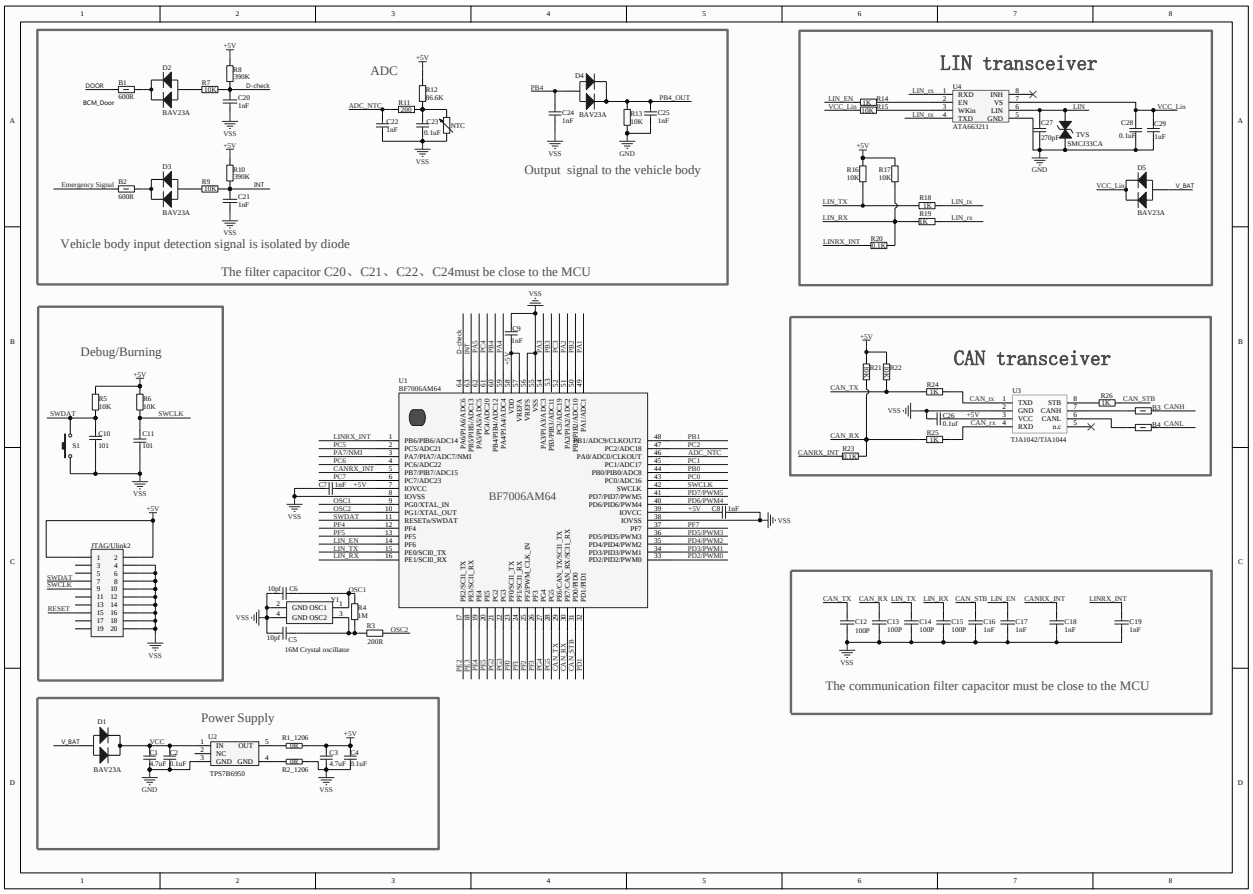
Indicator		Minimum	Typical	Maximum	Unit	Remarks
Rate		——	——	1	MHz	
Resolution		——	——	12	bit	
Sampling time		1	——	1024	Tadc_clk	
Conversion time		16	——	1062	Tadc_clk	
Input resistance	FILTER_R_SEL=0	——	0.8	——	KΩ	
	FILTER_R_SEL=1	——	5	——	KΩ	
	FILTER_R_SEL=2	——	1	——	KΩ	
Input capacitance	FILTER_R_SEL=0	——	5	——	pF	
	FILTER_R_SEL=1	——	7	——	pF	
	FILTER_R_SEL=2	——	15	——	pF	

6.9 Supply Current Characteristics

Indicator		Minimum	Typical	Maximum	Unit	Remarks
System work 32MHz		——	14.1	——	mA	Engineering test data, 25℃
System work 16MHz		——	11.8	——	mA	Engineering test data, 25℃
Sleeping mode	32MHz	——	5.9	——	mA	Engineering test data, 25℃
	16MHz	——	5.1	——	mA	Engineering test data, 25℃

DeepSleep mode	3.3V	——	25	——	uA	Engineering test data, 25℃
	5V	——	30	——	uA	Engineering test data, 25℃
RC128K	3.3V	——	4.7	——	uA	Engineering test data, 25℃
	5V					Engineering test data, 25℃
RC1M	3.3V	——	105	——	uA	Engineering test data, 25℃
	5V					Engineering test data, 25℃
XOSC	3.3V	——	179（8MHz） 245（16MHz）	——	uA	Engineering test data, 25℃
	5V	——	366（8MHz） 469（16MHz）	——	uA	Engineering test data, 25℃
PLL	3.3V	——	585	——	uA	Engineering test data, 25℃
	5V					Engineering test data, 25℃
ADC	3.3V	——	2	——	mA	Engineering test data, 25℃
	5V					Engineering test data, 25℃
BOR LVDT	3.3V	——	3.3	——	uA	Engineering test data, 25℃
	5V					Engineering test data, 25℃

7 Recommended reference applications



Note: The above application reference circuit is for reference design only.

8 Appendix A

Precautions for MCU use:

1. VCC power-on time from 0V to 3.3V, required to complete within 4ms;
2. BF7006AMXX VCC is powered on from 0V-3.1V, the GPIO port is in an uncertain state (the output follows the VCC voltage value or the output is low or the input is high-impedance state), pay attention to the protection of the peripheral components of the GPIO port;
   After VCC is powered on and is higher than 3.3V, the chip completes power-on reset, and GPIO is in the input high-impedance state by default. After the program runs, the GPIO port state is executed according to the settings in the program.

9 Appendix B

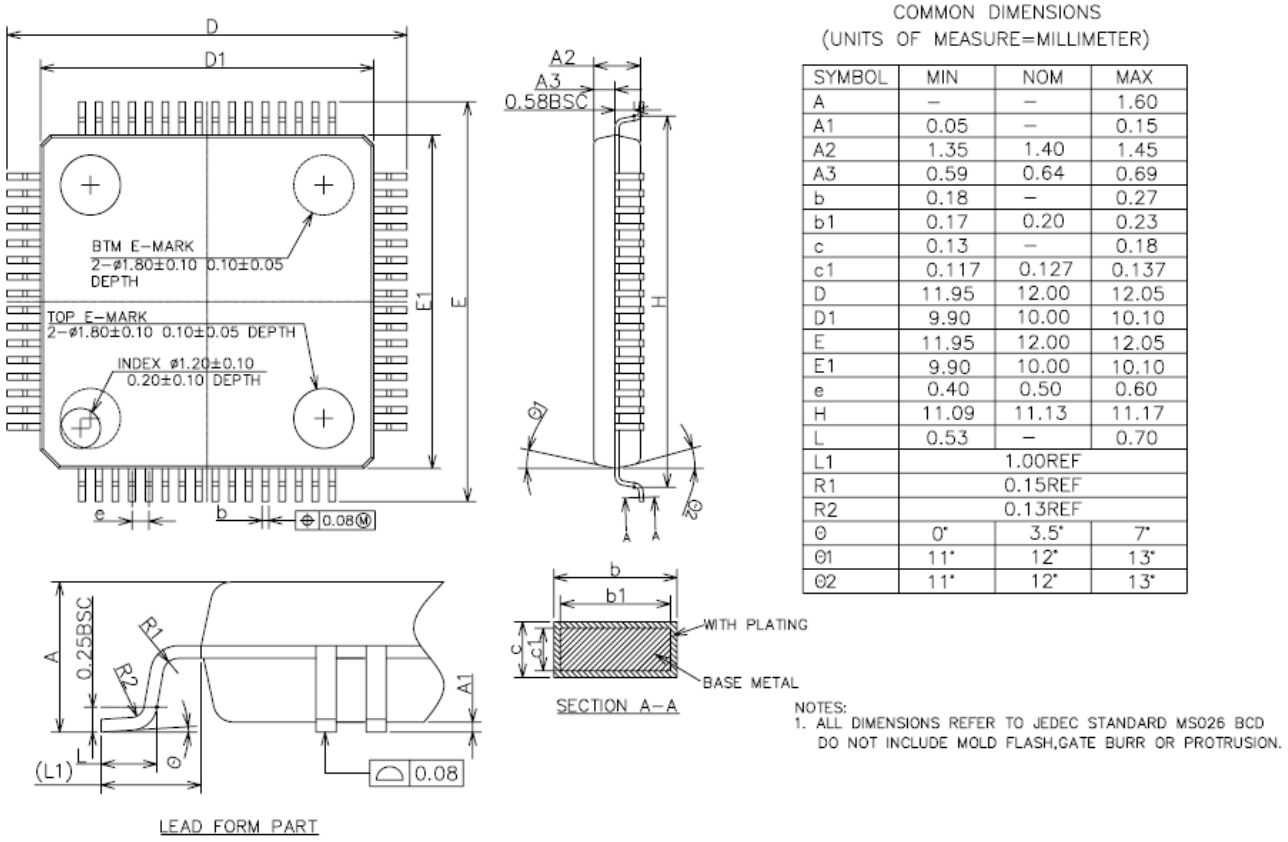
Revision history:

Edition	Date	The modified content
V1.0	2019/7/20	First edition
V1.1	2020/9/15	Application reference circuit update
V1.2	2020/12/7	Add LQFP48 pin diagram and package size
V1.3	2021/5/18	1. Increase chip power-on time requirements; 2. Added the description of the IO port state when the chip is powered on and reset in the GPIO description; 3. Add the appendix of MCU use precautions;
V1.4	2021/5/21	English version
/		
/		
/		
/		
/		

10 Package

10.1 Mechanical drawings

10.1.1 LQFP64

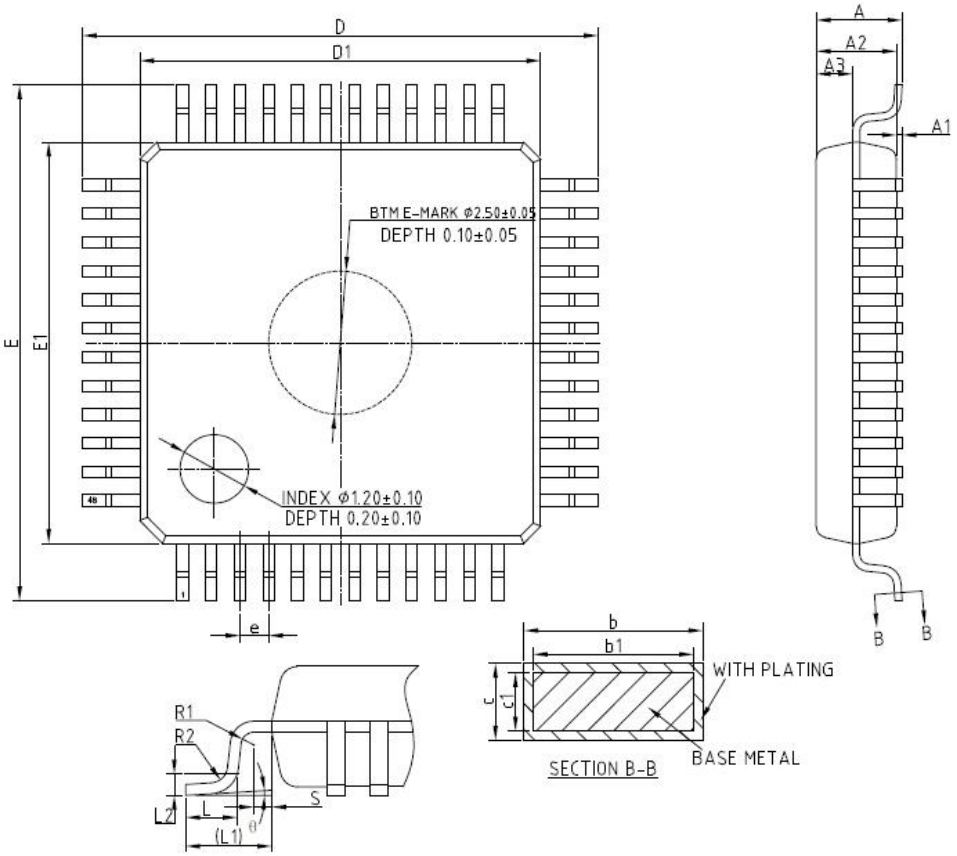


COMMON DIMENSIONS
(UNITS OF MEASURE=MILLIMETER)

SYMBOL	MIN	NOM	MAX
A	–	–	1.60
A1	0.05	–	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	–	0.27
b1	0.17	0.20	0.23
c	0.13	–	0.18
c1	0.117	0.127	0.137
D	11.95	12.00	12.05
D1	9.90	10.00	10.10
E	11.95	12.00	12.05
E1	9.90	10.00	10.10
e	0.40	0.50	0.60
H	11.09	11.13	11.17
L	0.53	–	0.70
L1	1.00REF		
R1	0.15REF		
R2	0.13REF		
θ	0°	3.5°	7°
θ1	11°	12°	13°
θ2	11°	12°	13°

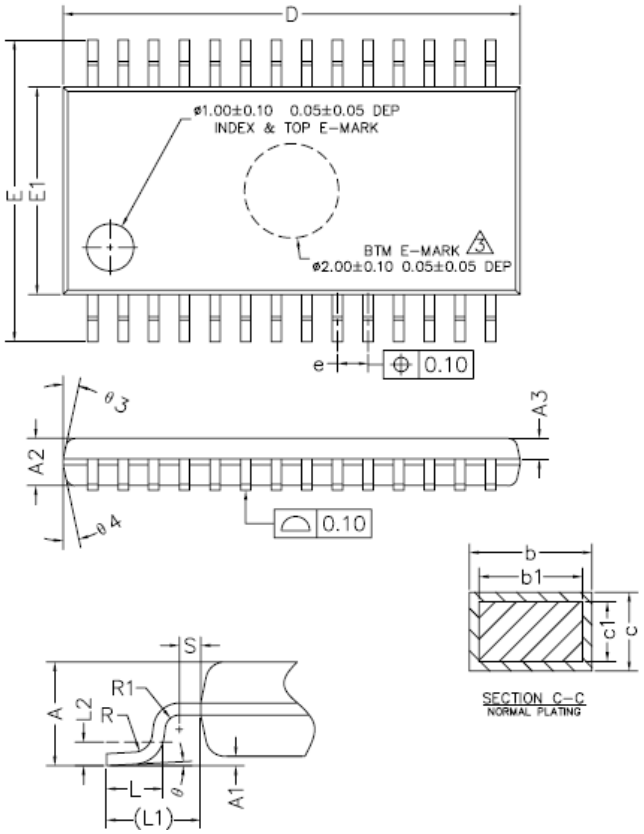
NOTES:
1. ALL DIMENSIONS REFER TO JEDEC STANDARD MS026 BCD
DO NOT INCLUDE MOLD FLASH,GATE BURR OR PROTRUSION.

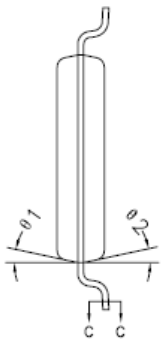
10.1.2 LQFP48



SYMBOL	MILLMETER		
	MIN	NOM	MAX
A	–	–	1.60
A1	0.05	–	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	–	0.27
b1	0.17	0.20	0.23
c	0.13	–	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	–	–
R2	0.08	–	0.20
S	0.20	–	–
θ	0°	3.5°	7°
θ 1	0°	–	–

10.1.3 TSSOP28



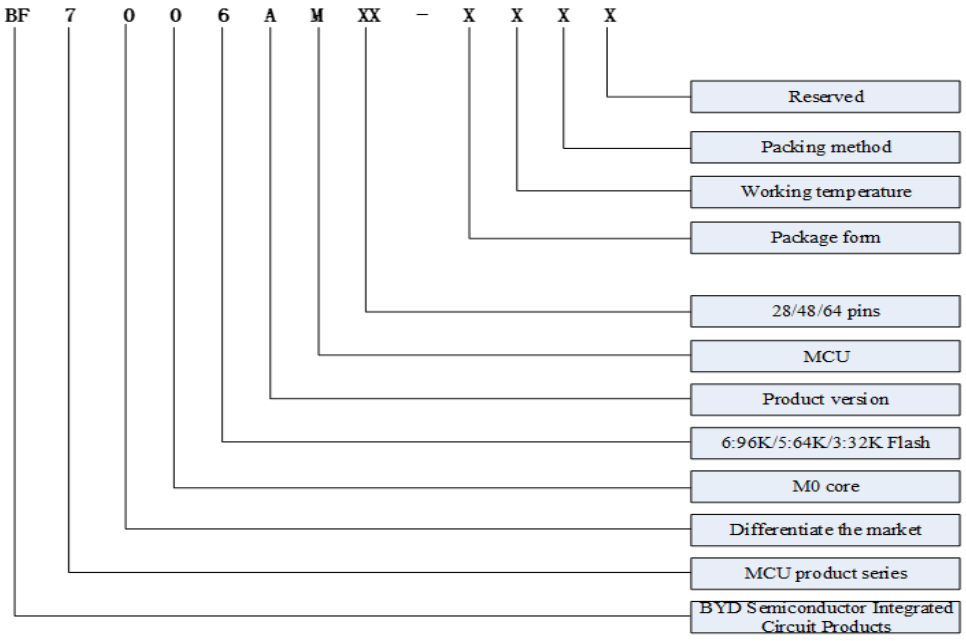


COMMON DIMENSIONS
(UNITS OF MEASURE=MILLIMETER)

SYMBOL	MIN	NOM	MAX
A	–	–	1.20
A1	0.05	–	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	–	0.29
b1	0.19	0.22	0.25
c	0.13	–	0.18
c1	0.12	0.13	0.14
D	9.60	9.70	9.80
E	6.20	6.40	6.60
E1	4.30	4.40	4.50
e	0.55	0.65	0.75
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R	0.09	–	–
R1	0.09	–	–
S	0.20	–	–
θ	0°	–	8°
θ1	10°	12°	14°
θ2	10°	12°	14°
θ3	10°	12°	14°
θ4	10°	12°	14°

11 Order information

Package	Work temperature		Package style	empty
S:SOP	Auto level	A: -40℃~+150℃	B: Taping	-
T:TSSOP		B: -40℃~+125℃	L: Material pipe	-
M:MSSOP		-	T: tray	-
L:LQFP		-	-	-
Q:QFN	Industrial level	K: -40℃~+85℃	-	-
B:BGA		J: -40℃~+105℃	-	-
D:DIP		L: -40℃~+125℃	-	-
-	Consumer level	P: -25℃~+70℃	-	-
-		Q: 0℃~+70℃	-	-



12 Disclaimer

1. The information in this document can be modified and updated without notifying the user.
2. BYD Semiconductor Co., Ltd. will do its best to ensure the high quality and high stability of the company's products. Nevertheless, due to the inherent characteristics of general semiconductor devices such as electrical sensitivity and vulnerability to external physical damage, our products may malfunction or fail under these conditions. When using our products, users are responsible for designing a safe and stable system environment in compliance with safety rules. Users can avoid possible accidents, fires and public injuries by removing redundant components, failure prevention and fire prevention measures. When the user uses the product, please follow the operating steps specified in the company's latest manual to use the product.
3. The company's product cannot and is prohibited to be applied to some special equipment that requires extremely high stability and quality, so as to avoid accidents such as casualties. Products that cannot be used include nuclear energy control equipment, aircraft and aviation devices, transportation equipment, traffic signal equipment, combustion control equipment, medical equipment, military equipment, and all safety equipment, etc. The company is not responsible for the losses and injuries caused by users in the non-product application range listed above.