BF7006AMXX 系列

BF7006AM64-LBTX

BF7006AM48-LBTX

BF7006AM28-TBBX

Datasheet

Rev1.3

2021-5-18

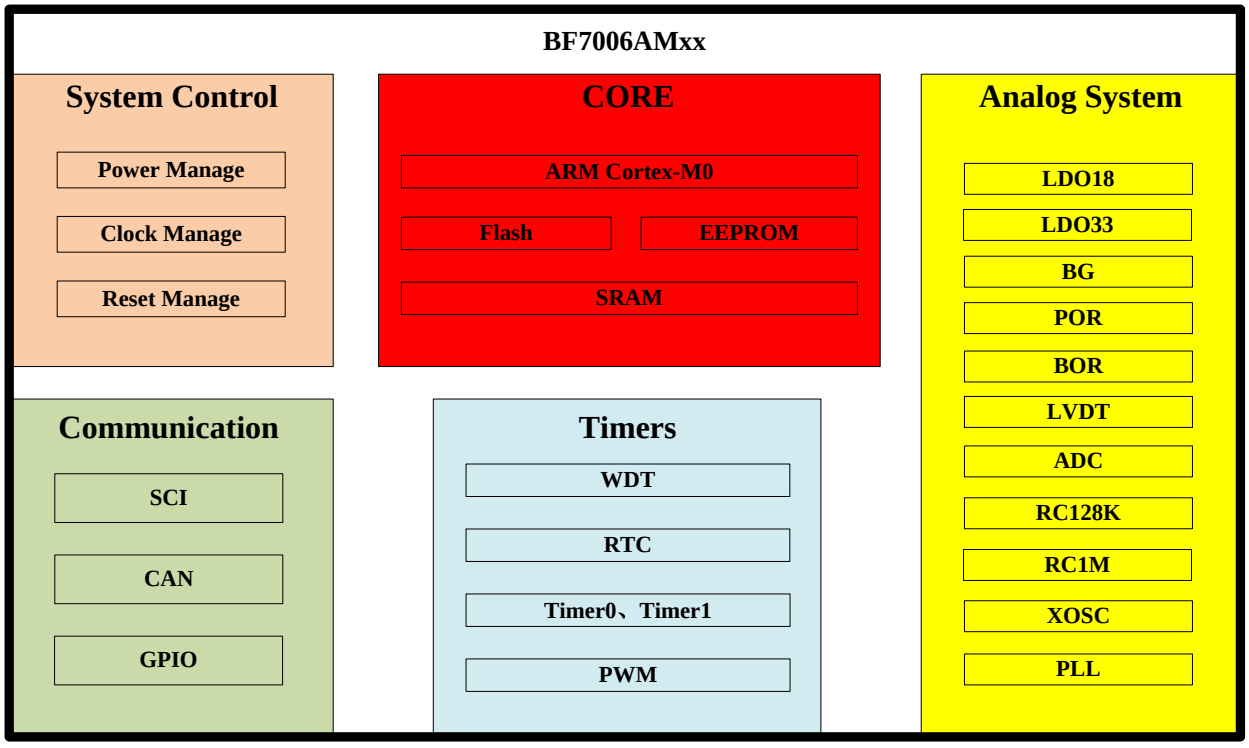
目录

1 芯片简介

BF7006AMXX 作为一款满足汽车 AECQ100 GRADE1 品质等级的 32 位通用 MCU，依照 ISO26262 ASIL-B 等级标准要求设计；其内部集成 CAN、LIN、UART 等多种通信模块，多路计数器、计时器及 PWM 功能，并包含有高精度 ADC，存储支持 EEPROM 即时数据保存等多种通用模块。封装形式为 LQFP64、LQFP48、TSSOP28，适合如电动车窗、电动座椅、雨刮、车灯、门锁等多种汽车电子车身控制应用。

1.1 系统组成概览

BF7006AMXX 芯片系统组成如下图：



1.2 芯片主要资源

BF7006AMXX 芯片主要资源如下，具体模块功能见各模块章节详细描述：

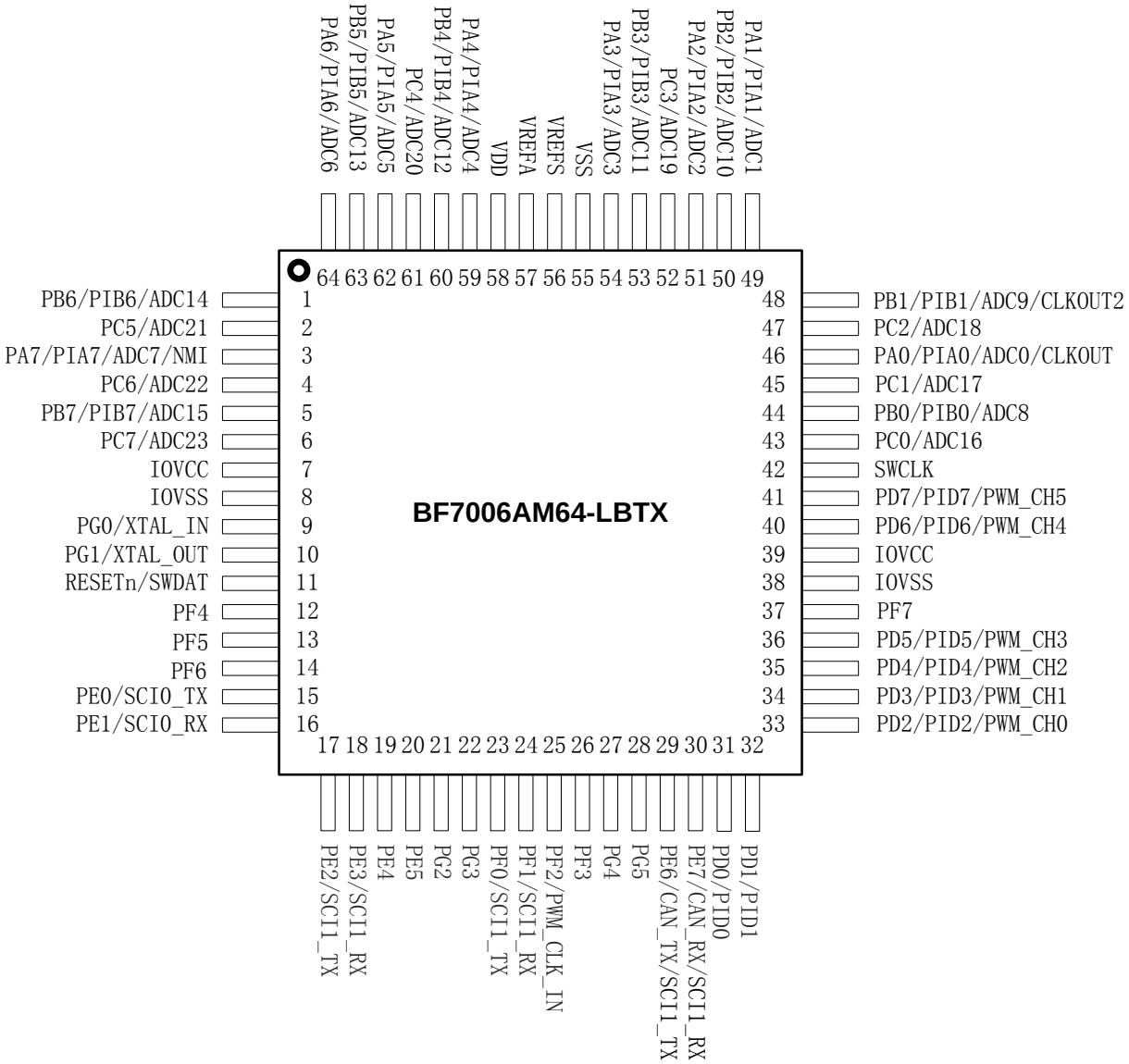
分类	特性	LQFP64	LQFP48	TSSOP28
系统	CPU	Cortex_M0	Cortex_M0	Cortex_M0
	频率	最大 32MHz	最大 32MHz	最大 32MHz
	外部中断数量	最多 24 通道	最多 24 通道	最多 14 通道
	IO 数量	最多 54 通道	最多 40 通道	最多 22 通道
	大电流驱动 IO	最多 6 通道	最多 6 通道	最多 6 通道
时钟源	内部	RC1MHz、RC128KHz、PLL	RC1MHz、RC128KHz、PLL	RC1MHz、RC128KHz、PLL
	外部	XTAL 8MHz/16MHz	XTAL 8MHz/16MHz	XTAL 8MHz/16MHz
存储体	FLASH	最大 96KB	最大 96KB	最大 96KB
	EEPROM	最大 2KB	最大 2KB	最大 2KB
	SRAM	最大 4KB	最大 4KB	最大 4KB
通信	CAN	最多 1 个	最多 1 个	最多 1 个
	SCI	最多 2 个	最多 2 个	最多 2 个
计数器/计时器	System Timer	最多 1 个，CPU 自带	最多 1 个，CPU 自带	最多 1 个，CPU 自带
	RTC	最多 1 个，32 位	最多 1 个，32 位	最多 1 个，32 位
	Timer	最多 2 个，16 位	最多 2 个，16 位	最多 2 个，16 位
	PWM	最多 6 通道，16 位	最多 6 通道，16 位	最多 6 通道，16 位
模拟模块	ADC	最多 24 通道，最大 12 位分辨率，最大 10 位精度，最大 1M 速率	最多 16 通道，最大 12 位分辨率，最大 10 位精度，最大 1M 速率	最多 8 通道，最大 12 位分辨率，最大 10 位精度，最大 1M 速率
调试	调试接口	ARM Serial Wire	ARM Serial Wire	ARM Serial Wire
	硬件断点数量	最多 2 个	最多 2 个	最多 2 个
功能安全	低压监测	3 挡位	3 挡位	3 挡位
	掉电复位	支持	支持	支持
	ECC 校验	6 位 ECC，FLASH、EEPROM	6 位 ECC，FLASH、EEPROM	6 位 ECC，FLASH、EEPROM
	存储体保护	FLASH、EEPROM	FLASH、EEPROM	FLASH、EEPROM
	CPU 死机监控	支持 CPU 死机复位	支持 CPU 死机复位	支持 CPU 死机复位
	时钟安全	支持外部晶振失效复位	支持外部晶振失效复位	支持外部晶振失效复位
	硬件看门狗	支持看门狗复位	支持看门狗复位	支持看门狗复位
	模块上电自检	CAN	CAN	CAN
主要电气特性	工作电压	3.3V~5.5V	3.3V~5.5V	3.3V~5.5V
	工作温度	-40℃~+125℃	-40℃~+125℃	-40℃~+125℃
	ESD 特性	HBM：≥6kv CDM：≥0.5kv	HBM：≥6kv CDM：≥0.5kv	HBM：≥6kv CDM：≥0.5kv
	Latch up 特性	100mA（常温）	100mA（常温）	100mA（常温）

2 管脚连接及功能说明

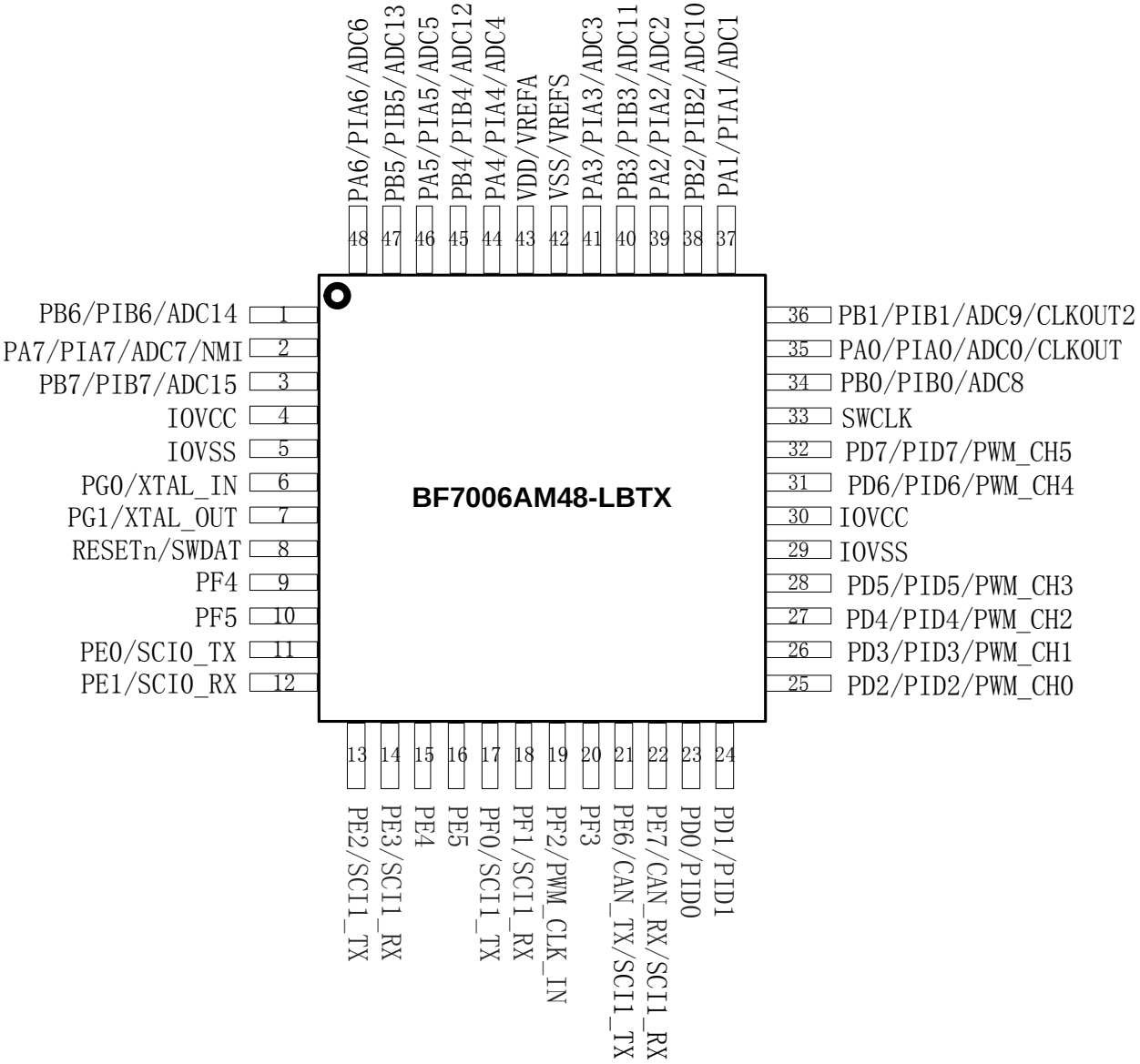
本章节描述 BF7006AMXX 芯片的封装形式及芯片管脚示意图、所有引脚的功能的详细说明以及各个管脚的功能复用情况。

2.1 管脚功能

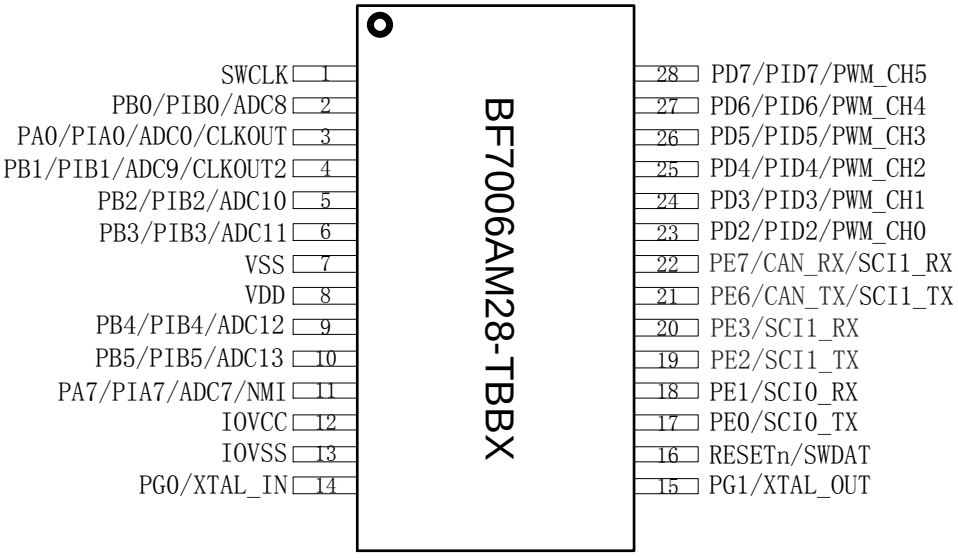
LQFP64 封装示意图：



LQFP48 封装示意图：



TSSOP28 封装示意图：



2.2 管脚功能复用关系

管脚优先级，从左到右，最低到最高								
LQFP64	LQFP48	TSSOP28	功能 1	功能 2	功能 3	功能 4	功能 5	功能 6
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	----

管脚优先级，从左到右，最低到最高								
LQFP64	LQFP48	TSSOP28	功能 1	功能 2	功能 3	功能 4	功能 5	功能 6
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 系列支持 ARM SW 双线调试协议，提供完整的上下位机、连接器及仿真工具。其中调试数据端口 SWDAT 与芯片外部复位功能端口 RESETN 复用于同一个管脚，该管脚默认为 RESETN 功能。当芯片连接调试工具时，系统会自动识别并切换该管脚为 SWDAT 功能；当芯片需要恢复 RESETN 功能时，需要重新对芯片进行上电。

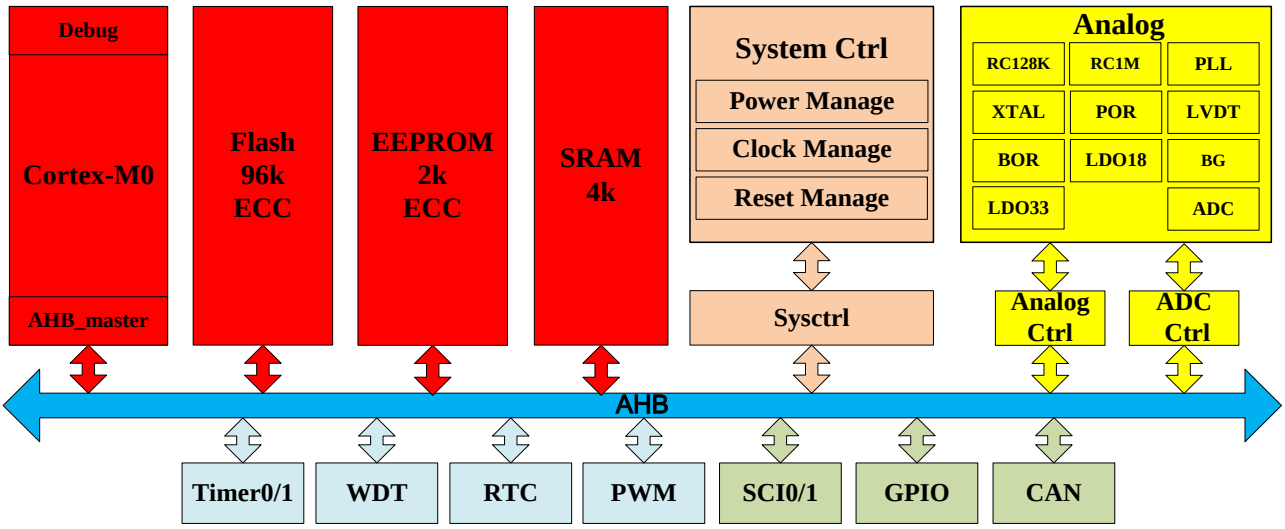
SCI1 模块映射为 3 路管脚，同一时间只能选择一路。

2.3 管脚功能说明

分类	信号名称	信号功能说明
电源管脚	IOVCC	IO 电源
	IOVSS	IO 电源地
	VDD	电源
	VSS	地
	VREFA	ADC 参考电源
	VREFS	ADC 参考电源地
系统管脚	RESETn/SWDAT	复位输入/调试数据端口（默认内部上拉）
	NMI	外部高级中断
	SWCLK	调试端口（默认内部上拉）
	CLKOUT	时钟输出，可以输出外部晶振时钟 8 分频、内部 250KHz 时钟（1MHz 分频）、内部 32KHz（128KHz 分频）时钟和 PLL 时钟 16 分频
	CLKOUT2	时钟输出 2，可以输出外部晶振时钟 16 分频和内部 1MHz 时钟的 4 分频
	XTAL_IN	晶振输入
	XTAL_OUT	晶振输出
通信接口	SCI0_RX	SCI0（UART / LIN）的输入
	SCI0_TX	SCI0（UART / LIN）的输出
	SCI1_RX	SCI1（UART / LIN）的输入
	SCI1_TX	SCI1（UART / LIN）的输出
	CAN_RX	CAN 的接收数据
	CAN_TX	CAN 的发送数据
计数器	PWM_CH0	高级计数器通道 0
	PWM _CH1	高级计数器通道 1
	PWM _CH2	高级计数器通道 2
	PWM _CH3	高级计数器通道 3
	PWM _CH4	高级计数器通道 4
	PWM _CH5	高级计数器通道 5
	PWM _CLK_IN	高级计数器外部计数时钟
ADC	ADC0~ADC23	ADC 的 24 个通道
GPIO	PAx~PGx	通用输入输出
	PIAx、PIBx、PIDx	外部中断

3 系统

BF7006AMXX 作为一款 32 位 Cortex_M0 核的 MCU，其设计架构基于 AMBA AHB2.0 总线集成，一定程度上简化了系统。系统主要包含 Cortex_M0 CPU、时钟管理单元、复位管理单元、功耗管理单元、中断管理单元、功能安全管理单元等系统功能组件。



3.1 CPU

CPU 为标准 32 位 Cortex_M0，详细请见 ARM Cortex_M0 用户手册，包括内部计时器 System Tik，此处不再赘述。

3.2 地址列表

BF7006AMXX 各个模块设备起始地址和终止地址分配如下：

设备名称		开始地址	结束地址
1	FLASH 主程序存储体	0x0000_0000	0x0001_7FFF
2	FLASH_NVR 存储体	0x0001_8000	0x0001_8FFF
3	SRAM 存储体	0x2000_0000	0x2000_0FFF
4	EEPROM 主数据存储体	0x4000_0000	0x4000_07FF
5	EEPROM_NVR 存储体	0x4000_0800	0x4000_08FF
6	FLASH/EEPROM 控制器	0x5000_0000	0x5000_FFFF
7	系统控制器	0x5001_0000	0x5001_FFFF
8	SCIO/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 控制器	0x5009_0000	0x5009_FFFF
14	GPIO	0x500A_0000	0x500A_FFFF
15	TIMER0/1	0x500B_0000	0x500B_FFFF

3.3 系统管理

BF7006AMXX 系统中包含中断管理、时钟与功耗管理、复位管理和安全管理四个部分。当中各个部分可配置实现系统的多种控制功能和管理，提供不同的应用需要。

3.3.1 中断管理

BF7006AMXX 中断基于 Cortex_M0 中断控制，外部增加相应功能扩展。主要特性为：

1. 各模块外设对应独立的中断源；
2. 外部共 24 个可用中断，其中 A，B，D 端口 24 个中断复用 1 个中断源并用中断标志位区分；
3. 系统中断 SYS_INTR 共有 9 个中断，分别为晶振启动初始化出错，晶振检测出错，FLASH_NVR 校验出错，FLASH 保护误操作，EEPROM 保护误操作，FLASH ECC 1 位出错，FLASH ECC 2 位出错，EEPROM ECC 1 位出错，EEPROM ECC 2 位出错，每位中断有对应的中断标志信号；
4. 支持 LVDT 低电压升压降压中断；
5. 支持 NMI 不可屏蔽中断；
6. 中断优先级可配；

7. 中断可使能与屏蔽；

3.3.1.1 中断分配

Cortex_M0 内部异常及中断此处不再赘述，系统中断号分配如下，确保程序编写对应正确的中断号：

CPU 中断列表	分配	对应状态寄存器位
IRQ0	---	
IRQ1	SYS_INTR 系统中断	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 中断	SYS_LVDT_IE_H SYS_LVDT_IE_L
IRQ5	PWM_CH0 中断	PWM_C0SC_IF
IRQ6	PWM_CH1 中断	PWM_C1SC_IF
IRQ7	PWM_CH2 中断	PWM_C2SC_IF
IRQ8	PWM_CH3 中断	PWM_C3SC_IF
IRQ9	PWM_CH4 中断	PWM_C4SC_IF
IRQ10	PWM_CH5 中断	PWM_C5SC_IF
IRQ11	PWM 溢出中断	PWM_SC_TOF
IRQ12	---	
IRQ13	---	
IRQ14	---	
IRQ15	---	
IRQ16	SCI0 ERR 中断	SCI_S1_RX_OVERFLOW_FLAG SCI_S1_NOSIE_ERR_FLAG
IRQ17	SCI0 RX 中断	SCI_S1_RX_FULL_FLAG SCI_S2_BREAK_CHECK_FLAG SCI_S2_RX_EDGE_FLAG
IRQ18	SCI0 TX 中断	SCI_S1_TX_EMPTY_FLAG SCI_S1_TX_COMP_FLAG
IRQ19	SCI1 ERR 中断	SCI_S1_RX_OVERFLOW_FLAG SCI_S1_NOSIE_ERR_FLAG

IRQ20	SCI1 RX 中断	SCI_S1_RX_FULL_FLAG SCI_S2_BREAK_CHECK_FLAG SCI_S2_RX_EDGE_FLAG
IRQ21	SCI1 TX 中断	SCI_S1_TX_EMPTY_FLAG SCI_S1_TX_COMP_FLAG
IRQ22	GPIO 外部中断	GPIO_PTSC_IF GPIO_ INTST（GPIOA） GPIO_ INTST（GPIOB） GPIO_ INTST（GPIOD）
IRQ23	ADC 中断	ADC_SC1_COCO
IRQ24	---	
IRQ25	RTC 中断	RTC_SC_IF
IRQ26	CAN WAKEUP 中断	CAN_CLRISR_WUPI CAN_IF_WUPI SYS_CAN_SPWKFLAG
IRQ27	CAN ERR 中断	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 中断	CAN_IF_RI
IRQ29	CAN TX 中断	CAN_IF_TI CAN_CLRISR_TI
IRQ30	Timer0 中断	TIMER_CFG_IF
IRQ31	Timer1 中断	TIMER_CFG_IF

3.3.1.2 中断使能和优先级

关于中断使能信号分两类：

1. 处理器CPU自带使能寄存器（ISER），用于控制送入到CPU的中断信号是否有效；
2. 工作模块使能寄存器，根据应用配置是否将外部中断与内部产生中断送入到CPU；
3. 外部NMI中断信号自能通过工作情况自定义是否送入到CPU；
4. 关于中断优先级设置，CPU内有8个32bit寄存器配置，每个中断入口有8bit寄存器选择优先级配置值，

默认优先级为中断0到31。详见Cortex_M0应用手册，具体不再赘述。

3.3.1.3 中断唤醒

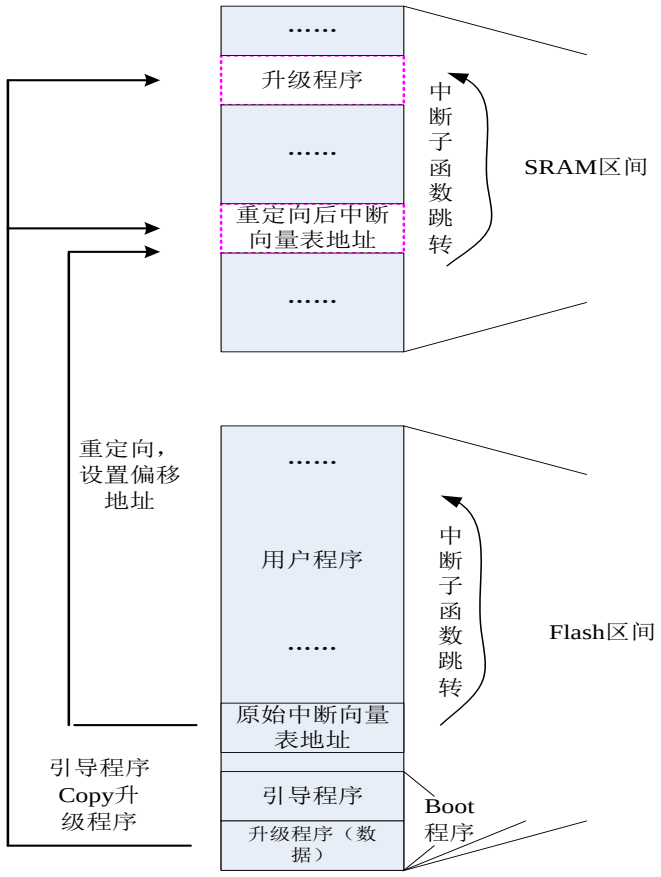
在系统低功耗模式下，关闭时钟及其他一些模拟模块以达到省功耗的目的，同时在低功耗模式下可以通过 LVDT 中断、NMI 中断、所有外部中断、CAN 中断、SCIO/1 中断、RTC 中断、WDT 复位来唤醒系统。所有可唤醒中断均为异步唤醒。根据系统不同的低功耗模式以及时钟源的选择，唤醒流程及所需时间均不相同，更具体的请见时钟及功耗管理章节。

3.3.1.4 中断向量重定向和在线升级

BF7006AMXX 支持系统向量重定向功能，用于更新用户程序后，系统可将中断列表地址映射至最新的用户程序中，正常执行用户程序的中断服务函数。

BF7006AMXX 提供两种在线升级方法：

1. 运行 BOOT 程序加载程序更新数据，在写入芯片过程中芯片 CPU 自动处于暂停状态，直至该笔用户程序更新数据成功写入 FLASH 中，以此循环；
2. 向量重定向功能还可用于在线升级操作，提供应用开发 BOOT 以及在线升级时使用：



在线升级过程中，存储在 FLASH 中的用户程序除 BOOT 程序外，用户的功能程序及中断服务程序都是需要被更新覆盖的。因此，在线升级过程中，如果需要用到中断（例如通过 CAN 进行在线升级，并且需要用到 CAN 的中断），则需要将中断向量表、中断服务程序重新定向（或者说映射）到另外一个地方。由于 FLASH 需要更新，又不能同时读写，所以一般都是定向（或者说映射）到 SRAM 中。同样的道理，由于 FLASH 在被更新的过程中，升级程序不能同时被 CPU 读出并执行，那么也需要将升级程序 Copy 到 RAM 中去执行。BOOT 程序由引导程序和升级程序组成，而升级程序又包括中断向量表、中断服务函数。由于升级程序不能在 FLASH 中被执行，因此，严格来说，存在 FLASH 中的升级程序可以认为只是引导程序要搬移的数据，只有当升级程序被搬移到 SRAM 中，并且被执行的时候，它才是真正意义上的升级程序。

BOOT 程序运行的过程指引：

1. CPU 复位后，开始运行 BOOT 程序；
2. 判断有无升级需求，如果没有升级需要，直接跳到用户功能程序；如果有升级需求，则开始执行引导程序；
3. 根据不同的在线升级方法更新用户程序；

升级程序开启中断之前要配置中断向量表的偏移地址，实现通过两组 32 位原始地址寄存器和两组 32 位映射地址寄存器，配置中断向量表的范围空间及映射后的中断向量表的范围空间；当 CPU 访问地址落在原始中断向量表的的地址空间内，则通过映射逻辑映射到映射地址空间内，进而能够正确进行中断响应。

3.3.2 复位管理

BF7006AMXX 由多种复位源来使得芯片系统进入复位状态，不同复位源进入不同的复位状态，主要分为两种：上电初始化复位和功能性全局复位。上电初始化复位发生后，将复位芯片所有单元以及寄存器；其它功能复位除系统关键配置和状态寄存器之外，复位所有功能单元。

主要特性：

1. 支持多种复位源；
2. 异步复位，同步释放，确保可靠性；
3. 每种复位源都有对应的的复位状态标志；
4. 外部复位支持复位信号滤波功能；

3.3.2.1 复位源

1. 上电复位：系统发生上电后模拟模块发出复位信号，复位所有单元；
2. 掉电复位：当使能掉电复位模块时，系统发生掉电，电压达到复位阈值后产生复位，复位所有单元；
3. 看门狗复位：看门狗定时器溢出后产生复位信号使系统复位；
4. 软件复位：通过写 CPU 内软件复位寄存器有效，使系统复位；
5. CPU 死机复位：CPU 程序出现紊乱，使内部 LOCKUP 信号有效，产生系统复位；
6. 外部管脚复位：通过拉低外部复位功能管脚使系统复位，该复位可滤波，防止外部干扰而造成的误复位；
7. 地址溢出复位：CPU 执行指令时寻址 FLASH 地址超出正常范围，发生地址溢出的情况使系统产生复位；
8. 晶振时钟起振失败复位：外部晶振起振失败（该功能通常用于芯片使用外部晶振作为系统时钟的情况）；
9. 晶振时钟失效复位：系统时钟故障监测单元监测到外部晶振失效时，发出系统复位（该功能通常用于芯片使用外部晶振作为系统时钟的情况）；

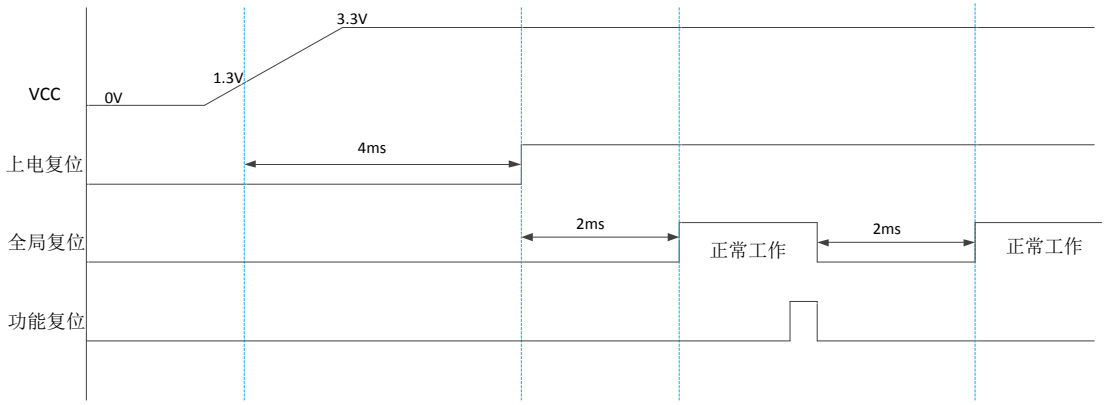
3.3.2.2 复位滤波

对于外部复位功能，为避免干扰毛刺信号使得芯片系统误进入复位状态，系统支持外部复位信号的滤波功能，其中滤波时钟为 RC32KHz。

当外部端口复位功能有效且需要外部复位滤波功能时，需确保 RC32K 时钟源处于工作状态，并且寄存器 SYS_EXFLT=1。滤波时间为 1 个有效的 RC_32KHz 时钟，滤波时间为 30us~60us。如果 RC32KHz 时钟源未打开且配置滤波功能使能时，外部复位功能将失效，复位源无法送入系统，应用需注意。

对于其它复位，芯片内部设计异步复位，2 级寄存器同步释放，确保复位信号可靠有效。

3.3.2.3 复位时序说明



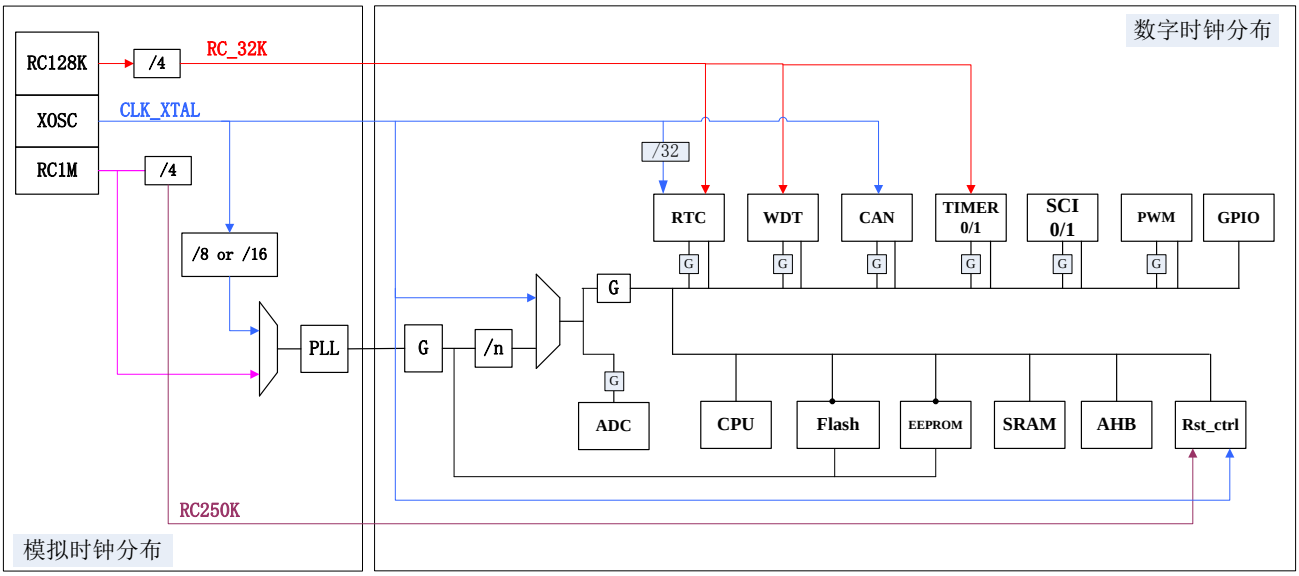
1. 芯片上电，VCC达到1.3V，上电复位信号有效，持续4ms；
2. 之后，系统开始进行全局复位，2ms之后，芯片全局复位完成，系统开始正常工作；
3. 工作期间发生其它功能（掉电，看门狗，地址溢出，软复位，CPU死机，晶振起振失败，晶振失效，外部管脚）复位时，产生有效的系统复位，发生全局复位。2ms之后，芯片全局复位完成，系统开始正常工作。

注意--VCC 上电从 0V 到 3.3V 时间，要求在 4ms 内完成。

3.3.3 时钟与功耗管理

BF7006AMXX 支持多种时钟源，同时系统工作时钟支持多种分频。各个外设模块支持时钟的单独控制，以便于更好的控制功耗和提高系统效率。同时，系统支持两种低功耗模式，尽可能地降低系统工作电流，并支持多种功能的唤醒机制。

3.3.3.1 时钟结构



时钟结构说明：

1. 系统包含 3 个模拟时钟源：RC128KHz、RC1MHz 和外部晶振，其中 RC128KHz 在模拟端经过 4 分频后形成 RC32KHz 时钟源送入数字系统使用；其中 RC1MHz 在模拟端经过 4 分频后形成 RC250KHz 时钟源送入数字系统使用；外部晶振典型支持 8MHz 和 16MHz；
2. 模拟 PLL 锁相环接入时钟源支持内部 RC1MHz 和外部晶振时钟，但接入时钟仅支持 1MHz，因此当 PLL 接入时钟源选择外部晶振时，必须根据不同的外部晶振频率进行 8 分频或者 16 分频至 1MHz；
3. 系统时钟为内部 PLL 输出时钟；PLL 输出时钟频率最大 32MHz，并可根据不同应用需求配置 SYS_CLK_SEL 进行分频；当选择外部晶振时钟作为系统时钟或者晶振时钟作为 PLL 接入时钟源时，在低功耗唤醒时，由于晶振起振时间影响，系统等待的时间将远大于选择内部 RC1MHz 作为 PLL 接入时钟源并且作为系统时钟的情况；

4. CAN 模块工作时钟支持外部晶振和内部系统时钟，通过程序配置，需要注意的是当选择为晶振作为工作时钟时，存在与系统运行时钟的跨时域问题，需要合理配置 SYS_CAN_DOMAIN 寄存器；当 CAN 模块时钟未供给至 CAN 模块时，CAN 模块的任何寄存器均无法被存取，存取动作将可能导致系统出现故障；
5. WDT 计数时钟为 RC32KHz 以及其 32 分频，软件可选；
6. RTC 计数时钟 RC32KHz 以及其 32 分频、外部晶振的 32 分频，软件可选；
7. 所有外设以及系统时钟可被门控，用于降低功耗；

3.3.3.2 时钟选择和切换

系统默认选择内部 RC1MHz 时钟作为 PLL 的接入时钟源，同时 PLL 输出的 2 分频（16MHz）作为系统时钟。外部晶振默认关闭。因此系统上电初始化前的工作时钟是基于内部 RC1MHz 产生的时钟。

如果系统在某种应用中需要高精准时钟源，可在初始化程序中配置打开晶振时钟并且将系统时钟切换为外部晶振时钟源，或者将 PLL 接入时钟源切换成外部晶振时钟。注意，由于晶振打开并且成功起振需要一定的稳定时间，因此在软件配置切换之前必须确保晶振时钟处于稳定的状态，否则系统将会出错。

同样的，如果在系统进入低功耗睡眠模式时，为降低功耗关闭外部晶振时钟，而此时系统工作时钟时基于外部晶振时钟的情况下，唤醒时间将包含晶振起振稳定时间，唤醒时间较长（具体见晶振稳定时间电气特性参数）。

3.3.3.3 系统工作模式

BF7006AMXX 有 4 种工作模式，可以根据不同的情况进行选择：

1. 正常模式（Active）：模块保持正常工作，各模块功能由软件配置控制；
2. 空闲模式（Sleeping）：CPU 和系统模块停止工作，其他外设模块功能由软件配置控制，支持中断唤醒；
3. 休眠模式（Sleepdeep）：此模式数字所有模块停止工作，部分模拟模块选择性工作（根据具体的唤醒条件配置），支持条件触发唤醒；
4. 调试模式（Debug）：用于测试调试，烧写程序。通过外部 ARM SW 调试接口直接进入；

不同工作模式对时钟源的控制：

工作模式	进入该模式的条件	时钟源状态	
正常模式（Active）	——	XTAL	取决于软件配置
		RC128K	取决于软件配置
		RC1MHz	工作
		PLL	工作
空闲模式（Sleeping）	配置 CPU 睡眠模式为普通睡眠 执行 WFI 指令	XTAL	取决于软件配置
		RC128K	取决于软件配置
		RC1MHz	工作
		PLL	工作
休眠模式（Sleepdeep）	配置 CPU 睡眠模式为深度睡眠 执行 WFI 指令	XTAL	取决于软件配置
		RC128K	取决于软件配置
		RC1MHz	关闭
		PLL	关闭
调试模式（Debug）	外部接口输入	XTAL	取决于软件配置
		RC128K	取决于软件配置
		RC1MHz	工作
		PLL	工作

不同工作模式对系统各模块的控制：

模块	正常工作	空闲模式	休眠模式
CPU	工作	PD	PD
AHB	工作	PD	PD
FLASH	工作	PD	PD
EEPROM	工作	PD	PD
SRAM	工作	PD	PD
SYS_CTRL	工作	PD	PD
TIMER0/1	软件控制	软件控制	PD
PWM	软件控制	软件控制	PD
RTC	软件控制	软件控制	软件控制
WDT	软件控制	软件控制	软件控制
CAN	软件控制	软件控制	PD
SCI0/1	软件控制	软件控制	PD
GPIO	软件控制	软件控制	PD
ADC	软件控制	软件控制	软件控制
LVDT	软件控制	软件控制	软件控制
BOR	软件控制	软件控制	软件控制

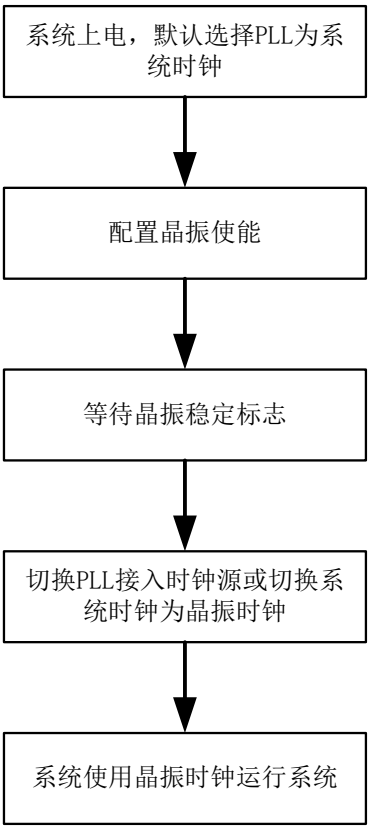
3.3.3.4 低功耗唤醒

1. 空闲模式唤醒：任何外设的中断、看门狗复位和外部复位均可唤醒系统。中断唤醒将使得系统从上次睡眠点继续运行，复位唤醒使得系统从空闲模式下复位并重新开始运行；
2. 睡眠模式唤醒：外部中断（包括 NMI）、RTC 中断、LVDT 低压中断、CAN 中断、SCI 中断、看门狗复位和外部复位可以唤醒系统，中断唤醒将使得系统从上次睡眠点继续运行，复位唤醒使得系统从空闲模式下复位并重新开始运行；
3. 睡眠模式下 CAN 唤醒：支持外部 CAN 通信显性电平唤醒，并可选择是否滤波（具体见通过 CAN 唤醒系统章节）
4. 睡眠模式下 SCI 唤醒：支持外部 SCI 通信管脚脉冲唤醒（具体见 SCI 中断唤醒章节）
5. 睡眠模式下 ADC 唤醒：具体见 ADC 睡眠唤醒章节

说明：由于睡眠模式系统关闭模拟大部分模块和全部数字模块，因此在唤醒时，首先系统需要确保电源和时钟是否稳定后才会正常唤醒 CPU 以及数字系统，因此唤醒需要的时间较长，包括电源稳定时间 60us、PLL 稳定时间 200us~500us（软件可配置）以及外部晶振稳定时间（若系统使用晶振作为系统工作时钟时，需要 4096 或 8192 个软件可选的晶振稳定周期和晶振起振稳定时间）。

3.3.3.5 晶振时钟初始化配置流程

BF7006AMXX 支持外部晶振时钟直接作为系统运行时钟，或者晶振时钟作为 PLL 接入时钟源，再使用 PLL 时钟作为系统时钟，已达到更高的应用精度要求。当外部晶振直接作为系统运行时钟时，最大频率不超 16MHz；当晶振时钟作为 PLL 接入时钟源，再使用 PLL 时钟作为系统时钟时，系统最大频率不超过 32MHz。系统初始化程序切换晶振时钟前，须保证晶振起振稳定。



3.3.4 功能安全管理

为更好的使得系统满足 ASILB 安全等级，BF7006AMXX 支持多种功能安全管理机制，从硬件层面实现关键逻辑功能的安全保护。

3.3.4.1 外部晶振失效监测

监测外部晶振失效有两种情况：

1. 晶振初始化起振失效监测：

   该功能使能后，晶振初始化启动时，内部 RC1MHz 的 4 分频时钟 RC250KHz 自动计时 100ms，100ms 之后监测晶振启动成功标志，如果监测到正确的晶振启动成功标志则认为晶振已成功启动，系统可继续运行；如果监测不到正确的晶振启动成功标志，则发出相应错误中断并根据寄存器 SYS_XTAL_INIT 配置是否复位系统（复位典型用于晶振作为系统时钟源时）；

2. 晶振工作时失效监测；

   晶振正常工作时，存在失效风险。当该功能使能时，系统使用内部 RC1MHz 的 4 分频时钟 RC250KHz 来监控晶振时钟，即使用晶振时钟（快速时钟）计数 RC250KHz 的 2 个周期（慢速时钟）。

考虑时钟受环境的偏差特性，监测功能设定合理计数值在一定区间内（寄存器 SYS_XTAL_CHKCNT 配置），当计数值在此区间内，则认为晶振正常；当计数值超出该计数区间，则认为晶振出现错误，并发出错误中断；当此错误连续出现 8 次及以上时，根据 SYS_XTAL_CHK 配置是否复位系统（复位典型用于晶振作为系统时钟源时）。

当晶振模块不使能时，监测功能将无法使用，应用需注意在关闭晶振或者进入睡眠模式时，该功能将无法正常使用。

3.3.4.2 低压及掉电监测

BF7006AMXX 支持系统供电监控，包含低压监测 LVDT 和掉电监测 BOR。系统提供 3 个档位的低压点，分别为 4.5V、4.0V 和 3.5V，以及 1 个掉电复位点 3.1V。当系统供电电压小于上述值时，为保证系统工作安全，系统会发出低压中断或者直接复位系统。

LVDT 提供两个中断标志位，分别为低压中断标志和恢复中断标志，可以更好地指示当前电压的状态。但是，仅有低压中断可将系统从睡眠模式中唤醒。

3.3.4.3 存储体保护和校验

BF7006AMXX 的 FLASH 和 EEPROM 支持擦写保护，防止误操作对程序尤其是 BOOT 程序的误擦除。保护空间大小由相应的寄存器 FLASH_LOCK_SIZE 和 EEPROM_LOCK_SIZE 配置控制，并且该寄存器的配置需要 EFLASH_UNLOCK 进保护解锁后才可进行。

当芯片连接调试功能时，保护功能自动解锁。

同时，FLASH 和 EEPROM 提供 32 位+6 位 ECC 的校验机制，可纠错 1 位或检错 2 位，从系统层面提高了存储体的可靠性。

FLASH 空间保护：

FLASH_LOCK_SIZE	地址保护范围	保护大小（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 空间保护：

EEPROM_LOCK_SIZE	地址保护范围	保护大小（bytes）
0x00	不保护	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 功能安全复位

BF7006AMXX 支持多种功能安全性复位，包括程序地址访问溢出复位、CPU 死锁保护复位、硬件看门狗复位。

1. 程序地址访问溢出复位：当程序指令访问超出限定的地址范围后，系统将认为此次操作非法，将发出地址溢出复位来防止程序跑飞；
2. CPU 死锁保护复位：基于 Cortex_M0 自带功能信号 LOCKUP 设计的复位；
3. 硬件看门狗复位：当程序规定时间内不喂狗，则认为系统处于死循环或者瘫痪状态，此时直接由硬件看门狗发出系统复位；系统采用独立的看门狗计数时钟，防止系统紊乱对看门狗运行造成影响。

3.3.4.5 ADC 温度检测通道

BF7006AMXX 包含 1 路内部通道，其支持检测芯片温度变化，不同温度该通道反馈的 ADC 转换数据不同。用户可根据此数据与温度的准线性关系来评估当前芯片温度，确保系统安全。具体配置见 ADC 章节。

3.3.5 寄存器列表

系统控制器模块地址区间：0x5001_0000~0x5001_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值

0x00	SYS_PTSEL	R/W	端口功能选择寄存器	0x00
0x04	SYS_XTAL_CTRL	R/W	晶振功能控制寄存器	0x00
0x08	SYS_PLL_SOURCE_SEL	R/W	PLL 时钟源控制选择寄存器	0xACB3
0x0C	SYS_CLK_SEL	R/W	数字时钟选择寄存器	0x3CA2
0x10	SYS_VECTOR_OFFSET	R/W	向量重定向偏移寄存器	0x0000_0000
0x14	SYS_CLK_PD	R/W	模拟工作使能寄存器	0x06
0x18	SYS_CAN_DOMAIN	R/W	CAN 跨时域计数时间配置寄存器	0x04
0x1C	SYS_CLK_OUT	R/W	模拟输出配置寄存器	0x0000
0x20	SYS_IO_LOCK	R/W	IO 功能锁定寄存器	0x01
0x24	SYS_LVDT_CRL	R/W	电压监测配置寄存器	0x07
0x28	SYS_EXRST	R/W	外部复位使能寄存器	0x01
0x2C	SYS_EXFLT	R/W	外部复位滤波寄存器	0x01
0x34	SYS_PERH_HALT	R/W	调试计数停止控制寄存器	0x01
0x38	SYS_PLL_T	R/W	PLL 稳定时间配置寄存器	0x00
0x40	SYS_CAN_CLKSEL	R/W	CAN 模块时钟选择寄存器	0x00
0x44	SYS_CAN_RST	R/W	CAN 独立复位寄存器	0x01
0x48	SYS_CAN_SPWKFLAG	R/W	系统唤醒 CAN 标志寄存器	0x00
0x100	SYS_RSTSTAT	R/W	复位状态寄存器	0x0001
0x104	SYS_INTEN	R/W	系统中断使能寄存器	0x0000
0x108	SYS_INTFLG	R/W	系统中断状态寄存器	0x0000
0x10C	SYS_XTAL_CHK	R/W	晶振工作失效监测寄存器	0x00
0x110	SYS_XTAL_CHKCNT	R/W	晶振工作失效监测区间配置寄存器	0x0002_0008
0x114	SYS_XTAL_INIT	R/W	晶振初始化失效监测寄存器	0x00
0x118	SYS_LVDT_IE	R/W	LVDT 中断使能寄存器	0x00
0x11C	SYS_LVDT_IF	R/W	LVDT 中断标志寄存器	0x00

3.3.6 寄存器详细描述

3.3.6.1 SYS_PTSEL 寄存器

地址	位	名称	R/W	复位值	描述
0x00	4:0	保留	——	——	保留

	1:0	SYS_PTSEL	R/W	0x0	SCI1 端口功能重定位选择 0x3：选择 PF0 和 PF1 为 SCI1 功能端口 0x2：选择 PE6 和 PE7 为 SCI1 功能端口 0x1：选择 PE2 和 PE3 为 SCI1 功能端口 0x0：选择 PF0 和 PF1 为 SCI1 功能端口

3.3.6.2 SYS_XTAL_CTRL 寄存器

地址	位	名称	R/W	复位值	描述
0x04	7	SYS_XTAL_CTRL_PD	RO	0x0	晶振工作状态标志 0x1 ：晶振处于正常工作状态 0x0 ：晶振处于关闭状态 由于时钟切换和晶振关闭需要跨时域多周期才能完成动作，所以应用中当系统配置 SYS_ANACLK_SEL 或 SYS_CLK_SEL 从外部晶振时钟源切换回内部时钟源时，再配置此位关闭外部晶振时钟后，不能立即执行进入睡眠的指令，需要检查此信号，当其为 0 时才能进行进入睡眠的操作，否则在睡眠模式下晶振无法正常关闭。
	6:4	SYS_XTAL_CTRL_CNT	R/W	0x0	晶振初始化稳定周期选择 0x0：4096 个晶振周期 其它：8192 个晶振周期
	3	SYS_XTAL_CTRL_INIT	RO	0x0	晶振初始化稳定标志 0x1：起振稳定 0x0：起振还未稳定 在进行时钟切换时，必须确保晶振起振稳定。
	2	保留	——	——	保留
	1	SYS_XTAL_CTRL_SLEEPPD	R/W	0x0	睡眠模式下配置晶振是否关闭 0x1：关闭 0x0：保持先前配置
	0	SYS_XTAL_CTRL_EN	R/W	0x0	晶振使能 0x1：使能 0x0：不使能

3.3.6.3 SYS_PLL_SOURCE_SEL 寄存器

地址	位	名称	R/W	复位值	描述
0x08	15:0	SYS_PLL_SOURCE_SEL	R/W	0xACB3	PLL 接入时钟源选择和配置 0xACB3：PLL 接入时钟源为 RC1MHz 0xCCD2：PLL 接入时钟源为外部晶振，接入 PLL 的晶振 16 分频

					0xBD5A：PLL 接入时钟源为外部晶振，接入 PLL 的晶振 12 分频 0x8D9C：PLL 接入时钟源为外部晶振，接入 PLL 的晶振 8 分频 PLL 接入时钟源须为 1MHz，当接入时钟源为外部晶振时应注意正确分频。

3.3.6.4 SYS_CLK_SEL 寄存器

地址	位	名称	R/W	复位值	描述
0x0C	15:0	SYS_CLK_SEL	R/W	0x3CA2	系统时钟源选择和配置 0x9ABD：系统时钟源选择 PLL，PLL 频率为 32MHz 0x3CA2：系统时钟源选择 PLL，PLL 频率为 16MHz 0xE78C：系统时钟源选择 PLL，PLL 频率为 8MHz 0x7C6B：系统时钟源选择外部晶振

3.3.6.5 SYS_VECTOR_OFFSET 寄存器

地址	位	名称	R/W	复位值	描述
0x10	31:0	SYS_VECTOR_OFFSET	R/W	0x0000_0000	向量重定向偏移地址

3.3.6.6 SYS_CLK_PD 寄存器

地址	位	名称	R/W	复位值	描述
0x14	7:3	保留	——	——	保留
	2	SYS_CLK_PDLVDT	R/W	0x1	配置是否关闭 LVDT 0x1：关闭 0x0：不关闭
	1	SYS_CLK_PDBOR	R/W	0x1	配置是否关闭 BOR 0x1：关闭 0x0：不关闭
	0	SYS_CLK_PD128K	R/W	0x0	配置是否关闭 RC128K 0x1：关闭 0x0：不关闭

3.3.6.7 SYS_CAN_DOMAIN 寄存器

地址	位	名称	R/W	复位值	描述
0x18	7:3	保留	——	——	保留
	2:0	SYS_CAN_DOMAIN	R/W	0x4	CAN 跨时域计数时间配置 0x4：适用于系统时钟 32MHz，外部晶振 8MHz 0x2：适用于系统时钟 32MHz，外部晶振 16MHz，或适用于系统时钟 16MHz，外部晶振 8MHz 0x1：适用于系统时钟 16MHz，外部晶振 16MHz，或适用于系统时钟 8MHz，外部晶振 8MHz，适用于系统时钟 8MHz，外部晶振 16MHz 当时钟配置完成后，根据具体配置选择上述寄存器值。

3.3.6.8 SYS_CLK_OUT 寄存器

地址	位	名称	R/W	复位值	描述
0x1C	15:6	SYS_CLK_OUTDUMMY	R/W	0x000	必须配置 0
	5	SYS_CLK_OUT_CLK2XTAL	R/W	0x0	CLKOUT2 端口输出晶振时钟 0x1：输出晶振时钟的 16 分频 0x0：不输出，端口 CLKOUT2 功能禁用
	4	SYS_CLK_OUT_CLK2RC250	R/W	0x0	CLKOUT2 端口输出 RC250KHz 时钟 0x1：输出 RC250KHz 时钟 0x0：不输出，端口 CLKOUT2 功能禁用
	3	SYS_CLK_OUTDUMMY2	R/W	0x0	必须配置 0
	2	SYS_CLK_OUT_CLKEN	R/W	0x0	CLKOUT 输出使能 0x1：使能 0x0：不使能，端口 CLKOUT 功能禁用
	1:0	SYS_CLK_OUT_CLKC	R/W	0x0	CLKOUT 输出时钟选择 0x3：外部晶振 8 分频 0x2：PLL 时钟 16 分频 0x1：输出 RC32KHz 0x0：输出 RC250KHz

3.3.6.9 SYS_IO_LOCK 寄存器

地址	位	名称	R/W	复位值	描述

0x20	7:1	保留	——	——	保留
	0	SYS_IO_LOCK	R/W	0x1	IO 口功能锁定 0x1：没有影响 0x0：锁定 IO 功能 锁定配置之前的输出数据及方向控制；当锁定前为输入方向，则外部端口的输入信号不产生影响。

3.3.6.10 SYS_LVDT_CRL 寄存器

地址	位	名称	R/W	复位值	描述
0x24	7:3	保留	——	——	保留
	2	SYS_LVDT_CRL_BORDELAY	R/W	0x1	BOR 掉电延迟选择 0x1：延迟 100us 0x0：不延迟
	1:0	SYS_LVDT_CRL_C	R/W	0x3	LVDT 档位选择 0x3：4.5V 档位 0x2：4.5V 档位 0x1：4.0V 档位 0x0：3.5V 档位

3.3.6.11 SYS_EXRST 寄存器

地址	位	名称	R/W	复位值	描述
0x28	7:1	保留	——	——	保留
	0	SYS_EXRST	R/W	0x1	外部复位端口使能 0x1：使能 0x0：不使能

3.3.6.12 SYS_EXFLT 寄存器

地址	位	名称	R/W	复位值	描述
0x2C	7:1	保留	——	——	保留
	0	SYS_EXFLT	R/W	0x1	外部复位端口滤波使能 0x1：使能，滤波 1~2 个 RC32K（128K 时钟 4 分频得到）时钟的时间 0x0：不使能 当芯片关闭 RC128KHz 时钟时，滤波功能无法实现，此时使能滤波后复位信号无法送入系统，应用需注意。

3.3.6.13 SYS_PERH_HALT 寄存器

地址	位	名称	R/W	复位值	描述
0x34	7:1	保留	——	——	保留
	0	SYS_PERH_HALT	R/W	0x1	调试时是否暂停看门狗 WDT 和 RTC 计数器 0x1：暂停 0x0：不暂停

3.3.6.14 SYS_PLL_T 寄存器

地址	位	名称	R/W	复位值	描述
0x38	7:2	保留	——	——	保留
	1:0	SYS_PLL_T	R/W	0x0	PLL 稳定时间选择 0x3：200us 0x2：300us 0x1：400us 0x0：500us 当系统退出空闲模式或者睡眠模式时，PLL 由于电源重新供给需要稳定时间，该寄存器提供应用等待 PLL 的稳定时间配置选择。

3.3.6.15 SYS_CAN_CLKSEL 寄存器

地址	位	名称	R/W	复位值	描述
0x40	7:1	保留	——	——	保留
	0	SYS_CAN_CLKSEL	R/W	0x0	CAN 模块时钟选择 0x1：系统时钟 0x0：外部晶振时钟（推荐使用）

3.3.6.16 SYS_CAN_RST 寄存器

地址	位	名称	R/W	复位值	描述
0x44	7:1	保留	——	——	保留
	0	SYS_CAN_RST	R/W	0x1	CAN 模块复位控制 0x1：不做操作 0x0：复位 CAN 模块 该位不会随着 CAN 模块复位后复位，需要手动再次配置为 1，否则 CAN 模块一直处于复位状态。

3.3.6.17 SYS_CAN_SPWKFLAG 寄存器

地址	位	名称	R/W	复位值	描述
0x48	7:1	保留	——	——	保留
	0	SYS_CAN_SPWKFLAG	R/W	0x0	标志系统睡眠模式下是否是通过 CAN 唤醒 0x1：CAN 导致系统从睡眠中唤醒 0x0：没有 CAN 系统唤醒发生 写该位清除标志。

3.3.6.18 SYS_RSTSTAT 寄存器

地址	位	名称	R/W	复位值	描述
0x100	16:9	保留	——	——	保留
	8	SYS_RSTSTAT_XTAL	R/W	0x0	晶振失效复位标志 0x1：发生复位 0x0：未发生复位 当该复位使能关闭时，当发生晶振失效时，此标志位也会置位。写 1 清除该标志。
	7	SYS_RSTSTAT_XTALINI	R/W	0x0	晶振初始化失败复位标志 0x1：发生复位 0x0：未发生复位 当该复位使能关闭时，当发生晶振初始化失败时，此标志位也会置位。写 1 清除该标志。
	6	SYS_RSTSTAT_ADDR	R/W	0x0	地址溢出复位标志 0x1：发生复位 0x0：未发生复位 写 1 清除该标志。
	5	SYS_RSTSTAT_EXT	R/W	0x0	外部复位标志 0x1：发生复位 0x0：未发生复位 写 1 清除该标志。
	4	SYS_RSTSTAT_LOCKUP	R/W	0x0	CPU 死锁复位标志 0x1：发生复位 0x0：未发生复位 写 1 清除该标志。
	3	SYS_RSTSTAT_SOFT	R/W	0x0	软件请求复位标志 0x1：发生复位 0x0：未发生复位 写 1 清除该标志。
	2	SYS_RSTSTAT_WDT	R/W	0X0	看门狗复位标志 0X1：发生复位 0X0：未发生复位 写 1 清除该标志。
	1	SYS_RSTSTAT_BOR	R/W	0X0	掉电复位标志 0X1：发生复位 0X0：未发生复位 写 1 清除该标志。
	0	SYS_RSTSTAT_POR	R/W	0x1	上电复位标志 0x1：发生复位 0x0：未发生复位

					写 1 清除该标志。

3.3.6.19 SYS_INTEN 寄存器

地址	位	名称	R/W	复位值	描述
0x104	16:9	保留	——	——	保留
	8	SYS_INTEN_ETER	R/W	0x0	EEPROM 出错 2 位及以上中断使能 0x1：使能 0x0：不使能
	7	SYS_INTEN_EOER	R/W	0x0	EEPROM 出错 1 位中断使能 0x1：使能 0x0：不使能
	6	SYS_INTEN_FTER	R/W	0x0	FLASH 出错 2 位及以上中断使能 0x1：使能 0x0：不使能
	5	SYS_INTEN_FOER	R/W	0x0	FLASH 出错 1 位中断使能 0x1：使能 0x0：不使能
	4	SYS_INTEN_EPOT	R/W	0x0	EEPROM 受保护地址非法擦写中断使能 0x1：使能 0x0：不使能
	3	SYS_INTEN_FPOT	R/W	0x0	FLASH 受保护地址非法擦写中断使能 0x1：使能 0x0：不使能
	2	SYS_INTEN_ADJ	R/W	0x0	上电配置字校验出错中断使能 0x1：使能 0x0：不使能
	1	SYS_INTEN_XTALCHK	R/W	0x0	外部晶振监测出错 1 次中断使能 0x1：使能 0x0：不使能
	0	SYS_INTEN_XTALINIT	R/W	0x0	外部晶振初始化失败中断使能 0x1：使能 0x0：不使能

3.3.6.20 SYS_INTFLG 寄存器

地址	位	名称	R/W	复位值	描述
0x108	16:9	保留	——	——	保留
	8	SYS_INTFLG_ETER	R/W	0x0	EEPROM 出错 2 位及以上中断标志 0x1：产生中断 0x0：未产生中断 写 1 清除该标志。
	7	SYS_INTFLG_EOER	R/W	0x0	EEPROM 出错 1 位中断标志 0x1：产生中断 0x0：未产生中断 写 1 清除该标志。
	6	SYS_INTFLG_FTER	R/W	0x0	FLASH 出错 2 位及以上中断标志 0x1：产生中断 0x0：未产生中断 写 1 清除该标志。
	5	SYS_INTFLG_FOER	R/W	0x0	FLASH 出错 1 位中断标志 0x1：产生中断 0x0：未产生中断 写 1 清除该标志。
	4	SYS_INTFLG_EPOT	R/W	0x0	EEPROM 受保护地址非法擦写中断标志 0x1：产生中断 0x0：未产生中断 写 1 清除该标志。
	3	SYS_INTFLG_FPOT	R/W	0x0	FLASH 受保护地址非法擦写中断标志 0x1：产生中断 0x0：未产生中断 写 1 清除该标志。
	2	SYS_INTFLG_ADJ	R/W	0x0	上电配置字校验出错中断标志 0x1：产生中断 0x0：未产生中断 写 1 清除该标志。
	1	SYS_INTFLG_XTALCHK	R/W	0x0	外部晶振监测出错 1 次中断标志 0x1：产生中断 0x0：未产生中断 此种情况晶振失效只能判断晶振失效复位标志。 写 1 清除该标志。
	0	SYS_INTFLG_XTALINIT	R/W	0x0	外部晶振初始化失败中断标志 0x1：产生中断 0x0：未产生中断 当发生晶振失效时系统时钟选晶振时，此标志位不会置 1；此种情

					况晶振失效只能判断晶振失效复位标志。写 1 清除该标志。

3.3.6.21 SYS_XTAL_CHK 寄存器

地址	位	名称	R/W	复位值	描述
0x10C	7:2	保留	——	——	保留
	1	SYS_XTAL_CHK_RSTEN	R/W	0x0	晶振失效后复位使能 0x1：使能 0x0：不使能
	0	SYS_XTAL_CHK_EN	R/W	0x0	晶振失效监测模块使能 0x1：使能 0x0：不使能

3.3.6.22 SYS_XTAL_CHKCNT 寄存器

地址	位	名称	R/W	复位值	描述
0x110	31:18	保留	——	——	保留
	17:9	SYS_XTAL_CHKCNT_H	R/W	0x020	晶振失效监测左区间周期值（外部晶振周期）
	8:0	SYS_XTAL_CHKCNT_L	R/W	0x008	晶振失效监测右区间周期值（外部晶振周期）

3.3.6.23 SYS_XTAL_INIT 寄存器

地址	位	名称	R/W	复位值	描述
0x114	7:2	保留	——	——	保留
	1	SYS_XTAL_INIT_RSTEN	R/W	0x0	晶振初始化失败后复位使能 0x1：使能 0x0：不使能
	0	SYS_XTAL_INIT_EN	R/W	0x0	晶振初始化失败监测模块使能 0x1：使能 0x0：不使能

3.3.6.24 SYS_LVDT_IE 寄存器

地址	位	名称	R/W	复位值	描述
0x118	7:1	保留	——	——	保留
	0	SYS_LVDT_IE	R/W	0x0	LVDT 使能 0x1：使能 0x0：不使能

3.3.6.25 SYS_LVDT_IF 寄存器

地址	位	名称	R/W	复位值	描述
0x11C	7:2	保留	——	——	保留
	1	SYS_LVDT_IE_H	R/W	0x0	LVDT 升压恢复中断标志 0x1：欠压事件恢复 0x0：未发生 写 1 该位清除标志位。
	0	SYS_LVDT_IE_L	R/W	0x0	LVDT 降压中断标志 0x1：发生欠压事件 0x0：未发生 写 1 该位清除标志位。 LVDT 低压中断可唤醒系统睡眠模式，唤醒后进入 LVDT 中断程序。仅此过程不会产生欠压标志。

3.3.7 程序范例

系统时钟选择：

```
void SystemInit (void)
{
    uint32_t clk_sel;
    if(__SYSTEM_CLOCK == PLL_32M)
        clk_sel = SYS_CLK_PLL_32M;  //选择 32M 时钟
    else if(__SYSTEM_CLOCK == PLL_16M)
        clk_sel = SYS_CLK_PLL_16M;  //选择 16M 时钟
    else if(__SYSTEM_CLOCK == PLL_8M)
        clk_sel = SYS_CLK_PLL_8M;   //选择 8M 时钟
    else if(__SYSTEM_CLOCK == XTAL_16M || __SYSTEM_CLOCK == XTAL_8M)
        clk_sel = SYS_CLK_XTAL;     //选择外部晶振时钟
    system_clk_sel(clk_sel);
}
```

系统复位：

```
void Reset_Flag()
{
    if(SYS_RSTSTAT & SYS_RSTSTAT_XTAL)       //晶振失效复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_XTAL;         //清除晶振失效复位标志
    }
    if(SYS_RSTSTAT & SYS_RSTSTAT_XTALINI)        //晶振初始化失败复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_XTALINI; //清除晶振初始化失败复位标志
    }
    if(SYS_RSTSTAT & SYS_RSTSTAT_ADDR)       //地址溢出复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_ADDR;         //清除地址溢出复位标志
    }
    if(SYS_RSTSTAT & SYS_RSTSTAT_EXT)            //外部复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_EXT;      //清除外部复位标志
    }
    if(SYS_RSTSTAT & SYS_RSTSTAT_LOCKUP)         //CPU 死锁复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_LOCKUP;  //清除 CPU 死锁复位标志
    }
    if(SYS_RSTSTAT & SYS_RSTSTAT_SOFT)       //软件复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_SOFT;         //清除软件复位标志
    }
    if(SYS_RSTSTAT & SYS_RSTSTAT_WDT)            //看门狗复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_WDT;          //清除看门狗复位标志
    }
    if(SYS_RSTSTAT & SYS_RSTSTAT_POR)                //上电复位
    {
        SYS_RSTSTAT |= SYS_RSTSTAT_POR;          //清除上电复位标志
    }
}
```

系统中断：

```
void System_Intstat()
{
    if(SYS_IF & SYS_INTFLG_ETER)         //EEPROM 出错 2 位及以上中断
    {
        SYS_IF |= SYS_INTFLG_ETER;       //清除 EEPROM 出错 2 位及以上中断标志
    }
```

```
    if(SYS_IF & SYS_INTFLG_EOER)          // EEPROM 出错 1 位中断
    {
        SYS_IF |= SYS_INTFLG_EOER;        //清除 EEPROM 出错 1 位中断标志
    }
    if(SYS_IF & SYS_INTFLG_FTER)          //FLASH 出错 2 位及以上中断
    {
        SYS_IF |= SYS_INTFLG_FTER;        //清除 FLASH 出错 2 位及以上中断标志
    }
    if(SYS_IF & SYS_INTFLG_FOER)          // FLASH 出错 1 位中断
    {
        SYS_IF |= SYS_INTFLG_FOER;        //清除 FLASH 出错 1 位中断标志
    }
    if(SYS_IF & SYS_INTFLG_EPOT)          //EEPROM 受保护地址非法擦写中断
    {
        SYS_IF |= SYS_INTFLG_EPOT;        //清除 EEPROM 受保护地址非法擦写中断标志
    }
    if(SYS_IF & SYS_INTFLG_FPOT)          // FLASH 受保护地址非法擦写中断
    {
        SYS_IF |= SYS_INTFLG_FPOT;        //清除 FLASH 受保护地址非法擦写中断标志
    }
    if(SYS_IF & SYS_INTFLG_ADJ)      //上电配置字检验出错中断
    {
        SYS_IF |= SYS_INTFLG_ADJ;         //清除上电配置字检验出错中断标志
    }
    if(SYS_IF & SYS_INTFLG_XTALCHK) //外部晶振检测出错 1 次中断
    {
        SYS_IF |= SYS_INTFLG_XTALCHK;//清除外部晶振检测出错 1 次中断标志
    }
    if(SYS_IF & SYS_INTFLG_XTALINIT)     //外部晶振初始化失败中断
    {
        SYS_IF |= SYS_INTFLG_XTALINIT;//清除外部晶振初始化失败中断标志
    }
}
```

4 存储体

BF7006AMXX 内部包含高可靠性的 96KB FLASH（不包含 ECC）、2KB EEPROM（不包含 ECC）和 4KB SRAM，其中 FLASH 和 EEPROM 均支持 6 位 ECC 校验机制和存储体保护功能，且由硬件自动进行计算和校验，无需软件参与，极大程度提高程序代码的安全性和可靠性。

BF7006AMXX 系统设计 FLASH 和 EEPROM 为独立接口，程序可同时操作 FLASH 和 EEPROM（同时擦写 FLASH 和 EEPROM 除外），无需等待。

4.1 FLASH

FLASH 为系统中程序存储的主要空间，由 FLASH_NVR 特殊存储区域和主程序存储区域组成。FLASH_NVR 特殊存储区域存储芯片重要关键修调数据，一旦被误改写将使芯片无法工作，因此该存储空间始终处于硬件保护状态。主程序空间由用户操作，用于存储软件程序和应用数据。

4.1.1 主要特性

主要特性如下：

1. 存储空间 4Mbytes 地址空间；
2. 每个 WORD 地址空间支持 ECC 校验；
3. 单个页空间大小 2K 地址空间；
4. 温度 85℃数据保持力至少 10 年；
5. 由于存储体带 ECC 计算和校验的缘故，写 FLASH 仅支持 WORD 地址跳变读写操作。

4.1.2 FLASH_NVR 信息

FLASH_NVR 地址空间为：0x0001_8000~0x0001_8FFF，其中存储了芯片关键修调数据，更改将可能导致芯片无法正常工作。

4.2 EEPROM

EEPROM 为系统中数据存储空间，由 EEPROM_NVR 特殊存储区域和主数据存储区域组成。EEPROM_NVR 特殊存储区域提供给客户使用，用户可保存芯片初始化读取的程序处理关键数据，同时主数据空间可由用户存储即时更新的应用数据，拓展应用。

4.2.1 主要特性

主要特性如下：

1. 存储空间 512 x 38bits（包含 ECC）；
2. 单个页空间大小 16 x 38bits（64Bytes 地址空间大小，包含 ECC）；
3. 读数据时间最大 35ns；
4. 单个行写时间 16~24us；
5. 单页擦除时间 16~24ms；
6. 整块擦除时间 16~24ms；
7. 温度 85℃数据保持力至少 10 年；

由于存储体带 ECC 计算和校验的缘故，写 FLASH 仅支持 word 地址跳变读写操作。

4.2.2 EEPROM_NVR 信息

EEPROM_NVR 地址空间为：0x4000_0800~0x4000_08FF，用户可用，存储具体数据由用户决定。

4.3 SRAM

主要特性如下：

1. 整体 1K x 32 Bits SRAM，4K 地址空间；
2. 支持 Byte、Halfword、Word 快速存取；

4.4 寄存器列表

FLASH/EEPROM 控制器模块地址区间：0x5000_0000~0x5000_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	EFLASH_SEL	R/W	擦写控制使能寄存器	0x0000
0x04	EFLASH_MODE	R/W	擦写控制方式寄存器	0x00
0x08	EFLASH_EBCFG	R/W	擦写控制功能寄存器	0x00
0x0C	FLASH_STATE	RO	FLASH 工作状态寄存器	0x01
0x10	EEPROM_STATE	RO	EEPROM 工作状态寄存器	0x01
0x200	EFLASH_ECC_CTRL	R/W	ECC 控制功能寄存器	0x01
0x204	EFLASH_UNLOCK	R/W	解保护密钥寄存器	0x0000
0x208	FLASH_LOC_KSIZE	R/W	FLASH 地址保护范围寄存器	0x00
0x20C	EEEPROM_LOCK_SIZE	R/W	EEPROM 地址保护范围寄存器	0x00

4.5 寄存器详细描述

4.5.1 EFLASH_SEL 寄存器

地址	位	名称	R/W	复位值	描述
0x00	15:0	EFLASH_SEL	R/W	0x0000	选择擦写 FLASH 还是 EEPROM 0xAA55： FLASH 0xCD78： EEPROM 此寄存器配置其它值无效。在完成擦写操作状态后自动复位为 0。

4.5.2 EFLASH_MODE 寄存器

地址	位	名称	R/W	复位值	描述
0x04	7:0	EFLASH_MODE	R/W	0x00	FLASH/EEPROM 擦写控制工作方式 0xA5：执行 FLASH 擦写，CPU 等待擦写完成后才可以执行程序 0x5A：执行 FLASH 擦写，此时 CPU 可以执行程序（程序在 SRAM 中的情况时可用） 0x3C：执行 EEPROM 擦写，此时 CPU 执行程序 此寄存器只有在 EFLASH_SEL 选定后才可以配置，其他情况无法写该寄存器，并维持复位值。此寄存器在完成擦写操作状态后硬件复位，回到初始值。当程序在 FLASH 中运行是是无法进行 FLASH 全部擦除的动作，因此此寄存器配置 00A5 时，不能进行整块擦除命令。

4.5.3 EFLASH_EBCFG 寄存器

地址	位	名称	R/W	复位值	描述
0x08	7:0	EFLASH_EBCFG	R/W	0x00	FlASH/EEPROM 擦写控制功能寄存器 0x55：页擦除操作 0xAA：块擦除操作 0x33：字写操作 0xCC：连续写操作，写个数 FLASH 不超过 32 个字，EEPROM 不超过 8 个字 此寄存器只有在 EFLASH_SEL 选定后才可以配置，其他情况无法写该寄存器，并维持复位值。此寄存器在完成擦写操作状态后硬件复位，回到初始值。在 EFLASH_MODE =0x00A5 模式下，不能配置 0xAA。 对于整片擦除，地址指向 NVR 时，NVR 空间和主存储空间全部擦除（EEPROM 提供用户可用的 NVR 空间，通过此配置可清除），地址指向主存储空间时，只全擦主存储空间（FLASH NVR 任何配置均无法擦除）。

4.5.4 FLASH_STATE 寄存器

地址	位	名称	R/W	复位值	描述
0x0C	7:0	FLASH_STATE	RO	0x00	FlASH 擦写完成标志 0x1：擦写完成 其它值无参考意义。

4.5.5 EEPROM_STATE 寄存器

地址	位	名称	R/W	复位值	描述
0x10	7:0	EEPROM_STATE	RO	0x00	EEPROM 擦写完成标志 0x1：擦写完成 其它值无参考意义。

4.5.6 EFLASH_ECC_CTRL 寄存器

地址	位	名称	R/W	复位值	描述
0x200	7:1	保留	——	——	保留
	0	EFLASH_ECC_CTRL	R/W	0x1	ECC 使能 0x1：使能 0x0：不使能

4.5.7 EFLASH_UNLOCK 寄存器

地址	位	名称	R/W	复位值	描述
0x204	31:0	EFLASH_UNLOCK	R/W	0x00	解除存储体的写入保护密钥 当配置为 0xAB23DC54 解除写保护寄存器控制，才可以配置 FLASH_LOCK_SIZE 和 EEEPROM_LOCK_SIZE 寄存器，其他值无法配置，配置完成后此寄存器清零。

4.5.8 FLASH_LOCK_SIZE 寄存器

地址	位	名称	R/W	复位值	描述
0x208	7:0	FLASH_LOCK_SIZE	R/W	0x00	FLASH 地址保护范围 0x00：保护 FLASH_NVR 区域 0x01：保护 FLASH_NVR +2K 主程序地址空间（从小端地址开始） 0x02：保护 FLASH_NVR +4K 主程序地址空间（从小端地址开始） 0x30：保护 FLASH_NVR +96K 主程序地址空间（从小端地址开始）

4.5.9 EEPROM_LOCK_SIZE 寄存器

地址	位	名称	R/W	复位值	描述
0x20C	7:0	EEPROM_LOCK_SIZE	R/W	0x00	EEPROM 地址保护范围 0x00：无任何保护 0x01：保护 EEPROM_NVR +64Bytes 数据地址空间（从小端地址开始） 0x02：保护 EEPROM_NVR +128Bytes 数据地址空间（从小端地址开始） …… 0x20：保护 EEPROM_NVR +2048Bytes 数据地址空间（从小端地址开始）

5 外设

BF7006AMXX 包含丰富的外设资源，其中支持多种通信功能，包括 CAN 通信、UART 通信和 LIN 通信；同时具有多个计数器/计时器，包括 RTC、WDT、Timer 和 Advance Timer；支持 54 个通用 GPIO 以及高精度高速度 ADC。本章详细介绍 BF7006AMXX 各外设单元的特性、功能以及详细寄存器说明；供应用人员开发使用。

5.1 SCI

5.1.1 主要特性

BF7006AMXX 最多支持 2 个 SCI 模块，其中一个具有独立功能端口，另外一个复用外部端口，同一时间内只允许有选择一组外部端口作为 SCI 使用。SCI 模块支持 UART 和 LIN2.0 协议通信。

1. 支持全双工，半双工串口通信；
2. 具有独立使能的双缓冲发射器和接收器；
3. 可编程波特率（13 位模数分频器）；
4. 中断驱动型或轮询操作：
   - 发送数据寄存器空，发送完成
   - 接收数据寄存器已满
   - 接收溢出、奇偶效验错误、帧错误和噪音错误
   - 闲置接收器检测

— 接收管脚上的活动边沿检测

5. 支持LIN的同步间隔段检测；
6. 支持硬件奇偶效验生成和检查；
7. 可编程8位或9位字符长度；
8. 可编程1位或2位STOP位长度；
9. 按闲置线路或地址标记的接收器唤醒；
10. 可选的13位中止字符生成/ 11位中止字符检测；
11. 可选的发射器输出极性和接收器输入极性；
12. 支持波特率自适应功能；

5.1.2 功能说明

本节详细介绍 BF7006AMXX SCI 主要功能。

5.1.2.1 波特率生成

波特率生成模数 bandrate={SCI_BDH[4:0],SCI_BDL}。

波特率计算公式：bandrate=0 时，不生成波特率时钟，当 bandrate=1~8191 时，SCI 波特率= BUSCLK/（16xbandrate）。BUSCLK 是系统时钟，也为 SCI 工作时钟，。

每次配置波特率寄存器均会清零内部 counter 计数器重新生成波特率信号。

通信要求发射器和接收器使用相同的波特率。

寄存器自身产生的计算误差：20KHz/2MHz=1%（波特率 20KHz，系统时钟 32MHz），波特率越小则误差越小，系统时钟频率越小误差越大，且系统时钟误差越大则计算误差越大。

通信允许的波特率偏差范围：8/11*16=4.5%。

本系统支持自动波特率匹配，在 LIN 协议中，同步段字符为 0x55。波特率检测时，从接受的 START 位的下降边沿开始测量，直到第 8 位数据位下降沿停止，一共 8 位，会在通信完毕后自动更新 bandrate，且可通过寄存器 SCI_BDH/SCI_BDL 读出。这里注意，接收同步段并自动匹配波特率，与接收功能同时进行，接收完毕该字符后会出接收满中断。波特率匹配之前的最大偏差不允许超过 40%，否则校准失效。

5.1.2.2 发射器功能

发射器输出管脚 Tx 闲置状态默认为逻辑高（复位后 SCI_C3[4]=0）。如果 SCI_C3[4]=1，发射器输出就被颠倒。

发射器可发送三种字符：前导闲置字符，中止字符，数据字符。三种字符排队发送，SCI_C2[4]位写 0 后再写 1 会排队前导闲置字符，SCI_C2[0]位写 1 后再写 0 会排队一个中止字符，写数据寄存器 SCI_D 会排队一个数据字符。

通过 SCI_C2[3]置位，发射器被使能。这会排队前导闲置字符，前导闲置字符是闲置状态的一个完整字符帧，根据 SCI_C1[4]和 SCI_C1[6]控制发送 12 位或 11 位或 10 位闲置字符（逻辑高）。在正常应用过程中需要发送闲置字符，程序会等待 SCI_S1[7]有效再进行设置，显示信息的最后字符已经移动到发送移位器，然后依次把 0 和 1 写入 SCI_C2[3]位。之后一旦移位器可用，该操作就立即将发送的闲置字符进行排队。注意：在 SCI_C2[3] =0 时只要移位器中的字符（包括三种字符）没有完成，SCI 发射器就不会停止发送。

通过把数据写入 SCI_D，程序把数据保存到发送数据缓冲器，会排队一个数据字符。SCI 发射器的中心元件是长度为 10 或 11 或 12 位（SCI_C1[4]和 SCI_C1[6]控制位中的设置）的发送移位寄存器。假设 SCI_C1[4]=0，选择正常的 8 位数据模式。在 8 位数据模式中，移位寄存器中有 1 个起始位、8 个数据位和 1 /2 个停止位。当发送移位寄存器可以用于新 SCI 字符时，在发送数据寄存器中等待的值被传输到移位寄存器，同时设置发送数据寄存器空 SCI_S1[7]标记，显示另外一个字符可以写入 SCI_D 的发送数据缓冲器。

通过将 SCI_C2[0]位写 1 后再写 0 会排队一个中止字符。中止字符是逻辑 0（10 位时间，包括启动和停止位）的全字符时间。13 位时间的较长中止符可以通过设置 SCI_S2[2]=1 进行使能。同样，SCI_C1[4]和 SCI_C1[6]都分别可以选择增加一位时间。一般来说，程序要等待 SCI_S1[7]有效再进行设置，以显示信息的最后一个字符已经移动到发送移位器，然后依次把 1 和 0 写入 SCI_C2[0]位。之后一旦移位器可用，该操作就立即对将发送的中止字符进行排队。如果当已进入队列的中止符进入移位器时 SCI_C2[0]仍然为 1，额外的中止字符会进入队列。

如果停止位移出 Tx 管脚后发送数据缓冲器中没有新字符（包括三种字符）在等待，发射器设置发送完成标记，进入闲置模式，Tx 处于高态，等待发送更多字符。

发送数据空中断产生条件包括：配置发射器使能 0 变 1 开启一次空中断，发送 FIFO 向移位寄存器传输时开启一次空中断。在传输过程中关闭发射器使能则会在将当前字符发送完毕后停止发送，会将之前的排队字符清除。发送完成中断产生条件：排队的字符均发送完毕后开启一次完成中断。

接收器功能

通过设置 SCI_S2[4]=1，接收器输入被反转。通过设置 SCI_C2[2]位，接收器被使能。

接收字符包括三种：数据字符，中止字符和闲置字符。

数据字符由逻辑 0 的起始位、8 个（或 9 个）数据位（LSB 先发）和逻辑 1 的停止位组成。在把停止位接收到接收移位器后，如果接收数据寄存器还未满，数据字符就被传输到接收数据寄存器，设置接收数据寄存器已满状态标记。如果此时已经设置了接收数据寄存器已满的 SCI_S1[5]，就设置溢出 SCI_S1[3]状态标记，新数据丢失。因为 SCI 接收器是双缓冲的，程序在设置 SCI_S1[5]后、读取接收数据缓冲器的数据前，有一个全字符时间，以避免接收器溢出。

当程序检测到接收数据寄存器已满时，它通过读 SCI_D 从接收数据寄存器中获取数据。

中止字符从 start 的 0 字符开始计数，直到 stop 位检测到 0 字符，SCI_S2[1]位选择是否使能 11 位中止字符检测，过程中一旦检测到管脚上的上升沿则清除计数。检测到足够的 0 字符（11/12/13 位），设置中止字符检测标记 SCI_S2[7]。

闲置字符根据 SCI_C1[2]位选择从停止/起始位后闲置字符位计数开始，是以接收器活动一段时间（SCI_S1[5]有效置位一次)后才开始检测的。一旦检测到 0 字符即清除计数，检测到足够的 1 字符(10/11/12 位)，设置闲置字符检测标记 SCI_S1[4]。

注：使能中止字符检测后只检测中止字符，不考虑数据接收，用于LIN协议流程控制；关闭中止字符检测使能后只接收数据，忽略中止字符检测。

5.1.2.3 接收器采样方法

SCI 接收器使用 16 倍波特率时钟进行采样。接收器通过以 16 倍波特率提取逻辑电平样本，以搜索 Rx 串行数据输入管脚上的下降边沿。下降边沿的定义是 3 个连续逻辑 1 采样后的逻辑 0 样本。16 倍波特率时钟用来把位时间划分为 16 个段，分别标记为 RT1 到 RT16。当定位了下降边沿时，还要从 RT3、RT5 和 RT7 中提取三个样本，以确保这是真正的起始位，而不仅仅噪音，如果这三个样本至少有两个样本为 0，接收器假设它与接收器字符同步，开始移位接收下面数据，如果不满足以上条件则退出状态机回到等待下降边沿状态。

接收器然后在 RT8, RT9 和 RT10 的每个位时间上进行采样，包括起始位和停止位，以决定该位的逻辑电平。逻辑电平是位时间期间提取的绝大多数样本的逻辑电平。在起始位中，如果 RT3、RT5 和 RT7 上的样本中至少有 2 个样本为 0，那么就假设该位为 0，即便在 RT8、RT9 和 RT10 上提取的一个或所有样本均为 1。

如果字符帧的任意位时间中的任意样本（起始位的 8 个样本 RT3~RT10，其它位的 3 个样本 RT8~RT10）不能与该位的逻辑电平保持一致，当收到的字符传输到接收数据缓冲器时，都设置噪音错误标记 SCI_S1[2]。

下降边沿检测逻辑不断寻找下降边沿，如果检测到边沿，样本时钟重新同步位时间。这样当出现噪音或不匹配波特率时，就可以提高接收器的可靠性。

5.1.2.4 接收器睡眠唤醒

接收器休眠唤醒是一种硬件机制，此机制的作用是利用硬件检测消除了处理不重要信息字符的软件开销。允许 SCI 接收器忽略用于不同 SCI 接收器的信息中的字符。

在这种应用系统中，所有接收器都估计每条信息的第一个字符，一旦确定该信息旨在用于不同接收器，它们就立即将逻辑 1 写入 SCI_C2[1]控制位。当设置了该位时，禁止设置与接收器有关的状态标记（当设置了 SCI_S2[3]位时，闲置位 IDLE 会被设置并产生中断）。

在接收器休眠状态下（软件设置 SCI_C2[1]位进入休眠），可以通过 SCI_C1[3]位选择唤醒方式（即硬件自动清除 SCI_C2[1]位），包括闲置字符唤醒和地址标记唤醒。

闲置字符检测见上面描述，一旦接收器检测到一个完整闲置字符，SCI_C2[1]被自动清除。唤醒后接收器会在下一个字符接收时设置相应状态标记。

地址标记唤醒是当接收器检测到已接收字符的最高位（在SCI_C1[4]=0模式中是第8位；在SCI_C1[4]=1模式中是第9位R8）中的逻辑1 时，SCI_C2[1]被自动清除。唤醒后接收器相关状态标记及中断在当前字符即可以被设置。

5.1.2.5 管脚连接模式

当 SCI_C1[7]= 1 时，SCI_C1[5]选择循环模式或单线模式。

循环模式：

循环模式独立于外部系统连接，有时用于检查软件，以帮助隔离系统问题。在该模式中，发射器输出内部连接到接收器输入，且 SCI 不使用 Rx 管脚。

单线模式：

在单线模式中，SCI_C3[5]位控制着Tx管脚上的串行数据方向。当SCI_C3[5]=0时，Tx管脚是SCI接收器的输入，连接接收器输入，当SCI_C3[5]=1时，Tx管脚是一个由发射器驱动的输出。

5.1.2.6 中断唤醒

本模块在系统低功耗模式下，可以配置打开接收边沿中断功能来唤醒系统。

在配置模块使能 SCI_EN=1，配置 Rx 管脚活动边沿中断使能 SCI_BDH[6]=1 的情况下，模块会异步检测 Rx 管脚边沿翻转， 当 Rx 管脚为低电平时（SCI_S2[4]=0）产生异步中断信号。

5.1.3 寄存器列表

BF7006AMXX 共有 2 个 SCI 模块，按照下面地址区间区分模块寄存器：

SCI0 模块地址区间：0x5004_0000~0x5004_3FFF

SCI1 模块地址区间：0x5004_4000~0x5004_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	SCI_BDH	R/W	SCI 波特率控制寄存器高	0x00
0x04	SCI_BDL	R/W	SCI 波特率控制低寄存器	0x00
0x08	SCI_C1	R/W	SCI 控制寄存器 1	0x00
0x0C	SCI_C2	R/W	SCI 控制寄存器 2	0x00
0x10	SCI_S1	RO	SCI 中断状态标记寄存器	0x00
0x14	SCI_S2	RO/RW	SCI 同步间隔段控制寄存器	0x00
0x18	SCI_C3	RO/RW	SCI 控制寄存器 3	0x00
0x1C	SCI_D	R/W	SCI 数据寄存器	0x00
0x20	SCI_EN	R/W	SCI 模块使能寄存器	0x00

5.1.4 寄存器详细描述

5.1.4.1 SCI_BDH 寄存器

地址	位	名称	R/W	复位值	描述
0x00	7	SCI_BDH_BREAK_CHECK_EN	R/W	0x0	间隔段检测中断使能 0x1：中断使能 0x0：中断禁止
	6	SCI_BDH_RX_EDGE_INT_EN	R	0x0	Rx 管脚活动边沿中断使能 0x1：中断使能 0x0：中断禁止
	5	SCI_BDH_RATE_AUTOMATCH_EN		0x0	波特率自动匹配使能 0x1：波特率自适应 0x0：固定波特率 同步段波特率自动匹配仅适用于 8 位数据 0x55，即下一笔通信必须是 0x55，会在通信完毕后自动更新波特率，且可通过寄存器 SCI_BDH/SCI_BDL 读出，使能后仅能同步一次，如需要再次同步，需要将使能位拉低一次再拉高。
	4:0	SCI_BDH_BPR_H		0x0	波特率模数除数寄存器高 5 位 波特率 bandrate={SCI_BDH[4:0],SCI_BDL}，bandrate=0 时不生成波特率时钟，当 bandrate=1~8191 时，SCI 波特率=系统总线时钟 BUSCLK/（16xbandrate）。

5.1.4.2 SCI_BDL 寄存器

地址	位	名称	R/W	复位值	描述
0x04	7:0	SCI_BDL_BPR_L	R/W	0x0	波特率模数除数寄存器低 8 位

5.1.4.3 SCI_C1 寄存器

地址	位	名称	R/W	复位值	描述
0x08	7	SCI_C1_CYCLE_MODE	R/W	0x0	循环模式使能 0x1：循环模式或单线模式，Tx 连接 Rx 0x0：正常双线模式
	6	SCI_C1_STOP_WIDTH	R/W	0x0	停止位数选择 0x1：2 位 0x0：1 位
	5	SCI_C1_SINGLE_TXD	R/W	0x0	单线模式使能 0x1：在 SCI_C1_CYCLE_MODE =1 时选择单线模式，Tx 管脚有效 0x0：内部循环模式，Tx 管脚无效
	4	SCI_C1_DATA_WIDTH	R/W	0x0	传输数据模式选择 0x1：9 位模式（第 9 位为奇偶校验位） 0x0：8 位模式
	3	SCI_C1_WAKE_SEL	R/W	0x0	接收器唤醒方式选择 0x1：地址标记唤醒 0x0：闲置线路唤醒
	2	SCI_C1_IDLE_SEL	R/W	0x0	闲置线路类型选择 0x1：停止位后闲置字符位计数开始 0x0：开始位后闲置字符位计数开始，计数 10 位时间（如果 SCI_C1_STOP_WIDTH =1 或 SCI_C1_DATA_WIDTH =1，则分别增加 1 位时间）
	1	SCI_C1_PARITY_EN	R/W	0x0	奇偶校验使能 0x1：奇偶效验使能 0x0：奇偶效验不使能。 奇偶校验使能用于接收器奇偶校验检查使能和发射器奇偶校验生成使能，适用于 8 位/9 位模式下。8 位模式下发射器发送的第 8 位为前 7 位奇偶校验生成的位，接收器校验前 7 位得到的值与第 8 位比较；9 位模式下发射器发送的 t8 为 8 位数据计算生成的位，接收器校验前 8 位数据得到的值与 r8 比较。
	0	SCI_C1_PARITY_ODD	R/W	0x0	奇偶校验选择 0x1：奇校验 0x0：偶校验 奇偶校验选择，奇效验表示数据字符中 1 的总数（包括奇偶校验位）是奇数。偶数表示数据字符中 1 的总数（包括奇偶校验位）是偶数。

5.1.4.4 SCI_C2 寄存器

地址	位	名称	R/W	复位值	描述
0x0C	7	SCI_C2_TX_EMPTY_INT_EN	R/W	0x0	发送缓存空中断使能 0x1：中断使能 0x0：中断禁止 中断使能不影响寄存器中中断标记置位，因此不开启中断可用于轮询模式。
	6	SCI_C2_TX_COMP_INT_EN	R/W	0x0	发送完成中断使能 0x1：中断使能 0x0：中断禁止 中断使能不影响寄存器中中断标记置位，因此不开启中断可用于轮询模式。
	5	SCI_C2_RX_FULL_INT_EN	R/W	0x0	接收满中断使能 0x1：中断使能 0x0：中断禁止 中断使能不影响寄存器中中断标记置位，因此不开启中断可用于轮询模式。
	4	SCI_C2_IDLE_INT_EN	R/W	0x0	闲置线路中断使能 0x1：中断使能 0x0：中断禁止 中断使能不影响寄存器中中断标记置位，因此不开启中断可用于轮询模式。
	3	SCI_C2_TX_EN	R/W	0x0	发射器使能 0x1：发射器打开 0x0：发射器关闭 通过向该位清 0 再置 1 可以排队发送一个闲置字符。
	2	SCI_C2_RX_EN	R/W	0x0	接收器使能 0x1：接收器打开 0x0：接收器关闭
	1	SCI_C2_RWU	R/W	0x0	接收器唤醒控制 0x1：接收器处于待机状态，等待唤醒条件 0x0：接收器正常运行 该位写入 1，将 SCI 接收器置于休眠状态，等待所选唤醒条件的自动硬件检测。唤醒条件既可以是信息间的闲置线路，也可以是某个字符中最高数据位中的逻辑 1。应用软件设置该位，也可以软件清零。一般应用硬件唤醒自动清除。
	0	SCI_C2_BREAK_TX	R/W	0x0	发送间隔段，先后将 1 和 0 写入该位，即在发送数据流中排入了一个间隔段，一旦移位器可用则发送一组中止字符。

5.1.4.5 SCI_S1 寄存器

地址	位	名称	R/W	复位值	描述
0x10	7	SCI_S1_TX_EMPTY_FLAG	RO	0x0	发送缓存空中断标记 0x1：发送缓存为空 0x0：发送缓存为满 写 SCI_D 可清除该标志。
	6	SCI_S1_TX_COMP_FLAG	RO	0x0	发送完成中断标记 0x1：发送完成，发射器闲置 0x0：发射器正在工作 发送新数据、重新配置发射使能、配置 SCI_C2_BREAK_TX 均清除该标志。
	5	SCI_S1_RX_FULL_FLAG	RO	0x0	接收满中断标记 0x1：接收缓存为满 0x0：接收缓存为空 读 SCI_D 可清除该标志。
	4	SCI_S1_IDLE_FLAG	RO	0x0	闲置线路中断标记 0x1：检测到闲置线路 0x0：未检测到闲置线路 在一段时间的活动后，当 SCI 接收线路已经闲置了一个全字符时间时，置位 IDLE。当 SCI_C1_IDLE_SEL =0，接收器在起始位后开始计数闲置位时间。因此，如果接收字符都为 1，这些位时间和停止位时间计数入接收器用于探测一个闲置线路所需逻辑高态（10/11/12 位时间，取决于 SCI_C1_STOP_WIDTH 和 SCI_C1_DATA_WIDTH 控制位）的全字符时间。当 SCI_C1_IDLE_SEL =1，接收器直到停止位后才开始计数闲置位时间。因此，停止位和前一字符末端的任何逻辑高时间不会计数入接收器用于探测一个闲置线路所需逻辑高的全字符时间。读 SCI_D 可清除该标志。
	3	SCI_S1_RX_OVERFLOW_FLAG	RO	0x0	接收溢出中断标记 0x1：接收溢出（新数据丢失） 0x0：没有溢出 当新的串行字符接收完毕要传输到接收数据寄存器时，原来接收的字符还没有从 SCI_D 读取，置位该标记。在这种情况下，新字符丢失（所有相关错误标记均丢失）。读 SCI_D 可清除该标志。
	2	SCI_S1_NOSIE_ERR_FLAG	RO	0x0	噪声中断标记 0x1：检测到噪声 0x0：未检测到噪声 接收器采样每位有 16 个样本，如果样本不一致则置位该标记（起始位的 8 个样本 RT3~RT10，其它位的 3 个样本 RT8~RT10，包括停止位）。读 SCI_D 可清除该标志。该位只记录当前存入数据寄存器中的数据的错误情况，数据更新状态会自动更新。

	1	SCI_S1_FRAME_ERR_FLAG	RO	0x0	帧错误中断标记 0x1：检测到帧错误 0x0：未检测到帧错误 当接收器在应该是停止位的时候检测到逻辑 0 时，置位该标记，同时接收满中断标记也会置位。读 SCI_D 可清除该标志。该位只记录当前存入数据寄存器中的数据的错误情况，数据更新状态会自动更新。
	0	SCI_S1_PARITY_ERR_FLAG	RO	0x0	奇偶校验错误中断标记 0x1：接收器奇偶校验错误 0x0：奇偶校验正确 读 SCI_D 可清除该标志。该位只记录当前存入数据寄存器中的数据的错误情况，数据更新状态会自动更新。

5.1.4.6 SCI_S2 寄存器

地址	位	名称	R/W	复位值	描述
0x14	7	SCI_S2_BREAK_CHECK_FLAG	RW	0x0	间隔段检测中断标记 0x1：检测到间隔段 0x0：未检测到间隔段 在 SCI_S2_BREAK_CHECK_EN =1 时并且检测到 LIN 中止字符时，就置位该标记。——检测 11/12/13 位的字符 0。软件直接将该寄存器位写 1 清除该标记。
	6	SCI_S2_RX_EDGE_FLAG	RW	0x0	Rx 管脚活动边沿中断标记 0x1：接收管脚上出现活动边沿 0x0：接收管脚上未出现活动边沿 软件直接将该寄存器位写 1 清除该标记。
	5	保留	——	——	保留
	4	SCI_S2_RX_INVERSION	RW	0x0	Rx 端数据反相 0x1：接收数据反转 0x0：接收数据不反转
	3	SCI_S2_RWU_IDLESEL	RW	0x0	接收唤醒闲置检测 0x1：在接收待机状态期间，检测到闲置字符时设置 IDLE 位， 0x0：在接收待机状态期间，检测到闲置字符时不设置 IDLE 位 用于控制接收器待机期间是否设置 IDLE 状态和中断。
	2	SCI_S2_BREAK_TX_SIZE	RW	0x0	间隔段生成位长度 0x1：用 13 位时间（如果 SCI_C1_STOP_WIDTH =1 或 SCI_C1_DATA_WIDTH =1，则分别增加 1 位时间）长度发送 0x0：用 10 位时间（如果 SCI_C1_STOP_WIDTH =1 或 SCI_C1_DATA_WIDTH =1，则分别增加 1 位时间）长度发送
	1	SCI_S2_BREAK_CHECK_EN	RW	0x0	间隔段检测使能

					0x1：在 11 位时间（如果 SCI_C1_STOP_WIDTH =1 或 SCI_C1_DATA_WIDTH =1，则分别增加 1 位时间）长度上检测 0x0：不检测 设置该位，在检测到 0x00 字符后 STOP 位如果仍旧检测到 0 字符则防止设置成帧错误和接收数据寄存器满标记，会继续检测下一个 0 字符。 在使能间隔段检测后如果检测到正常数据，则会正常接收，包括帧错误和接收满会正常置位。一旦检测到 0x00 字符且 STOP=0 时，会不接收数据，只检测间隔段。如果该位不使能，则所有数据正常接收，帧错误和接收满正常置位。
	0	SCI_S2_RX_ACTIVE_FLAG	RO	0x0	接收器活动标记 0x1：接收器活动 0x0：接收器闲置 当 SCI 接收器检测到有效起始位开始时，置位；当接收器检测到闲置线路时，清零；这种状态标记可以用来告诉系统当前是否正在接收 SCI 字符。

说明：

1. SCI_S2_BREAK_TX_SIZE、SCI_C1_STOP_WIDTH 和 SCI_C1_DATA_WIDTH 对应字符长度关系如下：

SCI_S2_BREAK_TX_SIZE	SCI_C1_STOP_WIDTH	SCI_C1_DATA_WIDTH	字符长度
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 寄存器

地址	位	名称	R/W	复位值	描述
0x18	7	SCI_C3_R8	RO	0x0	接收器的第 9 个数据，只读 该位为接收器的第 9 个数据位，当配置用于 9 位数据时，R8 可以视为数据寄存器的 MSB 左侧的第 9 个接收数据，需要在读取数据寄存器之前读取 R8，因为读数据寄存器能够完成自动的接收满标记清除，允许 R8 和 SCI_D 被新数据覆盖。
	6	SCI_C3_T8	R/W	0x0	发射器的第 9 个数据 发射器的第 9 个数据，当 SCI 配置用于 9 位数据时，T8 可以视为数据寄存器中缓冲数据的 MSB 左侧的第 9 个发送数据位。T8 应在数据寄存器写入前写入（如果它需要从它的原来值中修改）。如果 T8 不需要在新值（例如当它用于生成标记或奇偶效验）中修改，它就不需要在每次写数据寄存器前写入。当配置奇偶校验使能时，该位被用来作为奇偶校验位，是硬件自动生成的，不是写入的值。
	5	SCI_C3_TX_DIR	R/W	0x0	单线模式下 Tx 管脚方向选择 0x1：Tx 管脚是单线模式中的输出 0x0：Tx 管脚是单线模式中的输入
	4	SCI_C3_TX_INV	R/W	0x0	Tx 端数据反相 0x1：发送数据反转 0x0：发送数据不反转
	3	SCI_C3_RX_OF_INT_EN	R/W	0x0	接收溢出中断使能 0x1：中断使能 0x0：中断禁止
	2	SCI_C3_NOISE_ERR_INT_EN	R/W	0x0	噪声中断使能 0x1：中断使能 0x0：中断禁止
	1	SCI_C3_FRAME_ERR_INT_EN	R/W	0x0	帧错误中断使能 0x1：中断使能 0x0：中断禁止
	0	SCI_C3_PARITY_ERR_INT_EN	R/W	0x0	奇偶校验错误中断使能 0x1：中断使能 0x0：中断禁止

5.1.4.8 SCI_D 寄存器

地址	位	名称	R/W	复位值	描述
0x1C	7:0	SCI_DATA	R/W	0xFF	SCI 数据寄存器 读返回只读接收数据缓冲器的内容，写为写发送数据缓冲器。

5.1.4.9 SCI_EN 寄存器

地址	位	名称	R/W	复位值	描述
0x20	7:1	保留	——	——	保留
	0	SCI_ENABLE	R/W	0x0	SCI 模块使能寄存器 模块工作时钟门控使能 0x1：表示使能有效，打开模块工作时钟 0x0：关闭模块工作时钟，并且复位功能模块 在关闭模块使能的情况下，模块只能执行写读寄存器。系统低功耗模式下模块 Rx 边沿中断唤醒功能也必须在配置模块使能打开才有效。

5.1.5 程序范例

UART 功能：

```
//数据结构体初始化：
sci_parameter_struct sci0_parameter_init =
{
    SCI_BAUD_RATE_9600,              //设置波特率
    SCI_NORMAL_MODE,                 //正常工作模式
    SCI_STOP_BIT_1BIT,                   //1 位停止位数
    SCI_DATA_BIT_8BITS,              //8 位传输数据模式
    SCI_PARITY_DISABLE,              //奇偶校验不使能
    SCI_PARITY_ODD,                  //选择奇校验
    SCI_BREAK_TX_SIZE_13BITS,            //用 13 位时间长度发送
    SCI_BREAK_CHECK_ENABLE,      //间隔段检测使能
    SCI_RATE_AUTOMATCH_DISABLE, //固定波特率
    SCI_TX_ENABLE,                   //发送使能
    SCI_RX_ENABLE,                   //接收使能
    SCI_IDLE_SEL_STOPBIT,                //停止位后闲置字符位计数开始
    SCI_WAKE_SEL_ADDRRECE,           //地址标记唤醒
    SCI_TX_INV_DISABLE,              //发送数据不反转
    SCI_RWU_IDLESEL_DISABLE,             //在接收待机状态期间，检测到闲置字符时不设置 IDLE 位
    SCI_IE_TX_EMPTY | SCI_IE_TX_COMP | SCI_IE_RX_FULL | SCI_IE_BREAK_CHECK |
SCI_IE_RX_EDGE,/*发送缓存空中断使能、发送完成中断使能、
    接收满中断使能、间隔段检测中断使能、Rx 管脚活动边沿中断使能*/
    SCI_NVIC_ERR_ENABLE | SCI_NVIC_TX_DISABLE | SCI_NVIC_RX_ENABLE//错误中断使能、发送中断不
使能、接收中断使能
};

sci_parameter_struct sci1_parameter_init =
```

```
{
    SCI_BAUD_RATE_9600,              //设置波特率
    SCI_NORMAL_MODE,                 //正常工作模式
    SCI_STOP_BIT_1BIT,                   //1 位停止位数
    SCI_DATA_BIT_8BITS,                  //8 位传输数据模式
    SCI_PARITY_DISABLE,                  //奇偶校验不使能
    SCI_PARITY_ODD,                      //选择奇校验
    SCI_BREAK_TX_SIZE_13BITS,            //用 13 位时间长度发送
    SCI_BREAK_CHECK_ENABLE,      //间隔段检测使能
    SCI_RATE_AUTOMATCH_DISABLE,      //固定波特率
    SCI_TX_ENABLE,                       //发送使能
    SCI_RX_ENABLE,                       //接收使能
    SCI_IDLE_SEL_STOPBIT,                //停止位后闲置字符位计数开始
    SCI_WAKE_SEL_ADDRRECE,           //地址标记唤醒
    SCI_TX_INV_DISABLE,                  //发送数据不反转
    SCI_RWU_IDLESEL_DISABLE,             //在接收待机状态期间，检测到闲置字符时不设置 IDLE 位
    SCI_IE_TX_EMPTY | SCI_IE_TX_COMP | SCI_IE_RX_FULL | SCI_IE_BREAK_CHECK |
SCI_IE_RX_EDGE,/*发送缓存空中断使能、发送完成中断使能、
    接收满中断使能、间隔段检测中断使能、Rx 管脚活动边沿中断使能*/
    SCI_NVIC_ERR_ENABLE | SCI_NVIC_TX_ENABLE | SCI_NVIC_RX_DISABLE//错误中断使能、发送中断使
能、接收中断不使能
};

//主函数
int main(void)
{
    //UART 初始化
    sci1_port_sel(SCI1_PF0_PF1);             //选择 PF0 和 PF1 为 SCI1 功能端口
    sci_init(SCI1,sci1_parameter_init);
    sci_init(SCI0,sci0_parameter_init);      //选择 SCI0 或 SCI1 及相应的数据结构体
    sci_send_byte(SCI1,0x45);                //向 SCI 数据寄存器中写入 0x45
}

//子函数：
/***************选择 SCI1 功能端口******************/
void sci1_port_sel(uint8_t port_sel)
{
    SYS_PTSEL = port_sel;                    //将选择的端口组写入相应寄存器
}
/*******************SCI 初始化***********************/
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);
    /*将 SCI 初始化数据结构体中的值移位并写入波特率控制寄存器及控制与状态寄存器*/
    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);
        }
    }                                   //开启 SCI 错误中断、发送与接收中断
    sci_enable(scix);
    return SUCCESS;
}
/******************SCI 发送数据**********************/
void sci_send_byte(uint32_t scix,uint8_t dat)
```

```
{
    while(!(SCI_S1(scix) & SCI_S1_TX_EMPTY_FLAG));
    SCI_D(scix) = dat;                          //等待发送缓冲器为空后将数据写入数据寄存器
}
/*************SCI1 发送中断服务函数*****************/
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;
    }                                           //发送缓冲器为空则清除该中断标志
}
/*************SCI0 接收中断服务函数*****************/
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);           //清除间隔段检测中断标记
    }else if(state & SCI_IF_RX_FULL){
        tmp = SCI_D(SCI0);                      //清除接收满中断标记
    }else if(state & SCI_IF_RX_EDGE){
        sci_rx_edge_int_flag_clr(SCI0);     //清除 Rx 管脚活动边沿中断标记
    }
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);
    gpio_bit_set(GPIOE,GPIO_PIN_5);     //点灯
}
```

LIN 功能：

```
//LIN 发送数据数组及循环增量初始化定义
volatile uint8_t lin0_send_dat[11];
uint8_t i=0;
//数据结构体初始化
sci_parameter_struct sci_parameter_init =
{
    SCI_BAUD_RATE_9600,             //设置波特率
    SCI_NORMAL_MODE,                //正常工作模式
    SCI_STOP_BIT_1BIT,                  //1 位停止位数
    SCI_DATA_BIT_8BITS,             //8 位传输数据模式
    SCI_PARITY_DISABLE,             //奇偶校验不使能
    SCI_PARITY_ODD,                 //选择奇校验
    SCI_BREAK_TX_SIZE_13BITS,           //用 13 位时间长度发送
```

```
    SCI_BREAK_CHECK_ENABLE,      //间隔段检测使能
    SCI_RATE_AUTOMATCH_DISABLE, //固定波特率
    SCI_TX_ENABLE,                    //发送使能
    SCI_RX_ENABLE,                    //接收使能
    SCI_IDLE_SEL_STOPBIT,                 //停止位后闲置字符位计数开始
    SCI_WAKE_SEL_ADDRRECE,            //地址标记唤醒
    SCI_TX_INV_DISABLE,               //发送数据不反转
    SCI_RWU_IDLESEL_DISABLE,              //在接收待机状态期间，检测到闲置字符时不设置 IDLE 位
    SCI_IE_TX_EMPTY | SCI_IE_TX_COMP | SCI_IE_RX_FULL | SCI_IE_BREAK_CHECK |
SCI_IE_RX_EDGE,/*发送缓存空中断使能、发送完成中断使能、
    接收满中断使能、间隔段检测中断使能、Rx 管脚活动边沿中断使能*/
    SCI_NVIC_ERR_ENABLE | SCI_NVIC_TX_ENABLE | SCI_NVIC_RX_ENABLE//错误中断使能、发送中断使
能、接收中断使能
};
//主函数
int main(void)
{
    sci1_port_sel(SCI1_PF0_PF1);         //选择 PF0 和 PF1 为 SCI1 功能端口
    sci_init(SCI0,sci_parameter_init);
    sci_init(SCI1,sci_parameter_init);       //选择 SCI0 或 SCI1 及相应的数据结构体

    gpio_init(GPIOF,GPIO_MODE_OUT,GPIO_PIN_3);
    gpio_bit_set(GPIOF,GPIO_PIN_3); //LIN0_FLT 拉高
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_2);
    gpio_bit_set(GPIOG,GPIO_PIN_2); //LIN1_FLT 拉高
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_3);
    gpio_bit_set(GPIOG,GPIO_PIN_3); //LIN1_CS 拉高
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_4);
    gpio_bit_set(GPIOG,GPIO_PIN_4); //LIN0_CS 拉高

    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 给 SCI1 发送 11 个数据
}

//子函数：
/***************选择 SCI1 功能端口******************/
void sci1_port_sel(uint8_t port_sel)
{
    SYS_PTSEL = port_sel;                    //将选择的端口组写入相应寄存器
}
/*******************SCI 初始化**********************/
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);
    /*将 SCI 初始化数据结构体中的值移位并写入波特率控制寄存器及控制与状态寄存器*/
    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{
```

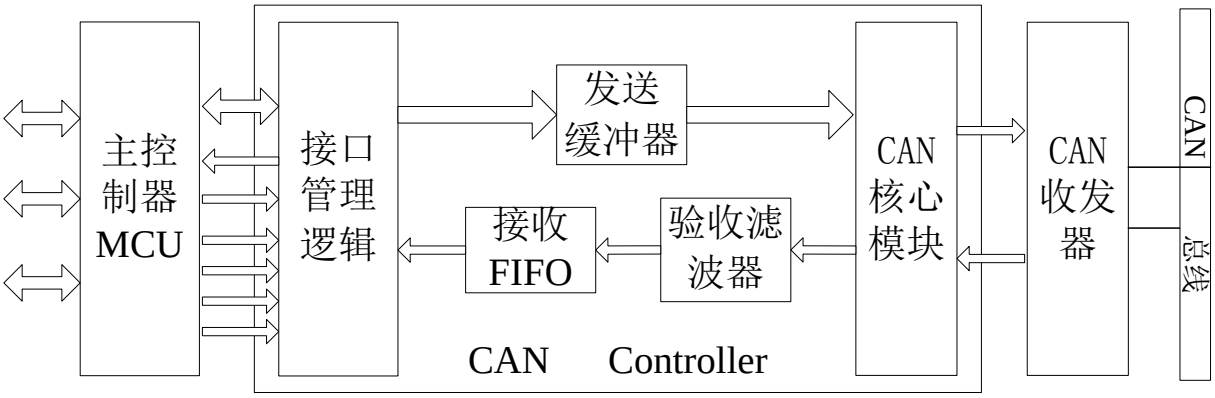
```
        if(temp & SCI_NVIC_ERR_ENABLE){
            NVIC_EnableIRQ(SCI1_ERR_IRQn);
        }
        if(temp & SCI_NVIC_TX_ENABLE){
            NVIC_EnableIRQ(SCI1_TX_IRQn);
        }
        if(temp & SCI_NVIC_RX_ENABLE){
            NVIC_EnableIRQ(SCI1_RX_IRQn);
        }
    }                                   //开启 SCI 错误中断、发送与接收中断
    sci_enable(scix);
    return SUCCESS;
}
/*****************LIN0 发送 LIN1 接收*******************/
void sci_lin_sci0tx_sci1rx(uint8_t *dat,uint8_t num)
{
    lin_dat = dat;
    lin_data_num = num;
    sci_lin_break_check_enable(SCI1);       //SCI1 间隔段检测使能
    sci_lin_break_trans(SCI0);              //SCI0 发送间隔段
    NVIC_EnableIRQ(SCI0_TX_IRQn);       //SCI0 开启发送中断
}
/***************LIN0 发送中断服务函数*****************/
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);//等待数据串发送完成即关闭发送中断
        }
        SCI_D(SCI0) = *lin_dat ++;
    }
}
/*****************LIN1 接收中断服务函数*****************/
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);           //清除间隔段检测中断标记
    }else if(state & SCI_IF_RX_FULL){
        tmp = SCI_D(SCI1);              //清除接收满中断标记
```

```
    }else if(state & SCI_IF_RX_EDGE){
        sci_rx_edge_int_flag_clr(SCI1);      //清除 Rx 管脚活动边沿中断标记
    }
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);
    gpio_bit_set(GPIOE,GPIO_PIN_5);      //点灯
}
```

5.2 CAN

本章提供了 CAN 控制器的内部框图、功能描述以及寄存器的介绍，以及应用参考指导。

CAN 是一种多主机方式的串行通讯总线，基本设计规范要求有高的位速率，高抗电磁干扰性，而且能够检测产生的任何错误。



5.2.1 主要特性

CAN控制器是一个应用广泛的现场总线，它具有以下几种特性：

1. 无损仲裁；
2. 极高的总线利用率，只要总线空闲，任何单元都可以开始发送报文；
3. 数据传输速率最高支持500Kbit/s；
4. 可根据报文的ID决定接收或屏蔽该报文；
5. 错误处理和检错机制；
6. 发送的信息遭到破坏后，可自动重发；
7. 节点在错误严重的情况下具有自动退出总线的功能；
8. 支持低功耗唤醒功能；
9. 标准帧和扩展帧信息的接收和传送（CAN2.0B）；
10. 在标准和扩展格式中都有单/双验收滤波器含屏蔽和代码寄存器（CAN2.0B）；
11. 可编程的错误限制报警（CAN2.0B）；
12. 增加消极错误中断、仲裁丢失中断和总线错误中断（CAN2.0B）；
13. 增加只听模式和环回自测试模式（CAN2.0B）；
14. 仲裁丢失中断以及详细的位位置（CAN2.0B）；
15. 模块时钟可配为系统时钟（见3.3.6.15），但可能引入PLL或RC模块偏差；

仲裁的机制确保信息和时间均不会损失。两个以上的单元同时开始发送消息时，对各消息ID 的每个位进行逐个仲裁比较。仲裁获胜（连续输出显性电平最多的单元，被判定为优先级最高）的单元可继续发送消息，仲裁失利的单元则立刻停止发送而进行接收工作。

5.2.2 功能说明

5.2.2.1 FIFO 功能

FIFO 为在 CAN 总线上发送的数据被载入 CAN 控制器的存储区，并分为发送缓冲器 FIFO 和接收缓冲器 FIFO。这些 FIFO 包含 5 个字节的标识符和帧信息，而最多可以包含 8 个数据字节。

—— 1 字节帧信息；

—— 2 个或 4 个标识符字节标准帧或扩展帧；

——最多 8 个数据字节；

CAN 地址	名称	组成和标注
0x40	帧信息	1 位说明如果报文包括一个标准帧或扩展帧 1 位远程传输请求位 4 位数据长度码表示数据字节的数量
0x44 0x48	标识符字节 1 标识符字节 2	标准帧：11 位标识符 扩展帧：16 位标识符
0x4C 0x50	标识符字节 3 标识符字节 4	仅扩展帧：13 个标识符
标准帧：0x4C ~ 0x68 扩展帧：0x54 ~ 0x70	数据字节 1-8	由数据长度代码说明最多 8 个数据字节

接收缓冲区，CAN控制器有多播功能，即主控制器发送报文，所有的CAN控制器都接收报文，接收之后都暂时存放在接收缓冲区中，滤波器自动检查标识符和数据字节，当接收的标识符和数据字节与滤波器的滤波设置完全匹配时，CAN控制器才会把接收缓冲区中的报文标识符和数据字节存放到接收缓冲器中，由于CAN节点的标识符是唯一的，所以只有唯一的节点接收。

5.2.2.2 接收缓存

CAN 控制器内部有两个缓存，分别存放接收数据及接受数据的字节数，可以防止来不及处理 CAN 接收数据时，做暂时保存。128*8bits 存储数据，32*8bits 储每笔信息长度。当接收数据时没有读取数据，缓存存储数据达到 96 个时，会发出溢出中断（当中断使能时产生溢出中断）。

5.2.2.3 中断

以下中断可使 CAN 控制器立即作用在某些状态上。一旦 CAN 产生中断，CAN 控制器就将中断输出设为高电平，直到主控制器通过读中断寄存器然后清中断。

1. 接收中断；
2. 发送中断；
3. 错误报警中断；
4. 数据溢出中断；
5. 唤醒中断；
6. 消极错误中断；
7. 仲裁丢失中断；
8. 总线错误中断；

具体中断说明如下：

1. 错误报警：总线关闭、发送接收错误计数超过错误报警限制（基本模式下为 96，扩展模式下为寄存器错误报警限制寄存器 CAN_EMLR 配置值）；
2. 数据溢出：接收 FIFO 大于 96 时发出数据溢出中断；
3. 消极错误：错误计数值大于等于 128 时发出消极错误中断；

表 1. 错误状态和计数值

单元错误状态	发送错误计数值（TEC）		接收错误计数值（REC）
主动错误状态	0～127	且	0～127
被动错误状态	128～255	或	128～255
总线关闭态	256～		—

4. 仲裁丢失：仲裁丢失时，会产生相应的仲裁丢失中断（中断允许）。同时，位流处理器的当前位位

置被捕捉送入仲裁丢失捕捉寄存器。

5. 总线错误：总线发生错误时被迫产生相应的错误中断（中断允许时）。同时，位流处理器的当前位置被捕捉送入错误代码捕捉寄存器。其中包括位错误、填充错误、CRC错误、格式错误、ACK错误。

表 9. 错误的种类

错误的种类	错误的内容	错误的检测帧（段）	检测单元
位错误	比较输出电平和总线电平（不含填充位），当两电平不一样时所检测到的错误。	• 数据帧（SOF~EOF） • 遥控帧（SOF~EOF） • 错误帧 • 过载帧	发送单元 接收单元
填充错误	在需要位填充的段内，连续检测到 6 位相同的电平时所检测到的错误。	• 数据帧（SOF~CRC 顺序） • 遥控帧（SOF~CRC 顺序）	发送单元 接收单元
CRC 错误	从接收到的数据计算出的 CRC 结果与接收到的 CRC 顺序不同时所检测到的错误。	• 数据帧（CRC 顺序） • 遥控帧（CRC 顺序）	接收单元
格式错误	检测出与固定格式的位段相反的格式时所检测到的错误。	• 数据帧（CRC 界定符、ACK 界定符、EOF） • 遥控帧（CRC 界定符、ACK 界定符、EOF） • 错误界定符 • 过载界定符	接收单元
ACK 错误	发送单元在ACK槽(ACK Slot)中检测出隐性电平时所检测到的错误（ACK 没被传送过来时所检测到的错误）。	• 数据帧（ACK 槽） • 遥控帧（ACK 槽）	发送单元

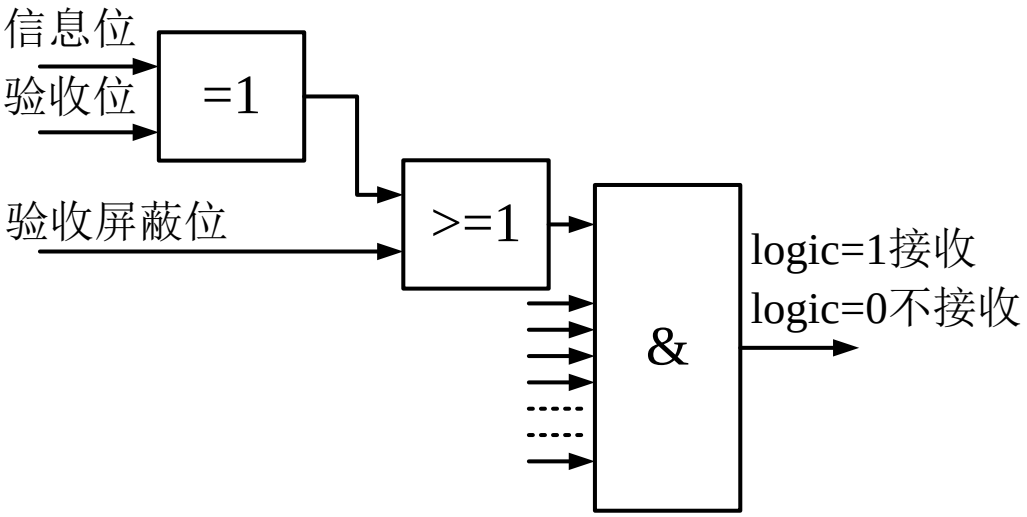
5.2.2.4 验收滤波

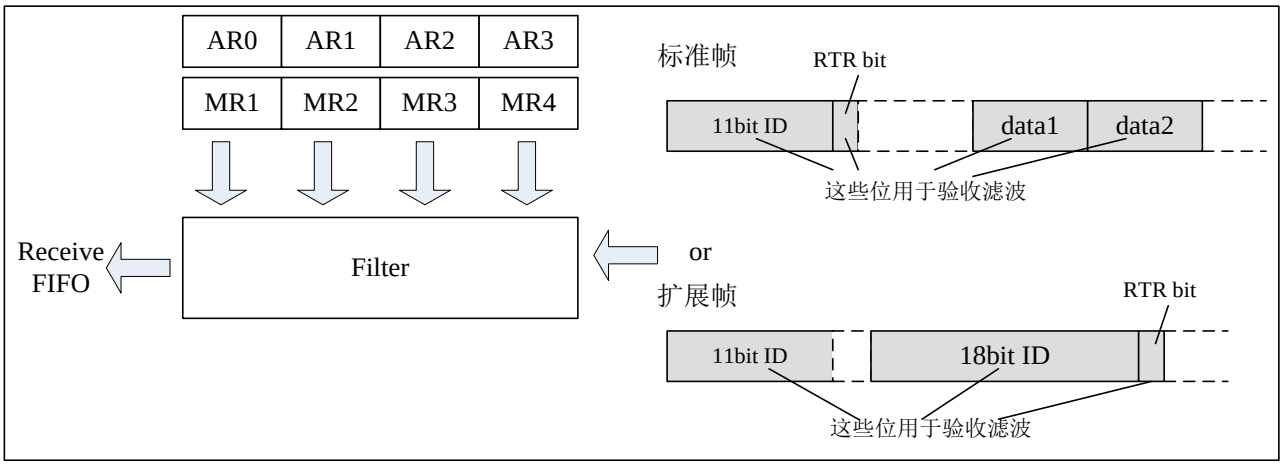
验收滤波由4个8位的验收码寄存器（CAN_IDAR0/CAN_IDAR1/CAN_IDAR2/CAN_IDAR3）和验收屏蔽寄存器（CAN_IDMR0/CAN_IDMR1/CAN_IDMR2/CAN_IDMR3）组成。这些寄存器可用于控制一个长的滤波器或两个短的滤波器。报文的哪些位用于验收滤波，取决于收到的帧（标准帧或扩展帧）和选择的滤波器模式（单滤波器或双滤波器）。标准帧的验收滤波可以包括RTR位甚至数据字节。对于不需要经过验收滤波的报文位，验收屏蔽寄存器必须相应的位位置上置1。

如果报文不包括数据字节（例如是一个远程帧或者数据长度码为零）但是验收滤波包括数据字节，则如果标识符直到 RTR 位都有效的话，报文会被接收。

单滤波器：

这种滤波器配置可以定义一个长滤波器（4 字节）。滤波器字节和信息字节之间位的对应关系取决于当前接收帧格式。





如果接收的是标准帧格式的信息，在验收滤波中只使用前两个数据字节来存放包括 RTR 位的完整的识别码。如果由于置位 RTR 位而导致没有数据字节，或因为设置相应的数据长度代码而没有或只有一个数据字节，信息也会被接收的。对于一个成功接收的信息，所有单个位的比较后都必须发出接受信号。

注意：CAN_IDMR1 和 CAN_IDAR1 的低四位是不用的。为了和将来的产品兼容，这些位可通过设置为 1 而定为“不影响”。

如下图例子所示（单滤波器滤波标准帧）：

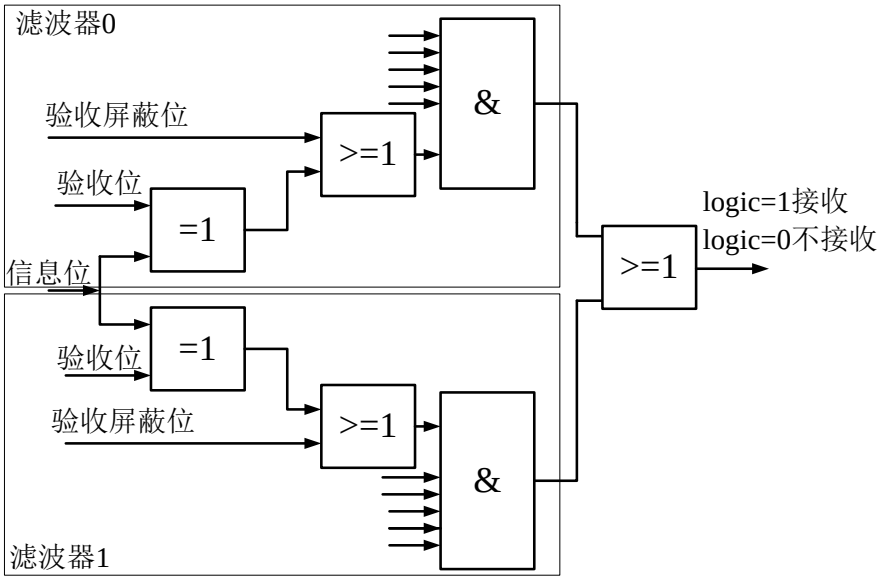
n	0	1(高四位)	2	3
CAN_IDARn	01XX X010	XXXX	XXXX XXXX	XXXX XXXX
CAN_IDMRn	0111 1010	1111	1111 1111	1111 1111
接收的报文（ID[28:18]，RTR）	01xx x010	xxxx		

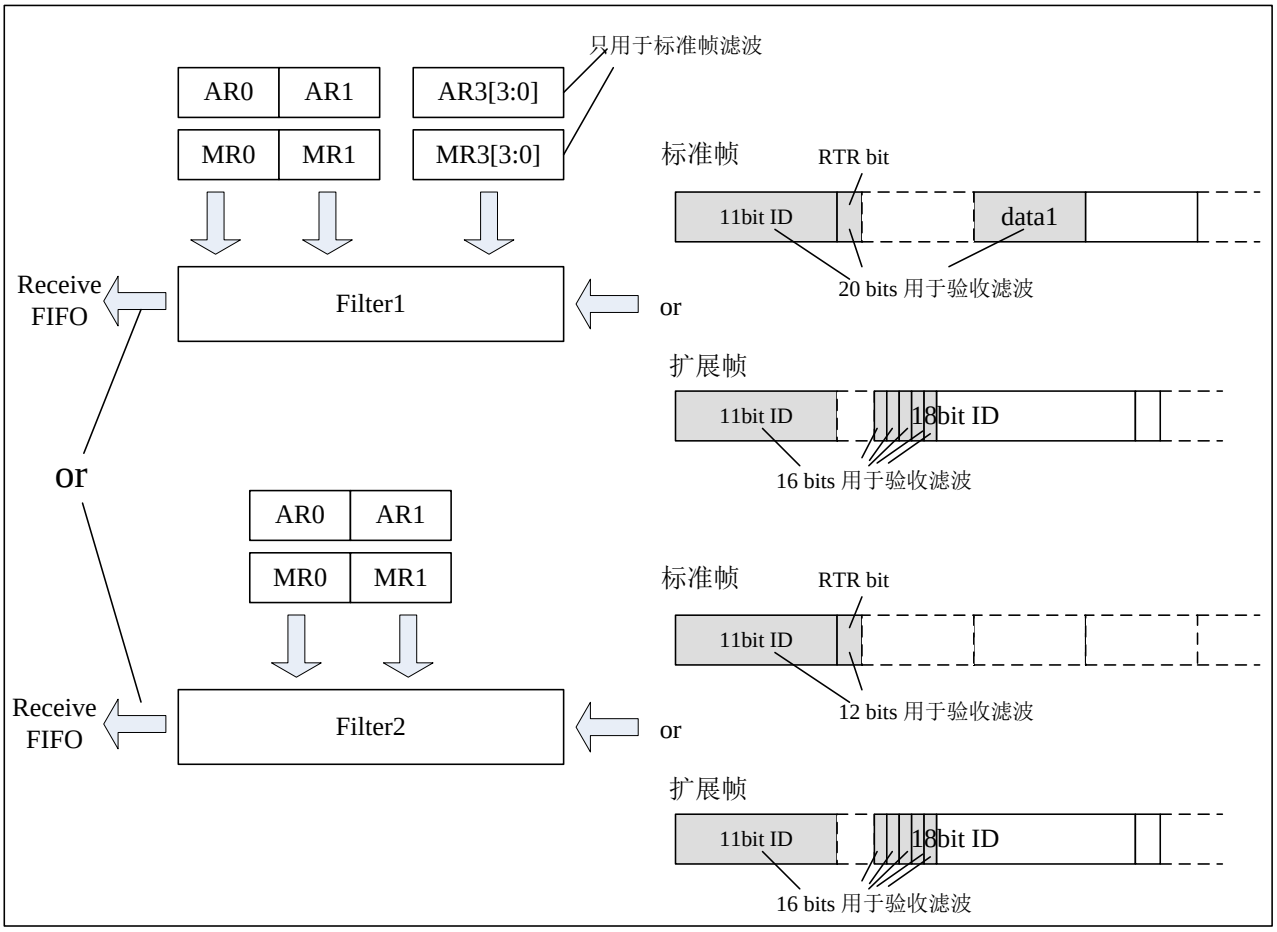
在验收屏蔽寄存器是“1”位置上，标识符相应的位可以是任何值，譬如远程发送请求位和数据字节1和2 的位。

扩展帧：如果接收的信息是扩展帧格式的，包括 RTR 位的全部识别码将被接受过滤使用。 为了成功接收信息，每个位的比较后都必须发出接受信号。CAN_IDMR3 的最低两位和 CAN_IDAR3 是不用的，这些位应该通过置位来定为“不影响”。

双滤波器：

这种配置可以定义两个短滤波器。一条接收的信息要和两个滤波器比较来决定是否放入接收缓冲器中。至少有一个滤波器发出接受信号，接收的信息才有效。滤波器字节和信息字节之间位的对应关系取决于当前接收的帧格式。





标准帧：如果接收的是标准帧信息，被定义的两个滤波器是不一样的，第一个滤波器比较包括 RTR 位的整个标准识别码和信息的第一个数据字节。第二个滤波器只比较包括 RTR 位的整个标准识别码。

为了成功接收信息，所有单个位的比较时应至少有一个滤波器表示接受。RTR 位置位或数据长度代码是 0 时表示没有数据字节存在，无论怎样，只要从开始到 RTR 位的部分都被表示接收，信息就可以通过滤波器 1。

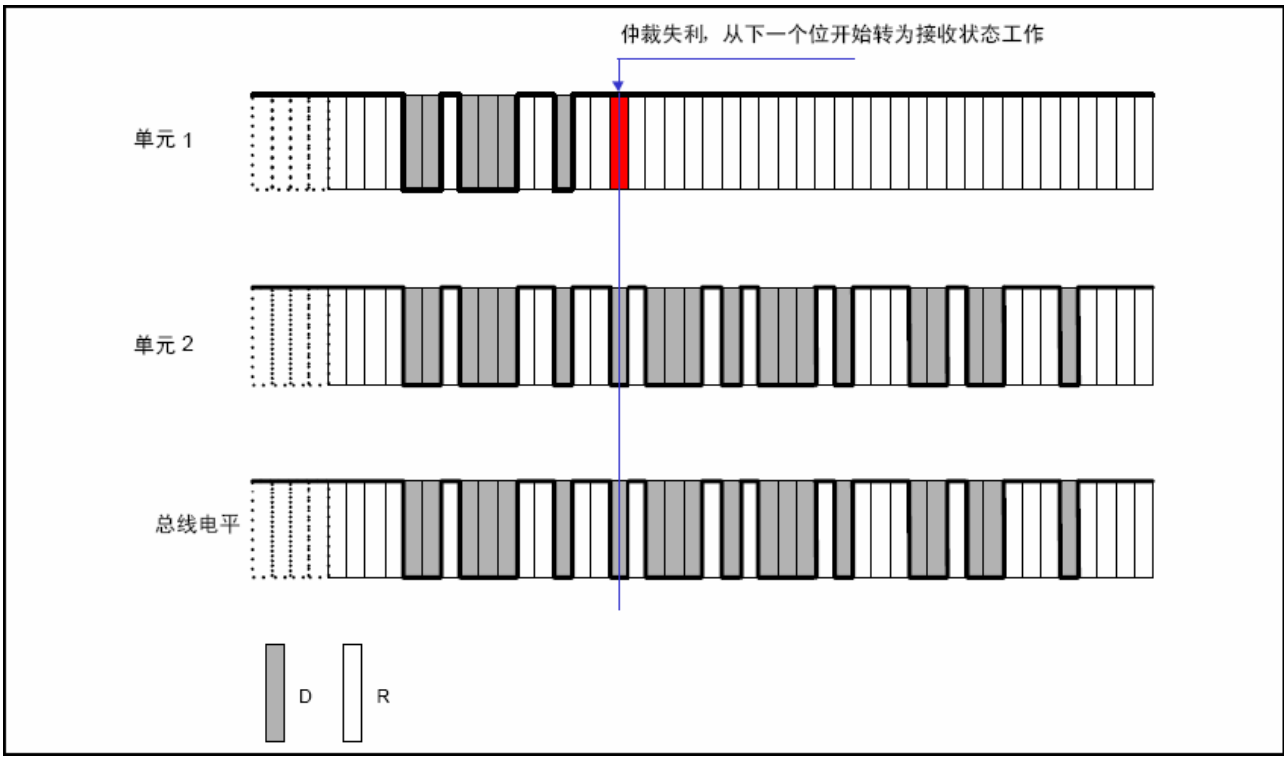
如果没有向滤波器请求数据字节过滤，CAN_IDMR1（AMR1）和 CAN_IDMR3（AMR3）的低四位必须被置为 1（不影响）。当使用包括 RTR 位的整个标准识别码时，两个滤波器都同样工作。

扩展帧：如果接收到扩展帧信息，定义的两个滤波器是相同的。两个滤波器都只比较扩展识别码的前两个字节。

为了能成功接收信息，所有单个位的比较时至少有一个滤波器表示接收。

帧类型	单滤波器模式	双滤波器模式	
标准	验收的报文位： - 11位标识符 - RTR位 -第一个数据字节8位 -第二个数据字节8位 使用的验收码寄存器和屏蔽寄存器： - CAN_IDAR0/ CAN_IDAR1（高四位）/ CAN_IDAR2/ CAN_IDAR3 - CAN_IDMR0/ CAN_IDMR1（高四位）/ CAN_IDMR2/ CAN_IDMR3 （接收屏蔽寄存器的未使用的位应设为1）	滤波器1 验收的报文位： - 11 位标识符 - RTR 位 -第一个数据字节8位 使用的验收码寄存器和屏蔽寄存器： - CAN_IDAR0/ CAN_IDAR1 或CAN_IDAR3的低四位 - CAN_IDMR0/ CAN_IDMR1 或CAN_IDMR3的低四位	滤波器2 用于验收测试的报文位： - 11 位标识符 - RTR位 使用的验收码寄存器和验收屏蔽寄存器： - CAN_IDAR2或CAN_IDAR3的高四位 - CAN_IDMR2或CAN_IDMR3的高四位
扩展	用于验收的报文位： -11位基本的标识符 -18位扩展的标识符 -RTR位 使用的验收码和验收屏蔽寄存器： - CAN_IDAR0/ CAN_IDAR1/ CAN_IDAR2或CAN_IDAR3的高六位 - CAN_IDMR0/ CAN_IDMR1/ CAN_IDMR2或CAN_IDMR3的高六位 （验收屏蔽寄存器的未使用的位应设为1）	滤波器1： 用于验收的报文位： -11位基本标识符 -扩展标识符的5个最高位 使用的验收码和验收屏蔽寄存器： - CAN_IDAR0/ CAN_IDAR1 和CAN_IDMR0/ CAN_IDMR1	滤波器2： 用于测试验收的报文位： - 11位基本的标识符 -扩展标识符的5 最高位 使用的验收码和验收屏蔽寄存器： - CAN_IDAR2/ CAN_IDAR3/ 和CAN_IDMR2/ CAN_IDMR3

5.2.2.5 仲裁



在总线空闲态，最先开始发送消息的单元获得发送权。多个单元同时开始发送时，各发送单元从仲裁段的第一位开始进行仲裁。连续输出显性电平最多的单元可继续发送。

CAN 采用无损仲裁的方式，当两个主机同时开始发送的新的一帧信息时，只在仲裁域中进行仲裁，当发送主机采样到总线的信号为显性电平，而自己发送的是隐性电平的时候，失去仲裁，停止发送信息，转为接收状态。

5.2.2.6 回环自测试

环回自测模式独立于外部系统的连接，有时用于检查软件，以帮助隔离系统问题。在该模式中，发送器输出内部连接到接收器输入。Rx 输入管脚被忽略， Tx输出进入隐性状态（逻辑1）。发送时， CAN 把它自己发送的报文作为从远程节点接收的报文。在该状态中，CAN将忽略CAN帧应答场中ACK间隙中发送的位，以确保正确接受它自己的报文。同时生成发送和接收中断。

5.2.2.7 CAN 自身睡眠唤醒

当CAN自身空闲时，也就是说该模块不接收任何报文且所有发送缓冲器空，可以通过设置CAN_CMR[5]位为1请求进入睡眠模式。在睡眠模式中，可以通过停止CAN的所有时钟（除了CPU端访问寄存器的时钟外）来降低功耗。

只有当 CAN 处于睡眠模式（CAN_CMR[5] = 1， CAN_SLPAK = 1）、唤醒功能使能（CAN_WUP[0] = 1）且唤醒中断使能 （CAN_IE[4]=1）时， CAN 唤醒中断才可能发生。

CAN控制器进入睡眠模式的时间取决于固定的同步延迟及其当前状态：

1. 如果有报文缓冲器等待发送，CAN 将继续发送，直到所有发送报文缓冲器空（成功发送或中止），然后再进入睡眠模式；

2. 如果CAN正在接收，它继续接收，并且一旦CAN总线空闲，就立即进入睡眠模式；

3. 如果CAN 既不在发送也不在接收，它会立即进入睡眠模式；

如果唤醒中断使能 CAN_WUP[0]位还未置位， CAN 将屏蔽它在 CAN 上检测到的任何信号。Rx 管脚因此被设置为隐性状态，这将把 CAN 锁在睡眠模式。CAN_WUP[0]必须在进入睡眠模式前设置，以便发挥作用。当检测到 CAN 总线上任意显性信号时 CAN 被唤醒，时钟被打开同时唤醒中断状态 CAN_IF[4]被置位。

只有当出现以下情形时，CAN才能够退出睡眠模式（唤醒）：

1. 如果出现CAN总线有效和CAN_WUP[0]= 1；

2. 清除CAN_CMR[5]位；

退出睡眠模式之后需配置唤醒功能使能CAN_WUP[0] = 0。

当CAN控制器处于睡眠模式时，通过配置CAN_WUP[1]来设置CAN总线上唤醒脉冲的长度。CAN_WUP[1]=0时，CAN被CAN总线上的任意显性信号唤醒，而CAN_WUP[1]=1时，CAN只有在CAN总线上的显性脉冲长度大于等于2us时才唤醒。2us通过外部晶振时钟计数滤波。

5.2.2.8 通过 CAN 唤醒系统

如果系统进入睡眠模式下（sleepdeep），只能通过外部显性脉冲唤醒进行唤醒，退出睡眠模式。此时，需配置CAN唤醒中断使能CAN_IE[4]=1，否则无法唤醒系统。

考虑到系统进入睡眠模式下，可能会被外部噪声信号误唤醒的情况，软件可通过配置CAN_SWU_FILT寄存器来选择是否进行硬件滤波，滤波时间为1.5us。此滤波仅作为系统唤醒时的CAN脉冲滤波，其它情况无

效。

5.2.3 寄存器列表

CAN 模块地址区间：0x5005_0000~0x5005_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	CAN_MOD	R/W	模式寄存器	0x01
0x04	CAN_CMR	R/W	命令寄存器	0x00
0x08	CAN_SR	RO	状态寄存器	0x3C
0x0C	CAN_IF	RO	中断寄存器	0x00
0x10	CAN_IE	R/W	中断使能寄存器	0x00
0x18	CAN_BTR0	R/W	总线定时寄存器 0	0x00
0x1C	CAN_BTR1	R/W	总线定时寄存器 1	0x00
0x20	CAN_SLPAK	RO	自身睡眠模式确认寄存器	0x00
0x24	CAN_WUP	R/W	自身唤醒寄存器	0x00
0x2C	CAN_ALC	RO	仲裁丢失捕捉寄存器	0x00
0x30	CAN_ECC	RO	错误代码捕捉寄存器	0x00
0x34	CAN_EMLR	R/W	错误报警限制寄存器	0x60
0x38	CAN_RXERR	R/W	RX 错误计数寄存器	0x00
0x3C	CAN_TXERR	R/W	TX 错误计数寄存器	0x00
0x40	CAN_IDAR0	R/W	验收代码寄存器 0	0x00
0x40	CAN_FRCTL	R/W	数据信息寄存器	0x00
0x44	CAN_IDAR1	R/W	验收代码寄存器 1	0x00
0x44	CAN_ID0	R/W	ID 寄存器 0	0x00
0x48	CAN_IDAR2	R/W	验收代码寄存器 2	0x00
0x48	CAN_ID1	R/W	ID 寄存器 1	0x00
0x4C	CAN_IDAR3	R/W	验收代码寄存器 3	0x00
0x4C	CAN_ID2	R/W	ID 寄存器 2	0x00
0x50	CAN_IDMR0	R/W	验收屏蔽寄存器 0	0x00
0x50	CAN_ID3	R/W	ID 寄存器 3	0x00
0x54	CAN_IDMR1	R/W	验收屏蔽寄存器 1	0x00
0x54	CAN_DATA0	R/W	数据寄存器 0	0x00
0x58	CAN_IDMR2	R/W	验收屏蔽寄存器 2	0x00
0x58	CAN_DATA1	R/W	数据寄存器 1	0x00
0x5C	CAN_IDMR3	R/W	验收屏蔽寄存器 3	0x00
0x5C	CAN_DATA2	R/W	数据寄存器 2	0x00
0x60	CAN_DATA3	R/W	数据寄存器 3	0x00
0x64	CAN_DATA4	R/W	数据寄存器 4	0x00
0x68	CAN_DATA5	R/W	数据寄存器 5	0x00
0x6C	CAN_DATA6	R/W	数据寄存器 6	0x00
0x70	CAN_DATA7	R/W	数据寄存器 7	0x00

偏移地址	寄存器名称	R/W	功能描述	初始值
0x74	CAN_RMC	RO	RX 信息计数器	0x00
0x78	CAN_ENABLE	R/W	使能寄存器	0x00
0x80	CAN_CLRISR	R/W	清中断状态寄存器	0x00
0x84	CAN_CLRECC	R/W	清除错误捕获码寄存器	0x00
0x88	CAN_SWU_FILT	R/W	系统唤醒滤波寄存器	0x00

5.2.4 寄存器详细描述

5.2.4.1 CAN_MOD 寄存器

地址	位	名称	R/W	复位值	描述
0x00	7:4	保留	——	——	保留
	3	CAN_MOD_AFM	R/W	0x0	验收滤波器模式： 0x1：选择单个验收滤波器(32位长度) 0x0：选择两个验收滤波器(每个有16位激活)
	2	CAN_MOD_STM	R/W	0x0	环回自测模式： 0x1 ：环回自测使能 0x0 ：环回自测禁止 当设置了该位时，CAN执行可用于自测操作的内部环回。发送器的位流输出从内部流回接收器。环回自测模式独立于外部系统的连接，有时用于检查软件，以帮助隔离系统问题。在该模式中，发送器输出内部连接到接收器输入。Rx输入管脚被忽略， Tx输出进入隐性状态(逻辑1）。发送时，CAN的运行如常，把它自己发送的报文作为从远程节点接收的报文。在该状态中，CAN将忽略CAN帧应答场中ACK间隙中发送的位，以确保正确接收它自己的报文，同时生成发送和接收中断。
	1	CAN_MOD_LOM	R/W	0x0	只听模式： 0x1：即使成功接收信息，CAN 控制器也不会向总线发送应答信号； 错误计数器停止在当前值 0x0：正常模式 只听模式不会向总线发送应答信号，数据也不会写入，不产生接收中断，但会产生总线错误（应答定义符错误）。
	0	CAN_MOD_RM	R/W	0x1	置位模式： 0x1：置位后，中止当前正在接收/发送的信息,进入置位模式，软件可配置置位模式下的相关寄存器 0x0：正常工作模式

5.2.4.2 CAN_CMR 寄存器

地址	位	名称	R/W	复位值	描述
0x04	7:6	保留	——	——	保留
	5	CAN_CMR_SPM	R/W	0x0	自身睡眠模式请求 0x1 ：睡眠模式请求，当 CAN 总线空闲时 CAN 进入睡眠模式 0x0 ：运行中，CAN正常工作 该位请求CAN进入睡眠模式，这是一个内部节电模式。当CAN总线空闲时，也就是说该模块不接收任何报文且所有发送缓冲器空，睡眠模式请求被受理。睡眠模式维持有效，直到该位被CPU清除或者根据CAN_WUP_ENABLE的设置，CAN检测到总线上有有效并自行清除。
	4	CAN_CMR_SRR	R/W	0x0	自接收请求 0x1：发送信息可被同时接收 0x0：无动作 验收滤波设置、发送中断、接收中断在此模式下同样有效。
	3	CAN_CMR_CDO	R/W	0x0	清除数据溢出 0x1：清除数据溢出状态位CAN_SR_DOS 0x0：无动作 溢出状态只在溢出过程中发生一次，如果在清除溢出状态时仍旧处于溢出的状况，则无法再产生溢出状态。该位配置后会自动清除。
	2	CAN_CMR_RRB	R/W	0x0	清数据缓冲器状态和接收中断 0x1：清数据缓冲器状态和接收中断 0x0：无动作 每一次读出接收缓存一笔数据时，须配置该位为1，否则无法继续读下一笔数据。该位自动清除。
	1	CAN_CMR_AT	R/W	0x0	中止发送 0x1：中止当前发送后的下一笔数据，用于停止自动重发动作 0x0：无动作 中止过程中，要想知道信息是否成功发送，可以通过发送完毕状态位来查看。不过，这应在在发送缓冲器状态位置1或产生发送中断之后。要注意的是即使因为发送缓冲器状态位变为0而使信息被中止也会产生发送中断。 该位配置后，当CAN_CMR_TR置位时会自动清除。
	0	CAN_CMR_TR	R/W	0x0	发送请求 0x1：发送信息 0x0：无动作 该位发送配置后自动清除，下笔发送需重新置1。

5.2.4.3 CAN_SR 寄存器

地址	位	名称	R/W	复位值	描述
0x08	7	CAN_SR_BS	RO	0x0	总线状态 0x1：总线关闭，CAN退出总线活动 0x0：总线开启，CAN加入总线活动 当发送错误计数器超过限制(255)，总线状态位被置为1(总线关闭)，CAN控制器将设置置位模式为 1，而且产生一个错误报警中断。此时发送错误计数器被置为127，接收错误计数器被清除。这种模式将会保持直到CPU将置位模式清除。完成这些之后，CAN控制器将通过发送错误计数器的减1计数以等待协议规定的最少时间（128个总线空闲信号）。之后总线状态位被清除（总线开启），错误状态位被置为0，错误计数器复位且产生一个错误报警中断。这期间读TX错误计数器可以得知关于总线关闭修复的状态信息。
	6	CAN_SR_ES	RO	0x0	出错状态 0x1：出错；至少出现一个错误计数器满或超过错误报警限制值的情况 0x0：两个错误计数器都在报警限制值以下 根据 CAN2.0B协议规定，在接收和发送期间检测到错误会影响错误计数器。至少有一个错误计数器满或超过CPU报警限制CAN_EMLR时错误状态位被置位。中断允许时，会产生错误报警中断。CAN_EMLR硬件复位后的默认值是 96。
	5	CAN_SR_TS	RO	0x0	发送状态 0x1：发送，CAN控制器在传送信息 0x0：空闲，没有要发送的信息
	4	CAN_SR_RS	RO	0x0	接收状态 0x1：接收，CAN控制器正在接收信息 0x0：空闲，没有正在接收的信息 如果接收状态位和发送状态位都是0，则CAN总线是空闲的。如果这两位都是1，则控制器正在等待下一次空闲。硬件启动后必须一直检测到11个连续的隐性位，直到空闲状态到来。总线关闭后会产生128个11位的连续隐性位。
	3	CAN_SR_TCS	RO	0x1	发送完毕状态 0x1：完毕，最近一次发送请求被成功处理 0x0：未完毕，当前发送请求未处理完毕 一旦发送请求位或自接收请求位被置1，发送完毕状态位就会被置0。发送完毕状态位会保持为0直到发送成功。
	2	CAN_SR_TBS	RO	0x1	发送缓冲器状态 0x1：释放；CPU可以向发送缓冲器写信息 0x0：锁定；CPU不能访问发送缓冲器，有信息正在等待发送或正在发送 如果CPU试图在发送缓冲器状态位是0时向发送缓冲器写，写入的字

					节将不被接受且在没有任何提示的情况下丢失。
	1	CAN_SR_DOS	RO	0x0	数据溢出状态 0x1：即将溢出，RXFIFO已经有超过96Bytes的数据 0x0：无溢出发生 当要接收的信息已经成功通过验收滤波器的时候，CAN控制器需要在RXFIFO中有足够的空间来存储信息描述符和每一个接收的数据字节。如果没有足够的空间来存储信息，信息就会丢失。CANRXFIFO大小为128Bytes，在96Bytes时会发出数据溢出状态。如果信息没有被成功接收（例如，由于错误），就没有数据溢出情况提示。
	0	CAN_SR_RBS	RO	0x0	数据缓冲器状态 0x1：RXFIFO有可用信息 0x0：已空，无可用信息 读出RXFIFO中的所有信息之后，此位被清除。

5.2.4.4 CAN_IF 寄存器

地址	位	名称	R/W	复位值	描述
0x0C	7	CAN_IF_BEI	RO	0x0	总线错误中断状态： 0x1：CAN控制器检测到总线错误且该中断使能时此位被置位 0x0：无总线错误中断
	6	CAN_IF_ALI	RO	0x0	仲裁丢失中断状态： 0x1：当 CAN 控制器丢失仲裁且该中断使能时此位被置位 0x0：无仲裁丢失中断
	5	CAN_IF_EPI	RO	0x0	消极出错中断状态： 0x1：当CAN控制器至少出现一个错误计数器满或超过错误报警限制值，或者从被动错误计数值又回到了主动错误计数值时，该中断使能时此位被置位 0x0：无信息出错中断
	4	CAN_IF_WUPI	RO	0x0	唤醒中断状态： 0x1：当 CAN 处于睡眠模式，唤醒使能打开时，检测到 CAN 总线有效时置位 0x0：处于休眠模式时未检测到唤醒
	3	CAN_IF_DOI	RO	0x0	数据即将溢出中断状态 0x1：当数据 FIFO 达到 96Byte 以上时且该中断使能时此位被置位 0x0：无数据即将溢出发生
	2	CAN_IF_EI	RO	0x0	错误中断状态 0x1：任意错误发生且该中断使能时置位 0x0：无错误发生
	1	CAN_IF_TI	RO	0x0	发送中断状态 0x1：发送缓冲器可以进行下一次发送数据填充且该中断使能时置位

					0x0：发送缓冲器正在工作
	0	CAN_IF_RI	RO	0x0	接收中断状态 0x1：当接收缓冲器不空和该中断使能时置位 0x0：接收缓冲器空 此位的功能和CAN_SR_RBS是等效的。所以CAN_CMR_RRB可以临时清除该位。如果执行释放命令后数据缓冲器中还有可用信息，该位又会被重新置位，直至缓冲器为空。

5.2.4.5 CAN_IE 寄存器

地址	位	名称	R/W	复位值	描述
0x10	7	CAN_IF_BEIE	R/W	0x0	总线错误中断使能 0x1：使能 0x0：不使能
	6	CAN_IF_ALIE	R/W	0x0	仲裁丢失中断使能 0x1：使能 0x0：不使能
	5	CAN_IF_EPIE	R/W	0x0	消极出错中断使能 0x1：使能 0x0：不使能
	4	CAN_IF_WUIE	R/W	0x0	唤醒中断使能 0x1：使能 0x0：不使能
	3	CAN_IF_DOIE	R/W	0x0	数据即将溢出中断使能 0x1：使能 0x0：不使能
	2	CAN_IF_EIE	R/W	0x0	错误中断使能 0x1：使能 0x0：不使能
	1	CAN_IF_TIE	R/W	0x0	发送中断使能 0x1：使能 0x0：不使能
	0	CAN_IF_RIE	R/W	0x0	接收中断使能 0x1：使能 0x0：不使能

5.2.4.6 CAN_BTR0 寄存器

地址	位	名称	R/W	复位值	描述
0x18	7:6	CAN_BTR0_SJW	R/W	0x0	同步跳转宽度 为了补偿在不同总线控制器的时钟振荡器之间的相位偏移，任何总线控制器必须在当前传送的相关信号边沿重新同步。同步跳转宽度定义了每一位周期可以被重新同步缩短或延长的时钟周期的最大数目。 同步跳转宽度时间 = 时钟预设值*（2 *CAN_BTR0_SJW [1]+ CAN_BTR0_SJW [0]+1)。
	5:0	CAN_BTR0_BRP	R/W	0x0	时钟预设值 CAN通信时钟周期 = 2*CAN模块时钟周期*（ CAN_BTR0_BRP+1）。

5.2.4.7 CAN_BTR1 寄存器

地址	位	名称	R/W	复位值	描述
0x1C	7	CAN_BTR1_SAM	R/W	0x0	内部采样次数 0x1：采样3次，取中间的一次为真实数据，提高可靠性 0x0：采样1次
	6:4	CAN_BTR1_TSEG2	R/W	0x0	时间段2，用于配置波特率
	3:0	CAN_BTR1_TSEG1	R/W	0x0	时间段1，用于配置波特率 CAN波特率 = 1/（2*CAN模块时钟周期*（1+ CAN_BTR0_BRP）*（3+ CAN_BTR1_TSEG1+ CAN_BTR1_TSEG2))

5.2.4.8 CAN_SLPAK 寄存器

地址	位	名称	R/W	复位值	描述
0x20	7:1	保留	——	——	保留
	0	CAN_SLPAK	RO	0x0	睡眠模式确认状态 0x1 ：已经进入睡眠模式 0x0 ：正常运行 该标记显示CAN模块是否已经进入睡眠模式。它用作CAN_CMR_SPM睡眠模式请求的握手标志。当CAN_CMR_SPM = 1，CAN_SLPAK = 1 时，睡眠模式是有效的。根据CAN_WUP_ENABLE设置，如果在处于睡眠模式检测到CAN总线有信号，CAN将清除该标志。清除CAN_CMR_SPM位也将复位CAN_SLPAK位。

5.2.4.9 CAN_WUP 寄存器

地址	位	名称	R/W	复位值	描述
0x24	7:2	保留	——	——	保留
	1	CAN_WUP_MODE	R/W	0x0	自身唤醒滤波 0x1：只有在 CAN 总线上的显性脉冲长度为大于 2us 时才唤醒。 0x0：CAN 被总线上的任意显性信号唤醒
	0	CAN_WUP_ENABLE	R/W	0x0	自身唤醒使能 0x1：使能 0x0：禁止

5.2.4.10 CAN_ALC 寄存器

地址	位	名称	R/W	复位值	描述
0x2C	7:5	保留	——	——	保留
	4:0	CAN_ALC_CODE	RO	0x0	仲裁丢失信息 仲裁丢失时，会产生相应的仲裁丢失中断。同时，位流的当前位位置被捕捉送入仲裁丢失捕捉寄存器。一直到软件读这个值，寄存器中的内容都不会变。直到仲裁丢失捕捉寄存器被读一次之后，新的仲裁丢失中断才有效。

CAN_ALC_CODE[4] ~ CAN_ALC_CODE[0]					说明
0	0	0	0	0	仲裁丢失在识别码的 bit1
0	0	0	0	1	仲裁丢失在识别码的 bit2
0	0	0	1	0	仲裁丢失在识别码的 bit3
0	0	0	1	1	仲裁丢失在识别码的 bit4
0	0	1	0	0	仲裁丢失在识别码的 bit5
0	0	1	0	1	仲裁丢失在识别码的 bit6
0	0	1	1	0	仲裁丢失在识别码的 bit7
0	0	1	1	1	仲裁丢失在识别码的 bit8
0	1	0	0	0	仲裁丢失在识别码的 bit9
0	1	0	0	1	仲裁丢失在识别码的 bit10
0	1	0	1	0	仲裁丢失在识别码的 bit11
0	1	0	1	1	仲裁丢失在 SRTR 位
0	1	1	0	0	仲裁丢失在 IDE 位
0	1	1	0	1	仲裁丢失在识别码的 bit12
0	1	1	1	0	仲裁丢失在识别码的 bit13
0	1	1	1	1	仲裁丢失在识别码的 bit14
1	0	0	0	0	仲裁丢失在识别码的 bit15
1	0	0	0	1	仲裁丢失在识别码的 bit16
1	0	0	1	0	仲裁丢失在识别码的 bit17
1	0	0	1	1	仲裁丢失在识别码的 bit18
1	0	1	0	0	仲裁丢失在识别码的 bit19
1	0	1	0	1	仲裁丢失在识别码的 bit20
1	0	1	1	0	仲裁丢失在识别码的 bit21
1	0	1	1	1	仲裁丢失在识别码的 bit22
1	1	0	0	0	仲裁丢失在识别码的 bit23
1	1	0	0	1	仲裁丢失在识别码的 bit24
1	1	0	1	0	仲裁丢失在识别码的 bit25
1	1	0	1	1	仲裁丢失在识别码的 bit26
1	1	1	0	0	仲裁丢失在识别码的 bit27
1	1	1	0	1	仲裁丢失在识别码的 bit28
1	1	1	1	0	仲裁丢失在识别码的 bit29
1	1	1	1	1	仲裁丢失在 RTR 位

5.2.4.11 CAN_ECC 寄存器

地址	位	名称	R/W	复位值	描述
0x30	7:6	CAN_ECC_MODE	RO	0x0	错误类 0x3：其他错误 0x2：填充错误 0x1：格式错误 0x0：位错误
	5	CAN_ECC_DIR	RO	0x0	错误方向 0x1：接收时发生错误 0x0：发送时发生错误
	4:0	CAN_ECC_CODE	RO	0x0	错误码 具体见下表。

CAN_ECC_CODE[4] ~ CAN_ECC_CODE[0]					错误
0	0	0	1	1	帧开始
0	0	0	1	0	ID28-ID21
0	0	1	1	0	ID20-ID18
0	0	1	0	0	SRTR 位
0	0	1	0	1	IDE 位
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 位
0	1	1	0	1	保留位 1
0	1	0	0	1	保留位 0
0	1	0	1	1	数据长度代码
0	1	0	1	0	数据区
0	1	0	0	0	CRC 序列
1	1	0	0	0	CRC 定义符
1	1	0	0	1	应答通道
1	1	0	1	1	应答定义符
1	1	0	1	0	帧结束
1	0	0	1	0	中止
1	0	0	0	1	活动错误标志
1	0	1	1	0	消极错误标志
1	0	0	1	1	支配控制位误差
1	0	1	1	1	错误定义符
1	1	1	0	0	过载标志

5.2.4.12 CAN_EMLR 寄存器

地址	位	名称	R/W	复位值	描述
0x34	7:0	CAN_EMLR	R/W	0x60	错误报警限制值 发出错误报警达到的计数值。该寄存器只能在 CAN 置位模式下才能写。

5.2.4.13 CAN_RXERR 寄存器

地址	位	名称	R/W	复位值	描述
0x38	7:0	CAN_RXERR	R/W	0x00	接收错误计数值 正常工作模式只读，该寄存器只能在 CAN 置位模式下才能写。 如果发生总线关闭，该寄存器被初始化。总线关闭期间，写这个寄存器无效。

5.2.4.14 CAN_TXERR 寄存器

地址	位	名称	R/W	复位值	描述
0x3C	7:0	CAN_TXERR	R/W	0x00	发送错误计数值 正常工作模式只读，该寄存器只能在 CAN 置位模式下才能写。 如果发生总线关闭，该寄存器被初始化为 127 用于计算总线定义的最小时间（128 个总线空闲信号）。这段时间里读该计数器将反映出总线关闭恢复的状态信息。总线关闭期间，写这个寄存器无效。 在总线关闭期间，写访问该寄存器 0-254 会清除总线关闭标志。复位模式被清除后控制器会等待一个 11 位的连续隐性位（总线空闲）。向该寄存器写入 255 会导致总线关闭事件。如果在总线关闭恢复之前又进入复位模式，总线关闭保持有效且该技术值被锁定。

5.2.4.15 CAN_IDAR0/CAN_IDAR1/CAN_IDAR2/CAN_IDAR3 寄存器

地址	位	名称	R/W	复位值	描述
0x40 0x44 0x48	7:0	CAN_IDAR0 CAN_IDAR1 CAN_IDAR2	R/W	0x00	验收代码寄存器，详细见验收滤波章节说明。该寄存器只能在 CAN 置位模式下写操作。

0x4C		CAN_IDAR3			

5.2.4.16 CAN_IDMR0/CAN_IDMR1/CAN_IDMR2/CAN_IDMR3 寄存器

地址	位	名称	R/W	复位值	描述
0x50 0x54 0x58 0x5C	7:0	CAN_IDMR0 CAN_IDMR1 CAN_IDMR2 CAN_IDMR3	R/W	0x00	验收屏蔽寄存器，详细见验收滤波章节说明。该寄存器只能在 CAN 置位模式下写操作。

5.2.4.17 CAN_RMC 寄存器

地址	位	名称	R/W	复位值	描述
0x74	7:6	保留	——	——	保留
	5:0	CAN_RMC	RO	0x00	接收信息计数器 反映了接收缓存中可用的信息数目，其值每次接收时加 1，每次释放接收缓冲器减 1。每次复位后，该寄存器清 0。

5.2.4.18 CAN_ENABLE 寄存器

地址	位	名称	R/W	复位值	描述
0x78	7:1	保留	——	——	保留
	0	CAN_ENABLE	R/W	0x00	CAN 模块使能 0x1：使能 0x0：不使能 CAN 使能时选择外部 IO 口自动切换为 CAN 功能。在配置关闭该位之前要配置 CAN 进入置位模式，再次配置该位之后再配置 CAN 退出置位模式，CAN 才可正常开始工作。

5.2.4.19 CAN_CLRISR 寄存器

地址	位	名称	R/W	复位值	描述
0x80	7	CAN_CLRISR_BEI	R/W	0x00	写 1 清除总线中断状态 BEI，读出的总是为 0
	6	CAN_CLRISR_ALI	R/W	0x00	写 1 清除仲裁丢失中断状态 ALI，读出的总是为 0
	5	CAN_CLRISR_EPI	R/W	0x00	写 1 清除消极错误中断状态 EPI，读出的总是为 0
	4	CAN_CLRISR_WUPI	R/W	0x00	写 1 清除唤醒中断状态 WUPI，读出的总是为 0
	3	CAN_CLRISR_DOI	R/W	0x00	写 1 清除数据溢出中断状态 DOI，读出的总是为 0
	2	CAN_CLRISR_EI	R/W	0x00	写 1 清除错误中断状态 EI，读出的总是为 0
	1	CAN_CLRISR_TI	R/W	0x00	写 1 清除发送中断状态 TI，读出的总是为 0
	0	保留	——	——	保留

5.2.4.20 CAN_CLRECC 寄存器

地址	位	名称	R/W	复位值	描述
0x84	7:1	保留	——	——	保留
	0	CAN_CLRECC	R/W	0x00	写 1 清除错误代码捕获寄存器 CAN_ECC，读出的总是为 0

5.2.4.21 CAN_FRCTL 寄存器

地址	位	名称	R/W	复位值	描述
0x40	7	CAN_FRCTL_FF	R/W	——	帧格式 0x1：扩展帧 0x0：标准帧
	6	CAN_FRCTL_RTR	R/W	0x00	远程请求 0x1：远程帧 0x0：数据帧
	5:4	保留	——	——	保留
	3:0	CAN_FRCTL_DLC	R/W		数据长度 在远程帧发送开始时由于 RTR 位被置位（远程），数据长度代码是不被考虑的。这使接收/发送的数据字节数目为 0。如果有两个 CAN 控制器使用同一个识别码同 时启动远程帧传送，数据长度代码必须正确说明以避免总线错误。 为了兼容，大于 8 的数据长度代码是不可用的。如果大于 8 将以 8 个字节计。 写该寄存器为写入要发送的数据，读该寄存器为读出接收的数据。

5.2.4.22 CAN_ID0/CAN_ID1/CAN_ID2/CAN_ID3 寄存器

地址	位	名称	R/W	复位值	描述
0x44 0x48 0x4C 0x50	7:0	CAN_ID0 CAN_ID1 CAN_ID2 CAN_ID3	R/W	0x00	CAN ID 号 写该寄存器为写入要发送的数据，读该寄存器为读出接收的数据。

5.2.4.23 CAN_DATA0~ CAN_DATA7 寄存器

地址	位	名称	R/W	复位值	描述
0x54 0x58 0x5C 0x60 0x64 0x68	7:0	CAN_DATA0 CAN_DATA1 CAN_DATA2 CAN_DATA3 CAN_DATA4 CAN_DATA5	R/W	0x00	CAN ID 数据 写该寄存器为写入要发送的数据，读该寄存器为读出接收的数据。

0x6C 0x70		CAN_DATA6 CAN_DATA7			

5.2.4.24 CAN_SWU_FILT 寄存器

地址	位	名称	R/W	复位值	描述
0x88	7:1	保留	——	——	保留
	0	CAN_SWU_FILT	R/W	0x00	系统睡眠模式下，CAN Rx 端口滤波使能 0x1：滤波 0x0：不滤波 该滤波通过模拟端口直接滤波，最大 1.5us。

5.2.5 程序范例

```
//主函数：
int main(void)
{
    gpio_init(GPIOG,GPIO_MODE_OUT,GPIO_PIN_5);
    gpio_bit_reset(GPIOG,GPIO_PIN_5);        //CAN_STB 拉低
    can_boot_config();//CAN 初始化配置，包括波特率在内的数据结构体的初始化和帧格式选择，数据帧的 id、长度、数据内容等的初始化
    can_send_frame();                        //CAN 发送帧数据

    can_sleep_config(1,0,ENABLE);            //设置 CAN 睡眠
    can_int_enable(CAN_IE_WUIE);             //唤醒中断使能

    if(can_sleep_request())                  //CAN 睡眠请求
    {
        gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);
        gpio_bit_set(GPIOE,GPIO_PIN_5); //点灯
    }

    sleep_deep();                            //系统进入睡眠模式
    CAN_FILTER_EN = 0x01;                  //设置 Rx 端口滤波唤醒

//  rtc_init(RTC_ENABLE | RTC_CLK_SEL_32K | RTC_INT_ENABLE | RTC_TRG_CLK_DIV(0),0x7D00);
    //RTC 初始化
}
//子函数：
/***************CAN 初始化参数设置*****************/
```

```
void can_boot_config()
{
        /*can parameter initialize*/
    can_parameter_struct can_parameter_init =
    {
        CAN_NOMARL,                 //工作模式
        CAN_BTR0_SJW_SET(0),            //同步跳转宽度
        CAN_BTR0_BRP_SET(1),            //CAN 波特率 = CAN 模块时钟频率 /
(2*(1+CAN_BTR0_BRP_SET(x))*(3+CAN_BTR1_TSEG1_SET(n)+CAN_BTR1_TSEG2_SET(m)))
        CAN_BTR1_TSEG1_SET(3),      //时间段，用于配置波特率
    //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),      //时间段，用于配置波特率
        CAN_BTR1_SAM_SET(1),            //内部采样次数
        100,                            //错误报警限制值
        CAN_WUIE_ENABLE | CAN_ERR_ALL_ENABLE | CAN_RIE_ENABLE | CAN_TIE_ENABLE,
                        //中断使能
        CAN_WU_NVIC_ENABLE | CAN_ERR_NVIC_ENABLE | CAN_RX_NVIC_ENABLE | CAN_TX_NVIC_ENABLE
            //外部中断使能
    };
        /*CAN 滤波器参数初始化*/
    can_filter_parameter_struct can_filter_parameter =
    {
        CAN_DOUBLE_FILTER,      //选择两个验收滤波器
        CAN_EXTENDED_FRAME, //标准帧
        CAN_DATA_FRAME,         //数据帧
        ENABLE,                 //RTR 位屏蔽使能
        /* 单滤波设置*/
        0x12345678,                 //单滤波 id,标准帧滤波:11 位有效;扩展帧滤波:29 位有效
        0xff,                           //滤波器验收数据 0,用于标准帧滤波
        0xff,                           //滤波器验收数据 1,用于标准帧滤波
        0xffff,                     //单滤波屏蔽验收 id,标准帧滤波:11 位有效;扩展帧滤波:29 位
有效
        0xff,                           //滤波器屏蔽验收数据 0,用于标准帧滤波
        0xff,                           //滤波器屏蔽验收数据 1,用于标准帧滤波
        /* 双滤波设置 */
        0x12345678,                 //双滤波 id0,标准帧滤波:11 位有效;扩展帧滤波:29 位 ID  有
效验收位 ID28~13
        0x98765432,                 //双滤波 id1,标准帧滤波:11 位有效;扩展帧滤波:29 位 ID  有
效验收位 ID28~13
        0xff,                           //滤波器验收数据 0,用于标准帧滤波
        0xffff,                     //双滤波屏蔽验收 id0,标准帧滤波:11 位有效;扩展帧滤波:16 位
有效
```

```
        0xffff,                         //双滤波屏蔽验收 id1,标准帧滤波:11 位有效;扩展帧滤波:16 位
有效
        0xff                            //滤波器屏蔽验收数据 0,用于标准帧滤波
    };
    can_clock_sel(CAN_XTAL_CLOCK);//选择外部晶振作为时钟源
    can_init(can_parameter_init,can_filter_parameter)       ;//CAN 寄存器初始化配置
    can_transmit_data.tx_ff = CAN_STANDARD_FRAME;   //标准帧,扩展帧
    can_transmit_data.tx_ft = CAN_DATA_FRAME;           //数据帧
    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 传输数据
}
/***************CAN 发送帧数据******************/
void can_send_frame()
{
    can_transmit_data.tx_ff = can_receive_data.rx_ff;       //标准帧,扩展帧
    can_transmit_data.tx_ft = can_receive_data.rx_ft;           //数据帧
    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 传输数据
    can_int_flag_clr(CAN_IF_TI);
    can_transmit_request();                              //CAN 数据传输请求
    while((can_interrupt_flag_get() & CAN_IF_TI) == 0x00);  //等待发送完成
    can_int_flag_clr(CAN_IF_TI);                         //清除发送完成中断标志位
}
/***************CAN 自身睡眠唤醒配置*****************/
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;
    }                                                    //设置自身唤醒使能
    if(wakeup_mode){
        CAN_WUP |= CAN_WUP_MODE;
    }else{
        CAN_WUP &= ~CAN_WUP_MODE;
    }                                                    //设置自身唤醒滤波
    if(filter_en)
        CAN_FILTER_EN = 0x01;
    else
        CAN_FILTER_EN = 0x00;                            //设置睡眠模式下 Rx 端口滤波使能
}
/***************CAN 中断使能*****************/
void can_int_enable(uint8_t intstate)
{
    CAN_IE |= intstate;                                  //设置 CAN 中断使能
}
/***************CAN 自身睡眠请求*****************/
ErrorStatus can_sleep_request()
{
    uint32_t timeout = 400;
    CAN_CMR |= CAN_CMR_SPM;                      //自身睡眠请求
    while(!CAN_SLPAK){                               //等待进入睡眠
        timeout --;
```

```
        if(timeout == 0){
            return ERROR;
        }
    }
    return SUCCESS;
}
/***************系统睡眠*****************/
void sleep_deep()
{
    SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
    __WFI();                                        //使系统进入睡眠模式
}
```

5.3 PWM

本模块主要实现输出频率占空比可调的 PWM 波形功能和输入捕获功能，同时也可作为计数器使用。

5.3.1 主要特性

1. 16位向上或向下计数器；
2. 支持最多6路PWM通道；
3. 每个通道支持输出比较或边缘对齐PWM模式波形输出，支持设置、清除、切换输出比较操作，PWM输出的极性可选；
4. 每个通道支持上升、下降或任何边输入捕捉触发；
5. 在所有通道上，支持中央对齐脉冲宽度调制；
6. 定时器时钟源，可选为系统时钟、晶振时钟或外部时钟输入，支持时钟 1/2/4/8/16/32/64/ 128分频；
7. 支持每个通道一个功能中断以及计数器溢出中断；
8. 每个通道口和外部时钟口单独使能选通，支持PWM计数工作时不使能通道口或外部时钟口，它们均可单独配置作为GPIO功能使用；
9. 支持支持PWM计数触发ADC采样；

5.3.2 功能说明

本节详细介绍 BF7006AMXX PWM 主要功能。

5.3.2.1 计数器工作模式

模块内含一个 16 位计数器，计数器时钟源可选择外部输入时钟或系统时钟或晶振时钟，其中系统时钟或晶振时钟由系统配置，本模块只配置选择外部输入时钟或系统时钟，见寄存器 PWM_SC_CLKS_SEL 描述。时钟可以配置预分频（1/2/4/8/16/32/64/128）。

模块工作控制时钟始终是系统时钟，当选择计数时钟为外部输入时钟时，实际是由系统时钟采样输入时钟信号来控制计数。中断信号由系统时钟产生，不受外部输入时钟影响。

计数器有两种计数模式。选择中央对齐 PWM 时，计数器以向上/向下计数模式运行。 否则，计数器作为简单的向上计数器运行。作为向上计数器时，定时器计数器从 0x0000 计数到其终端计数，然后从 0x0000 重新开始。终端计数为 0xFFFF 或 PWM_MOD 中的模数值。

选择中央对齐 PWM 模式时，计数器从 0x0000 向上计数到其终端计数，然后向下计数到 0x0000，在那里又开始向上计数。0x0000 和终端计数值均为正常长度计数（一个定时器时钟周期长度）。在这种模式下，定时器溢出标记 PWM_SC_TOF 在终端计数周期结束时被设置（当计数器变为下一个更低计数值时）。

计数器溢出会产生溢出中断 PWM_SC_TOF。在没有模数限制，也不在 PWM_SC_CPWMS=1 模式时，16 位定时器计数器从 0x0000 计数到 0xFFFF，然后在下一个计数时钟周期内溢出为 0x0000。在从 0xFFFF 过渡到 0x0000 时，PWM_SC_TOF 被设置。设置了模数限制时，PWM_SC_TOF 在从模数寄存器中设置的值向 0x0000 过渡时设置。当处于中央对齐 PWM 模式时（PWM_SC_CPWMS=1），PWM_SC_TOF 标记在计数器到达模数寄存器中设置的计数值结束改变方向时被设置（也就是说从模数寄存器中设置的值向下一个更低计数值过渡时）。这与 PWM 周期结束对应（0x0000 计数值与周期中央对应）。

计数器可随时通过将任何值写入 PWM_CNT 寄存器来手动复位。

无论以下哪种通道工作模式，计数器均会自动重载计数，不会自行停止，直到配置 PWM_SC_CLK_SEL=0 停止。如果模式寄存器值为 0x0000，则计数器进入自由运行模式，会计数到 0xffff。此外均为模式计数方式，即计数到模式寄存器值后重新开始计数。

5.3.2.2 输入捕获模式

在非中央对齐模式下，每个通道工作模式单独可配。中央对齐模式下，所有通道默认输出 PWM 波形功能。

利用输入捕捉功能，可捕捉外部事件发生的时间。当输入捕捉通道的管脚上发生活动边沿时，可将计数

器的内容锁入到通道值寄存器中。上升边沿、下降边沿或任何边沿都可选择作为触发输入捕捉的使能边沿。

输入捕捉的信号经过同步（跨时域系统时钟打两拍）后再判断实现捕获功能。

在输入捕捉模式下，通道值寄存器为只读。

输入捕捉事件（检测到选定边沿）后会设置标记位。该位可选择生成 CPU 中断请求。

5.3.2.3 输出比较模式

利用输出比较功能，可生成具有可编程位置、极性、持续时间和频率的定时脉冲。当计数器达到输出比较通道的通道值寄存器中的值时，可设置、清除或切换通道管脚，即可选择用于强制将管脚设置为零或 1、反转管脚电平。

刚完成选择输出比较切换模式时，管脚上的以前的值一直被驱动，直到发生下一个输出比较事件，然后管脚被切换。

中断标志在每次计数器与通道值寄存器中的 16 位值匹配时被设置。

5.3.2.4 边沿对齐模式

这类 PWM 输出使用计数器的向上计数模式（PWM_SC_CPWMS=0）；当其他通道被配置用于输入捕捉或输出比较功能时，也可使用它。这个 PWM 信号的周期由模数寄存器 PWM_MOD 的值加 1 确定。占空比由通道寄存器 PWM_CxV 中的设置确定。这个 PWM 信号的极性由 PWM_CxSC_ELS 控制位中的设置确定。0%和 100%的占空比都是可能的。

脉冲宽度 = PWM_CxV

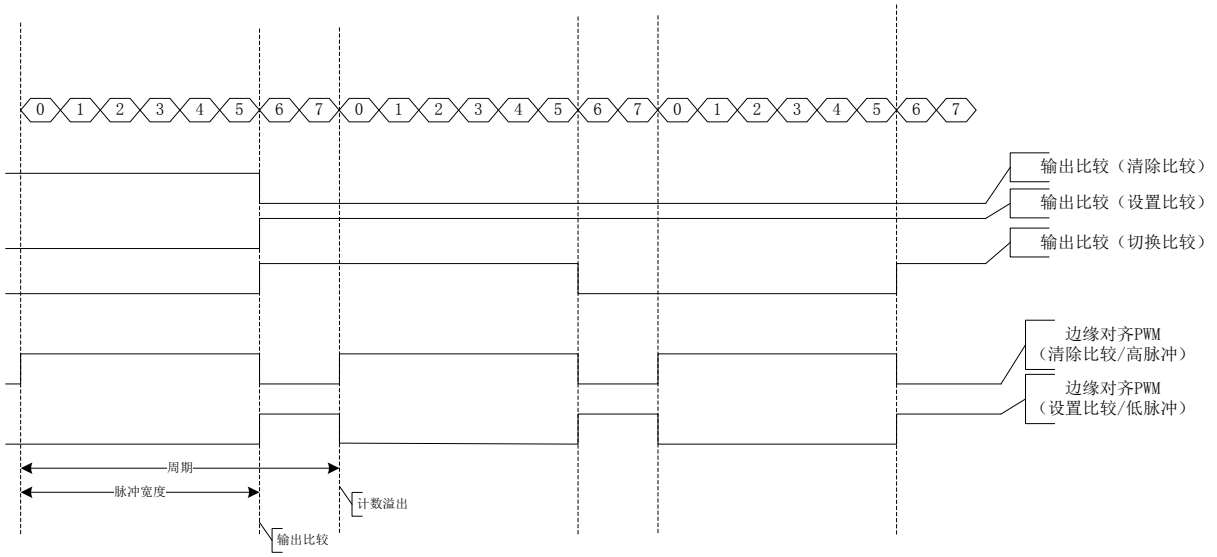
周期 = PWM_MOD + 1

通道寄存器中的输出比较值决定 PWM 信号的脉冲宽度（占空比）。模数溢出和输出比较间的时间间隔为脉冲宽度。如果 PWM_CxSC_ELS=0，计数器溢出强迫 PWM 信号进入高态；而输出比较强制 PWM 信号进入低态。如果 PWM_CxSC_ELS=1，计数器溢出强制 PWM 信号进入低态；而输出比较强制 PWM 信号进入高态。

当通道值寄存器被设为 0x0000 时，占空比为 0%。通过将通道计数器 PWM_CxV 设置为大于模数设置的值可实现 100%的占空比。

中断标志在计数器与标志占空比周期结束的通道值寄存器匹配时被设置。

详细计数及波形输出见下面示意图：



5.3.2.5 中央对齐模式

16 位模数寄存器值的两倍设置 PWM 输出周期，而通道值寄存器设置一半占空比持续时间。定时器计数器向上计数，直到达到模数值，然后向下计数直到达到 0。向下计数的情况下，计数与通道值寄存器匹配时，PWM 输出进入有效状态。向上计数的情况下，计数与通道值寄存器匹配时，PWM 输出进入无效状态。这类 PWM 信号被称为中央对齐，因为所有通道的活动占空比的中心与计数值 0 对齐。

这类 PWM 输出使用计数器的向上/向下计数模式。PWM_CxV 中的输出比较值决定 PWM 信号的脉冲宽度（占空比），而 MOD 中的值决定周期。PWM_CxSC_ELS 将决定 PWM_SC_CPWMS 输出的极性。

脉冲宽度 = 2 x PWM_CxV

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

如果通道值寄存器PWM_CxV为零，占空比将为0%。如果PWM_CxV是正值大于模数设置，占空比将为100%，因为占空比比较将不会发生。

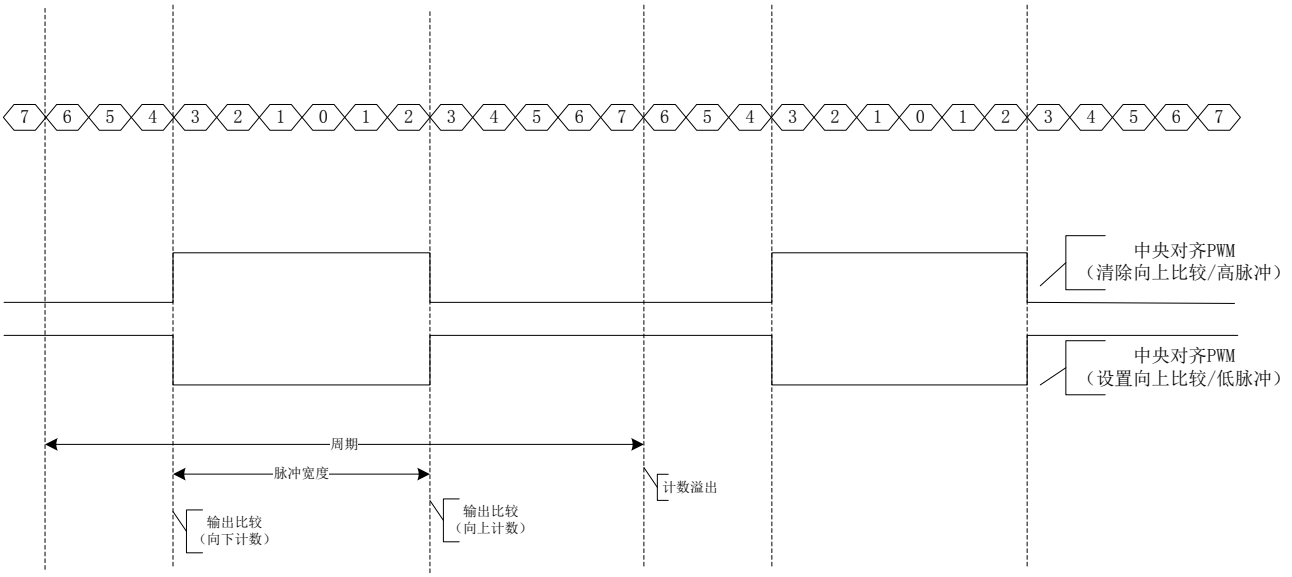
通道寄存器中的输出比较值决定 PWM_SC_CPWMS 信号的脉冲宽度（占空比）。如果 PWM_CxSC_ELS=0，当向上计数时发生数值比较会强制 PWM_SC_CPWMS 输出信号进入低态；当向下计数时发生数值比较会强制输出进入高态。计数器达到 PWM_MOD 中的模数设置后开始向下计数直到 0，再开始向上计数到模数设置。这将周期设置为 PWM_MOD 的两倍。

当计数器以向上/向下计数模式运行时，输入捕捉、输出比较和边缘对齐 PWM 功能没有意义。因此这意味着当 PWM_SC_CPWMS=1 时，所有使能通道必须用于 PWM_SC_CPWMS 模式中。

通道输出比较中断标志在占空比周期时间开始和结束时（定时器计数器与通道值寄存器匹配时）被设置。PWM_SC_TOF 在计数器方向在终端计数（模数寄存器中的值）结束时从向上计数变为向下计数时被设置。在这种情况下，PWM_SC_TOF 与 PWM 周期结束相对应。

注：在中央对齐模式下，计数过程中更新计数周期会造成无效电平的长度前后不一致，有效电平长度不变，即以 0 为中心对齐的电平长度不变。

详细计数及波形输出见下面示意图：



5.3.3 寄存器列表

PWM 模块地址区间：0x5006_0000~0x5006_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	PWM_SC	R/W	计数器控制寄存器	0x00
0x04	PWM_CNT	R/W	计数值寄存器	0x0000
0x08	PWM_MOD	R/W	计数器模数寄存器	0xFFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x0c	PWM_C0SC	R/W	通道 0 控制寄存器	0x00
0x10	PWM_C0V	R/W	通道 0 计数值寄存器	0x0000
0x14	PWM_C1SC	R/W	通道 1 控制寄存器	0x00
0x18	PWM_C1V	R/W	通道 1 计数值寄存器	0x0000
0x1c	PWM_C2SC	R/W	通道 2 控制寄存器	0x00
0x20	PWM_C2V	R/W	通道 2 计数值寄存器	0x0000
0x24	PWM_C3SC	R/W	通道 3 控制寄存器	0x00
0x28	PWM_C3V	R/W	通道 3 计数值寄存器	0x0000
0x2c	PWM_C4SC	R/W	通道 4 控制寄存器	0x00
0x30	PWM_C4V	R/W	通道 4 计数值寄存器	0x0000
0x34	PWM_C5SC	R/W	通道 5 控制寄存器	0x00
0x38	PWM_C5V	R/W	通道 5 计数值寄存器	0x0000
0x3C	PWM_ADCV	R/W	ADC 触发计数值寄存器	0x0000

5.3.4 寄存器详细描述

5.3.4.1 PWM_SC 寄存器

地址	位	名称	R/W	复位值	描述
0x00	7	PWM_SC_TOF	R/W	0x0	计数器定时溢出标志位 0x1：溢出 0x0：未溢出 写0清除该位。
	6	PWM_SC_TOIE	R/W	0x0	计数器定时溢出中断使能 0x1：打开中断使能 0x0：关闭（可用于轮询）
	5	PWM_SC_CPWMS	R/W	0x0	计数方式选择 0x1：表示向上计数后向下计数（中央对齐） 0x0：表示向上计数
	4:3	PWM_SC_CLK_SEL	R/W	0x0	预分频器输入的计数器时钟源 0x3：表示选择外部输入时钟计数 0x2：表示选择系统时钟计数 0x1：表示选择系统时钟计数 0x0：表示关闭计数器
	2:0	PWM_SC_CLK_DIV	R/W	0x0	计数时钟的2的 PWM_SC_CLK_DIV次幂分频

5.3.4.2 PWM_CNT 寄存器

地址	位	名称	R/W	复位值	描述
0x04	15:0	PWM_CNT	R/W	0x0000	计数值寄存器 记录当前计数值，系统复位或写该寄存器均清零计数器。

5.3.4.3 PWM_MOD 寄存器

地址	位	名称	R/W	复位值	描述
0x08	15:0	PWM_MOD	R/W	0xFFFF	计数器模数寄存器 配置计数器周期，系统复位或写PWM_CNT寄存器均清零计数器。写入模数寄存器前需关闭计数器或复位计数器，以避免造成首次计数器溢出发生时间的混乱。

5.3.4.4 PWM_CxSC（x=0~5）寄存器

地址	位	名称	R/W	复位值	描述
0x0C 0x14 0x1C 0x24 0x2C 0x34	7	PWM_CxSC_IF	R/W	0x0	通道x中断状态 0x1：表示发生输入捕获或输出比较事件 0x0：表示未发生中断 通道x用作输入捕捉通道的情况下，通道x管脚上发生有效触发边沿时会设置这个位。通道x为输出比较或边缘对齐/中央对齐PWM通道时，计数器寄存器中的值与通道x计数值寄存器中的值匹配时会设置该位。通道x用作边缘对齐/中央对齐PWM通道而占空比被设置为0%或100%的情况下，匹配时将不设置该位。
	6	PWM_CxSC_IE	R/W	0x0	通道x中断使能 0x1：打开 0x0：关闭（用于轮询）
	5:4	PWM_CxSC_MS	R/W	0x0	通道x模式选择（见下表）
	3:2	PWM_CxSC_ELS	R/W	0x0	通道x边沿/电平极性选择（见下表）
	1:0	保留	——	——	保留

模式与极性选择				
PWM_SC_CPWMS	PWM_CxSC_MS	PWM_CxSC_ELS	模式	功能
x	x	00	无效	
1	xx	10	中央对齐 PWM	High-true 脉冲（清除向上比较输出）
		x1		Low-true 脉冲（设置向上比较输出）
0	00	01	输入捕获模式	仅在上升边沿捕捉
		10		仅在下降边沿捕捉
		11		在上升或下降边沿捕捉
	01	01	输出比较模式	切换比较输出
		10		清除比较输出
		11		设置比较输出
	1x	10	边缘对齐 PWM	High-true 脉冲（清除比较输出）
		x1		Low-true 脉冲（设置比较输出）

5.3.4.5 PWM_CxV（x=0~5）寄存器

地址	位	名称	R/W	复位值	描述
0x10 0x18 0x20 0x28 0x30 0x38	15:0	PWM_CxV	R/W	0x0	通道计数值寄存器 记录输入捕获的计数值或配置输出比较的计数值。 在输入捕捉模式下，该寄存器记录捕捉到有效边沿时的计数器数值，向通道值寄存器的任何写入操作在输入捕捉模式下将被忽视。 在输出比较或 PWM 模式下，该寄存器用于配置每个通道比较输出的计数值。 计数器不在工作模式时，寄存器在写操作后直接更新。否则（计数器在工作状态中），寄存器在写操作后要等到计数器从 PWM_MOD-1 变为 PWM_MOD 时更新。

5.3.4.6 PWM_ADCV 寄存器

地址	位	名称	R/W	复位值	描述
0x3C	15:0	PWM_ADCV	R/W	0x0000	ADC触发计数值寄存器 配置ADC的PWM触发方式下，触发条件满足的计数值。 当 ADC 选择内部 PWM 硬件触发方式时，该寄存器才有意义。用于配置触发条件计数值，当计数值达到此数值时送出标志位给 ADC 采样，只有清除溢出中断操作（写 PWM_SC_TOF=0）才能清除该标志位。 在 PWM_SC_CLK_SEL =0 时（计数器不在工作状态），寄存器在写操作后直接更新。否则（计数器在工作状态中），寄存器在写操作后要等到计数器从 PWM_MOD-1 变为 PWM_MOD 时更新。根据更新一致性机制，当配置计数器控制寄存器配为不同值时会清除当前更新。

5.3.5 程序范例

```
uint8_t aaa,bbb;//PWM 模式选择参数
//主函数：
int main(void)
{
    /*PWM 初始化*/
    aaa = 2;bbb = 2;
    /*PWM 初始化，包括 PWM_INT_ENABLE（PWM 中断初始化）、PWM_CPWMS_UNIDIR（选择向上计数）、
PWM_CLK_SEL_SYS（选择系统时钟作为时钟源）、PWM_CLK_DIV_32（时钟 32 分频）、2000（周期为 2ms）*/
    pwm_init(PWM_INT_ENABLE | PWM_CPWMS_UNIDIR | PWM_CLK_SEL_SYS | PWM_CLK_DIV_32,2000);
    /*PWM 通道初始化，包括 PWM_CHn_INT_ENABLE（PWM 通道中断使能）、PWM_CHn_MS(aaa)（设置边沿对
齐）、PWM_CHn_ELS(bbb)（清除比较输出）、200（占空比为 10%，即 200/2000，500、800、1100...以此类
推）*/
    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);
}
//子函数：
/*****************PWM 初始化*********************/
void pwm_init(uint8_t pwm_sc,uint16_t pwm_mod)
{
    PWM_SC = pwm_sc;                                    //PWM 控制寄存器配置
    PWM_MOD = pwm_mod;                                 //PWM 模数寄存器配置
    if(pwm_sc & PWM_SC_TOIE){
        NVIC_EnableIRQ(PWM_TOF_IRQN);
    }else{
        NVIC_DisableIRQ(PWM_TOF_IRQN);
    }                                                   //定时溢出中断配置
}
/****************PWM 通道初始化******************/
void pwm_ch0_init(uint8_t pwm_ch_sc,uint16_t pwm_ch_cnt)
{
    PWM_COSC = pwm_ch_sc;                               //通道 0 控制寄存器配置
    PWM_COV = pwm_ch_cnt;                               //通道 0 计数值配置
    if(pwm_ch_sc & PWM_CnSC_IE){
        NVIC_EnableIRQ(PWM_CH0_IRQn);
    }else{
        NVIC_DisableIRQ(PWM_CH0_IRQn);
    }                                                   //通道 0 中断配置
```

```
}
/*************PWM 计数溢出中断服务函数****************/
void PWM_TOF_IRQHandler(void)
{
    pwm_tof_clr();                                  //清除计数溢出中断标志
}
/***************PWM 通道中断服务函数******************/
void PWM_CH0_IRQHandler(void)
{
    pwm_ch0_if_clr();                                //清除通道 0 中断标志位
}
```

5.4 TIMER

BF7006AMXX 包含 2 个基本定时器 Timer0 和 Timer1，提供应用基本的定时计数需求。

5.4.1 主要特性

1. 16位向上计数器；
2. 计数时钟可选择系统时钟分频、内部32KHz时钟；
3. 支持自动重载模式；

5.4.2 功能说明

TIMER 模块起定时作用，其内部主要结构为一个 16 位的计数器，通过对输入时钟的计数达到定时的功能，TIMER 的计数原则为累加计数，当计数器计数到设定值时产生中断；TIMER 的计数时钟可选择系统时钟和内部 RC 时钟；TIMER 有两种工作模式：单次定时模式和自动重装载模式，无论哪种模式，计时完成均会产生中断。

5.4.3 寄存器列表

TIMER0 模块地址区间：0x500B_0000~0x500B_3FFF

TIMER1 模块地址区间：0x500B_4000~0x500B_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	TIMER_CFG	R/W	计数器控制寄存器	0x00
0x04	TIMER_MOD	R/W	计数器模数寄存器	0x0000
0x08	TIMER_CNT	R/W	计数值寄存器	0x0000

5.4.4 寄存器详细描述

5.4.4.1 TIMER_CFG 寄存器

地址	位	名称	R/W	复位值	描述
0x00	7	保留	——	——	保留
	6	TIMER_CFG_IF	R/W	0x0	中断标志位，写 1 清零 关闭 TIMER 不能清除该位。
	5	TIMER_CFG_IE	R/W	0x0	中断使能 0x1：使能 TIMER 中断 0x0：关闭 TIMER 中断
	4:3	TIMER_CFG_CLK_DIV	R/W	0x0	TIMER 分频（仅在 TIMER 选择 sys_clk 时使用） 0x3：系统时钟 8 分频 0x2：系统时钟 4 分频 0x1：系统时钟 2 分频 0x0：系统时钟
	2	TIMER_CFG_CLK_SEL	R/W	0x0	TIMER 时钟选择寄存器 0x1：选择 RC32KHz 0x0：选择系统时钟
	1	TIMER_CFG_RLD	R/W	0x0	TIMER 自动重载使能寄存器 0x1：自动重载模式 0x0：手动模式
	0	TIMER_CFG_EN	R/W	0x0	TIMER 计数使能寄存器 0x1：开启 0x0：停止 在手动重载模式下会在定时完成后硬件自动清零该寄存器，扫描过程中配置该寄存器会重新计数。

5.4.4.2 TIMER_MOD 寄存器

地址	位	名称	R/W	复位值	描述
0x04	15:0	TIMER_MOD	R/W	0x0000	TIMER溢出值配置寄存器，扫描过程中配置该寄存器会重新计数。

5.4.4.3 TIMER_CNT 寄存器

地址	位	名称	R/W	复位值	描述
0x08	15:0	TIMER_CNT	RO	0x0000	TIMER计数寄存器

5.4.5 程序范例

```
//主函数:
int main(void)
{
    gpio_init(GPIOB,GPIO_MODE_OUT,GPIO_PIN_6);  //PB6 设置为输出
    gpio_bit_set(GPIOB,GPIO_PIN_6);             //PB6 输出高电平
    gpio_bit_reset(GPIOB,GPIO_PIN_6);               //PB6 输出低电平
    /*定时器初始化，包括 TIMERx（定时器选择，可选择定时器 0 或定时器 1）、IMER_INT_ENABLE（定时器中断使能）、IMER_CLK_SYS_DIV2（时钟 2 分频）、TIMER_AUTO_RLD_ENABLE（自动重装载使能）、TIMER_ENABLE（定时器使能）、0x3E80（定时器溢出值，0x3E80 即 16000，每计满 16000 产生一次溢出中断，按时钟频率计算则每 1ms 产生一次中断）*/
    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  \
        );
}
//子函数:
/*****************定时器初始化*******************/
void timer_init(uint32_t timerx,uint8_t timer_cfg,uint16_t timer_mod)
{
    TIMER_CFG(timerx) = timer_cfg;              //配置定时器控制寄存器
    TIMER_MOD(timerx) = timer_mod;              //配置定时器模数寄存器
    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);
    }                                           //定时器中断使能配置
}
uint8_t timer0_f = 0;   //电平翻转状态位
uint16_t timer0_d = 0;  //电平翻转延时参数
/*****************定时器 0 中断服务函数*******************/
void TIMER0_IRQHandler(void)
{
    timer_intflag_clr(TIMER0);                  //清除中断标志位
    timer0_d++;
    if(timer0_d == 500)
    {
        timer0_d = 0;
        timer0_f = ~timer0_f;                   //电平延时翻转
    }
    if(timer0_f)
        gpio_bit_reset(GPIOB,GPIO_PIN_6);
    else
        gpio_bit_set(GPIOB,GPIO_PIN_6);         //LED 灯闪烁
}
/*****************定时器 1 中断服务函数*******************/
void TIMER1_IRQHandler(void)
{
    timer_intflag_clr(TIMER1);                  //清除中断标志位
        timer0_f = ~timer0_f;
    if(timer0_f)
        gpio_bit_reset(GPIOB,GPIO_PIN_6);
    else
        gpio_bit_set(GPIOB,GPIO_PIN_6);         //电平翻转
}
```

5.5 RTC

RTC为32位计数器。该模块可用于计时计数或任务调度功能。此外，它可以提供周期性叫醒服务而不需要外部组件。

5.5.1 主要特性

1. 32 位向上计数器；
2. 计数时钟可选择 1KHz 和 32KHz 和外部晶振时钟的 32 分频；
3. 支持睡眠模式下 RTC 计时唤醒系统；
4. 支持 RTC 硬件计时触发 ADC 采样；

5.5.2 功能说明

本节详细介绍 BF7006AMXX RTC 主要功能。

5.5.2.1 基本功能

RTC支持32位向上计数，同时支持系统低功耗下RTC唤醒和触发ADC帧模式唤醒。

MCU复位后，计数器被设置为0x00，RTC关闭。1KHz内部时钟被选择为默认时钟源。

三种时钟源可由软件选择：1K内部时钟、外部时钟和32K内部时钟。

5.5.2.2 帧唤醒模式

RTC中断可用于唤醒ADC帧工作模式，固定周期来唤醒ADC进行外部采样来唤醒系统。具体见ADC章节。

5.5.3 寄存器列表

RTC 模块地址区间：0x5007_0000~0x5008_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	RTC_SC	R/W	状态控制寄存器	0x0000
0x04	RTC_CNT	R/W	计数寄存器	0x0000_0000
0x08	RTC_MOD	R/W	模数寄存器	0x0000_0000

5.5.4 寄存器详细描述

5.5.4.1 RTC_SC 寄存器

地址	位	名称	R/W	复位值	描述
0x00	15:10	保留	——	——	保留
	9	RTC_SC_EN	R/W	0x0	RTC工作使能 0x1：RTC工作 0x0：RTC不工作
	8	保留	——	——	保留
	7	RTC_SC_IF	R/W	0x1	实时中断标记 0x1：已达到RTC溢出值寄存器中规定的值 0x0：未达到RTC溢出值寄存器中规定的值。 写入逻辑数1可清除此位和中断请求。
	6:5	RTC_SC_CLK_SEL	R/W	0x0	计数时钟源选择 0x3：实时时钟源为内部32KHz时钟 0x2：实时时钟源为内部32KHz时钟 0x1：实时时钟源为外部晶振时钟32分频 0x0：实时时钟源为1KHz时钟 计数过程中动态变更该位可清除RTC_CNT计数器。选择时钟源时，确保时钟源被正常使能（如果适用），以确保RTC的正确运行。由于时钟切换稳定性要求，RTC时钟切换选择1KHz时钟，要等待2ms才能正常工作。因此，推荐在需要再次配置RTC时，应先关闭RTC，再进行配置。
	4	RTC_SC_IE	R/W	0x0	中断使能 0x1：使能 0x0：不使能
	3:0	保留	——	——	保留

5.5.4.2 RTC_CNT 寄存器

地址	位	名称	R/W	复位值	描述
0x04	31:0	RTC_CNT	R/W	0x0	计数器的当前值。

5.5.4.3 RTC_MOD 寄存器

地址	位	名称	R/W	复位值	描述
0x08	31:0	RTC_MOD	R/W	0x0	计数器的产生中断的比较值。写该寄存器可清除RTC_CNT。实际应用禁止配置小于3，否则时序会出错，频繁中断。

5.5.5 程序范例

```
//主函数：
int main(void)
{
    gpio_init(GPIOB,GPIO_MODE_OUT,GPIO_PIN_6);  //PB6 设置为输出
    gpio_bit_set(GPIOB,GPIO_PIN_6);             //PB6 输出高电平
    gpio_bit_reset(GPIOB,GPIO_PIN_6);               //PB6 输出低电平
    /*RTC 初始化，包括 RTC_ENABLE（RTC 使能）、RTC_CLK_SEL_32K（选择 32KHz 内部时钟源）、
RTC_INT_ENABLE（RTC 中断使能）、RTC_TRG_CLK_DIV(0)（不触发 ADC 采样）、0x7D00（RTC 中断比较值，
0x7D00 即 32000，计满 32000 产生一次中断，按时钟频率计算则 1s 产生一次中断）*/
    rtc_init(RTC_ENABLE | RTC_CLK_SEL_32K | RTC_INT_ENABLE | RTC_TRG_CLK_DIV(0),0x7D00);
}
//子函数：
/******************RTC 初始化*********************/
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)      //选择外部晶振作为时钟源
    {
        if(!(SYS_XTAL_CTRL & SYS_XTAL_CTRL_INIT)){          //晶振起振未稳定
            rval = xtal_init();                                     //晶振初始化配置
            if(rval == ERROR){
                rtc_sc &= ~RTC_SC_CLK_SEL;
                rtc_sc |= RTC_CLK_SEL_1K;                       //初始化失败则选择 1K 内部时
钟作为时钟源
            }
        }
    }
    RTC_SC = rtc_sc;                                            //RTC 状态控制寄存器配置
    RTC_MOD = rtc_mod;                                          //RTC 模数寄存器配置
    if(rtc_sc & RTC_SC_IE){
        NVIC_EnableIRQ(RTC_IRQn);
    }else{
        NVIC_DisableIRQ(RTC_IRQn);
```

```
    }                                                          //RTC 中断使能配置
}
/*****************RTC 中断服务函数*******************/
void RTC_IRQHandler(void)
{
    rtc_int_flag_clr();                                        //清除 RTC 中断标志位
    timer0_f = ~timer0_f;
    if(timer0_f)
        gpio_bit_reset(GPIOB,GPIO_PIN_6);
    else
        gpio_bit_set(GPIOB,GPIO_PIN_6);                        //LED 灯闪烁
}
```

5.6 WDT

WDT 是一个 16 位向上计数的计数器。当系统软件不能正常执行时，WDT 看门狗将强制进行系统复位。为了防止从 WDT 定时器发起系统复位，应用软件必须定期复位 WDT 计数器。如果应用程序在超时前未能复位 WDT 计数器，这时会生成一个系统复位，强迫系统回到已知起点。

WDT 计数时钟为内部 32KHz 时钟或分频的 1KHz 时钟,系统上电后，WDT 默认是打开的，建议用户先关闭看门狗，根据应用需要进行配置后再开启 WDT。

5.6.1 主要特性

1. 溢出周期（16 位）可编程功能；
2. 计数时钟可选择 32KHz 和其 32 分频；
3. 喂狗操作唯一指令，非法指令喂狗将导致看门狗复位，防止误操作；
4. 支持窗口喂狗模式；
5. 寄存器配置保护解锁功能，避免误配置看门狗寄存器；
6. 系统低功耗模式下，支持看门狗复位唤醒；
7. 调试模式下可配置看门狗暂停工作；

5.6.2 功能说明

本节详细介绍 BF7006AMXX WDT 主要功能。

5.6.2.1 基本功能

系统复位后，WDT默认是打开的，WDT默认配置为32K Hz内部时钟，WDT的溢出值寄存器值为0xFFFF。

如果没有定期喂狗，将会出现系统复位。为了更新WDT计数器，软件必须在计数器溢出前（小于等于溢出值）进行顺序写入，否则会强制复位。

5.6.2.2 窗口模式

软件在完成它的主控制循环程序比想象的要快时，可能会在某些应用场合出现问题。因此根据应用需求采用窗口模式，窗口模式下如果喂狗操作太早，WDT将会导致系统复位。

当窗口模式有效时，WDT必须在计数器达到一个期望的最小值(该值最大为计时器溢出值的75%，当用户配置WDT_WINVAL值大于计数器溢出值75%时，系统会默认为计时器溢出值的75%）以后才能进行喂狗操作。提前写入，WDT会复位系统。该最小值由WDT_WINVAL寄存器决定。

该功能通常也应用于代码调试。

5.6.2.3 看门狗的配置保护

喂狗操作必须往 WDT_CNT 写入 0x55AA，否则写入非 0x55AA 以外的其他值会立即导致复位。

看门狗的控制寄存器、溢出值寄存器、窗口寄存器都是在系统复位后“只限写一次”。如需更新看门狗配置必须在下一次系统复位后。该规定提供健全的保护机制确保跑飞的程序无法乱配置看门狗。看门狗的配置顺序要求：先配置窗口寄存器、溢出值寄存器，接着配置其他控制寄存器，并确保WDT_CS_UPDATE=0。即使应用程序使用WDT所有控制状态寄存器的默认复位设置，用户也必须在复位初始化过程中进行写入操作到所有控制状态寄存器，以便在该设置中锁定。

在某些特定情况下，用户希望不通过复位来重新配置WDT或者关闭WDT使能。WDT提供二次配置机制，通过配置WDT_CS_UPDATE=1，用户可在任何时刻执行“顺序解锁”指令，在256个系统时钟内重新配置WDT，而不需要复位系统。

当WDT_CS_UPDATE=1时，可往WDT_CNT写入0x6BC3,接着在256个系统时钟内完成WDT新的配置，新的配置在256个系统时钟后才生效。当解锁计数器计数未达到256时，再次往WDT_CNT写入0x6BC3，解锁计数器会从0重新计数，以防256个系统时钟不够用。

5.6.2.4 低功耗唤醒

低功耗模式下，WDT 复位可唤醒系统，并从起始位置开始运行。但重新配置 WDT 需要 256 个系统时钟，在重新配置 WDT 后，如果需要进入低功耗模式，需要在这之前插入延时。这可确保 MCU 在进入低功耗模式之前，WDT 的新配置已生效，否则 MCU 有可能无法从低功耗模式下唤醒。

5.6.2.5 调试模式

系统处于调试模式时，可配置看门狗是否暂停工作，用于专门调试看门狗或者调试时避免看门狗的影响。

5.6.3 寄存器列表

WDT 模块地址区间：0x5008_0000~0x5008_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	WDT_CS	R/W	控制状态寄存器	0x0180
0x04	WDT_CNT	R/W	计数器寄存器	0x0000
0x08	WDT_TOVAL	R/W	溢出值寄存器	0xFFFF
0x0C	WDT_WINVAL	R/W	窗口寄存器	0x0000

5.6.4 寄存器详细描述

5.6.4.1 WDT_CS 寄存器

地址	位	名称	R/W	复位值	描述
0x00	15	WDT_CS_WINEN	R/W	0x0	看门狗窗口模式使能 0x1：窗口模式打开 0x0：窗口模式关闭
	14:9	保留	——	——	保留
	8	WDT_CS_CLKSEL	R/W	0x1	看门狗时钟选择 0x1：32KHz内部晶振时钟 0x0：1KHz内部晶振时钟
	7	WDT_CS_EN	R/W	0x1	看门狗使能 0x1：使能 0x0：不使能
	6	保留	——	——	保留
	5	WDT_CS_UPDATE	R/W	0x0	允许更新WDT配置 0x1：允许更新WDT配置，软件在执行解锁顺序写入操作后，可在256个系统时钟内重新更新WDT配置 0x0：允许更新WDT配置，在复位后对WDT初始化配置，后续不可以更新WDT配置
	4:2	保留	——	——	保留
	1	WDT_CS_SLEEP	R/W	0x0	空闲模式看门狗使能 0x1：使能 0x0：不使能
	0	WDT_CS_DEEPSLEEP	R/W	0x0	睡眠模式看门狗使能 0x1：使能 0x0：不使能

5.6.4.2 WDT_CNT 寄存器

地址	位	名称	R/W	复位值	描述
0x04	15:0	WDT_CNT	R/W	0x0	16位的WDT计数器，可随时通过软件进行读取该寄存器的数值。不能直接通过写操作更改该寄存器值，但写操作可以实现以下功能： 写入0x55AA，可清除WDT计数器。非0x55AA无法喂狗并且系统复位。 当WDT_CS_UPDATE=1时，可往WDT_CNT写入0x6BC3,接着在256个系统

					时钟内完成WDT新的配置。

5.6.4.3 WDT_TOVAL 寄存器

地址	位	名称	R/W	复位值	描述
0x08	15:0	WDT_TOVAL	R/W	0xFFFF	溢出比较值寄存器。严禁向该寄存器配置计数周期过短的值，否则系统会频繁进入复位。

5.6.4.4 WDT_WINVAL 寄存器

地址	位	名称	R/W	复位值	描述
0x0C	15:0	WDT_WINVAL	R/W	0x0	窗口溢出值 该值最大为计时器溢出值的75%，当用户配置值大于计数器溢出值75%时，系统会默认为计时器溢出值的75%。 该寄存器配置为0时相当于非窗口模式工作。该寄存器在实际应用中不应配置过小，否则看门狗会频繁进入复位且无法喂狗。

5.6.5 程序范例

```
//主函数:
int main()
{
    wdt_config(WDT_WIN_ENABLE | WDT_CLOCK_32KHZ | \
               WDT_UPDATA_ENABLE | WDT_SLEEP_DISABLE | \
               WDT_DEEPSLEEP_DISABLE | WDT_ENABLE);/*看门狗初始化，包 WDT_WIN_ENABLE（窗口模式使能）、WDT_CLOCK_32KHZ（选择 32KHz 时钟源）、WDT_UPDATA_ENABLE（允许更新 WDT 配置）、WDT_SLEEP_DISABLE（低功耗模式不使能）、WDT_DEEPSLEEP_DISABLE（睡眠模式不使能）、WDT_ENABLE（看门狗使能）*/
    wdt_overflow_count(50000);              //配置溢出比较值
    wdt_overflow_count_win(30000);          //配置窗口溢出值
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);//PE5 设置为输出

    gpio_bit_set(GPIOE,GPIO_PIN_5);      //PE5 输出高电平
    delay(1000);                            //延时
    gpio_bit_reset(GPIOE,GPIO_PIN_5);       //PE5 输出低电平
    while(1)
    {
        wdt_clear();                        //定时喂狗可防止程序频繁进入复位状态
```

```
    }
}
//子函数：
/*****************延时函数*******************/
void delay(uint16_t time)
{
    uint16_t i=0;
    while(time--)
    {
        for(i = 0;i < 1000;i++);
    }
}
/***************看门狗初始化*****************/
void wdt_config(uint16_t wdt_cs)
{
    WDT_CS = wdt_cs;                    //WDT 控制状态寄存器配置
}
/*************看门狗溢出值配置***************/
void wdt_overflow_count(uint16_t wdt_cnt)
{
    WDT_TOVAL = wdt_cnt;                //WDT 溢出值寄存器配置
}
/*************窗口溢出值配置***************/
void wdt_overflow_count_win(uint16_t wdt_cnt)
{
    WDT_WINVAL = wdt_cnt;               //WDT 窗口寄存器配置
}
/***************喂狗函数*****************/
void wdt_clear()
{
    WDT_CNT = 0x55AA;                   //WDT 计数寄存器写入 0x55AA，可清除 WDT 计数器
}
```

5.7 GPIO

BF7006AMXX 拥有丰富的外部 GPIO 资源，最多支持 54 个通用 GPIO，分别对应 A0~A7，B0~B7，C0~C7，D0~D7，E0~E7，F0~F7，G0~G5。

5.7.1 主要特性

1. 数据方向控制，并支持输入高阻态；
2. 支持内部上拉电阻功能；

3. 支持外部 GPIO 中断功能，包含上升沿、下降沿、高电平、低电平中断类型；
4. 支持 NMI 外部中断，中断优先级最高；

5.7.2 功能说明

本节详细介绍 BF7006AMXX GPIO 主要功能。

5.7.2.1 基本功能

BF7006AMXX GPIO 为通用的 GPIO 端口，支持普通的 IO 输出输入功能。端口 A、B、D 同时支持外部中断功能，共 24 个外部中断。中断类型可选，包括上升沿、下降沿、高电平、低电平四种中断类型。同时，所有 GPIO 均支持内部上拉功能，上拉电阻 20K。

BF7006AMXX VCC 上电从 0V-3.1V 区间，GPIO 端口处于不确定态（输出跟随 VCC 电压值或者输出低或者输入高阻态），需注意 GPIO 端口的外围元器件保护；VCC 上电高于 3.3V 后，芯片完成上电复位，GPIO 端口状态按照程序里设置执行；

BF7006AMXX VCC 电压下降至 3.1V，产生掉电复位后，GPIO 端口默认为输入高阻态。

5.7.2.2 不可屏蔽中断

NMI 作为优先级最高的不可屏蔽中断，复用在 PA7 端口。(NMI 中断说明请见 ARM Cortex_M0 相关资料)。系统对 NMI 控制功能扩展如下：

1. NMI 可被使能或禁能；
2. NMI 支持四种中断类型：上升沿、下降沿、高电平、低电平中断；
3. NMI 功能优先级最高；
4. 同样支持内部上拉电阻功能；

5.7.3 寄存器列表

GPIO 模块地址区间：0x500A_0000~0x500A_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	GPIO_PTD（GPIOA）	R/W	端口 A 数据寄存器	0x00
0x04	GPIO_PTDD（GPIOA）	R/W	端口 A 方向寄存器	0x00
0x08	GPIO_PTPE（GPIOA）	R/W	端口 A 上拉使能寄存器	0x00

0x10	GPIO_PTSC（GPIOA）	R/W	端口 A 中断控制和状态寄存器	0x00
0x14	GPIO_PTPS（GPIOA）	R/W	端口 A 中断使能寄存器	0x00
0x18	GPIO_PTES（GPIOA）	R/W	端口 A 中断类型选择寄存器	0x00
0x1C	GPIO_PTD（GPIOB）	R/W	端口 B 数据寄存器	0x00
0x20	GPIO_PTDD（GPIOB）	R/W	端口 B 方向寄存器	0x00
0x24	GPIO_PTPE（GPIOB）	R/W	端口 B 上拉使能寄存器	0x00
0x2C	GPIO_PTSC（GPIOB）	R/W	端口 B 中断控制和状态寄存器	0x00
0x30	GPIO_PTPS（GPIOB）	R/W	端口 B 中断使能寄存器	0x00
0x34	GPIO_PTES（GPIOB）	R/W	端口 B 中断类型选择寄存器	0x00
0x38	GPIO_PTD（GPIOC）	R/W	端口 C 数据寄存器	0x00
0x3C	GPIO_PTDD（GPIOC）	R/W	端口 C 方向寄存器	0x00
0x40	GPIO_PTPE（GPIOC）	R/W	端口 C 上拉使能寄存器	0x00
0x48	GPIO_PTD（GPIOD）	R/W	端口 D 数据寄存器	0x00
0x4C	GPIO_PTDD（GPIOD）	R/W	端口 D 方向寄存器	0x00
0x50	GPIO_PTPE（GPIOD）	R/W	端口 D 上拉使能寄存器	0x00
0x58	GPIO_PTSC（GPIOD）	R/W	端口 D 中断控制和状态寄存器	0x00
0x5C	GPIO_PTPS（GPIOD）	R/W	端口 D 中断使能寄存器	0x00
0x60	GPIO_PTES（GPIOD）	R/W	端口 D 中断类型选择寄存器	0x00
0x64	GPIO_PTD（GPIOE）	R/W	端口 E 数据寄存器	0x00
0x68	GPIO_PTDD（GPIOE）	R/W	端口 E 方向寄存器	0x00
0x6C	GPIO_PTPE（GPIOE）	R/W	端口 E 上拉使能寄存器	0x00
0x74	GPIO_PTD（GPIOF）	R/W	端口 F 数据寄存器	0x00
0x78	GPIO_PTDD（GPIOF）	R/W	端口 F 方向寄存器	0x00
0x7C	GPIO_PTPE（GPIOF）	R/W	端口 F 上拉使能寄存器	0x00
0x84	GPIO_PTD（GPIOG）	R/W	端口 G 数据寄存器	0x00
0x88	GPIO_PTDD（GPIOG）	R/W	端口 G 方向寄存器	0x00
0x8C	GPIO_PTPE（GPIOG）	R/W	端口 G 上拉使能寄存器	0x00
0x94	GPIO_INTST（GPIOA）	R/W	端口 A 中断标志寄存器	0x00
0x98	GPIO_INTST（GPIOB）	R/W	端口 B 中断标志寄存器	0x00
0x9C	GPIO_INTST（GPIOD）	R/W	端口 D 中断标志寄存器	0x00
0xA0	GPIO_NMISC	R/W	NMI 中断控制和状态寄存器	0x00

5.7.4 寄存器详细描述

5.7.4.1 GPIO_PTDD（GPIOx）寄存器

地址	位	名称	R/W	复位值	描述
0x04 0x20	7：0	GPIO_PTDD（GPIOA） GPIO_PTDD（GPIOB）	R/W	0x00	端口的数据方向 0x1：输出

0x3C		GPIO_PTDD（GPIOC）			0x0：输入
0x4C		GPIO_PTDD（GPIOD）			
0x68		GPIO_PTDD（GPIOE）			
0x78		GPIO_PTDD（GPIOF）			
0x88		GPIO_PTDD（GPIOG）			

5.7.4.2 GPIO_PTD（GPIOx）寄存器

地址	位	名称	R/W	复位值	描述
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	端口数据 对于配置为输入的端口管脚，读数返回管脚上的逻辑电平。对于配置为输出的端口管脚，读数返回写入寄存器的最后一个值。 写入值被锁定在该寄存器的所有位中。对于配置为输出的端口管脚，逻辑电平驱动相应的 MCU 管脚。 复位默认为高抗阻输入。

5.7.4.3 GPIO_PTPE（GPIOx）寄存器

地址	位	名称	R/W	复位值	描述
0x08 0x24 0x40 0x50 0x6C 0x7C 0x8C	7：0	GPIO_PTPE（GPIOA） GPIO_PTPE（GPIOB） GPIO_PTPE（GPIOC） GPIO_PTPE（GPIOD） GPIO_PTPE（GPIOE） GPIO_PTPE（GPIOF） GPIO_PTPE（GPIOG）	R/W	0x00	上拉使能 0x1：使能 0x0：不使能 对于配置为输出的端口管脚，这些位不会产生影响，同时内部上拉器件被禁止。

5.7.4.4 GPIO_PTSC（GPIOx）寄存器

GPIO_PTSC（GPIOA）、GPIO_PTSC（GPIOB）、GPIO_PTSC（GPIOD）					
地址	**位**	**名称**	**R/W**	**复位值**	**描述**
0x10 0x2C 0x58	7:4	保留	——	——	保留
	3	GPIO_PTSC_IF	R/W	0x0	端口类型中断标志 0x1：检测到端口中断 0x0：未检测到端口中断 显示是否检测到某一类型端口中断。写入对其没有任何影响，请区别于各个端口位中断标志。
	2	GPIO_PTSC_ACK	R/W	0x0	端口类型中断清除 0x1：清除中断 0x0：无影响 读数总为 0。
	1	GPIO_PTSC_IE	R/W	0x0	端口类型中断使能 0x1：端口中断使能 0x0：端口中断禁止
	0	GPIO_PTSC_TRGMOD	R/W	0x0	端口中断检测模式模式 0x1：端口同时检测边沿和电平 0x0：端口只检测边沿 当 GPIO 选择边沿时，IO 口输入长电平，中断标志拉高标志只识别边沿，标志位清除后等待下一个边沿再出中断拉高标志位。

5.7.4.5 GPIO_PTPS（GPIOx）寄存器

GPIO_PTPS（GPIOA）、GPIO_PTPS（GPIOB）、GPIO_PTPS（GPIOD）					
地址	**位**	**名称**	**R/W**	**复位值**	**描述**
0x14 0x30 0x5C	7：0	GPIO_PTPS（GPIOA） GPIO_PTPS（GPIOB） GPIO_PTPS（GPIOD）	R/W	0x00	每个端口类型的 8 个位的中断使能 0x1：使能 0x0：不使能 该寄存器是各个端口每一位的中断使能。

5.7.4.6 GPIO_PTES（GPIOx）寄存器

GPIO_PTES（GPIOA）、GPIO_PTES（GPIOB）、GPIO_PTES（GPIOD）					
地址	位	名称	R/W	复位值	描述
0x18 0x34 0x60	7：0	GPIO_PTES（GPIOA） GPIO_PTES（GPIOB） GPIO_PTES（GPIOD）	R/W	0x00	端口中断类型选择 0x1：检测中断生成的上升边沿/高电平 0x0：检测中断生成的下降边沿/低电平

5.7.4.7 GPIO_INTST（GPIOx）寄存器

GPIO_INTST（GPIOA）、GPIO_ INTST（GPIOB）、GPIO_ INTST（GPIOD）					
地址	位	名称	R/W	复位值	描述
0x94 0x98 0x9C	7：0	GPIO_ INTST（GPIOA） GPIO_ INTST（GPIOB） GPIO_ INTST（GPIOD）	R/W	0x00	各个端口位中断标志 0x1：产生中断 0x0：无中断标志 当产生中断时，对应位拉高变为1，对该位写1清零。

5.7.4.8 GPIO_NMISC 寄存器

GPIO_PTSC（GPIOA）、GPIO_PTSC（GPIOB）、GPIO_PTSC（GPIOD）					
地址	位	名称	R/W	复位值	描述
0xA0	7	保留	——	——	保留
	6	GPIO_NMISC_NMPDD	R/W	0x0	中断请求（NMI）上拉器件禁止 0x1：NMI 中断上拉功能禁止 0x0： NMI 中断上拉器件使能； 该位仅控制 NMI 功能下的上拉功能。
	5	GPIO_NMISC_EDG	R/W	0x0	NMI 中断类型选择。 0x1：上升边沿，高电平 0x0：下降边沿，低电平
	4	GPIO_NMISC_PE	R/W	0x0	NMI 中断端口使能 0x1：外部端口配置功能为 NMI 中断 0x0：外部端口不配置功能为 NMI 中断 配置使能前应用应先清除 NMI 中断标志，防止误进入中断。
	3	GPIO_NMISC_IF	RO	0x0	NMI 中断标志 0x1：产生了中断

					0x0：无中断
	2	GPIO_NMISC_ACK	R/W	0x0	用来确认中断请求事件。写入 0 则没有任何意义或影响。读总是返回 0。写 1 清除中断标志。电平中断无法清除标志，直至电平中断失效。
	1	GPIO_NMISC_IE	R/W	0x0	NMI 中断使能 0x1：当 NMI 有效时产生中断请求 0x0：当 NMI 有效时时不产生中断请求（使用轮询）
	0	GPIO_NMISC_MOD	R/W	0x0	NMI 中断模式选择 0x1：所有模式的中断 0x0：下降边沿或上升边沿的中断

5.7.5 程序范例

```
//主函数：
int main(void)
{
    gpio_init(GPIOA,GPIO_MODE_IPU,GPIO_PIN_5);              //PA5 设置为输入
    gpio_init(GPIOE,GPIO_MODE_OUT,GPIO_PIN_5);              //PE5 设置为输出
    gpio_init(GPIOA,GPIO_MODE_OUT,GPIO_PIN_2);              //PA2 设置为输出

    gpio_trigge_mode(GPIOA,GPIO_TRG_FALLING,GPIO_PIN_5);    //PA5 设置为下降沿触发中断
    gpio_interrupt_set(GPIOA,GPIO_PIN_5,ENABLE);                //PA5 中断使能

    while(1)
    {
        /*在硬件电路上 PA2 连接 PA5，当 PA2 进行电平翻转时，会改变 PA5 输入电平状态，触发外部中断，进入 GPIO 中断服务函数*/
        gpio_bit_set(GPIOA,GPIO_PIN_2);
        gpio_bit_reset(GPIOA,GPIO_PIN_2);
    }
}
//子函数：
/*****************GPIO 初始化函数*******************/
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;                 //配置端口为浮空输入模式
            break;
        case GPIO_MODE_IPU:
            GPIO_PTDD(gpio_periph) &= ~pin;
            GPIO_PTPE(gpio_periph) |= pin;                      //配置端口为上拉输入模式
```

```
            break;
        case GPIO_MODE_OUT:
            GPIO_PTDD(gpio_periph) |= pin;                          //配置端口为输出模式
            break;
        default:
            break;
    }
}
/*********GPIO 中断触发模式选择函数**********/
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;
    }                                                               //非 A,B,D 组端口返回 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;                          //高电平触发中断
            break;
        case GPIO_TRG_LOW:
            GPIO_PTSC(gpio_periph) |= GPIO_PTSC_TRGMOD;
            GPIO_PTES(gpio_periph) &= ~pin;                         //低电平触发中断
            break;
        case GPIO_TRG_RISING:
            GPIO_PTSC(gpio_periph) &= ~GPIO_PTSC_TRGMOD;
            GPIO_PTES(gpio_periph) |= pin;                          //上升沿触发中断
            break;
        case GPIO_TRG_FALLING:
            GPIO_PTSC(gpio_periph) &= ~GPIO_PTSC_TRGMOD;
            GPIO_PTES(gpio_periph) &= ~pin;                         //下降沿触发中断
            break;
        default:
            return ERROR;
    }
    /* gpio trigge pin enable */
    GPIO_PTPS(gpio_periph) |= pin;                                  //端口触发中断使能
    return SUCCESS;
}
/************GPIO 端口中断使能*************/
void gpio_interrupt_set(uint32_t gpio_periph,uint8_t pin,FunctionalState value)
{
        /* gpio trigge 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);
    }                                                    //端口中断配置
}
/************GPIO 端口输出高电平*************/
void gpio_bit_set(uint32_t gpio_periph,uint8_t pin)
{
    GPIO_PTD(gpio_periph) |= pin;                              //端口输出高电平
}
/************GPIO 端口输出低电平*************/
void gpio_bit_reset(uint32_t gpio_periph,uint8_t pin)
{
    GPIO_PTD(gpio_periph) &= ~pin;                             //端口输出低电平
}
/************GPIO 中断服务函数*************/
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);                      //电平翻转
    }
    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);                    //清除中断标志位
}
```

5.8 ADC

模数转换器 ADC 能够将连续的模拟信号转换成离散的数字信号，该芯片 ADC 是 12 位数逐次逼近式 ADC，可配置的寄存器使得时序的控制更加容易，可以方便的应用到车载 MCU 中，与外设进行通信。

5.8.1 主要特性

1. 分辨率12位的线性逐次逼近ADC
2. 共28个模拟输入端口，其中外部24个；
3. 12位和8位右对齐输出格式；
4. 单次转换和连续转换；
5. 采样时间和转换时间可配置；
6. 支持硬件触发ADC采样功能；
7. 与小于、大于或等于可编程值自动比较的中断；
8. 支持系统睡眠时触发ADC有效采样系统；

5.8.2 功能说明

本节详细介绍 BF7006AMXX ADC 主要功能。

5.8.2.1 基本功能

复位期间或当ADC_SC1_ADCH=0x11111时，ADC模块禁止。当已经完成当前转换，而下一次转换还未发起时，模块进入最低功耗状态。

ADC可以对软件选择的任意通道实施模数转换。转换完成后，结果保存在数据寄存器ADC_DATA中。然后置位转换完成标记ADC_SC1_COCO，如果已经使能了转换完成中断ADC_SC1_AIEN =1，则触发中断。

ADC 模块能够自动地把转换结果与比较寄存器的内容进行比较。通过设置 ADC_SC2_ACFE0 或 ADC_SC2_ACFE1 位并结合任意一种转换模式和配置一起运行，就使能了比较功能。

5.8.2.2 通道分配

模拟输入共有28个，其中24个外部ADC通道，内部具有温度传感器和内部隙带通道。管脚控制寄存器ADC_APCTL用来禁止对作为模拟输入的管脚的I/O控制功能。如果选择成预留通道则固定为管脚VREFS输入。

ADC_SC1_ADCH通道选择配置	通道	输入
00000~10111	AD0~AD23	ADC0~ADC23
11000~11001	AD24~AD25	保留
11010	AD26	温度传感器
11011	AD27	BG（1.267V典型）
11100	保留	保留
11101	VREFA	VREFA
11110	VREFS	VREFS
11111	模块禁止	模块禁止

5.8.2.3 触发方式

ADC控制器有两种触发可供选择：软件触发和硬件触发。通过ADC_SC2_ADTRG用来选择转换的触发类型。当选择软件触发时，配置通道后就能发起转换。当选择硬件触发时，如下硬件触发后就能发起转换。

硬件触发有三种，通过寄存器位ADC_SC2_ADHTS和ADC_IWK来选择：

- 内部中断触发——当ADC硬件触发使能时，RTC计数器溢出触发ADC转换；
- 内部中断触发——PWM计数触发ADC转换（5.3.4.6）。该触发不支持系统低功耗睡眠模式；
- 外部中断NMI触发ADC转换；

当系统提供了硬件触发源且已使能了硬件触发时，硬件触发的上升沿上就发起转换。如果转换进行过程中再出现上升沿，该上升沿就被忽略。在连续转换模式下，只是触发连续转换的第一个上升沿有效。硬件触发功能可以结合任意转换模式和配置一起运行。

5.8.2.4 转换控制

ADC 转换可以在 12 位或 8 位模式下进行，由 ADC_CFG_MODE 位决定。转换可以由软件或硬件触发发起。此外，ADC 模块采样时间可设置，也可配置连续转换，可配置将转换结果与软件设定的比较值自动比较的方式。

转换模式：

1. 正常工作时，选择是否进行连续转换，且连续转换为不间断连续转换同一ADC通道；
2. 系统低功耗模式下，仅支持硬件触发单次转换模式，在触发前ADC处于低功耗模式，触发事件后等待100us后发出一次ADC转换时序，循环触发一次转换一次。

发起转换：

1. 选择软件触发或者硬件触发；
2. 如果使能连续转换，在当前转换完成会就自动发起一个新转换。在软件触发操作中，连续转换在选择好通道后就开始，并且一直持续直到被中止。在硬件触发操作中，连续转换在硬件触发事件后开始，并且一直持续直到被中止。

ADC转换过程中不允许切换模式，如果需要，请先关闭ADC_SC1_ADCH，完成模式切换配置后再打开ADC_SC1_ADCH。

完成转换：

当转换结果被传输到数据结果寄存器ADC_DATA中时，转换完成。

在配置数据比较功能时，如果未达到比较条件，不会置位ADC_SC1_COCO；其它情况下均置位ADC_SC1_COCO。通过置位ADC_SC1_COCO标识，如果置位ADC_SC1_COCO时ADC_SC1_AIEN =1，就会触发中断。

中止转换：

当出现下列情况时，进行中的任何转换都将中止：

1. 重新配置通道选择；
2. ADC_SC2、ADC_CFG、ADC_CV0、ADC_CV1控制和配置寄存器写入。这表明出现运行模式更改，当前转换因此无效；
3. MCU复位；当转换被中止时，数据寄存器ADC_DATA的内容不变，保持上次成功转换完成后传输的值。如果由于芯片复位，ADC_DATA则为复位值。

5.8.2.5 自动比较

ADC可以配置比较功能来检查上限或下限。在进行完输入采样并转换后，结果与比较值ADC_CV0或ADC_CV1比较。

比较条件通过ADC_SC2_ACFGT0或ADC_SC2_ACFGT1设定，满足比较条件后就设置ADC_SC1_COCO，转换结果直接传输到ADC_DATA中。

同时，ADC还支持双条件比较功能，在比较功能有效ADC_SC2_ACFE=1的情况下才有效。ADC_SC2_ACFGT1决定第二路比较条件，同样可配比较上限和下限，ADC_CV1对应第二路比较值。两路比较使能时，需要两路比较结果均满足，才会置位ADC_SC1_COCO。转换结果直接传输到ADC_DATA中。

比较功能可以用来监控通道的电压，这时MCU可能处于低功耗模式。在满足比较条件时，ADC中断会唤醒MCU系统。

5.8.2.6 睡眠唤醒

系统睡眠模式下，仅支持硬件触发单次转换模式。在触发前ADC处于关闭状态，触发事件后先打开ADC，等待100us后发出一次ADC采样和转换时序，循环触发一次转换一次。

硬件触发发起转换，如果ADC中断使能，转换完成事件（设置比较功能时，发生比较匹配事件）就会置位ADC_SC1_COCO，生成ADC中断，把MCU系统从睡眠模式中唤醒。

根据不同是否配置自动比较，可以应用到以下几种场景：

1. 设置自动比较功能，在进入睡眠模式前配置ADC关闭，达到触发条件时，系统先打开系统时钟源，稳定500us后将ADC时钟打开，ADC控制器先打开ADC，等待100us后开启一次ADC转换，完成后硬件自动设置ADC关闭并等待下一次硬件触发条件满足，循环进行，转换结果达到比较条件则置位ADC_SC1_COCO，生成ADC中断，唤醒MCU系统；
2. 关闭自动比较功能，在进入睡眠模式前配置ADC关闭，达到触发条件时，系统先打开系统时钟源，稳定500us后将ADC时钟打开，ADC控制器先打开ADC，等待100us后开启一次ADC转换，完成后直接设置ADC_SC1_COCO，生成ADC中断，唤醒MCU系统。
3. 其它配置情况，进入睡眠模式后，不会启动ADC扫描。

如若要使用ADC唤醒模式，需在MCU配置进入睡眠模式之前，需要依次配置ADC_SC2_ADTRG=1选择硬件触发类型，ADC_SCSC1_ADCO=0选择单次转换模式，ADC_SC2_ADHTS选择硬件触发源，配置ADC_PD =1，配置打开相应的ADC通道使能ADC_SCSC1_ADCH，最后配置系统进入睡眠模式。同样，ADC也支持可以配置RTC计

时溢出触发ADC采样，也可以配置外部NMI中断触发ADC采样。从系统睡眠模式退出后，会关闭ADC使能，需要软件或硬件触发来重新开始转换。

5.8.2.7 时序时间说明

ADC 时序时间遵循以下要求：

Timer1=（ADC_CKC_SAMDEL+2）*单个ADC时钟时间；

Sample_Timer= （ADC_SPT+1）*单个ADC时钟时间；

Time2=（ADC_CKC_WNUM +3）*单个ADC时钟时间；

Time3=（12+2）*单个ADC时钟时间（12位模式）

Time3=（8+2）*单个ADC时钟时间（8位模式）

时序要求:（ADC_CKC_WNUM +3）*单个ADC时钟时间> 4*ADC采样触发器的时钟

ADC转换时间：T= Timer1+ Sample_Time+ Time2+ Time3

5.8.2.8 自测试功能

电容自测试：配置执行校准流程即可完成此测试，校验得到的数据与校准码是否一致。

短路自测试：测试短路。配置不同通道执行正常转换即可完成此测试功能。可配置是否校准，可配数据位宽，校验得到的转换数据是否符合标准。

测试流程：

1. 配置通道“11101”（VREFA），正常转换完成，得到数据1；
2. 配置通道“11110”（VREFS），正常转换完成，得到数据2；校验数据1/2是否满足要求：数据1==0xfff（12位数据模式）/0xff（8位数据模式），数据2==0x0。

开路自测试：需要配置进入自测试模式，分别测试每一路ADC通道的连接（每个通道包括提前采样VREFA/VREFS共两次转换），可配是否校准，可配数据位宽，校验得到的转换数据是否符合标准。

测试流程：

1. 配置进入自测试模式，配置通道“00000”的第一次转换（VREFA），转换完成得到数据1；
2. 配置该通道的第二次转换（VREFS），转换完成得到数据2；

3. 配置通道“00001”的第一次转换（VREFA），转换完成得到数据1；

4. 配置该通道的第二次转换（VREFS），转换完成得到数据2；

依次将所有通道执行两次自测试转换，校验每个通道数据1和数据2是否一致，数据一致说明测试正常。

自动比较功能不区分哪种模式，无论有没有校准，是否是测试模式，均可配自动比较功能。几种测试模式是互斥的，即不能同时配置自测试、校准使能、电容测试使能。配置校准使能时，无论其它配置如何，均需要进行校准运算，即校准使能的优先级最高。

5.8.3 寄存器列表

ADC 模块地址区间：0x5009_0000~0x5009_FFFF

偏移地址	寄存器名称	R/W	功能描述	初始值
0x00	ADC_SC1	R/W	状态和控制寄存器 1	0x1F
0x04	ADC_SC2	R/W	状态和控制寄存器 2	0x01
0x08	ADC_DATA	R	数据结果寄存器	0x000
0x0C	ADC_CV0	R/W	比较值 0 寄存器	0x000
0x10	ADC_CV1	R/W	比较值 1 寄存器	0x000
0x14	ADC_CFG	R/W	配置寄存器	0x00
0x18	ADC_APCTL	R/W	管脚使能寄存器	0x000000
0x1C	ADC_SPT	R/W	采样时间配置寄存器	0x002
0x78	ADC_CKC	R/W	模拟时钟及间隔控制寄存器	0x00
0x80	ADC_PD	R/W	模拟关断控制寄存器	0x1
0x84	ADC_TEST	R/W	自测试模式配置寄存器	0x0
0x8C	ADC_IWK	R/W	硬件内部触发选择寄存器	0x0
0x90	ADC_FRSEL	R/W	输入信号滤波选择控制寄存器	0x0

5.8.4 寄存器详细描述

5.8.4.1 ADC_SC1 寄存器

地址	位	名称	R/W	复位值	描述
0x00	7	ADC_SC1_COCO	RO	0x0	转换完成标志 0x1：转换完成 0x0：转换未完成或未产生转换

					COCO标记是只读位，在每次转换完成时置位，读取ADC_DATA时，该位就被清除。
	6	ADC_SC1_AIEN	R/W	0x0	中断使能 0x1：转换完成中断使能 0x0：转换完成中断禁止 该位用来使能转换完成中断。当ADC_SC1_COCO已置位且该位为1时，中断将被触发。
	5	ADC_SC1_ADCO	R/W	0x0	连续转换使能 0x1：进行连续转换 0x0：进行单次转换
	4:0	ADC_SC1_ADCH	R/W	0x1F	输入通道选择 通道选择位全部设置为1时，通道全部关闭。该功能能够彻底禁止ADC，隔离所有输入通道。用此种方式终止连续转换，可以防止多余的的一次单次转换。当禁止连续转换时，就不需要通过把通道选择位都设置为1的方式使ADC处于低功耗状态，因为当转换完成时模块会自动进入低功耗状态。

5.8.4.2 ADC_SC2 寄存器

地址	位	名称	R/W	复位值	描述
0x04	7	ADC_SC2_ADACT	RO	0x0	转换状态 0x1：转换正在进行 0x0：转换未进行 该位显示正在进行一个转换。当开始转换时就会置位A，当转换完成或中止时就会自动清除。
	6	ADC_SC2_ADTRG	R/W	0x0	转换触发类型选择 0x1：选择硬件触发 0x0：选择软件触发
	5	ADC_SC2_ACFE0	R/W	0x0	比较器0比较功能使能 ： 0x1：比较功能使能 0x0：比较功能禁止
	4	ADC_SC2_ACFGT0	R/W	0x0	比较器0比较条件选择 0x1：当输入大于或等于比较值时触发 0x0：当输入小于比较值时触发
	3	ADC_SC2_ACFE1	R/W	0x0	比较器1比较功能使能 ： 0x1：比较功能使能 0x0：比较功能禁止
	2	ADC_SC2_ACFGT1	R/W	0x0	比较器1比较条件选择 0x1：当输入大于或等于比较值时触发 0x0：当输入小于比较值时触发

	1	ADC_SC2_ADHTS	R/W	0x0	ADC硬件触发选择 0x1：外部中断请求（NMI）管脚 0x0：内部中断请求（包括RTC溢出或PWM定时溢出中断）
	0	ADC_SC2_COREN	R/W	0x1	12位数据校准使能 0x1：使能 0x0：不使能 仅控制12位转换模式下，数据是否经过校准运算。

5.8.4.3 ADC_DATA 寄存器

地址	位	名称	R/W	复位值	描述
0x08	11:0	ADC_DATA	RO	0x0	ADC转换的数据

5.8.4.4 ADC_CV0 寄存器

地址	位	名称	R/W	复位值	描述
0x0C	11:0	ADC_CV0	R/W	0x0	比较器0的比较值

5.8.4.5 ADC_CV1 寄存器

地址	位	名称	R/W	复位值	描述
0x10	11:0	ADC_CV1	R/W	0x0	比较器1的比较值

5.8.4.6 ADC_CFG 寄存器

地址	位	名称	R/W	复位值	描述
0x14	7	保留	——	——	保留
	6:4	ADC_CFG_ADIV	R/W	0x0	时钟分频选择 选择生成ADC时钟所使用的分频。
	3	保留	——	——	保留
	2	ADC_CFG_MODE	R/W	0x0	转换模式选择 0x1：12位转换

					0x0：8位转换
	1:0	保留	——	——	保留

ADC_CFG_ADIV	分频倍数	系统时钟频率 32MHz	系统时钟频率 16MHz	系统时钟频率 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 寄存器

地址	位	名称	R/W	复位值	描述
0x18	23:0	ADC_APCTL0 ~ ADC_APCTL23	R/W	0x0	ADC0~ADC23端口功能使能 0x1：该端口ADC输入功能打开 0x0：该端口ADC输入功能关闭

5.8.4.8 ADC_SPT 寄存器

地址	位	名称	R/W	复位值	描述
0x1C	9:0	ADC_SPT	R/W	0x2	ADC采样时间配置 采样时间 = （ADC_SPT + 1）个ADC时钟 注：配置时需保证ADC_CKC_WNUM *ADC时钟周期>5*(系统时钟的ADC_CKC_CKV配置周期)

5.8.4.9 ADC_CKC 寄存器

地址	位	名称	R/W	复位值	描述
0x78	7	ADC_CKC_SAMBG	R/W	0x0	采样时序与比较时序间隔选择 0x1：间隔1个ADC时钟 0x0：间隔0
	6:4	ADC_CKC_SAMDEL	R/W	0x0	采样延迟时间选择 0x7：16个ADC时钟 0x6：14个ADC时钟 0x5：12个ADC时钟 0x4：10个ADC时钟 0x3：8个ADC时钟 0x2：4个ADC时钟 0x1：2个ADC时钟 0x0：0个ADC时钟
	3:2	ADC_CKC_WNUM	R/W	0x0	采样完毕后距离转换间隔时间选择 0x3：6个ADC时钟 0x2：5个ADC时钟 0x1：4个ADC时钟 0x0：3个ADC时钟
	1:0	ADC_CKC_CKV	R/W	0x0	模拟输入时钟信号分频 0x3：系统工作时钟8分频 0x2：系统工作时钟4分频 0x1：系统工作时钟2分频 0x0：系统工作时钟1分频

5.8.4.10 ADC_PD 寄存器

地址	位	名称	R/W	复位值	描述
0x80	7:1	保留	——	——	保留
	0	ADC_PD	R/W	0x0	关闭模拟ADC以降低功耗 0x1：关闭 0x0：不关闭

5.8.4.11 ADC_TEST 寄存器

地址	位	名称	R/W	复位值	描述
0x84	7:2	保留	——	——	保留
	1	ADC_TEST_NUM	R/W	0x0	自测试模式计数 0x1：第二次转换 0x0：第一次转换
	0	ADC_TRST_EN	R/W	0x0	自测试模式使能 0x1：使能 0x0：不使能

5.8.4.12 ADC_IWK 寄存器

地址	位	名称	R/W	复位值	描述
0x8C	7:1	保留	——	——	保留
	0	ADC_IKW	R/W	0x0	ADC 硬件触发内部源选择 0x1：选择 PWM 计数触发 0x0：选择实时计数器(RTC) 溢出触发

5.8.5 ADC 参考应用流程

ADC参考应用流程：

1. ADC初始化

a) 配置 ADC 的扫描模式、ADC 数据位宽、ADC 时钟配置、采样时钟数；
b) ADC 中断触发扫描使能及中断触发源配置；
c) ADC 比较器使能及参数配置；
d) ADC 通道选择作为模拟输入配置；
e) ADC 中断使能；

2、 配置 ADC 转换通道，并开启 ADC 转换
3、 读取 ADC 转换数据
a) 中断模式：在中断服务函数中读取转换数据；
b) 查询模式：等待 ADC 转换完成后，读取 ADC 数据结果寄存器；
4、 执行步骤 2，转换下一通道

注意：连续转换模式下，只进行当前通道的连续转换，并且转换完成后自动进行下一次的转换，不需要开启 ADC 转换。

6 电气特性

6.1 温度指标

指标	最小值	典型值	最大值	单位	备注
工作温度	-40	——	+125	℃	
存储温度	-55	——	+150	℃	
焊接温度	——	——	+260	℃	

6.2 潮敏指标

指标	最小值	典型值	最大值	单位	备注
潮敏等级 MSL	——	3	——	——	

6.3 ESD 指标

指标	最小值	典型值	最大值	单位	备注
人体模型 ESD HBM	-6000	——	+6000	V	
机器模型 ESD CDM	-500	——	+500	V	
闩锁 LATCH UP	-50	——	+50	mA	全温

6.4 电源指标

指标	最小值	典型值	最大值	单位	备注
工作电源电压 VDD	3.3	——	5.5	V	
工作电源电流 IDD	——	——	120	mA	
模拟电源电压 VDDA	3.3	——	5.5	V	
ADC 参考电压 VREFA	——	VDDA	——	V	

注：VREFA 电压须等于 VDDA

6.5 输入输出

指标	最小值	典型值	最大值	单位	备注
输出高电压 VOH	90%VDD	——	——	V	
输出低电压 VOL	——	——	10%VDD	V	
输出延迟	——	——	70	ns	
输入高电压 VIH	70%VDD	——	——	V	
输入低电压 VIL	——	——	30%VDD	V	
输入延迟	——	——	5	ns	
所有 IO 驱动总电流之和	——	——	120	mA	
内部上拉电阻	——	20K	——	Ω	
输入迟滞电压	10%VDD	——	——	V	
内阻	——	——	80	Ω	
普通驱动能力	——	——	5	mA	
大驱动能力 PB0~PB5	——	——	20	mA	
IO 漏电流	——	0.1	15	uA	

6.6 上掉电及电压检测

指标	最小值	典型值	最大值	单位	备注
上电电压 POR（恢复）	0.96	1.30	1.99	V	工程测试数据
上电电压 POR（掉电）	0.84	1.10	1.81	V	工程测试数据
上电延迟	4	6	8	ms	工程测试数据
掉电电压 BOR（恢复）	——	3.14	——	V	工程测试数据
掉电电压 BOR（掉电）	——	3.08	——	V	工程测试数据
掉电延迟	——	85	——	us	
低压检测电压 LVDT1	——	4.5	——	V	
低压检测电压 LVDT2	——	4.0	——	V	
低压检测电压 LVDT3	——	3.5	——	V	
低压检测延迟	——	10	——	us	

6.7 时钟源

指标	最小值	典型值	最大值	单位	备注
RC1M 周期偏差（修调前）	-15%	——	+15%	——	-40~+125℃
RC1M 周期偏差（修调后）	-3%	——	+3%	——	-40~+125℃
RC128K 周期偏差（修调前）	-30%	——	+30%	——	-40~+125℃
RC128K 周期偏差（修调后）	-15%	——	-15%	——	-40~+125℃

外部晶振 XOSC	——	8 or 16	——	MHz	-40~+125℃
外部晶振起振时间	——	3.1（8M） 3.1（16M）	——	ms	工程测试数据
PLL 频率偏差	——	——	——	——	-40~+125℃
PLL 锁定时间	——	100	——	us	-40~+125℃

6.8 模数转换 ADC

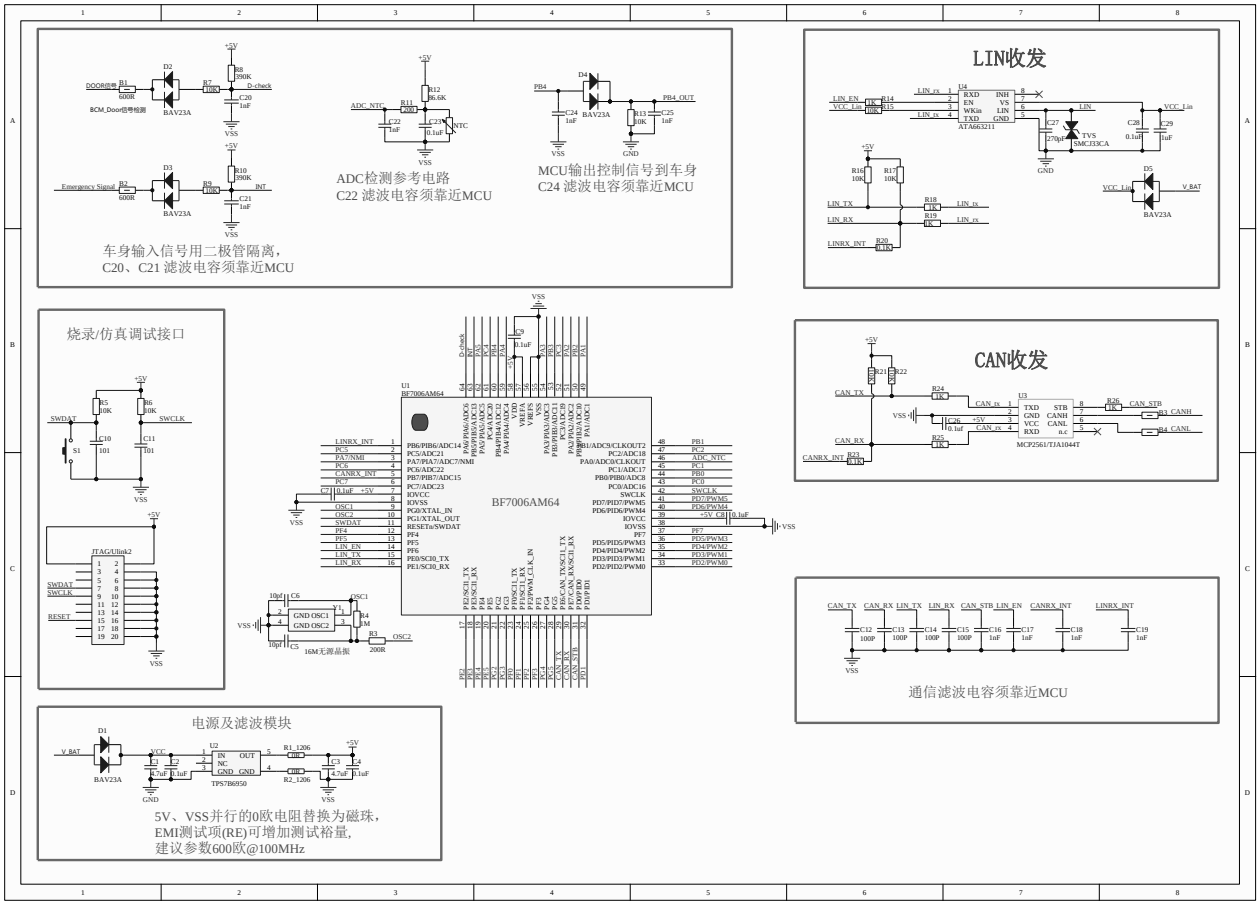
指标		最小值	典型值	最大值	单位	备注
速率		——	——	1	MHz	
分辨率		——	——	12	bit	
采样时间		1	——	1024	Tadc_clk	
转换时间		16	——	1062	Tadc_clk	
输入阻抗	FILTER_R_SEL=0	——	0.8	——	KΩ	
	FILTER_R_SEL=1	——	5	——	KΩ	
	FILTER_R_SEL=2	——	1	——	KΩ	
输入电容	FILTER_R_SEL=0	——	5	——	pF	
	FILTER_R_SEL=1	——	7	——	pF	
	FILTER_R_SEL=2	——	15	——	pF	

6.9 工作电流

指标		最小值	典型值	最大值	单位	备注
系统工作 32MHz		——	14.1	——	mA	工程测试数据，25℃
系统工作 16MHz		——	11.8	——	mA	工程测试数据，25℃
Sleeping 睡眠模式	32MHz	——	5.9	——	mA	工程测试数据，25℃
	16MHz	——	5.1	——	mA	工程测试数据，25℃
DeepSleep 睡眠模式	3.3V	——	25	——	uA	工程测试数据，25℃
	5V	——	30	——	uA	工程测试数据，25℃
RC128K	3.3V	——	4.7	——	uA	工程测试数据，25℃
	5V					工程测试数据，25℃
RC1M	3.3V	——	105	——	uA	工程测试数据，25℃
	5V					工程测试数据，25℃
XOSC	3.3V	——	179（8MHz） 245（16MHz）	——	uA	工程测试数据，25℃
	5V	——	366（8MHz） 469（16MHz）	——	uA	工程测试数据，25℃
PLL	3.3V	——	585	——	uA	工程测试数据，25℃
	5V					工程测试数据，25℃

ADC	3.3V	——	2	——	mA	工程测试数据，25℃
	5V					工程测试数据，25℃
BOR LVDT	3.3V	——	3.3	——	uA	工程测试数据，25℃
	5V					工程测试数据，25℃

7 应用参考电路



注：以上应用参考电路仅供参考设计。

8 附录 A

MCU 使用注意事项：

1. VCC 上电从 0V 到 3.3V 时间，要求在 4ms 内完成；
2. BF7006AMXX VCC 上电从 0V-3.1V 区间，GPIO 端口处于不确定态（输出跟随 VCC 电压值或者输出低或者输入高阻态），需注意 GPIO 端口的外围元器件保护；
   VCC 上电高于 3.3V 后，芯片完成上电复位，GPIO 默认处于输入高阻态，待程序运行后，GPIO 端口状态按照程序里设置执行。

9 附录 B

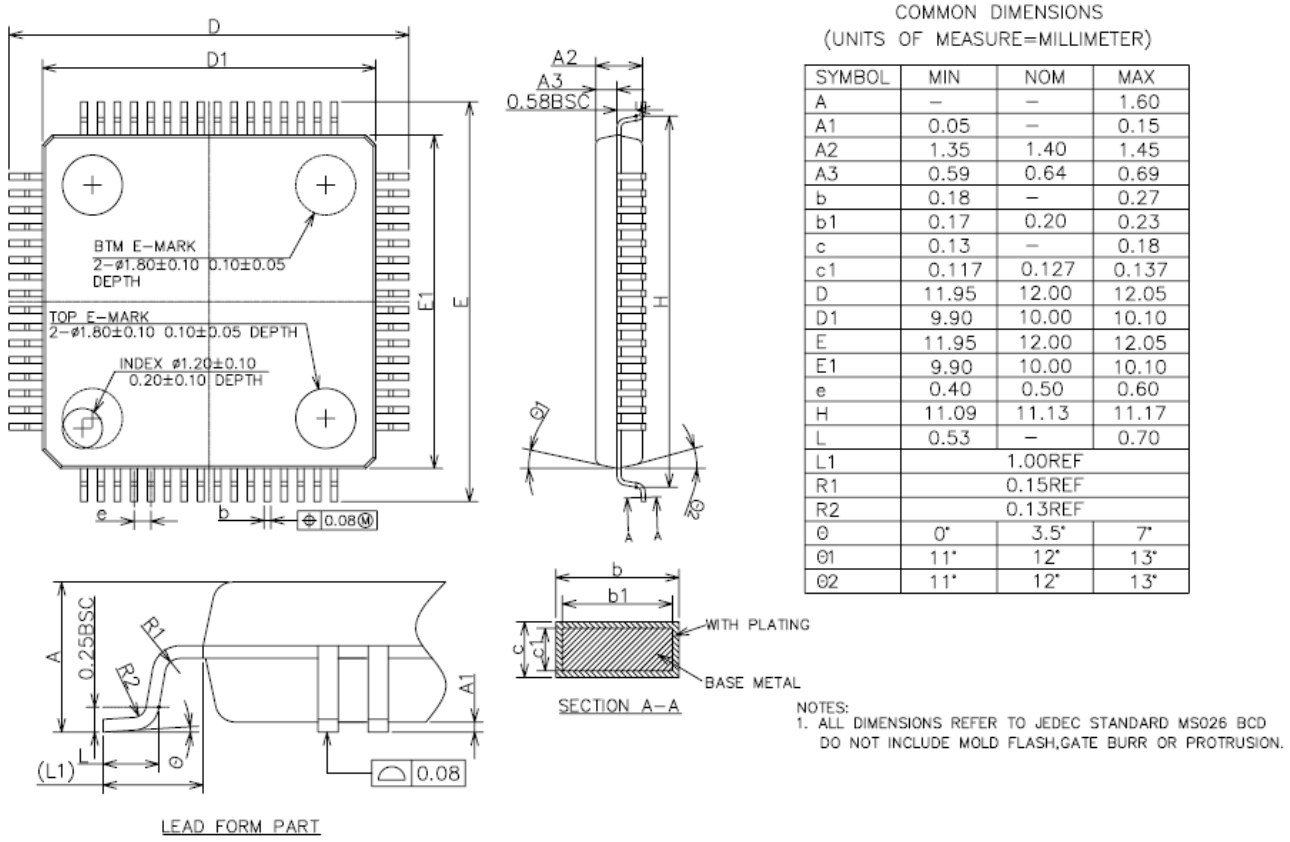
修订历史记录

版 次	日 期	修 改 内 容
V1.0	2019/7/20	初版
V1.1	2020/9/15	应用参考电路更新
V1.2	2020/12/7	增加 LQFP48 引脚图及封装尺寸
V1.3	2021/5/18	1. 增加芯片上电时间要求； 2. GPIO 描述中增加芯片上电复位时 IO 端口状态描述； 3. 增加 MCU 使用注意事项附录；
/		
/		
/		
/		
/		
/		

10 封装

10.1封装外形图展示

10.1.1 LQFP64

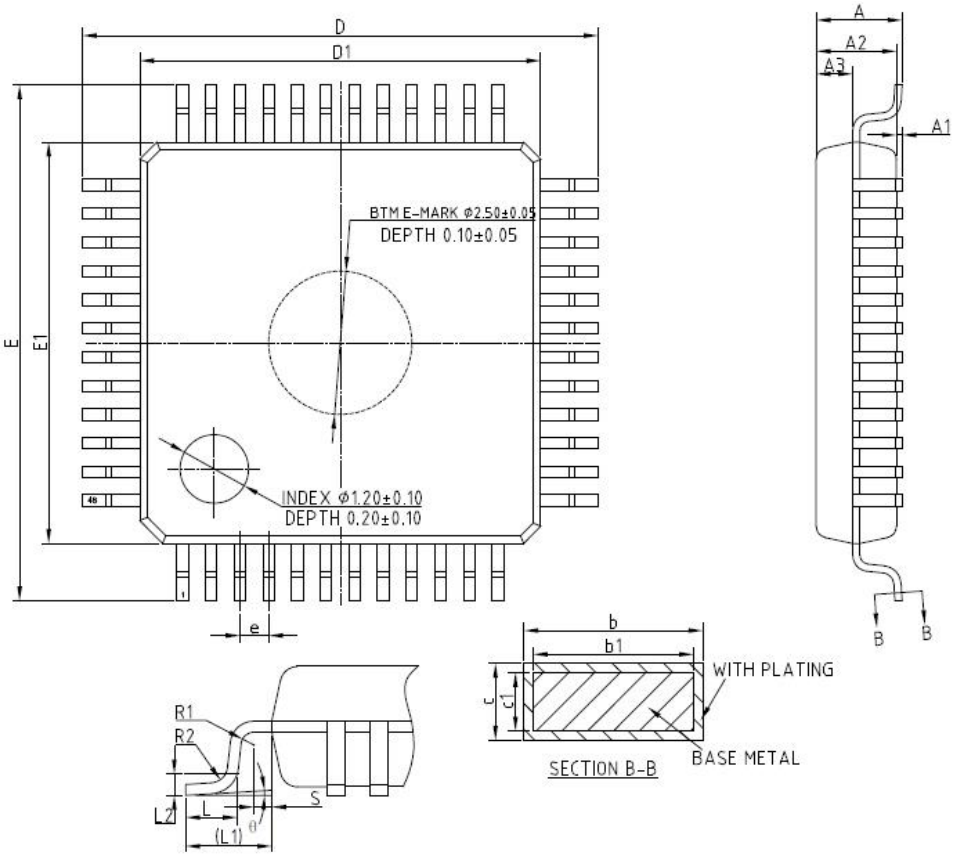


COMMON DIMENSIONS
(UNITS OF MEASURE=MILLIMETER)

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°

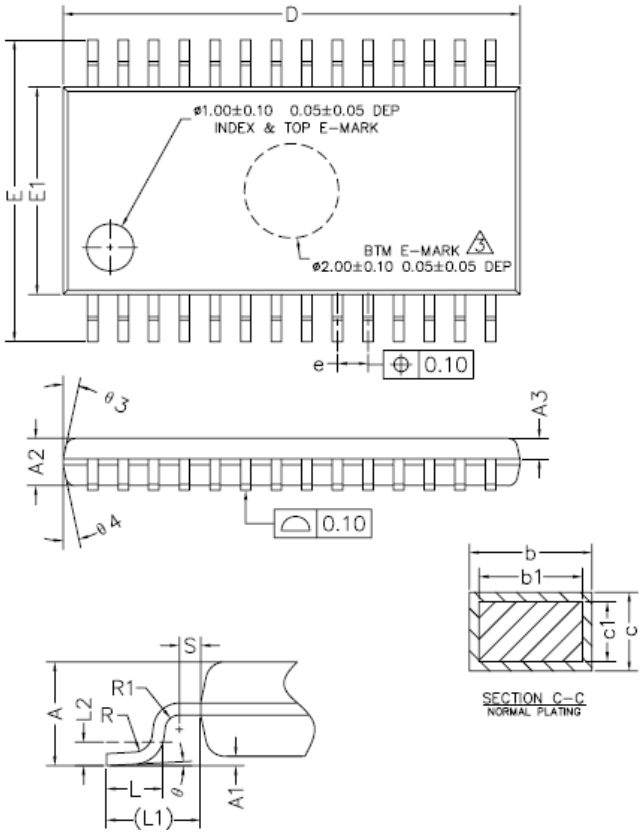
NOTES:
1. ALL DIMENSIONS REFER TO JEDEC STANDARD MS026 BCD
DO NOT INCLUDE MOLD FLASH,GATE BURR OR PROTRUSION.

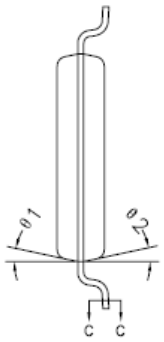
10.1.2 LQFP48



	MILLMETER		
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	0.12	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



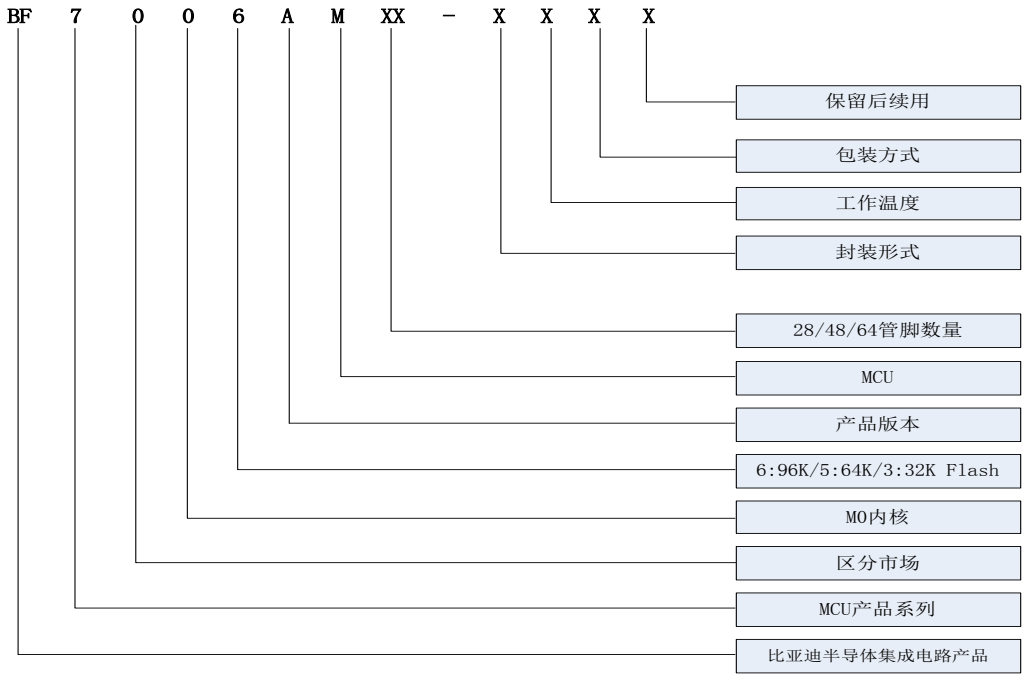


COMMON DIMENSIONS
(UNITS OF MEASURE=MILLIMETER)

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 订货信息

封装形式	工作温度		包装形式	保留后续用
S:SOP	车规级	A：-40℃~+150℃	B:编带	-
T:TSSOP		B：-40℃~+125℃	L:料管	-
M:MSSOP		-	T:托盘	-
L:LQFP		-	-	-
Q:QFN	工业级	K：-40℃~+85℃	-	-
B:BGA		J：-40℃~+105℃	-	-
D:DIP		L：-40℃~+125℃	-	-
-	消费级	P：-25℃~+70℃	-	-
-		Q：0℃~+70℃	-	-



12 免责声明

1、此文档中的信息可以在不通知用户时进行修改及更新
2、比亚迪半导体有限公司将竭尽最大的努力保证本公司产品的高质量与高稳定性。尽管如此，由于一般半导体器件的电气敏感性及易受到外部物理伤害等固有特点，本公司产品有可能在这些情况下出现故障或失效。当使用本公司产品时，使用者有责任遵从安全规则来设计一个安全及稳定的系统环境。使用者可通过去除多余器件、故障预防及火灾预防等措施来避免可能发生的意外、火灾及公共伤害。在用户使用该产品时，请遵从本公司最新说明书上规定的操作步骤来使用该产品。
3、本公司该产品不能及禁止应用在一些需要极高稳定性及质量的特殊设备上，以免导致人员伤亡等意外发生。产品不能应用范围包括原子能控制设备、飞机及航空器件、运输设备、交通信号设备、燃烧控制设备、医药设备、军工设备以及所有安全性设备等等。使用者在以上列举的非产品应用范围内使用时造成的损失与伤害，本公司概不负责。