STR 71x 微控制器参考手册

版本 7

2005 年 11 月

1 简介

这本手册为应用开发者提供了有关怎样使用 STR71X 微处理器存储器和外围设备的完整的信息。

STR71xF 是具有不同内存容量、封装和外围设备的微处理器系列。

有关订购信息、机械和电气设备的特性请参考 STR71x 数据手册。

有关内部闪存的编程、擦除和保护请参考 STR7 闪存编程参考手册。

有关 ARM7TDMI 内核的信息请参考 ARM7TDMI 技术参考手册。

相关文件：

可从 www.arm.com 获得的信息：

ARM7TDMI 的技术参考手册

可从 http://www.st.com 获得的信息：

STR71x 数据手册

ARM7 闪存编程参考手册

AN1775-STR71x 硬件开发入门

上述信息只是有选择的列表，有关 STR71X 应用信息的完整列表可见 http://www.st.com。

目 录

2 存储器

2.1 存储器结构

ARM 本地总线作为主系统总线，连接 CPU、存储器和系统服务模块，而低功耗 APB rev.E 被用作外设总线。

本地总线系统中包括了 CPU，RAM，Flash，外部存储器接口(EMI)和功率、复位和时钟控制单元（PRCCU）。

ARP 桥（APB1 和 APB2）为两组外围设备提供接口。当访问时钟速率低于 ARM7 核的 ARP 外设时，会在 CPU 时钟内自动插入等待状态。

程序存储器、数据存储器、寄存器和 I/O 口都被组织在同一个 4GB 的线性地址空间内。

在存储器中，字节以小端（Little Endian）方式处理，在一个字中的最低序号字节被认为是该字的最低字节，而最高序号的字节是该字的最高字节。

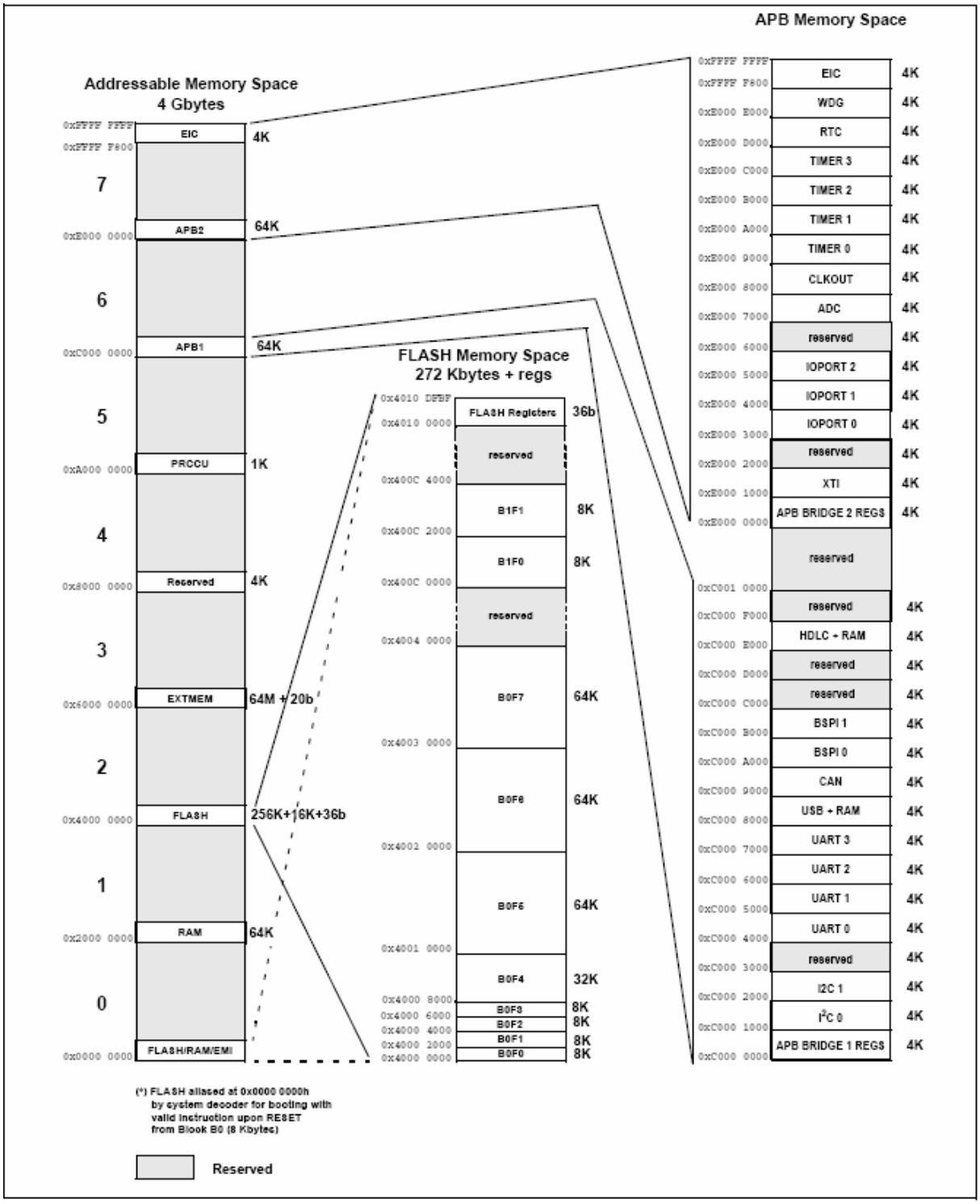
图 1 所示为 STR71x 的存储器映射图。至于外设寄存器的详细地址映射，请参阅相关章节。

可寻址存储空间被分成 8 大块，用存储器地址线 A[31:0]的最高位 A[31:29]进行选择。

- 000=Boot 存储器
- 001=ARM 存储器
- 101=Flash 存储器
- 100=外部存储器
- 100=保留存储器
- 101=PRCCU 寄存器
- 110=APB 桥 1—串行通信外设
- 111= APB 桥 2—系统外设和中断控制器

2.1.1 存储器映射

图 1 存储器映射



2.1.2 存储器段的映射

表 1 存储器段映射

STR71x Memory Blocks	Block Base Address	Section
Boot Memory	0x0000 0000	Section 2.1.5
RAM	0x2000 0000	Section 2.1.6
Flash	0x4000 0000	Refer to STR7 Flash Programming Reference Manual
External Memory Interface (EMI)	0x6000 0000	Section 2.3.7
PRCCU	0xA000 0000	Section 3.7
APB1	0xC000 0000	See table 2
APB2	0xE000 0000	See table 3

2.1.3 APB1 存储器映射

APB1 基地址=0xC000 0000h

表 2 APB1 存储器映射

Pos	STR71x APB1 Peripheral	Address Offset	Peripheral Register Map
0	APB1 Bridge Configuration registers	0x0000	Section 17.3
1	I2C0	0x1000	Section 10.6
2	I2C1	0x2000	Section 10.6
3	Reserved		
4	UART0	0x4000	Section 12.5
5	UART1 + SMARTCARD Interface	0x5000	Section 12.5 and Section 13.6
6	UART2	0x6000	Section 12.5
7	UART3	0x7000	Section 12.5
8	USB RAM	0x8000	Section 14.6
	USB Registers	0x8800	
9	CAN	0x9000	Section 9.7
10	BSPI0	0xA000	Section 11.11
11	BSPI1	0xB000	Section 11.11
12	Reserved		
13	Reserved		
14	HDLC Registers	0xE000	Section 15.4
	HDLC RAM	0xE800	
15	Reserved		

2.1.4 APB2 存储器映射

APB2 基地址=0xE000 0000h

表 3　APB2 存储器映射

Pos	STR71x APB2 Peripheral	Address Offset	Peripheral Register Map
0	APB2 Bridge Configuration registers	0x0000	Section 17.3
1	External Interrupts (XTI)	0x1000	Section 5.4.5
2	Reserved	0x2000	Section 4.2.1
3	IOPORT0	0x3000	Section 4.2.1
4	IOPORT1	0x4000	Section 4.2.1
5	IOPORT2	0x5000	Section 4.2.1
6	Reserved	0x6000	
7	ADC	0x7000	Section 16.5
8	CKOUT	n.a.	
9	TIMER0	0x9000	Section 8.7
10	TIMER1	0xA000	Section 8.7
11	TIMER2	0xB000	Section 8.7
12	TIMER3	0xC000	Section 8.7
13	RTC	0xD000	Section 6.5
14	WDG	0xE000	Section 7.5
15	EIC	0xF800	Section 5.2

注意：EIC在存储器映射中有另外一个映像，它可通过偏移0xFFFFF800来寻址。这可用来从ARM7中断向量（0x00000018）跳转到EIC_IVR寄存器，该寄存器指向有效的中断程序（见第5节）。

2.1.5 启动存储器

当退出复位时有 3 种启动模式可由芯片引脚的配置来选定（见启动配置）。

- **Flash 启动模式**：在这种模式下，Flash 被同时映射到两个存储器段 010 和 000。系统从 Flash 的 0 簇 0 扇区启动。
- **RAM 启动模式**：在这种模式下，RAM 被同时映射到两个存储器段 001 和 000，系统从 RAM 内存启动。这在调试过程中非常有用，RAM 可通过外部 JTAG 控制器或开发系统（仿真器）进行预加载。
- **外部存储器启动模式**：在这种模式下，外部存储器被同时映射到两个存储器段 011 和 000，系统从外部存储器段 0 启动。

2.1.6 RAM

STR71x带有64k字节全静态同步RAM，可按字节、半字（16位）或全字（32位）进行存取。RAM起始地址为0x2000 0000。
在RAM启动模式下（参见2.1.5）RAM起始地址被同时映射到0x0000 0000h和0x2000 0000h。

利用PCU_BOOTCR寄存器中的BOOT[1:0]位，可在运行过程中重新改变RAM映射为RAM启动模式配置。这一点对于管理中断向量和服务程序特别有用，可以把它们复制到RAM中，修改和访问它们，即使在Flash不可用时（也就是闪存编程或擦除时）也可以做到。

2.1.7 Flash

Flash闪存模块按簇（bank）和扇区组织，如表4所示。

表4 Flash闪存模块组织

Bank	Sector	Addresses	Size (bytes)
Bank 0 256 Kbytes Program Memory	Bank 0 Flash Sector 0 (B0F0)	0x00 0000 - 0x00 1FFF	8K
	Bank 0 Flash Sector 1 (B0F1)	0x00 2000 - 0x00 3FFF	8K
	Bank 0 Flash Sector 2 (B0F2)	0x00 4000 - 0x00 5FFF	8K
	Bank 0 Flash Sector 3 (B0F3)	0x00 6000 - 0x00 7FFF	8K
	Bank 0 Flash Sector 4 (B0F4)	0x00 8000 - 0x00 FFFF	32K
	Bank 0 Flash Sector 5 (B0F5)	0x01 0000 - 0x01 FFFF	64K
	Bank 0 Flash Sector 6 (B0F6)	0x02 0000 - 0x02 FFFF	64K[1)]
	Bank 0 Flash Sector7 (B0F7)	0x03 0000 - 0x03 FFFF	64K[1)]
Bank 1 16 Kbytes Data Memory	Bank 1 Flash Sector 0 (B1F0)	0x0C 0000 - 0x0C 1FFF	8K
	Bank 1 Flash Sector 1 (B1F1)	0x0C 2000 - 0x0C 3FFF	8K
Flash Control Registers	Flash Control/Data Registers	0x10 0000 - 0x0010 0017	24
	Flash Protection Registers	0x10 DFB0 - 0x0010 DFBC	12

[1)] 在128K版本中不可用。

Bank 0 是用来存放程序的存储器。扇区B0F0-B0F7用作Boot扇区，可以被写保护以免被误写。

Bank 1 包括16k字节数据存储器：它分为2个扇区（每个8k字节）。可在此区域进行编程应用数据。

可以对Bank 0和Bank 1 单独进行编程，也就是说，可从一个簇读出而对另一个簇写入。

对Flash存储器可以进行保护，防止各种不想要的存取（读/写/擦除）。

对闪存进行编程可采用在电路中编程和在应用中编程方式，请参阅STR7闪存编程参考手册。

2.1.7.1 Flash突发和低功耗模式

STR71x的Flash闪存有两种读写方式：突发模式和低功耗模式。利用PCU_PWRCR寄存器中的FLASHLP位可以打开或关闭突发模式。

在突发模式中，可在最高器件工作频率下以零等待进行连续读写，不连续的访问需用1等待。

当突发模式关闭时，Flash工作在低功耗(LP)模式下，所有访问的最高速率为33MHz零等待。

2.1.7.2 Flash掉电模式

根据应用需要，可以选择停止模式或低功耗等待中断模式来关闭Flash电源（参阅STR7闪存编程手册）。否则，在停止模式下，闪存自动降低功耗，并可在唤醒后立即进行读操作。

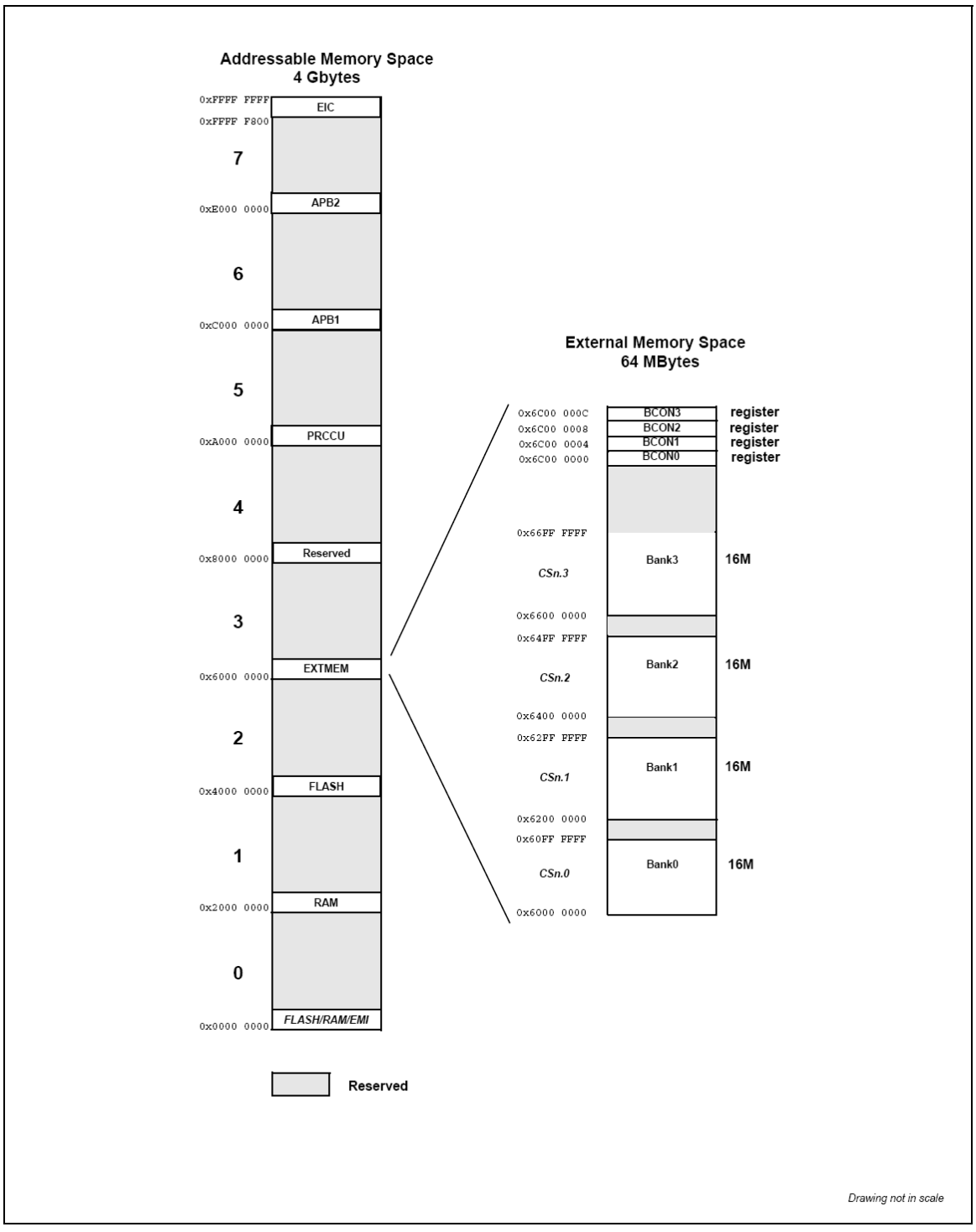
2.1.8 外部存储器

外部存储器接口可提供接口给外部存储器件，例如ROM，FLASH， SRAM或外接外围设备。

根据对A[26：25]的解码将外部存储器空间划分为4个段（Bank），每一个段可寻址到16M字节的外部存储器。对这些段之一进行寻址时将激活相应的芯片选择输出CSN[3：0]。在芯片外可得到EMI地址总线A[23:0]，以及控制信号WEN[1：0]（字节写操作使能）和RDN（读操作使能）。

将数据读出/写入外部存储器段所需要的传输时间长度可通过对各段的控制寄存器的编程来控制，编程能确定访问一个段所用的等待状态的数目。存取时间也与所选的外部总线宽度有关。每个寄存器也用一个标志位来指明某个特定存储器段可否被存取。

图 2. 外部存储器映射



2.2 启动配置

在 STR71x 中提供了三种启动模式，可通过三个输入引脚 BOOTEN，BOOT0 和 BOOT1 来激活。

BOOTEN 是一个专用引脚，正常情况下必须通过一个 10K 的电阻接地。当 BOOTEN=0 时，器件处于 FLASH 启动模式，其 BOOT[1:0]管脚不起作用，可以是任意值。

当 BOOTEN=1 时，当外部 RSTINn 引脚被释放后，其 BOOT[1:0]管脚电平值在第二个 CK 时钟的上升沿后被锁存。这些值用来构造设置器件的 BOOT 模式，如表 5 所示。

表 5. 启动模式

BOOTEN	BOOT1	BOOT0	Mode	Boot Memory Mapping	Note
0	any	any	FLASH	FLASH mapped at 0h	• System executes code from Flash
1	0	0			
1	1	0	RAM	RAM mapped at 0h	• System executes code from internal RAM • For Lab development.
1	1	1	EXTMEM	EXTMEM mapped at 0h	• System executes code from external memory

注意：当启动为如下设置 BOOTEN=1，BOOT1=0 和 BOOT0=1 时，系统启动处于保留的 ST 启动模式。

在下面的部分中，用简写 B[1：0]表示 BOOT[1：0]引脚，且若不另外说明时，认为 BOOTEN 为 1。

2.2.1 FLASH 启动模式

这是标准的操作模式；若 BOOTEN=0，进入这种模式不必控制 BOOT[1:0]。

这种模式也可以通过强迫其外部管脚 B1=0 和 B0=0 来进入。当外部 Reset 信号被释放时，这种状态将被锁存。

用户需要自己开发在应用编程（IAP）程序，并确定不同扇区的最佳用途。例如，B0F0 和 B0F1 可用来作为用户的启动加载器。当采用这种模式时，至少 B0F0 要预先被编程，因为系统启动是从 Flash 的 B0F0 扇区开始的。

注意：为了保证最大程度的安全，建议闪存擦除和编程子程序不要保存在 Flash 内部，而是在 ICP 程序开始时将其从外部工具加载到 RAM 中。

2.2.2 RAM 和 EXTMEM 启动模式

提供 RAM 模式是为了便于应用开发。RAM 模式通过强迫外部引脚 B1=1 和 B0=0 进入，而 EXTMEM 模式通过强迫外部引脚 B1=1 和 B0=1 来进入。当外部复位释放后，引脚状态锁存在第二个 CK 时钟脉冲上。

在 EXTMEM 模式下，系统从外部存储器的段 0（CSN0 有效）启动。外部存储器同时也映射到地址 0h(见 PCU_BOOTCR 寄存器)。

在RAM模式下，系统启动从内部RAM执行，RAM在地址0h处也可见（见PCU_BOOTCR寄存器）。必须事先在RAM中预加载启动代码，比如通过一种开发系统（MultiICE™ 或同类）。

要由用户的应用系统来为 B0，B1 提供适当的信号，因为在复位阶段没有专门的控制器 I/O 来控制 B0 和 B1。

2.3 外部存储器接口（EMI）

2.3.1 EMI 总线接口信号描述

EMI 外部总线信号详见表 6。

表 6. EMI 总线接口信号

Name	I/O	Description
A[23:0]	O	External interface address bus
D[15:0]	I/O	External intere data bus
RDn	O	Active low read signal for external memory. It maps to the OE_N input of the external components
WEn.0	O	External write enable signal. When '0', enables write operation to 8 LSBs (bits 7:0) of external memory
WEn.1	O	External write enable signal. When '0', enables write operation to bits 15:8 of external memory
CSn.0	O	Active low chip select for bank 0.
CSn.1	O	Active low chip select for bank 1.
CSn.2	O	Active low chip select for bank 2.
CSn.3	O	Active low chip select for bank 3.

2.3.2 EMI 存储器映射

EMI 存储器映射如表 7 所示。每个段使用 A[23:0]中的全部 24 位，提供 16M 字节的可寻址空间。外部存储器空间的基地址为 EMI_BASE = 0x6000 0000。

表 7. EMI 存储器映射

Address Range	Description	Addressable Size (Bytes)	Bus Width (bit)
0x6000 0000 - 0x60FF FFFF	Bank 0 - BOOT (CSn.0)	16M	16 bit access only[1)]
0x6200 0000 - 0x62FF FFFF	Bank 1 (CSn.1)	16M	8/16 SW Selectable
0x6400 0000 - 0x64FF FFFF	Bank 2 (CSn.2)	16M	8/16 SW Selectable
0x6600 0000 - 0x66FF FFFF	Bank 3 (CSn.3)	16M	8/16 SW Selectable
0x6C00 0000 - 0x6C00 0010	Internal Registers	n/a	16 bit access only

注意 1：如果外部存储器用于引导操作，那么它必须为 16 位，因为 CSN0 存储器段已经硬件连接为只能提供 16 位存储器接口。

2.3.3 EMI 可编程时序

EMI 中的每个存储器段可以为其任何一个读周期或写周期添加可编程的等待状态数目（最高为 15）：等待个数是软件可配置的，经由 EMI_BCONx 寄存器(x=0,..,3)中的 C_LENGTH 位场来进行。

根据不同的外部存储器数据总线宽度选择，可有两种访问类型：单周期访问和多周期访问。

单周期访问：

对于实现为单个外总线操作的写周期（例如，一个16位写操作，写入到配置成16位宽的段）来说，一个单一访问需要（等同于片选信号有效的长度）的总长度（以EMI内部时钟单位来衡量），将取决于在哪个存储器段上执行该周期。可用下面的公式进行计算：

CSn[3:0] length (Write) = C_LENGTH+1

这个值也是产生的等待状态的数目。因此，对于性能评价来说，每个外部访问将持续(Int_Access_length + CSn[3:0] length)个MCLK周期，其中，Int_Access_length可在ARM7程序员手册中找到，其取决于访问的类型。

对于转换为单一外部总线操作的读周期（例如，从配置为16位宽的段中进行16位读），一个单一访问（等于片选信号宽度）所要求的总长度（以EMI内部时钟单位来计量），可用下面的公式进行计算：

CSn[3:0] length (Read) = C_LENGTH+2

如前所述，每个外部访问将持续(Int_Access_length + CSn[3:0] length)个MCLK周期。

多周期访问：

对于转换为多个（N个）外总线周期（例如，当一个32位写操作写入到配置成8位宽的区域时，N=4）的读周期或写周期来说，总的外部总线周期持续时间可由下面的公式得到：

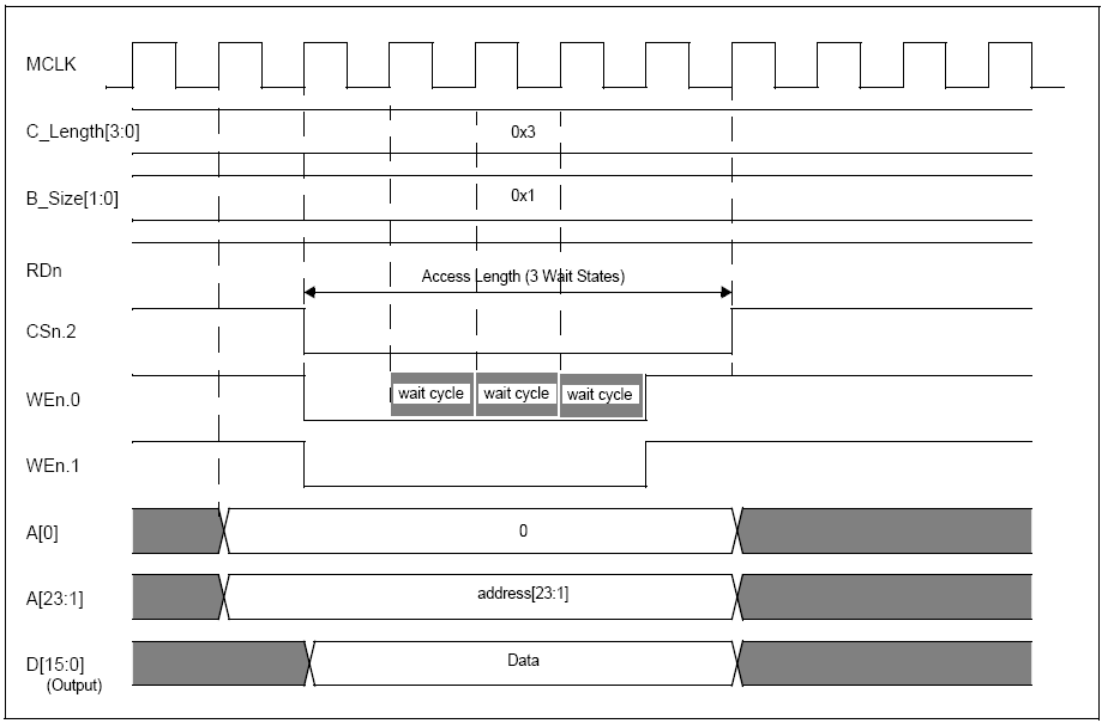
Total CS length (Read/Write Bank x) = (Length of the single cycle)*(N)

在这种情况下，总的访问持续时间为（Int_Access_length + Total CS length)个MCLK周期。

2.3.4 写访问举例

在图 3 中，将在 16 位外部存储器中执行 16 位的写操作。从图可知，两条外部写选通脉冲在写周期的持续时间中都有效。这是个单周期写操作，因此外部地址中的 LSB 没有改变，并使用 5 MCLK 个周期来完成操作（C_LENGTH=3）。

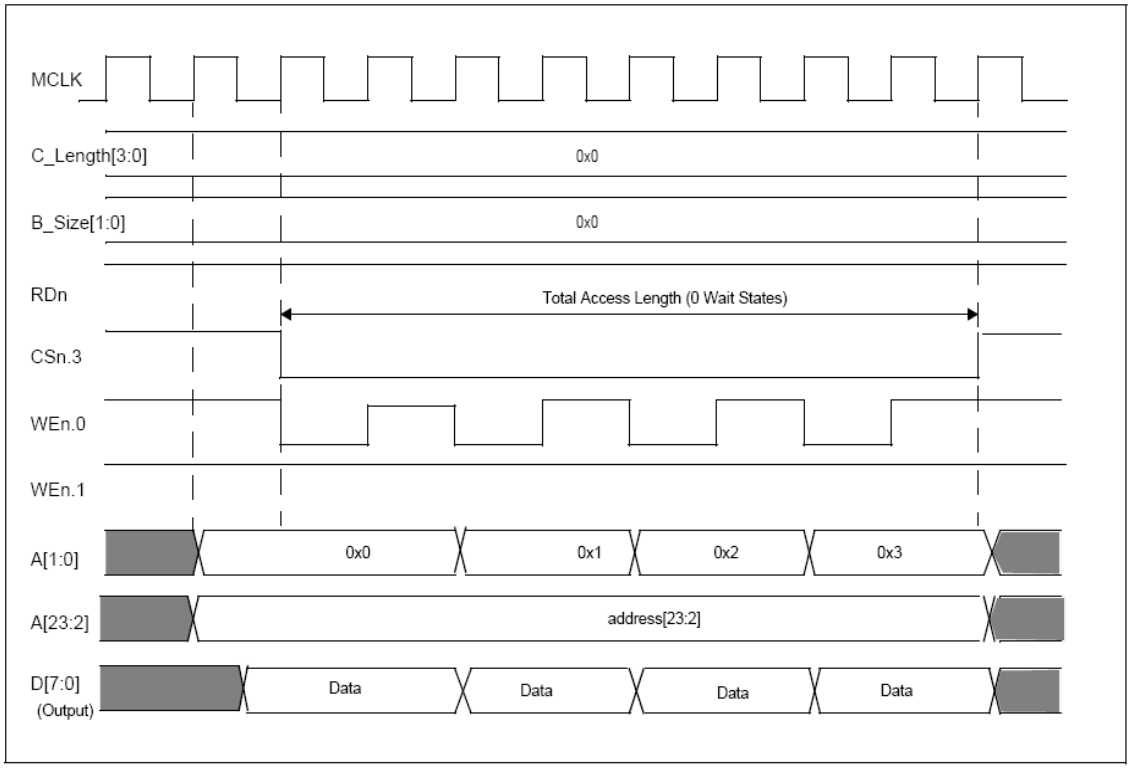
图 3. 16 位存储器上的 16 位写周期，3 等待状态



在图 4 中，将执行一个 32 位的写操作。然而在这个例子里，外部总线宽度只有 8 位（B_SIZE=0）。因而需要 4 个写周期才能完成执行全部的 32 位写操作。外部地址的最低两位由读-写控制器修改，每次写都不同，第一个写操作从“00”开始，然后每次写递增，直至“11”结束。

由于外部数据总线中只用到低 8 位，每个写操作都只有 WEn.0 有效。

图 4.　8 位存储器上的 32 位写周期，没有等待状态

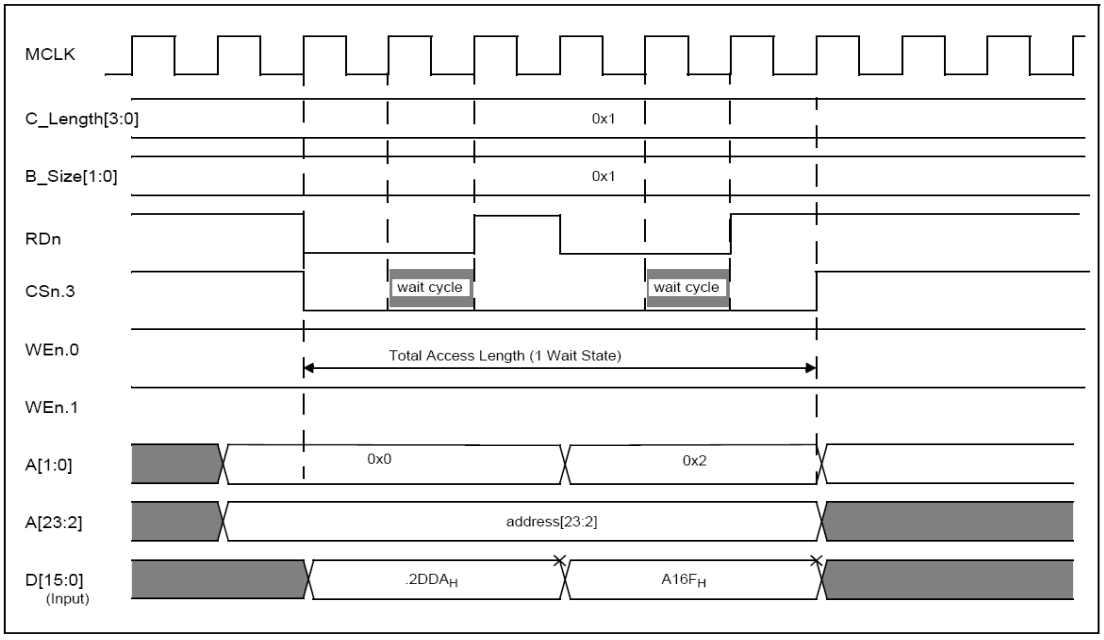


在本例中使用了 CS3n；需要注意的是，CS3n 保持有效直至所有的写周期都完成。而 WEn.0 信号在每个写周期之间都有 1 个 MCLK 周期长的无效间隔。

2.3.5 读访问举例

下面的图表示一个基本的读操作。在读操作情况下只需要一个外部读出选通脉冲（RDn）。在这个例子中，将执行32位的读操作。然而其外部总线尺寸为16位(B_SIZE = 1)，因此，EMI的外部设备要执行两个连续的读操作。第一个读操作的结果锁存在EMI块的内部（本例中为“2DDA”，即低16位数据），以便在执行完第二个16位读操作后可返回正确的32位数据（即“A16F2DDA”）。

图 5.　16-比特存储器中的 32-比特读循环，1 等待状态



对于第一个读操作，“00”被分配给 A[1:0]，对于第二个操作，其增量为 2（即“10”）。

2.3.6 寄存器描述

2.3.6.1 Bank n 配置寄存器（EMI_BCONn）

基地址：0x6C00 0000

地址偏移：

– BCON0: 0x00h – BCON1: 0x04h

– BCON2: 0x08h – BCON3: 0x0Ch

复位值：

– BCON0: 0x803Dh – BCON1: 0x003Dh

– BCON2: 0x003Eh – BCON3: 0x003Ch

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BE	reserved									C_LENGTH[3:0]				B_SIZE[1:0]	
rw	-									rw				rw	

Bank n 配置寄存器（BCONn）是一个用于设置 Bank n 操作的 16 位读/写控制寄存器。BCONn 控制比特描述如下：

比特 15 BE：Bank n 始能

0：该段（Bank）不可用

1：该段可用

比特 5:2 **C_LENGTH[3:0]:**周期长度

C_LENGTH 域选择将被插入到任何一个在 Bank n 中执行的读/写周期中的等 待状态的数目。任何读周期或写周期中的总 CS 长度将等于 C_LENGTH+1 个 EMI 内部时钟的周期。

0x0h = 0 等待状态

0x1h = 1 等待状态

0x2h = 2 等待状态

....

0xFh =15 等待状态

比特1:0 **B_SIZE[1:0]:**总线宽度

B_SIZE域定义了用于访问Bank n的有效外部总线宽度。

0x00b =8-比特

0x01b =16-比特

0x10b =保留（不能用）

0x11b =保留（不能用）

2.3.7 EMI 寄存器映射

图 8. EMI 寄存器映射

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	BCON0	BE	reserved									C_LENGTH[3:0]				B_SIZE[1:0]	
4	BCON1	BE	reserved									C_LENGTH[3:0]				B_SIZE[1:0]	
8	BCON2	BE	reserved									C_LENGTH[3:0]				B_SIZE[1:0]	
C	BCON3	BE	reserved									C_LENGTH[3:0]				B_SIZE[1:0]	

基地址=0x6C00 0000

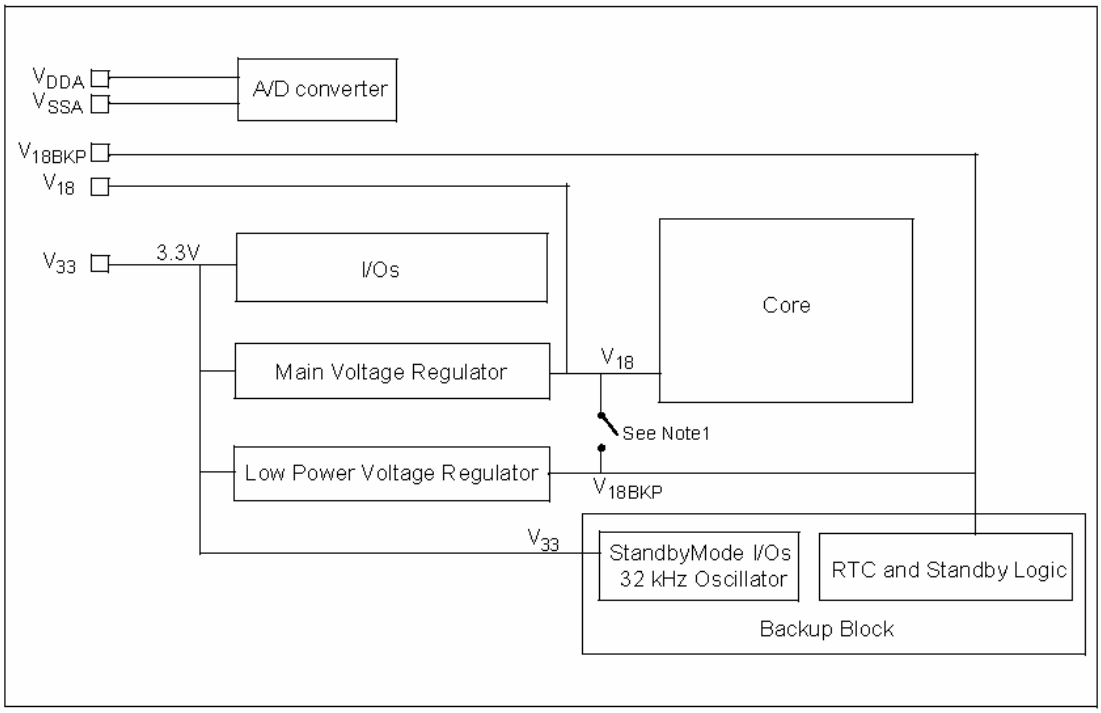
3 电源、复位和时钟控制单元

3.1 电源

本芯片用外部3.3V电源供电，所有I/O均可工作在3V。有两个内置稳压器：主稳压器和低功耗稳压器，为核心逻辑电路提供工作电压。

引脚V_{DDA}和V_{SSA}为A/D转换器提供参考电压。

图6. 电源概要



注释1：在正常工作模式下，V18与V18BKP短接。在待机模式下，V18区域与V18BKP区域是断开的。

注释2：两个V18引脚必须连接外部稳压电容。不支持对V18引脚连接外部1.8V电源的方式（参见STR71x数据手册）。

3.1.1 外部供电电压V18BKP的选用

在待机模式（见3.5.5节）中，主稳压器关闭，低功耗稳压器向备用模块供电。如果除了V33还有外部1.8V电源（V18BKP）可用时，在软件控制下可以使低功耗稳压器旁路，提高了系统的供电效率。

此种情况下V18BKP不可以直接连接，必须通过一个二极管连接，以避免当器件不处于待机模式时出现冲突。

由于要为I/O供电，V33在待机模式下不能关断，因为这时nSTDBY， nRSTIN，WAKEUP引脚必须保持有效。

3.2 稳压器

3.2.1 主稳压器

通过位于I/O环以内的P沟道晶体管，主稳压器(MVR)可以向器件提供足够的电流，使其运行在任意模式下。它有一个带隙基准源进行热补偿，有100 μA（典型值）的静态功耗。

注意：

1. MVR在待机模式下自动关断。

2. 当器件进入停止模式或LPWFI模式，MVR可被设为自动关断（用PCU_PWRCR寄存器的LPVRWFI位），从而使低功耗稳压器作为唯一的电源。

3. 通过PCU_PWRCR寄存器的LPVRWFI位MVR可被关断。在此设定下，器件仅由低功耗稳压器供电，最大允许工作频率为1MHz，且PLL被禁用。

4. 当PCU_PWRCR寄存器的VROK位由硬件置位时，主稳压器电源输出电压可保证稳定在规定电压值上。

3.2.2 低功耗稳压器

只有当处于待机、停止或LPWFI模式时才应该使用分离的低功耗稳压器。其设计与主稳压器不同，产生一个不稳定、没有热补偿的大约1.6V的电压，其输出电流通常不足以让器件工作在正常模式下。因为这个限制，当主稳压器关闭时PLL被自动禁用，最大允许工作频率为1MHz。

在待机模式下（见3.1.1节），当为了维持实时钟和唤醒逻辑电路工作用外部电源通过V_{18BKP}引脚为芯片提供1.8V供电时，低功耗电压调节器也可以被关掉。

对于主电压调节器和低功耗电压调节器来说，电压的稳定通过外接电容来实现，电容分别连接到V_{18}（主调节器）和V_{18BKP}（低功耗调节器）引脚。主电压调节器的推荐最小电容值为10 μF（低串联电阻的钽电容）加上33nF陶瓷电容，低功耗电压调节器的为1 μF。注意芯片和电容的距离要尽量短，还要注意将串联电感限制在60nH以内。

注释：

主电压调节器和低功耗电压调节器各自有一个低电压检测电路，当相应的调节电压值(V_{18} 或 V_{18BKP})降低到低于1.35V (+/- 10%)时，检测电路会使器件保持在复位状态。

3.3 复位

上电时，nRSTIN引脚必须被外部复位电路拉低直到V_{33}达到数据手册要求的最小值。

当下列事件发生时复位管理器使MCU复位：

- 硬件复位，由nRSTIN引脚低电平触发。
- 软件复位，通过设定RCCU_SMR寄存器的HALT位产生（该位用RCCU_CCR寄存器的SRESEN位和ENHALT位来使能）。
- 供电电压降落至低于任一电压调节器的低电压检测电路的阈值（参考STR71x数据手册）。
- 看门狗计数终了条件。
- 器件处于STANDBY模式下WAKEUP引脚的触发（参考3.5.5）。
- 器件处于STANDBY模式下实时时钟闹钟的触发（参考3.5.5）。

引起最近复位的事件在RCCU_CFR寄存器中标示出来：相应的位被置位。硬件触发的复位和低电压检测器复位将使所有的位复位。

硬件复位会覆盖掉其他情况，强行使系统复位。复位状态下，内部寄存器置为其设定的复位值。I/O引脚进入复位配置。

来自nRSTIN引脚的复位是异步的：只要一变为低复位周期即被启动。

若看门狗模式被打开，假如在设定时间长度结束而没有向相应寄存器写入特定的代码，片上定时器/看门狗就会产生一个复位（参见看门狗说明）。

nRSTIN引脚再次变高后，在退出复位状态前需要计数2048个CLK时钟加上8个CLK2（参考3.4）（有可能多加一个CLK时钟周期，这取决于nRSTIN引脚的上升沿和CLK第一个上升沿之间的延迟），参考图7。

复位阶段结束时，可程序计数器被置为由存储器中位置0x0h处的复位向量给出的地址。

图7 一般复位时序

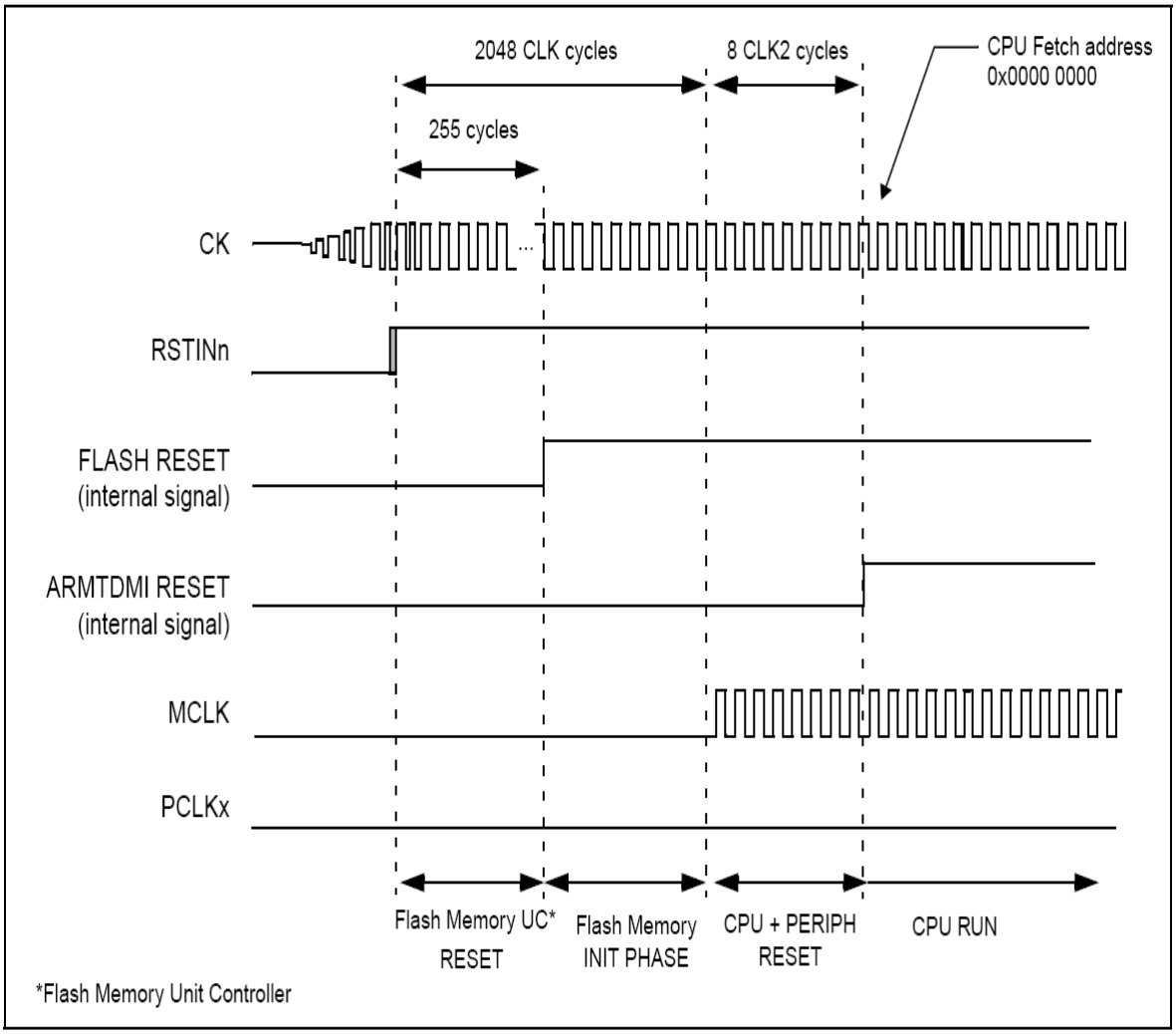
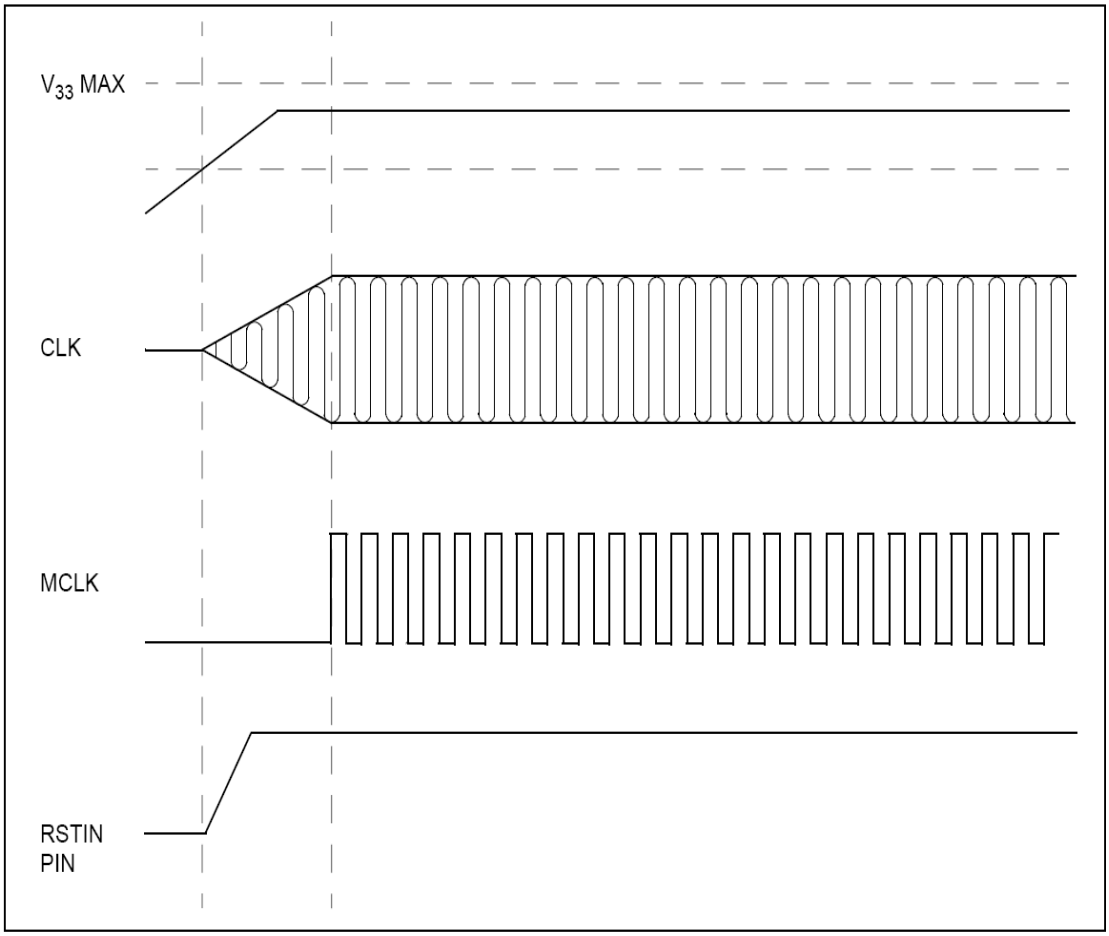


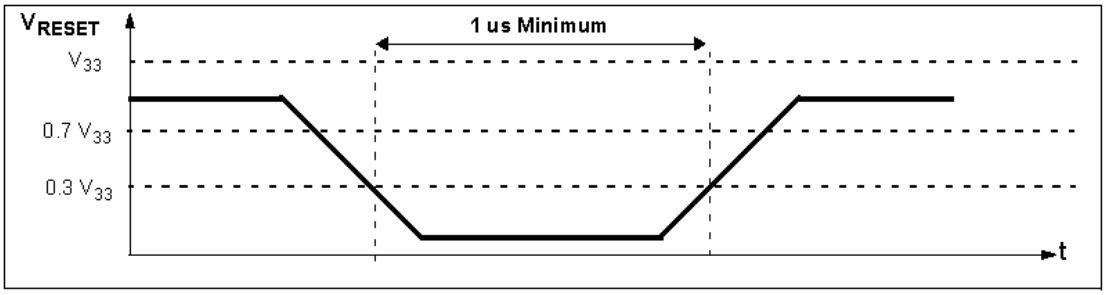
图8 时钟启动次序和复位时序



3.3.1 复位引脚时序

为了改善器件的抗噪声性能，复位输入引脚有一个带有滞环特性的施密特触发器输入电路。虚假的复位事件被模拟滤波器屏蔽掉，从而保证在nRSTIN引脚上的所有宽度小于100ns的短脉冲（单个突发）不会被系统当作有效的复位脉冲。另一方面，建议在nRSTIN引脚上提供一个持续时间至少为1 μs的有效复位脉冲，来确保该异步脉冲被完整锁存。这意味着所有宽度在100ns到1 μs的脉冲对于器件的影响是不确定的：有可能被认定有效，也可能被过滤掉。

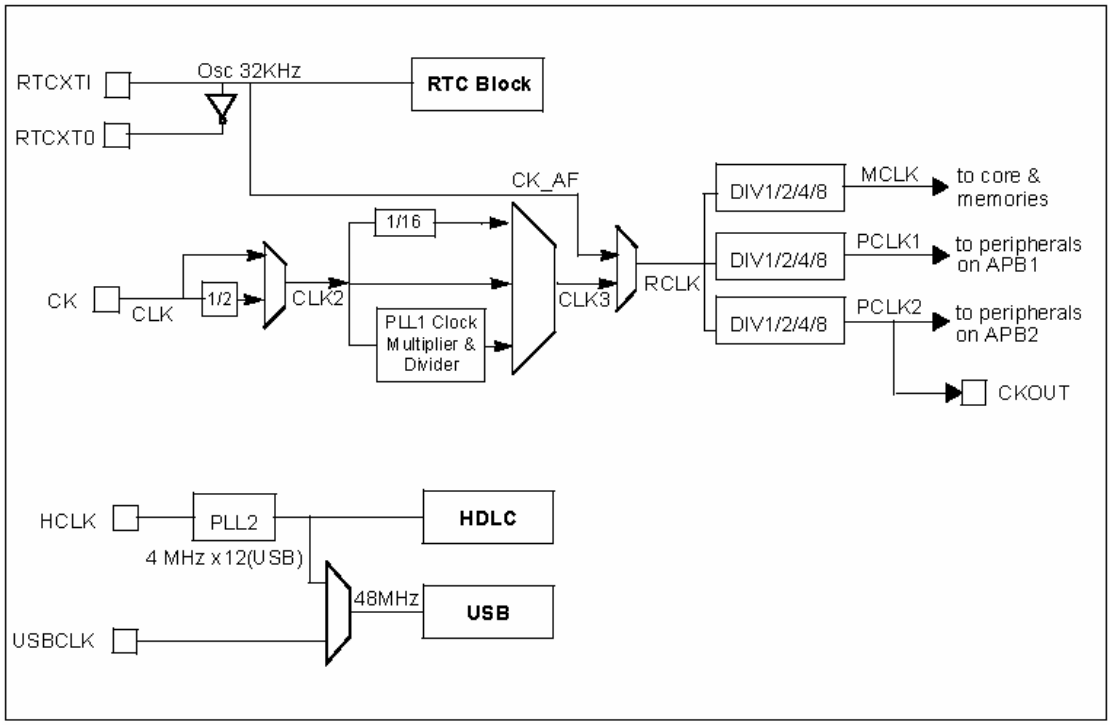
图9 建议施加在nRSTIN引脚的信号



3.4 时钟

下图给出STR71x时钟分布图。

图10 时钟分布图



源时钟CLK通过CK引脚提取自外部振荡器。在低功耗模式下这个时钟可被关断（见3.5节）。

系统PLL (PLL1)用于内部倍频，产生合适的工作频率。RCLK是系统PLL的输出，或者是替换源CK_AF (32KHz RTC clock)，后者需要被使能。

器件内存在几个时钟不同的时钟域：

- 主时钟MCLK区域，包括CPU，内部存储器，外存储器接口，PRCCU寄存器（不包括以下注释2说明的RCCU寄存器）。
- PCLK1区域，包括APB1外围设备（串行通信外设），列在存贮器映射表中。
- PCLK2区域，包括APB2外围设备（系统外设），列在存贮器映射表中。

每个区域都可以通过编程各时钟分频器而独立使用不同的频率。由于CPU子系统有专门的分频器，允许软件动态改变CPU工作频率，从而能根据应用需要来调整计算速度和功耗，又能维持外设的稳定工作。

映射在APB存储器空间的片上外设使用独立于MCLK的RCLK输出的2，4，或8倍分频。当访问这些外设的寄存器时，总线桥会自动加入等待状态。

注意时钟可独立地对每个外设使能或关闭。每个外设也可在软件控制下复位（参见17节）。

注释：

1. 如果CPU时钟MCLK比外设时钟 PCLK1和 PCLK2慢，则禁止访问相应外设寄存器。
2. 如果MCLK时钟分频器被设定的预分频值不是1（即RCLK频率与MCLK频率不同），就不可能访问RCCU寄存器，因为它们总由RCLK提供时钟。
3. 对非密集操作，可以关闭PLL1，且可以对CLK2进行16分频，从而在维持快速中断响应的同时允许低功耗工作。
4. 系统模块(ARM7TDMI、PRCCU、片上存储器和总线桥)由MCLK (总线时钟)驱动，不能用软件关闭，目的是保证基本功能。

一个32KHz振荡器用来维持实时时钟(RTC)，其具有可编程WAKEUP报警。在不需要时也可被设为无效；当有效时不会受到任何低功耗模式转换的影响。

HDLC外设可选择从外部引脚接收基准时钟，可动态地改变频率而与CPU操作无关。内部锁相环 (PLL2)允许使用低频外部信号，从而减少功耗和噪声。

USB接口需要精确的48 MHz时钟基准。可以通过外部USBCLK引脚引入，或用内部PLL2乘于一个外部低速基准产生，假如PLL2不被HDLC接口使用。

注意：假如不用USB接口，设置PCU_PLL2CR寄存器的0，1，2位来关闭PLL2（降低功耗）。

警告：为减少功耗，**RCCU_PER**寄存器的0，1，3位在初始化阶段时必须被应用软件复位。这些位在复位时被硬件使能，仅仅是为了测试的目的。

3.4.1 PLL1时钟倍频器

CLK信号驱动一个可编程的除二电路。如果RCCU_CFR寄存器中的DIV2控制位被置位（复位状态）CLK2等于CLK除于2，假如DIV2被复位CLK2等于CLK。实际上，除二电路的使用是为了确保50%占空比信号。

当PLL有效时，它将CLK2乘于12，16，20，或24，取决于RCCU_PLL1CR寄存器MX[1:0]位的值。被乘的时钟然后被一个1到7的因数除，由RCCU_PLL1CR寄存器的DX[2:0]位的值确定。

当DX[2:0]位被编程为111，且RCCU_PLL1CR寄存器的FREEN位被设为1，PLL环路开放，提供一个慢速回馈时钟，其取决于MX[1:0] 和 FREF_RANGE位（见3.4.2.1和表9）。假如改为DX[2:0]=’111’ 和 FREEN为‘0’，则PLL被关断。

倍频器包含对CLK2 和 PLL时钟输出的频率比较器，用来验证PLL时钟处于稳定（锁定状态）。当进入锁定状态时RCCU_CFR寄存器的LOCK位被置位，并且只要PLL是锁定的就维持为1。假如由于某种原因（例如MX[1:0]位的改变、PLL或CLK2的停止或重启等等）失去原来锁定的编程频率，该位就重新变为0。只有当系统LOCK位是1时才有可能选择PLL时钟。如果LOCK位返回0，即使CSU_CKSEL位是1系统时钟也切换回到CLK2。PLL的选择还会受到主电压调节器状态的限制：仅当PCU_PWRCR寄存器的VROK位是1时，即电压调节器提供了一个稳定的供电电压，PLL才可以被选择（参见电压调节器指标）。设定RCCU_CFR寄存器的CSU_CKSEL位允许选择倍频器时钟作为系统时钟，但前述的两种条件必须满足。

当编程PLL倍频器和分频器因数时，注意不要超过每个时钟的最大允许工作频率值。请参考数据手册中的参数。

MCLK没有最低的限制。然而，当系统时钟频率太低时，有些外设就会产生错误的操作。

3.4.1.1 PLL自由运行模式

PLL能够提供低精度时钟PLLCLK，对较慢的程序有用。频率范围从125kHz到500kHz，取决于MX[1:0]位和FREF_RANGE。这个模式由RCCU_PLL1CR寄存器的FREEN位和DX[2:0]位使能。当PLL关断且FREEN位是1，即所有这四位都被置位时，PLL提供这种时钟。此时钟的选择仍由CSU_CKSEL位管理，但不受RCCU_CFR寄存器的LOCK位和 PCU_PWRCR寄存器的VROK位限制。为了避免PLL时钟不可预测的行为，用户只有在PLL时钟不是系统时钟，即当CSU_CKSEL位是0时，才可设置和清除自由运行模式。

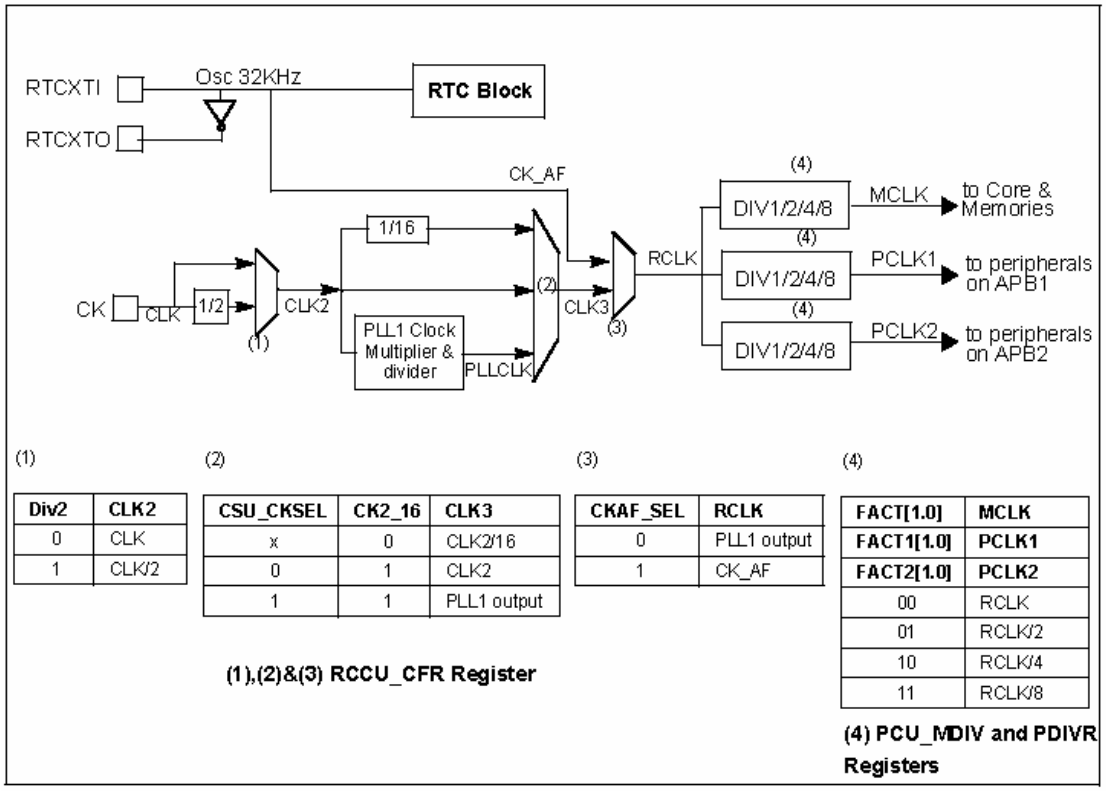
表9 PLL自由运行模式频率

	Free Running Mode frequency	
MX[1:0]	FREF_RANGE=0	FREF_RANGE=1
'01, '11'	125 kHz	250 kHz
'00, '10'	250 kHz	500 kHz

3.4.2 设置时钟

下图描述了用于设定时钟的PRCCU寄存器的编程。

图11 PRCCU编程



(1)

Div2	CLK2
0	CLK
1	CLK/2

(2)

CSU_CKSEL	CK2_16	CLK3
x	0	CLK2/16
0	1	CLK2
1	1	PLL1 output

(3)

CKAF_SEL	RCLK
0	PLL1 output
1	CK_AF

(1),(2)&(3) RCCU_CFR Register

(4)

FACT[1.0]	MCLK
FACT1[1.0]	PCLK1
FACT2[1.0]	PCLK2
00	RCLK
01	RCLK/2
10	RCLK/4
11	RCLK/8

(4) PCU_MDIV and PDIVR Registers

3.4.2.1 时钟配置复位状态

在复位状态下，RCCU_CFR值为8008h，PCU_MDIVR和PCU_PDIVR寄存器的值（FACT位）是0000h。因此，在复位状态，时钟配置为DIV2=1，CK2_16=1，MCLK、PCLK1和PCLK2都工作在外部时钟CLK2。

3.4.2.2 典型时钟配置示例

例如，为通过外部16 MHz的CK获得48MHz的MCLK，

- 设置MX[1:0] = 01，将CLK2乘12
- 设置DX[2:0] = 001，除2
- 设定FREF_RANGE = 1

3.4.2.3 PRCCU工作模式

表10 **PRCCU**工作模式

MODE	RCLK	DIV2	CSU_ CKSEL	MX[1:0]	DX[2-0]	CK2_16	CKAF_ SEL	WFI_C KSEL
PLL x 24	CLK2 x 24/N	1	1	1 0	N={DX}+1	1	0	X
PLL x 20	CLK2 x 20/N	1	1	0 0	N={DX}+1	1	0	x
PLL x 16	CLK2 x 16/N	1	1	1 1	N={DX}+1	1	0	x
PLL x 12	CLK2 x 12/N	1	1	0 1	N={DX}+1	1	0	x
SLOW 1	CLK2	1	0	X	X	1	0	x
SLOW 2	CLK2 /16	1	X	X	X	0	0	x
SLOW3	CK_AF	X	X	X	X	X	1	x
WFI	**If LPOWFI=0, no changes occur on RCLK**							
LOW-POWER WFI 1	CLK2 /16	1	X	X	X	X	X	0
LOW-POWER WFI 2	CK_AF	1	X	X	X	X	X	1
RESET	CLK2	1	0	00	111	1	0	0

注释：关于低功耗模式SLOW,LPWFI请参见3.5节。

3.4.3 中断的产生

对下列事件PRCCU产生一个中断请求。

表11 PRCCU中断

Event	Description	Event trigger	Interrupt Mask	Event Flag
CK_AF Switching	CK_AF selected or deselected as RCLK source	CK_AF bit in RCCU_CCR register toggles	EN_LOCK bit in RCCU_CCR register	CKAF_I bit in RCCU_CFR register
CLK2/16 Switching	CLK2/16 selected or deselected as RCLK source	CK2_16 bit in RCCU_CFR register toggles	EN_CK2_16 bit in RCCU_CCR register	CK2_16_I bit in RCCU_CFR register
Lock	PLL1 becomes locked or unlocked	LOCK bit in RCCU_CFR register toggles	EN_LOCK bit in RCCU_CCR register	LOCK_I bit in RCCU_CFR register
Stop	CLK restarts after waking up from Stop mode		EN_STOP bit in RCCU_CCR register	STOP_I bit in RCCU_CFR register

当这些事件中任何一个发生时，RCCU_CFR寄存器的相应等待位变为1，中断请求送到中断控制器。需要由用户去复位等待位，此操作作为中断子程序开始的第一条指令。等待位只能被清除（通过写1清除）。每个中断可以通过复位RCCU_CCR寄存器的相应位而被屏蔽掉。

3.5 低功耗模式

3.5.1 慢速模式

在慢速模式下，可以通过降低主时钟来减少功耗。在慢速模式下可以继续使用芯片的所有功能，

只是降低了速度。

要进入慢速模式，RCLK频率可被强迫设定为CLK2，CLK2除16，或CK_AF (32KHz时钟)。使用最后一种时钟需要置位CKAF_ST，表明实时时钟被选择且实际存在。

为了减少功耗，可以设定RCCU_PLL1CR寄存器的DX[2:0]位使PLL1关断。

注释：

1.当选择32KHz CK_AF时钟时，PLL1可被设为自动关断，只要置位RCCU_CFR寄存器中的CKSTOP_EN位即可。

2.当选择作为32KHz系统时钟时，可通过使用GPIO引脚停止外部振荡器来减少功耗。

3.5.2 WFI模式

在**WFI**模式下，通过停止芯核工作来减少功耗。程序停止执行但外设保持运行，且寄存器内容被保护。当有中断请求送到EIC时器件恢复工作，程序重新开始执行。

为进入WFI模式，软件要向RCCU_SMR寄存器WFI位写0。

一个中断请求必须被EIC认可才能从WFI模式中唤醒。

3.5.3 LPWFI模式

LPWFI(低功耗等待中断)是WFI和慢速模式的结合。当进入此模式时下列事件发生：

- 芯核的MCLK时钟停止。
- 外设被CLK2_16或CK_AF (32KHz)时钟驱动。
- PLL自动关闭。

要从LPWFI模式下唤醒，必须有一个中断请求被EIC认可。

注释：

1.在LPWFI模式下，通过下列途径降低功耗：

- 设定PCU_PWRCR寄存器的LPVRWFI位，停止主电压调节器工作。
- 设定FLASH_CR0寄存器的PWD位，使FLASH处于掉电模式（参考STR7闪存编程参考手册）。
- 当选择CK_AF时，停止外部振荡器。

2.从LPWFI模式唤醒后，在LPWFI模式下被选择的时钟(CK2_16 或 CK_AF时钟)仍然是系统时钟(RCLK)。

请看图12和图13给出的示例。

图12 使用CK_AF的LPWFI示例

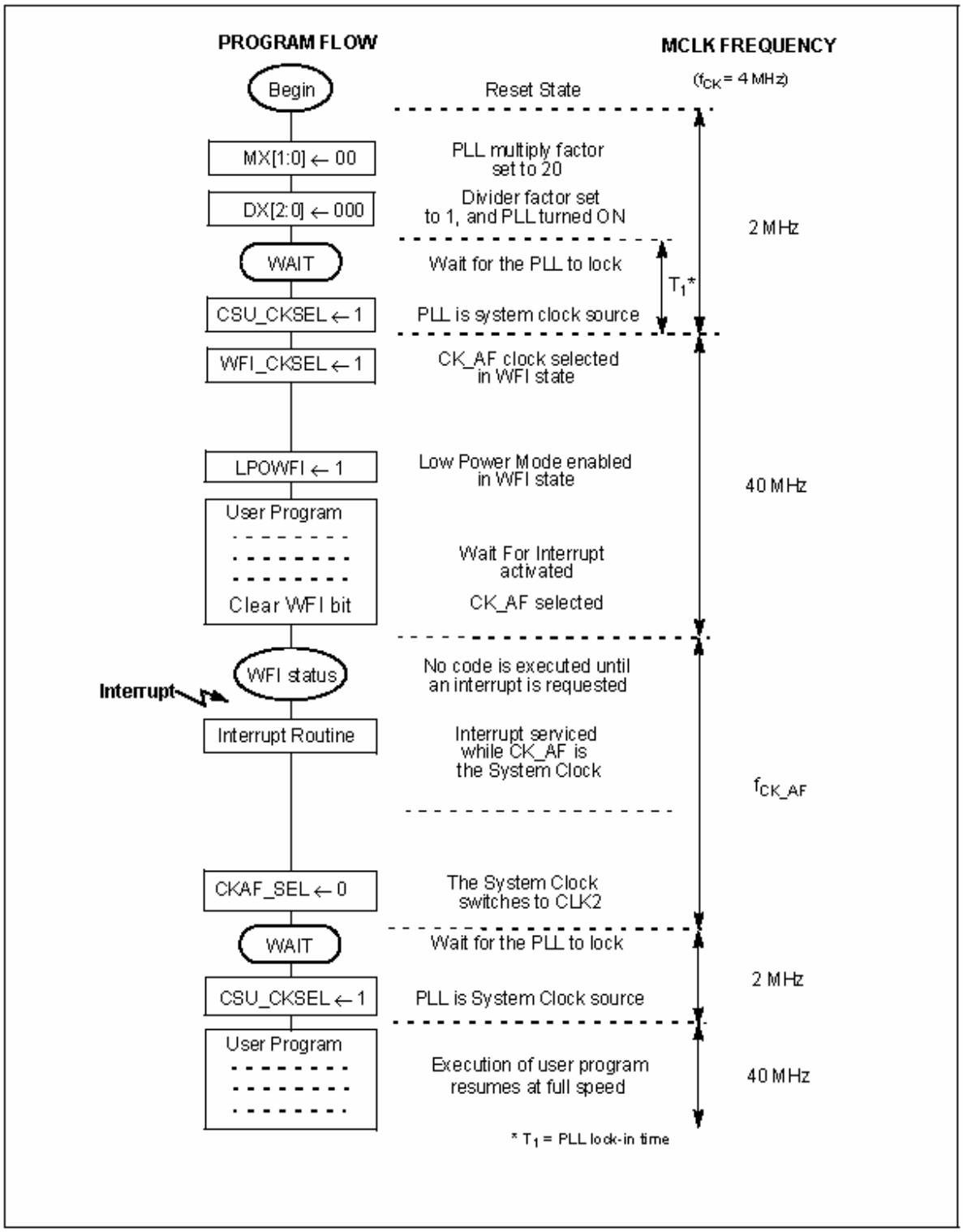
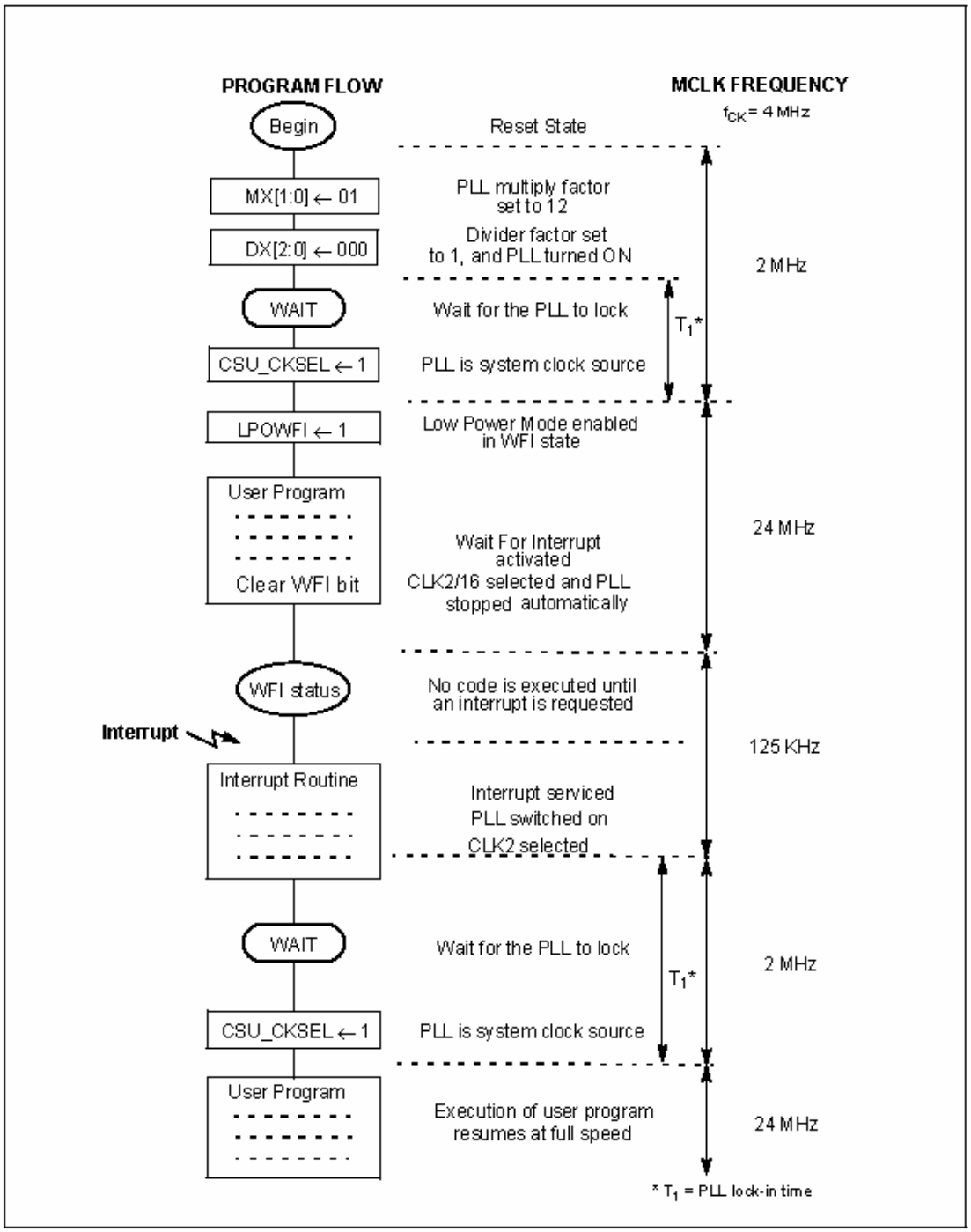

PROGRAM FLOW
MCLK FREQUENCY
(f_{CK} = 4 MHz)
Begin
Reset State
MX[1:0] ← 00
PLL multiply factor set to 20
DX[2:0] ← 000
Divider factor set to 1, and PLL turned ON
2 MHz
WAIT
Wait for the PLL to lock
T_1*
CSU_CKSEL ← 1
PLL is system clock source
WFI_CKSEL ← 1
CK_AF clock selected in WFI state
LPOWFI ← 1
Low Power Mode enabled in WFI state
40 MHz
User Program
Wait For Interrupt activated
Clear WFI bit
CK_AF selected
WFI status
No code is executed until an interrupt is requested
Interrupt
Interrupt Routine
Interrupt serviced while CK_AF is the System Clock
f_{CK_AF}
CKAF_SEL ← 0
The System Clock switches to CLK2
WAIT
Wait for the PLL to lock
2 MHz
CSU_CKSEL ← 1
PLL is System Clock source
User Program
Execution of user program resumes at full speed
40 MHz
* T_1 = PLL lock-in time

图13 使用CLK2/16的LPWFI示例



3.5.4 停止模式

在停止模式下，停止RCLK（芯核和外设时钟）而不复位器件，因而保存了MCU状态（除RCCU_CFR寄存器的CSU_CKSEL位和STOP_I位以外）。

要进入停止模式，必须执行XTI一章描述的Stop位设置顺序操作。器件将保持在此模式直到唤醒输入有效重启程序执行。

停止模式唤醒事件后，经2048时钟周期的延迟后MCU恢复程序的执行。

从停止模式唤醒时，RCCU_CFR寄存器的STOP_I位被置位，如开放了中断则会产生一个中断。

注释：

1.在STOP模式PLL1自动关闭。

2.在STOP模式下通过以下途径降低MCU功耗：

- 通过设置PCU_PWRCR寄存器的LPVRWFI位停止主电压调节器。
- 通过设定FLASH_CR0寄存器的PWD位使FLASH处于掉电模式。
- 用GPIO引脚停止外部振荡器。

3.从STOP模式退出时必须使能外部振荡器。

当处于STOP模式下复位发生时，时钟重启。

注意：假定MCLK=PCLK，在启动STOP位置位序列操作后的(N+6)*MCLK周期之内不能发生停止模式唤醒事件，其中N是完成设定停止位序列操作所需要的周期数（细节请参考“STOP模式进入条件”），否则唤醒事件将会被忽视。

3.5.5 待机模式

主电压调节器控制逻辑负责管理掉电/唤醒序列操作，确保进入和退出超低功耗的待机模式时平滑过渡。

在此模式下，供电电源如正常一样通过V33引脚给定。主电压调节器被关断，V18区域（器件的内核）供电关闭（外引脚V18上的电压降为0）。后援模块，包括实时钟和唤醒逻辑，由低功率电压调节器独立供电。

参看第3.1.1节:选择使用外部V18BKP供电。

掉电过程的启动可通过软件命令，即设定PCU_PWRCR寄存器的PWRDWN位（软件待机进入），或通过外部强制nSTDBY引脚为0（硬件待机进入）。

注释　nSTDBY是一个双向漏极开路的引脚，要求一个外部上拉。

注释　由于在待机模式下器件内核的V18供电关断，系统RAM内容丢失，看门狗无效。

注释　当进入待机模式时，V18区域（器件的内核）电源关闭，而属于后援（Backup）模块的电路仍然保持供电。当离开此模式时，所有属于V18区域的逻辑电路从复位状态开始启动。这意味着包含影响后援模块行为的位的寄存器(PCU_PWRCR, RTC_ALR and RTC_PRL)一般来说并不反映它的状态。事实上，这些位被锁存到后援区域，以便在待机期间保持它们的值。但是当离开待机模式时，这些值并没有传回到V18域的逻辑电路中去，因此V18逻辑只能显示出复位值。

注释　任何向实时时钟的写操作至少需要两个32kHz时钟周期才能完成。假如软件设置实时时钟寄存器，只有在实时时钟寄存器写操作完成后才能进入待机模式。这可由RTC_CRL寄存器的RTOFF位来监视。

注释　当进入待机模式时要确保唤醒引脚被拉低。

3.5.5.1 软件待机进入

图14和图15显示了在通过软件方式进入待机模式时，在外部信号电平和软件层次上事件发生的次序。

- 所有连接主内核（V18区域）与后援模块的信号被强制为地电平以避免电气损坏。
- V18BKP区域与V18区域断开，因此后援模块仅由低功耗调节器供电。
- 主电压调节器被关断。
- 所有属于内核的I/O被强制为高阻抗。
- nSTDBY引脚被器件强制为低。

图14 用WAKEUP退出的软件待机模式进入

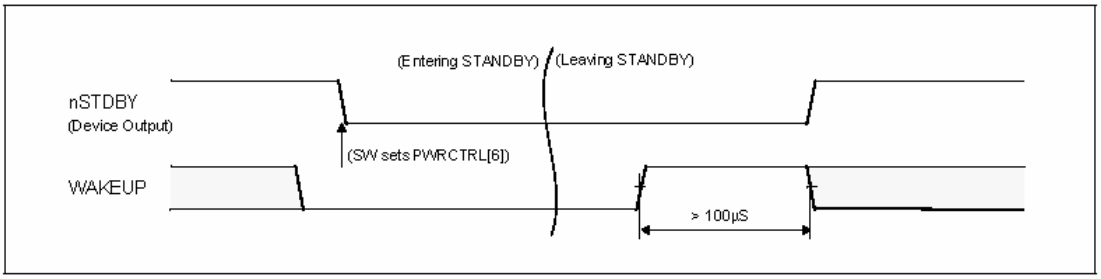


图15 用nRSTIN退出的软件待机模式进入

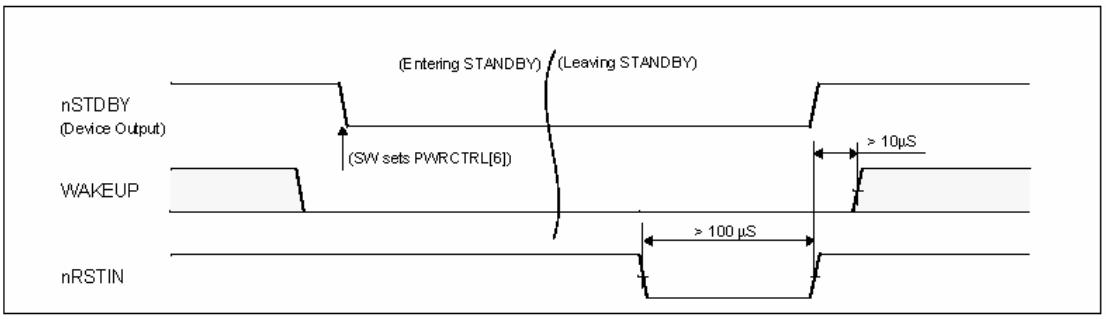
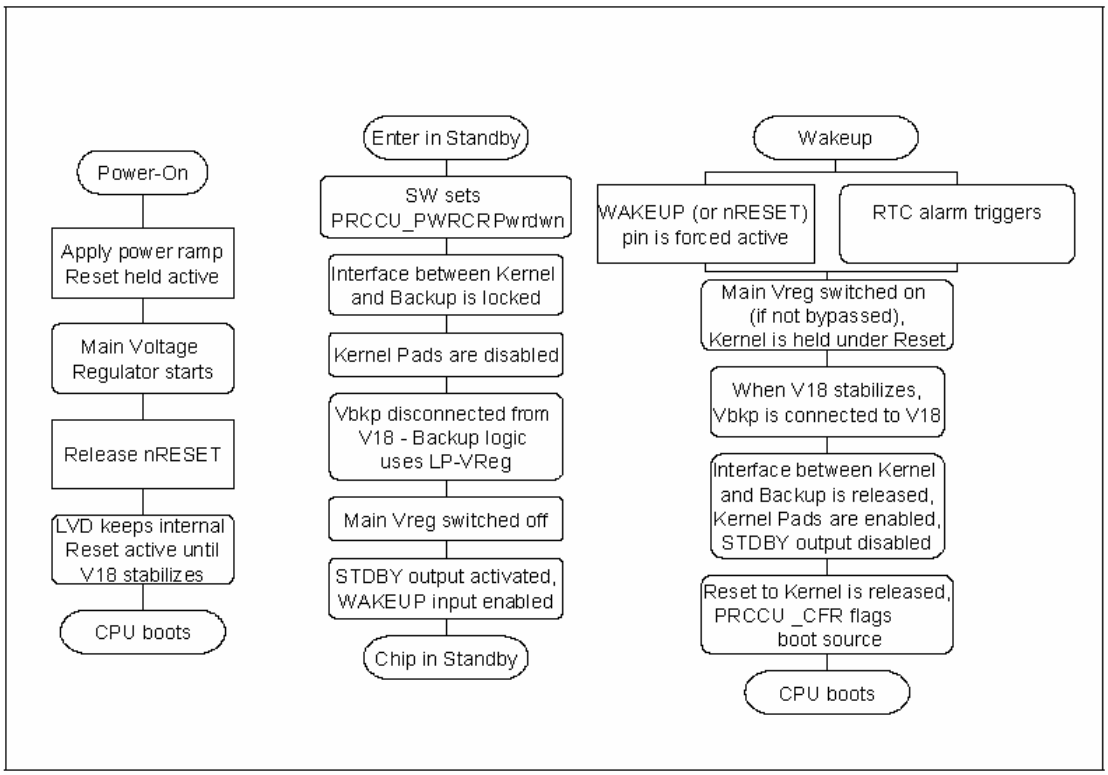


图16 软件的掉电-唤醒序列流程图



唤醒事件可由RTC闹钟或外部唤醒引脚产生。在此情形下，需要脉冲宽度至少有100µS。另一个唤醒源是外部nRSTIN引脚。

唤醒事件重新接通内核的电源。内核被保持在复位状态下，直到内部电压调节到正常值。此时，内核和后援模块的接口重新连接，CPU从复位时序重新开始。RCCU_CFR寄存器的标志位将表明唤醒来源(RTC, WAKEUP引脚, nRSTIN引脚, 看门狗，软件)。

注意：

1.如WAKEUP引脚是高电平，不允许进入待机模式。
2.在待机模式下，RTC闹钟将总是引起唤醒事件而与屏蔽位设置无关。

3.5.5.2 硬件待机进入

在通过硬件进入待机模式时,时序是相同的，但由外部nSTDBY引脚触发而启动，因此该引脚作为输入。在此情况下，需要WAKEUP和nSTDBY引脚的两个上升沿来退出待机模式。WAKEUP 上升沿使主电压调节器重新接通，而nSTDBY引脚上升沿将释放对器件的V18区域的内部复位。
注释：
1．在待机模式硬件进入方式下，WAKEUP和nSTDBY上升沿之间的最小时间间隔是100μS。
2．在待机模式下，RTC报警将总是引起唤醒事件而不管屏蔽位如何设置。
3．如果WAKEUP引脚为逻辑高电平，将不允许进入待机模式。禁止WAKEUP引脚保持为高的同时强制nSTDBY为低，在WAKEUP和nRSTIN脉冲前后也禁止强制nSTDBY为高。
4．复位事件（nRSTIN触发）较nSTDBY优先级更高。因此，复位触发将强制从待机模式退出。假如当nSTDBY有效时触发nRSTIN，则器件从待机模式退出。当nSTDBY保持持续低电平时给出复位脉冲，器件将在nRSTIN上升沿后再次进入待机模式。
5．要用WAKEUP引脚唤醒系统，其信号脉冲宽度必须保持高有效电平至少100μS。少于100μS的脉冲宽度将会影响系统。

图17 用WAKEUP退出的硬件待机模式进入

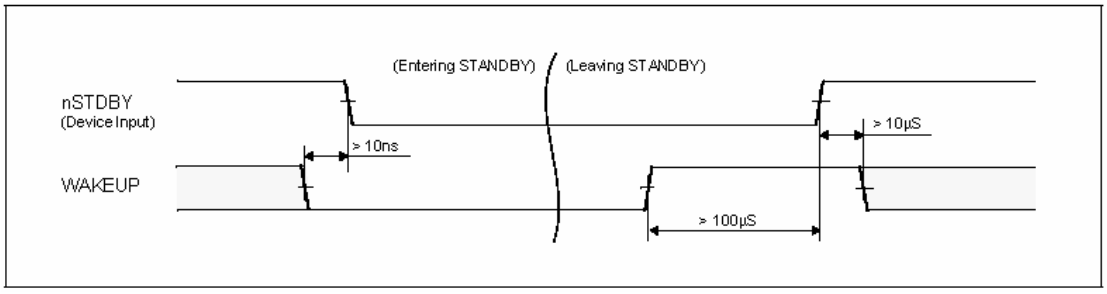
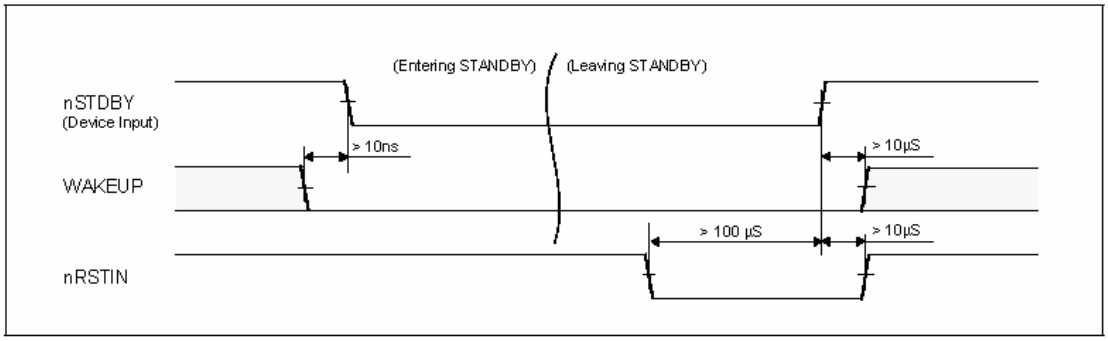


图18用nRSTIN退出的硬件待机模式进入



3.5.5.3 唤醒/RTC报警复位

当唤醒事件从待机模式重新启动器件且电源重新供给V18区域时，复位被触发。复位来源由RCCU_CFR寄存器的一位置位来说明。

3.6 寄存器描述

本节将使用如下缩略语：

read/write (rw)	Software can read and write this bit
read only (r)	Software can only read this bit, writing to it has no effect.
read/clear (rc_w0)	Software can read as well as clear this bit by writing '0'. Writing '1' has no effect.

3.6.1 时钟控制寄存器（RCCU_CCR）

基地址：0xA000 0000h
地址偏移量：0x00h
复位值：0x0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	EN_HALT	EN_STOP	EN_CK2_16	EN_CKAF	EN_LOCK	-	-	-	SRESEN	CKAF_SEL	WFI_CKSEL	LOP_WFI
					rw	rw	rw	rw				rw	rw	rw	rw

位 32-12=保留，永远读为 0。

位 11=EN_HALT：使能 HALT 位。
0：没有软件复位
1：当软件在 RCCU_SMR 寄存器中设置了 HALT 位且 SRESEN=1 时，软件复位

位 10=EN_STOP：停止中断屏蔽位。
0：停止中断请求无效。
1：停止中断请求有效。

位 9=EN_CK2_16：CK2_16 中断屏蔽位。
0：CK2_16 切换中断无效。
1：CK2_16 切换中断有效。

位 8=EN_CKAF：CKAF 中断屏蔽位。
0：CKA 切换中断无效。
1：CKA 切换中断有效。

位 7=EN_LOCK：LOCK 中断屏蔽位。
0：LOCK 转换中断无效。
1：LOCK 转换中断有效。

位 6:4=保留，永远读作 0。

位 3=SRESEN：软件复位使能
0：没有软件复位
1：当软件在 RCCU_SMR 寄存器中设置 HALT 位且 EN_HALT=1 时，软件复位。

位 2=CKAF_SEL：替换功能时钟选择。
0：不选择 CK_AF 时钟。
1：选择 CK_AF 时钟。
注意：要检测选择是否生效，请检测 CKAF_SET 是否设置。如果不存在实时钟，选择将不会生效。

位 1=WFI_CKSEL：WFI 时钟选择。
如果 LPOWFI=1，这一位选择应用于低功耗 WFI 模式下的时钟。
0：MCLK 在低功耗 WFI 时是 CLK2/16。
1：MCLK 在低功耗 WFI 时是 CK_AF，如果它存在的话。实际上这一位设置在 WFI 模式下的 CKAF_SEL。

位 0=LPOWFI：在等待中断时为低功耗模式。
0：在 WFI 无效时为低功耗模式。当 WFI 被执行后，MVLK 不改变。
1：当 WFI 指令执行时器件进入低功耗模式。在这一状态中的时钟依赖于 WFI_CKSEL。

3.6.2 时钟标记寄存器（RCCU_CFR）

基地址：0xA000 0000h
地址偏移量：0x08h
复位值：0000 8408h 在一个外部唤醒复位之后
0000 8208h 在一个电压调节器低电平检测器复位之后
0000 8088h 在一个 RTC 报警唤醒复位之后
0000 8048h 在一个看门狗复位之后
0000 8042h 在一个软件复位之后
0000 8008h 在一个外部（上电）复位之后

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV2	STOP_I	CK2_16_I	CKAF_I	LOCK_I	WKP_RES	LVD_RES	-	RTC_ALARM	WDG RES	SOFT RES	CKSTOP_EN	CK2_16	CKAF_ST	LOCK	CSU_CKSEL
rw	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	rw	rw	r	r	rw

位 31: 16=保留，永远读作 0。

位 15=DIV2：OSCIN 被 2 除。
这个位控制工作在 CLK 信号下的除 2 电路。
0：不对 CLK 信号进行分频
1： CLK 被 2 分频。

位 14=STOP_I：停止中断等待位。
这一位只能被清除。
0：没有停止中断请求等待。
1：有停止中断请求等待。

位 13=CK2_16_1：CK2_16 切换中断等待位。
这一位只能被清除。
0：没有 CK2_16 中断请求等待。
1：有 CK2_16 中断请求等待。

位 12=CKAF_1：CK_AF 切换中断等待位。
这一位只能被清除。
0：没有 CK_AF 切换中断请求等待。
1：有 CK_AF 切换中断请求等待。

位 11=LOCK_1：锁定中断等待位。
这一位只能被清除。
0：没有锁定中断请求等待。
1：有锁定中断请求等待。

位 10=WKP_RES：外部唤醒标志。
这一位是只读位。
0：没有唤醒复位发生。
1：复位由一个外部唤醒事件在待机模式下产生。

位 9=**LVD_RES**：电压调节器低电压检测器复位标志。
这一位是只读位。
0：没有电压调节低电压检测器复位发生。
1：电压调节低电平监测器复位发生。

位 8=保留，永远读作 0。

位 7=**RTC_ALARM**：实时时钟报警复位标志。
这一位是只读位。
0：没有 RTC 报警事件发生。
1：复位由一个 RTC 告警在待机模式下产生。

位 6=**WDG_RES**：看门狗复位标志。
这一位是只读位。
0：没有看门狗复位发生。
1：有看门狗复位发生。

位 5=**SOFTRES**：软件复位标志。
这一位是只读位。
0：没有软件复位发生。
1：有软件复位发生。

位 4=**CKSTOP_EN**：时钟停止使能。
这一位由软件来设置和清除。
0：即使 CK_AF 被选择，PLL1 仍保持有效。
1：如果 CK_AF 被选择，则关闭 PLL1。

位 3=**CK2_16**：CLK2/16 选择
0：CLK2/16 被选择为 RCLK 源，且 PLL 关闭。
1：RCLK 源是 CLK2（或者 PLL 的输出由 CSU_CKSEL 的值决定）。

位 2=**CKAF_ST**：CK_AF 状态。
这一位是只读位。
0：PLL 时钟、CLK2 或 CLK2/16 是 RCLK 源（由 CSU_CKSEL 和 CK2_16 位决定）。
1：CK_AF 是 RCLK 源。

位 1=LOCK：PLL 锁定。
这一位是只读位。
0：PLL 是关断的，或者未被锁定而不可以被选为 RCLK 源。
1：PLL 是锁定的。

位 0=**CSU_CKSEL**：CSU 时钟选择
这一位由软件来设置和清除。在以下情况下，这一位由硬件复位：
- 位 DX[2:0](RCCU_PLL1CR)被设置成 111；
- 晶振被停止（由硬件或软件）；
- CK2_16 位（RCCU_CFR）被强制为‘0’。
0：CLK2 提供系统时钟
1：如果 LOCK 位和 VROK 位是‘1’，PLL 倍频器提供系统时钟。
如果 FREEN 位被设置，这一位选择的时钟独立于 LOCK 位和 VROK 位。
注意：设置 CKAF_SEL 位覆盖掉其他时钟选择。清除 CK2_16 位将覆盖 CSU_CKSEL 选择（见图 11）。

3.6.3 PLL 配置寄存器（RCCU_PLL1CR）

基地址：0xA000 0000h
地址偏移量：0x08h
复位值：0000 0007h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	FREEN	FREF_ RANGE	MX1	MX0	-	DX2	DX1	DX0
								rw	rw	rw	rw		rw	rw	rw

位 31：8=保留，永远读作 0。

位 7=FREEN：PLL 自由运行模式使能
0：自由运行模式无效。在这种情况下，PLL 的运行仅取决于 MX[1：0]位和 DX[2：0]位。
1：自由运行模式有效。在这种模式下，当所有的三个 DX[2：0]位被设置，PLL 不会停止，而是会提供一个低频率备份时钟，由 CSU_CKSEL 位选择；工作在这种模式下不需要 LOCK 和 VROK 位被设置，而且与 MX[1：0]位的设置无关。

位 6=FREF_RANGE：参考频率范围选择器位
0：配置 PLL 适应输入频率（CLK2）1.5—3MHz。
1：配置 PLL 适应输入频率（CLK2）大于 3MHz。

位 5：4=MX[1：0]：PLL 倍频因数。
参考表格 12 来设置 MX 位。

表 12 PLL 倍频因数

MX1	MX0	CLK2 x
1	0	24
0	0	20
1	1	16
0	1	12

注意：建议在修改 MX 的值之前，不选择并关闭 PLL。

位 3=保留，永远读作 0。

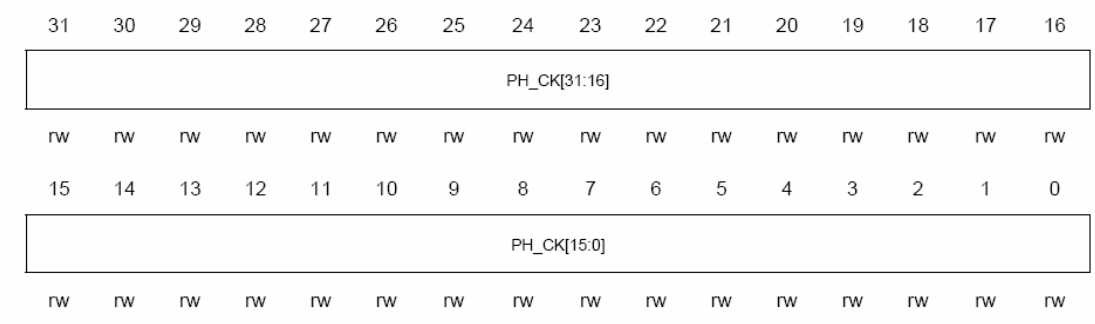
位 2：0=DX[2：0]：PLL 输出时钟分频系数。参考表 13 来设置 DX 位。

表 13 PLL 分频系数

DX2	DX1	DX0	RCLK
0	0	0	PLLCK / 1
0	0	1	PLLCK / 2
0	1	0	PLLCK / 3
0	1	1	PLLCK / 4
1	0	0	PLLCK / 5
1	0	1	PLLCK / 6
1	1	0	PLLCK / 7
1	1	1	FREEN= 0: CLK2 (PLL OFF, Reset State) FREEN=1: PLL in Free Running mode

3.6.4 外围使能寄存器（RCCU_PER）

基地址：A000 0000h
地址偏移量：1Ch
复位值：0001 FFFFh



位 32—0=PH_CK[32：0]：外围时钟有效。

0：外围时钟被关闭，停止外围设备。
1：外围时钟有效，允许外围设备工作。
寄存器位和外围设备的关系在表 14 中给出。

表 14 外设时钟管理

PH_CKEN Reg. Status	Peripheral Stopped
PH_CK[1:0] = 0	not used[1)]
PH_CK[2] = 0	EMI
PH_CK[3] = 0	not used[1)]
PH_CK[4] = 0	USB KERNEL
PH_CK[16:5] = 0	not used[1)]
PH_CK[31:17] = 0	not used

1) 为降低功耗，这些位应该在复位后用软件清除。

3.6.5 系统模式寄存器(RCCU_SME)

基地址：A000 0000h
地址偏移量：20h
复位值：0000 0001h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	HALT	WFI
														rw	rw

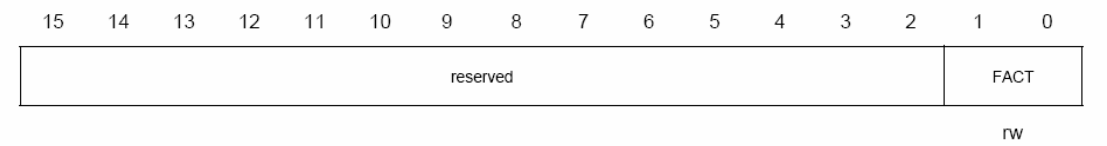
位 31：2=保留，永远读为 0。

位 1=HALT：停止。
0：无作用。
1：如果 SRESEN 位和 ENHALT 位在 RCCU_CCR 寄存器中设置，产生一个软件复位。

位 0=WFI：等待中断模式。
0：进入 WFI(等待中断)模式。在这种模式下，CPU 保持空闲状态直到一个中断请求被 EIC 认可。当这些发生时，这一位又被置为‘1’。这就意味着该位一旦被复位，仅能由硬件来置‘1’。
1：无作用。
警告：如果所有的 EIC 中断通道都被屏蔽，清除这一位将无限期地停止程序执行，除非器件被复位。因此必须确保在清除 WFI 位之前至少有一个中断通道打开。

3.6.6 MCLK 分频器控制（PCU_MDIVR）

地址偏移量：40h
复位值：0000h



这个寄存器根据表 15 为主系统时钟 MCLK 设置预分频因子。它可以由软件在任何时间写入,用于动态调节工作频率.

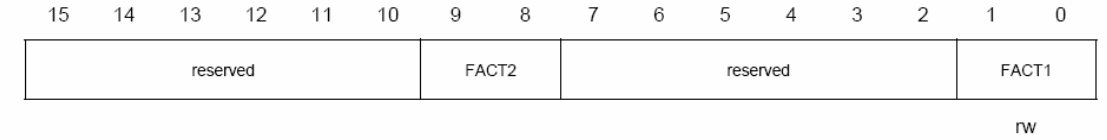
位 15: 2=保留，永远读作 0。

位 1：0=FACT[1：0]：分频系数
表 15 时钟预分频值

FACT	Prescaling ratio
00	1 - Default, no prescaling, MCLK = RCLK
01	2 - MCLK = RCLK / 2
10	4 - MCLK = RCLK / 4
11	8 - MCLK = RCLK / 8

3.6.7 外设时钟分频控制寄存器（PCU_PDIVR）

地址偏移量：44h
复位值：0000h



这个寄存器为两个 APB 时钟设置预分频因子，PCLK1 的外围设备属于 APB1 组，PCLK2 的外围设备属于 APB2 组；预分频因子的值在表 15 中列出。可以在任何时候由软件写入。FACT1 和 FACT2 的选择是相互独立的。
注意：外设时钟速度必须等于或低于 CPU 时钟速度，但是必须低于或等于数据手册中规定的值。用户必须确保满足这一条件，否则会发生不可预测现象。

位 15：10=保留，永远读作 0。

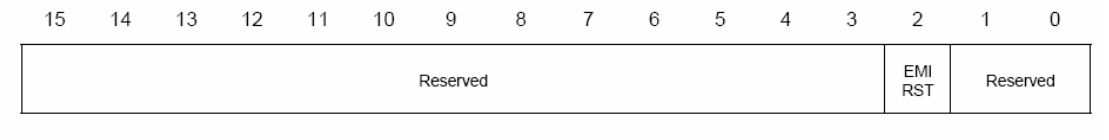
位 9：8=FACT2[1：0]：APB2 外围设备的分频系数

位 7：2=保留，永远读作 0。

位 1：0=FACT2[1：0]：APB1 外围设备的分频系数

3.6.8 外设复位控制寄存器（PCU_PRSTR）

地址偏移量：48h
读/写
复位值：0000h



这个寄存器允许对大多数系统模块单独强制触发一个复位。不是所有的系统模块都可由软件复位，这是为了确保器件行为一致。

位 15：4：保留，永远读作 0。

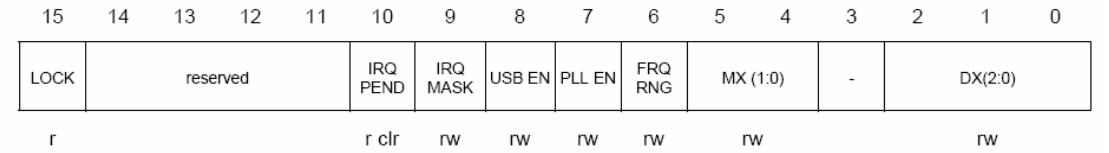
位 3=为工厂测试保留。为了降低功耗，这一位必须由软件设置。

位 2=EMIRST
如果这一位被设为逻辑 1，外部存储器接口将被强置为复位状态。复位的激活/结束与系统时钟 MCLK 同步。
如果这一位被设为逻辑 0，EMI 正常工作。

位 1：0=为工厂测试保留。为了降低功耗，这一位必须由软件设置。

3.6.8.1 PLL2 控制寄存器（PCU_PLL2CR）

地址偏移量：4Ch
复位值：0033h



这个寄存器控制 PLL2 的运行，专用于 HDLC 或 USB 模块。

位 15=LOCK：PLL2 锁定
只读。当 PLL2 锁定输入参考时钟并且提供一个稳定的频率输出时，这一位由硬件设置。如果设置了位 9 IRQ MASK 打开中断的话，这一位的任何改变都会产生一个中断请求。

位 14：11=保留

位 10=IRQ PEND：锁定变化 CPU 中断请求等待
由硬件设置，用软件仅可清除。当这一位被设定时，锁定状态改变引起的中断请求正等待处理。这个中断请求映射到 PRCCU 的中断向量。通过把这一位写入 1 可以去掉等待的请求。

位 9=IRQ MASK：锁定变化 CPU 中断请求使能
当这一位复位(缺省值)时，PLL2 不产生中断请求；当该位置 1 时，锁定状态的任何改变都会产生一个中断请求，通过读这个寄存器来清除等待的请求。

位 8=USBEN：USB 用 PLL 时钟使能
当这一位复位(缺省值)时，用于 USB 的 48MHz 参考时钟连接到 USBCLK 引脚；当该位置 1 时，USB 的参考时钟通过 PLL2 由 HCLK 引脚来提供；输入频率，倍频和分频系数必须正确选则以保证满足 USB 标准要求的精度。

位 7=PLLEN：选择 PLL
当这一位复位(缺省值)时，PLL 被旁路，但没有关闭。HCLK 直接驱动内部逻辑（HDLC 或 USB）。当 PLLEN 被设置，PLL 输出被选择作为逻辑电路的时钟源。如果位 15（LOCK）被复位，则禁止将这一位置 1。如果 PLL 未被锁定，不论什么原因，这一位将由硬件清除。
注意：为了关闭 PLL2 来降低功耗，设置 DX[2：0]位。

位 6=FRQRNG：PLL2 频率范围选择
当 PLL 输入频率（HCLK 管脚）在 3—5MHz 范围内时，这一位须由软件置 1。
当 PLL 输入频率（HCLK 管脚）在 1.5—3MHz 范围内时，这一位须由软件清除。

位 5：4=MX[1：0]：PLL 倍频因子
参考表 16 来配置倍频设定。

表 16 PLL 倍频因子

MX1	MX0	CLOCK2 x
1	0	28
0	0	20
1	1	16
0	1	12

位 3=未使用；当读时为‘0’。

位 2：0=DX[2：0]：PLL 输出时钟分频因子。参考表 17 来设置分频器，其中 PLL 时钟代表倍频器部分的输出。

表 17 PLL 分频因子

DX2	DX1	DX0	CK
0	0	0	PLL CLOCK / 1
0	0	1	PLL CLOCK / 2
0	1	0	PLL CLOCK / 3
0	1	1	PLL CLOCK / 4
1	0	0	PLL CLOCK / 5
1	0	1	PLL CLOCK / 6
1	1	0	PLL CLOCK / 7
1	1	1	BYPASS (PLL OFF)

3.6.9 引导配置寄存器(PCU_BOOTCR)

偏移地址：50h
复位值：0000 00p1 1100 00bb
其中 p：依赖于封装，b：依赖于 BOOT 引脚值（如下）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res.						PKG64	res.	HDLC	CAN	ADC EN	LPOW DBG EN	USB FILT EN	SPI0 EN	BOOT	
						r	r	r	r	rw	rw	rw	rw	rw	

这个寄存器包括引导选项和其他全局配置控制。

引导选项是在复位时被锁定的外部 BOOT 引脚的值，这些位的值可被软件看到；之后还可以修改它们来建立一个不同地配置（存储器映射）。详细描述请看 2.2 节。

配置比特是固定的（硬件）器件选项，用户不可以修改。这些位可让软件读出配置。

位 15：10：保留

位 9=PKG64：芯核安装在 64 引脚封装中
硬件配置。只读。
当这一位被清除时，芯片被安装在一个 144 管脚的封装中。
当这一位被设置时，芯片被安装在一个 64 管脚的封装中，而且所有的非连接管脚都强置于空闲模式（输入和输出无效），忽略在 IO 端口控制寄存器中的值。

位 8=保留。

位 7=HDLC：HDLC 起作用
硬件配置。只读。
当这一位被清除时，HDLC 接口控制器无效。
当这一位被设置时，HDLC 接口控制器有效。

位 6=CAN：CAN 起作用
HW 配置。只读。
当这一位被清除时，CAN 接口控制器无效。
当这一位被设置时，CAN 接口控制器有效。

位 5=ADC EN：ADC 有效
读/写
当这一位被清除时（复位值），模数转换器的模拟部分关闭（掉电），保持最小静态功耗。
当这一位被设置时，模数转换器的模拟部分有效，可以应用。设置完这一位后，在第一个有效转换之前需要一个 1ms 的开始时间。在任何低功耗模式下（WFI，STOP），这一位应该由软件复位以关闭 ADC。

位 4=LPOWDBGEN：使能为 STOP 模式预留的调试界面。
读/写。

当器件处在 STOP 模式时，所有的内部时钟被冻结，包括进入到 ARM7 的 MCLK，这样就不能回应一个来自仿真器的调试请求（DBGRQS 引脚，或通过 JTAG 接口的命令）。

当这一位被设置时，调试请求输入会强制芯片立即离开 STOP 模式，打开内部时钟，因此允许仿真器对系统实施控制。

当这一位被清除时（复位值），这一功能无效，在 STOP 模式下的调试请求被忽略。

位 3=USBFILT EN：使能 USB 待机滤波器

读/写。
USB 收发器的作用是一个低通滤波器，用于改善当总线处在 STANDBY 模式时的抗噪声性能。
当这一位被清除时（复位值），滤波器无效。
当这一位被设置时，滤波器有效（由 USB 总线的 STANDBY 条件确定）。

位 2=SPI0 EN：使能 SPI0
读/写。
当这一位被清除时（复位值），SPI0 接口控制器关闭，UART3 和 I2C1 可以使用引脚 P0.0 到 P0.3。
当这一位被设置时，SPI0 接口控制器有效，将会访问引脚 P0.0 到 P0.3（如果它们被编程为替换功能）。在这种情况下 UART3 和 I2C1 不可用。

位 1：0=BOOT[1：0]：引导模式
这些位报告引导配置情况，引导配置可由用户通过引导管脚[1：0]或通过 JTAG 编程序来选择时，其编码列在表 18 中。

表 18 引导模式

BOOT[1:0]	BOOT MEMORY	NOTES
00	FLASH	Default
01	BOOT	
10	RAM	
11	EXTERNAL	

除了它们在存储器映射图上的正常位置以外，对应于 BOOT 值的存储器段也被映射到地址 0000.0000， CPU 在完成复位动作后，开始从这个区域中读取代码。

软件随后可以修改这些位，把任何一段内存映射到地址 0000.0000 。

3.6.10 电源控制寄存器（PCU_PWRCR）

偏移地址：54h
复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WR EN	BUSY	WKUP ALRM	VR OK	-	-	FLASH LP	LVD_DIS	OSC BYP	PWR DWN	LPVR BYP	LPVR WFI	VR BYP	-	-	-
rw	r	r	r			rw	rws	rw	rw	rw	rw	rw			

位 15=WREN：寄存器写使能

这一位作为一个安全机制避免对此寄存器的误写操作。

当 WREN=0(缺省值)，寄存器中的所有位都被写保护，不可以被改变。这一位仅由软件设置，不可清除。

当 WREN= 1 ，列表中标明 RW 的位可写。一个定时间隔启动，允许在一个 64 个 MCLK 周期时间内进行寄存器写操作。过了这个时间间隔之后，这一位被硬件置位。这一位不影响读操作。

位 14=BUSY：备份逻辑忙——编程进行中
只读
备份逻辑是异步的，这个位作为寄存器和相关逻辑握手状态的标志。
当 BUSY=0（缺省值），可以通过对寄存器进行写操作来对备份逻辑进行编程。
当 BUSY=1，一个以前的写操作尚未完成，禁止向寄存器写入信息。

位 13=WKUP-ALRM：唤醒或告警正在有效
只读
当外部 WAKEUP 引脚启动或一个内部唤醒源启动，硬件会将这一位置 1 。在这种情况下降低功耗的命令无效。
当 WKUP-ALRM=0（缺省值），可以开始降低功率顺序操作。

位 12=VROK：主要调节器 OK
只读。
当 VROK=0，主电压调节器是不稳定的，器件的供电电压不能保证在规定的范围内。
当 VROK=1，主电压调节器已稳定，电压达到规定范围。
注意：对于任何会影响到器件的时钟和功率状态的配置改变，软件应该在改变之前和改变之后检查这一位。

位 11：10=保留，永远读作 0。

位 9=FLASH LP：Flash 低功耗（低速度）模式选择
只有当位 15（WREN）用同一个写指令置 1 时，这一位才可以写。
当这一位被清除（缺省值）时，Flash 工作于 FAST 模式，对于顺序访问使用突发模式，允许以零等待状态工作于最高器件频率，对非连续存储器访问产生一个等待周期。
当本位被置位时，Flash 进入低功耗模式，突发模式无效。不会产生任何等待状态。低功耗（LP）模式可以被用于不高于 33MHz 的工作频率(MCLK)下。
注意：如果本位被置 1 的同时 MCLK 的运行速度高于器件最大限定速率，则不能保证器件的正确工作。

位 8=LVD DIS：电压调节器低电压检测无效
只有当位 15（WREN）被设置时，这一位才可以写。这是个粘性的位：一旦被设置，它就不能被软件清除——只有复位将清除它。
当这一位被清除（缺省值）时，如果供电电压低于任何一个电压调节器的低电压检测器的阈值电压时，就会产生一个复位信号。
当这一位被设置后，电压降落将不会产生复位信号。
注意：为了安全起见，仅当低功耗电压调节器关闭时(LPVRBYP 位=1)这一位才允许被设置。

位 7=OSC BYP：32KHz 振荡器旁路使能
只有当位 15（WREN）被设置时，这一位才可写。
当这一位被清除（缺省值）时，32KHz 振荡器被使能，为实时时钟提供一个时钟源，并且为整个系统提供一个备份时钟源。

位 6=PWRDWN：启动掉电模式
只有当位 15（WREN）被设置时，这一位才可写。
当 PWRDWN=0（缺省值）时，芯片工作于正常模式。
当 PWRDWN=1 时，芯片进入掉电模式。主电压调节器关闭，器件内核供电断开。
备用（低功耗）电压调节器仍保持工作状态，供电给备用部分：实时间时钟和唤醒逻辑。

位 5=LPVRBYP：低功耗调节器旁路
只有当位 15（WREN）被设置时，这一位才可写。
当 LPVRBYP=0（缺省值）时，备用（低功耗）电压调节器保持工作状态，准备接管低功耗模式。

当 LPVRBYP=1 时，备用（低功耗）电压调节器关闭（旁路），备用逻辑通过 V18kp 管脚由一个外部 1.8V 电源供电。
注意：如果设置了本位而没有正确地连接外部供电电源的话，将产生不可预测的结果，包括器件的永久损坏。

位 4=LPVRWFI：等待中断模式下的低功耗电压调节器。
只有当位 15（WREN）被设置时，这一位才可写。
当 LPVRWFI =0（缺省值）时，主电压调节器除了在掉电模式下，一直是启动的。
当 LPVRWFI=1 时，主电压调节器在 STOP, LP_WFI（参看 3.5 节）低功耗模式下关闭，此时备份（低功耗）电压调节器为器件供电。

位 3=VRBYP：主调节器旁路
只有当位 15（WREN）被设置时，这一位才可写。
当 VRBYP =0（缺省值）时，主电压调节器启动，为器件供电。
当 VRBYP=1 时，主电压调节器无条件关闭。在这种情况下，器件仅由低功耗电压调节器供电。在这种配置中，最大允许工作频率是 1MHz，并且 PLL 无效。

位 2：0＝保留，永远读作 0。

3.7 PRCCU 寄存器映射

表 19. PRCCU 寄存器映射

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RCCU_CCR	-	-	-	-	-	EN_STOP	EN_CK2_16	EN_CKAF	EN_LOCK	-	-	-	SRESEN	CKAF_SEL	WFI_CKSEL	LOP_WFI
8	RCCU_CFR	DIV2	STOP_I	CK2_16_I	CKAF_I	LOCK_I	WKP_RES	LVD_RES	-	RTC_ALARM	WDG RES	SOFT RES	CKSTOP_EN	CK2_16	CKAF_ST	LOCK	CSU_CKSEL
18	RCCU_PLL1CR	-	-	-	-	-	-	-	-	FREEN	FREF_RANGE	MX1	MX0	-	DX2	DX1	DX0
1C	RCCU_PER	-	-	-	-	-	-	-	-	-	-	-	PH_CK4	PH_CK3	PH_CK2	PH_CK1	PH_CK0
20	RCCU_SMR																WFI
40	PCU_MDIVR	reserved														FACT	
44	PCU_PDIVR	reserved						FACT2		reserved						FACT1	
48	PCU_RSTR	RST[15:0]															
4C	PCU_PLL2CR	LOCK	reserved				IRQ PEND	IRQ MASK	USB EN	PLL EN	FRQ RNG	MX (1:0)		-	DX(2:0)		
50	PCU_BOOTCR	reserved						PKG 64	-	HDLC	CAN	ADC EN	LPOW DBG EN	USB FILT EN	SPI0 EN	BOOT	
54	PCU_PWRCR	WR EN	BUSY	WKUP ALRM	VR OK	-		FLASH LP	LVD DIS	OSC BYP	PWR DWN	LPVR BYP	LPVR WFI	VR BYP	-	-	-

基地址请看表 1。

4 I/O 端口

4.1 功能描述

每个通用 I/O 端口有 3 个 16 位配置寄存器（PC0,PC1,PC2）和一个 16 位数据寄存器（PD）。

依据每个 I/O 端口在数据手册的器件引脚描述表中列出的具体硬件特性，每个端口都可以独立地定义为输入、输出和替换功能等。

每个 I/O 端口位可任意编程，但 I/O 端口寄存器必须按 16 位字来访问，以字节或比特方式访问是不允许的。

图 19 说明了一个 I/O 端口位的基本结构

图 19 I/O 端口位的基本结构

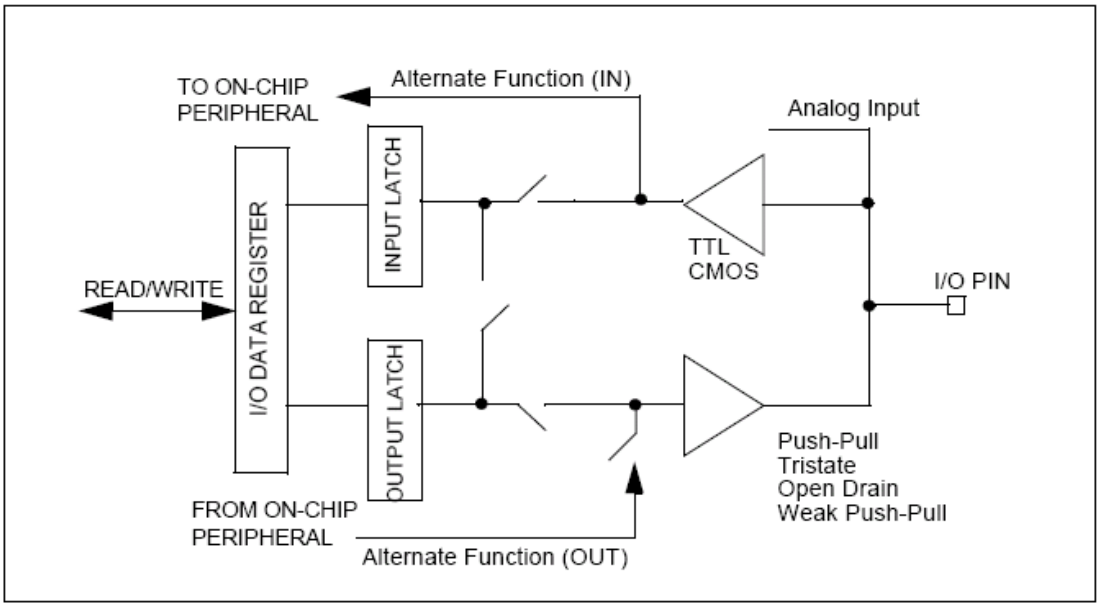


表 20 端口位配置表

Port Configuration Registers (bit)	Values							
PC0(n)	0	1	0	1	0	1	0	1
PC1(n)	0	0	1	1	0	0	1	1
PC2(n)	0	0	0	0	1	1	1	1
Configuration	HiZ/AIN	IN	IN	IPUPD	OUT	OUT	AF	AF
Output	TRI	TRI	TRI	WP	OD	PP	OD	PP
Input	AIN	TTL	CMOS	CMOS	N.A.	N.A.	CMOS	CMOS

说明：

AF： 替换功能　　OD： 漏极开路

AIN： 模拟输入　　OUT： 输出

CMOS：CMOS 输入　　PP： 推挽

HiZ： 高阻　　TRI： 三态

IN: 输入　　TTL： TTL 输入

INUPD：输入上拉/下拉　　WP： 弱推挽

N.A： 不适用。在输出模式，一个端口读访问将得到输出锁存值。见图 22。

通用输入输出端口（GPIO）

系统复位时 I/O 端口配置为通用状态（存储器映射 I/O）。

当向 I/O 数据寄存器写数据时，这些数据总是被加载到输出锁存。输出锁存保持着这些要输出的数据，而输入锁存扑获出现在 I/O 引脚上的数据。

一个对 I/O 数据寄存器的读访问将读取输入锁存或输出锁存，取决于这个端口位的配置是输入还是输出。

I/O 替换功能（AF）

每个引脚的替换功能在数据手册中列出。如果一个端口位被配置为替换功能，这个引脚将不再与输出锁存相连，而是连接到片上外设的输出信号。

要使用替换功能，还必须在外设控制寄存器中打开这个功能。每个引脚只能使用一种替代功能。

- 对于替换功能的输入，端口位可以配置为输入模式或者替换功能模式
- 对于替换功能的输出或者输入-输出，端口位必须配置为替换功能模式

外部中断/唤醒线

一些端口拥有外部中断功能（见数据手册）。要使用外部中断，端口必须配置为输入模式。关于中断/唤醒的更多信息，请参看第 5 节。

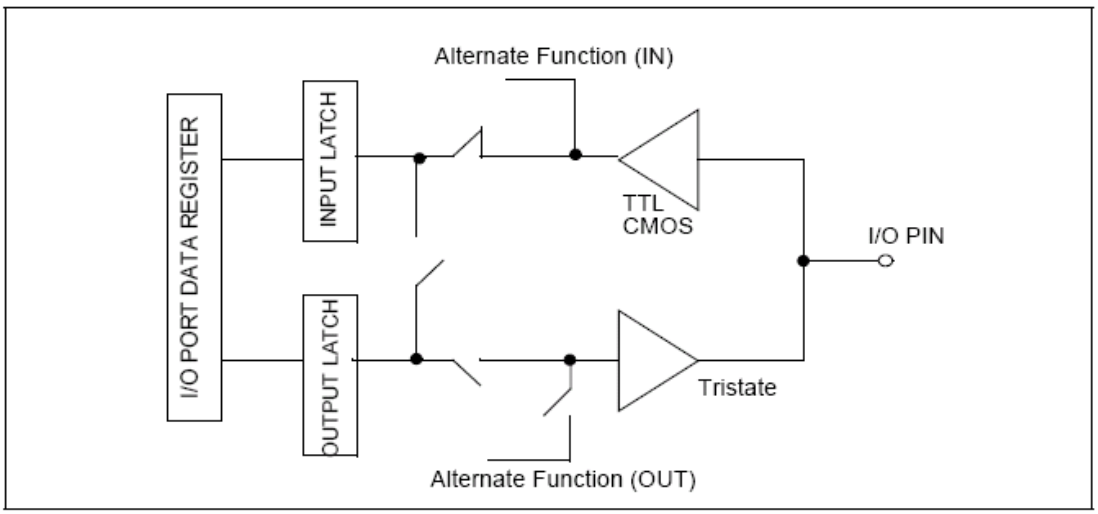
4.1.1 输入模式配置

当 I/O 端口配置为输入时：

- 输出缓冲器强制为三态。
- I/O 引脚上的数据在每一时钟周期被采样到输入锁存。
- 对 I/O 数据寄存器的读操作将得到输入锁存中的值。

图 20 说明了 I/O 端口的输入模式配置。

图 29 输入模式配置



4.1.2 输入上拉/下拉模式配置

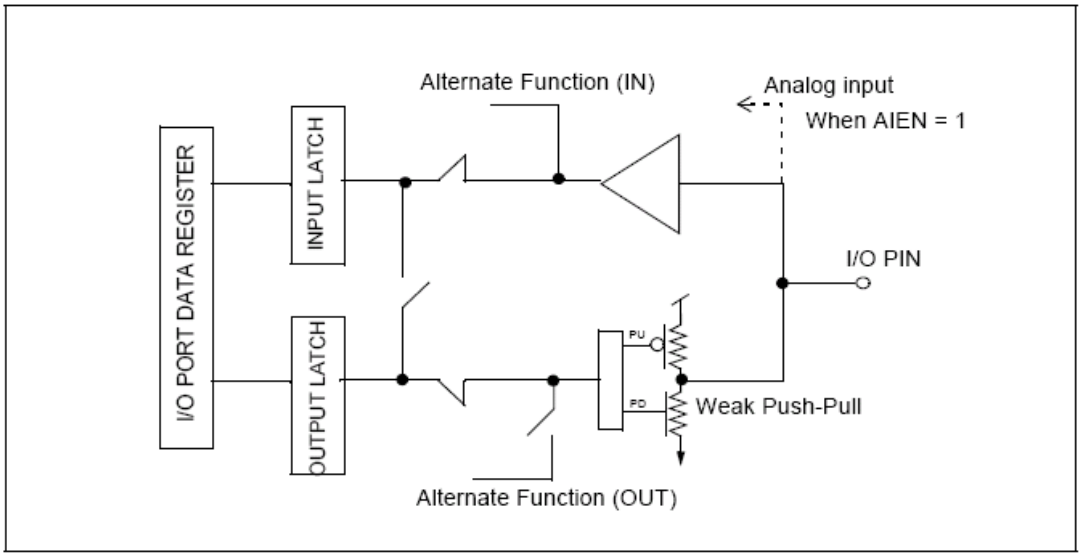
当 I/O 端口接通为输入上拉/下拉模式时：

- 输出缓冲配置为弱推挽，软件可在输出锁存中写入适当的电平来建立所需要的弱上拉或下拉。

■ 在输出锁存中的数据驱动 I/O 引脚（逻辑 0 建立弱下拉，逻辑 1 建立弱上拉）。
■ 对 I/O 数据寄存器的读操作得到输入锁存的值。

图 21 说明了 I/O 端口的输入上拉/下拉模式（PUPD）配置：

图 21 输入上拉/下拉模式配置

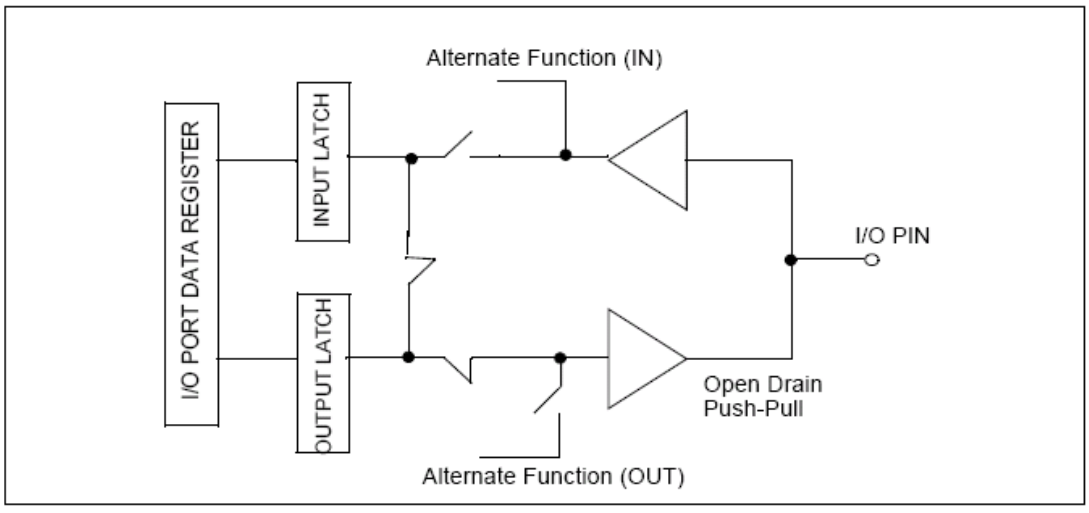


4.1.3 输出模式配置

当 I/O 端口配置为输出模式时：
■ 输出缓冲器接通为漏极开路或者推挽模式
■ 输出锁存的数据驱动 I/O 引脚
■ 对 I/O 数据寄存器的读操作得到输出锁存的值。

图 22 说明了 I/O 端口位的输出模式配置

图 22 输出模式配置



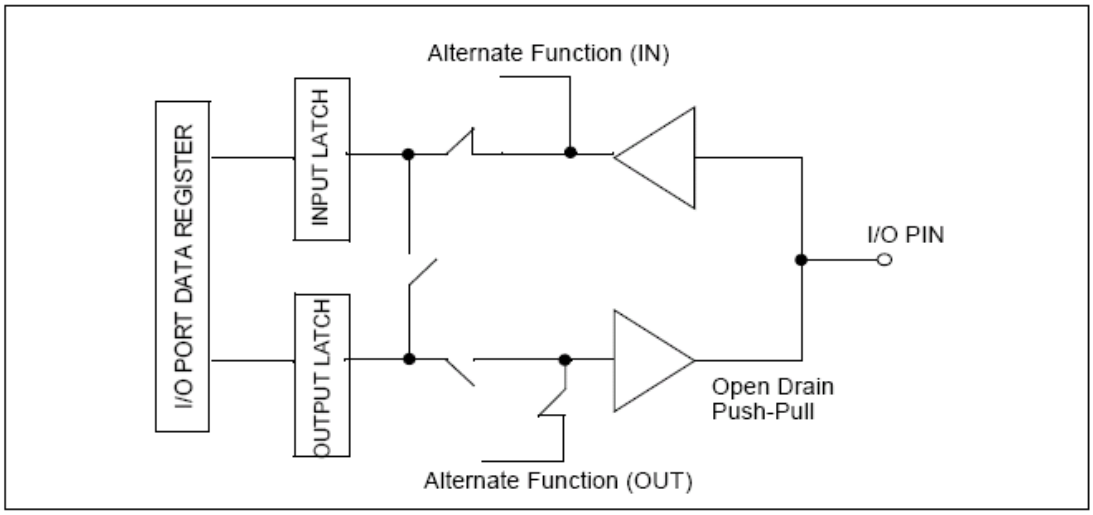
4.1.4 替换功能模式配置

当 I/O 端口配置为替换功能模式时：

- 输出缓冲器接通为漏极开路或者推挽方式
- 输出缓冲器由来自外设的信号驱动（替换功能的输出）
- 出现在 I/O 引脚上的数据每一时钟周期被采样到输入锁存
- 对 I/O 数据寄存器的读操作得到输入锁存的值。

图 23 说明了 I/O 端口位的替换功能模式的配置

图 23 替换功能模式的配置



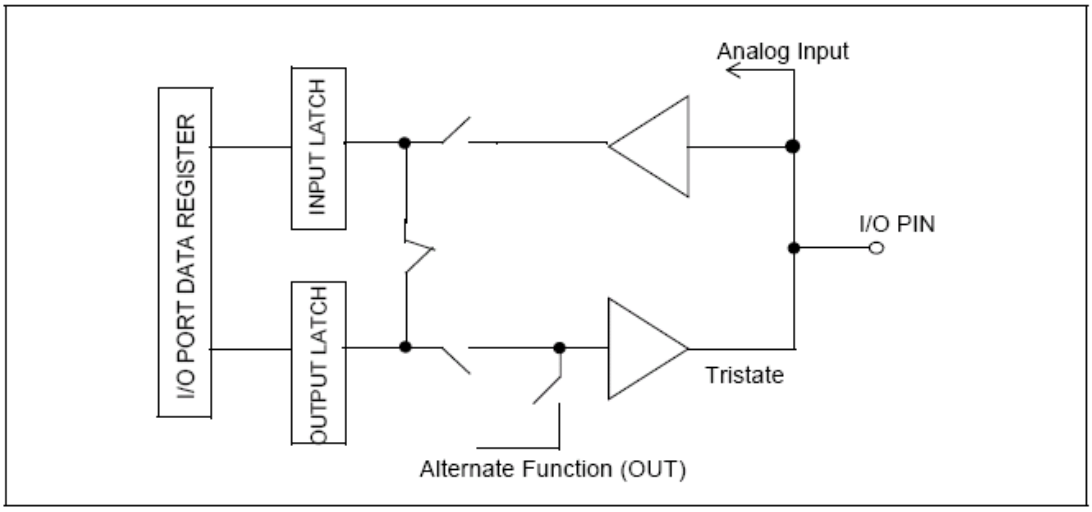
4.1.5 高阻-模拟输入模式配置

当 I/O 端口配置为高阻-模拟输入模式时：

- 输出缓冲器强制为三态
- 输入缓冲器无效（替换功能的输入强制为一固定值）
- 模拟输入可接模拟外设的输入
- 对 I/O 数据寄存器的读操作得到输出锁存的值。

图 24 显示了 I/O 端口位配置为高阻-模拟输入模式时的配置：

图 24 高阻-模拟输入模式时的配置



4.2 寄存器描述

I/O 端口寄存器不能以字节方式访问。

端口配置寄存器 0（PC0）

偏移地址： 00h

复位值： FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C015	C014	C013	C012	C011	C010	C09	C08	C07	C06	C05	C04	C03	C02	C01	C00
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15：0=C0[15：0] 端口配置位

见表 20 配置 I/O 端口方式

端口配置寄存器 1（PC1）

偏移地址： 04h

复位值： FFFFh（对 GPIO0 为 7FFFh）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C115	C114	C113	C112	C111	C110	C19	C18	C17	C16	C15	C14	C13	C12	C11	C10
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15：0=C1[15：0] 端口配置位

见表 20 配置 I/O 端口方式

端口配置寄存器 2（PC2）

偏移地址： 08h

复位值： 0000h（对 GPIO2 为 0001h）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C215	C214	C213	C212	C211	C210	C29	C28	C27	C26	C25	C24	C23	C22	C21	C20
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15：0=C2[15：0] 端口配置位

见表 20 配置 I/O 端口方式

I/O 数据寄存器（PD）

偏移地址： 0Ch

复位值： （*）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15：0=D[15：0] ：I/O 数据位

向这个寄存器的写访问总是将数据写进输出锁存。

读操作时，在输入模式和替换功能模式下从输入锁存读取数据，在输出模式和高阻模式下从输出锁存读取数据。

（*）代表取决于外部硬件的配置

4.2.1 I/O 端口寄存器映射

下表总结了对每个 I/O 端口实现的寄存器。

表 21 I/O 端口寄存器

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC0	C0[15:0]															
4	PC1	C1[15:0]															
8	PC2	C2[15:0]															
C	PD	D[15:0]															

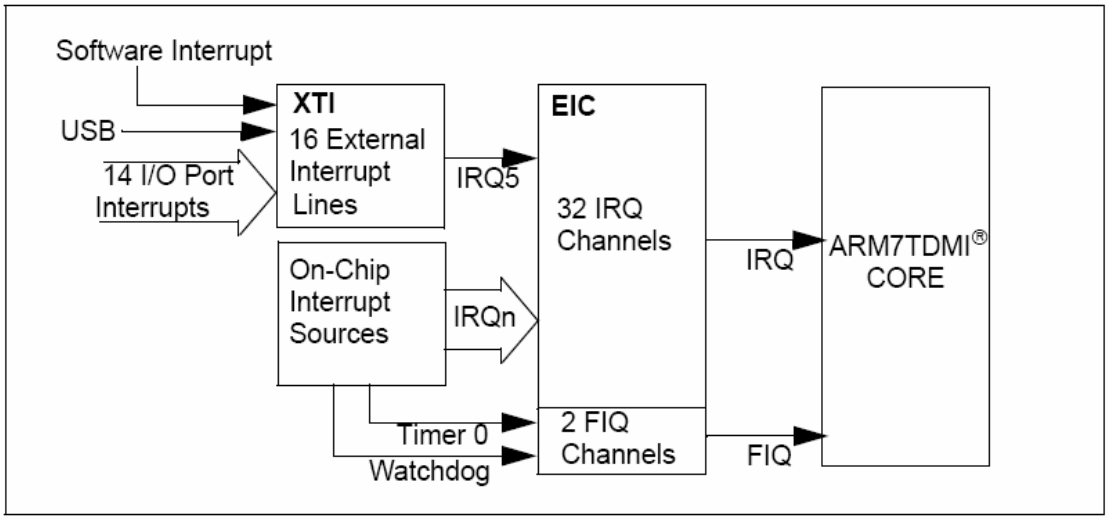
有关基地址请参看表 3 “APB2 存储器映射”。

5 中断

ARM CPU 提供两个等级的中断：用来适应快速、短延迟响应中断处理的 FIQ（快速中断请求），和适用于更普遍情况中断的 IRQ（中断请求）。

STR71x 中断管理系统提供两个中断控制块：EIC 和 XTI，如图 25 所示。

图 25 中断管理概要



5.1 中断延迟时间

一旦中断请求产生（来自于外部中断源或者是片上外设），在中断处理程序启动之前，中断请求本身要经过 3 个不同的阶段。中断延迟时间可以看作是这三个阶段延迟的总和：

- 由于输入阶段同步引起的延迟时间。这个同步逻辑可能存在（例如，对外部中断输入信号的同步过程），也可能不存在（例如，片上中断请求），取决于中断来源。与这个阶段关联的延迟时间为 0 或者是两个时钟周期。
- EIC 本身的延迟时间。这个过程有 2 个时钟周期的延迟，
- ARM7TDMI 中断处理逻辑的引起的延迟时间（参阅在 www.arm.com 网站上的文档）。

表 22 EIC 中断延迟（以时钟周期计）

	SYNCH. STAGE		EIC STAGE
	min	max	
FIQ	0	2	2
IRQ			

5.2 增强的中断控制器（EIC）

增强的中断控制器（EIC）完成对多中断通、中断优先仲裁和中断向量的硬件处理。它提供如下功能：

- 32 个可屏蔽中断通道，映射到 ARM CPU 的 IRQ 中断请求线路，
- 每个映射为 IRQ 的中断通道有 16 个可编程优先级，
- 硬件支持中断嵌套（最多可支持 16 级中断请求嵌套），包含内部硬件嵌套堆栈，
  - 2 个可屏蔽中断通道，映射到 ARM CPU 的 FIQ 中断请求线，既没有优先级也没有矢量化
  - 在寄存器偏址 0x18h 处的跳转指令，指向具有最高级中断优先级的 ISR 起始地址（由用户

定义）

- 16 个从 XTI 块产生的外部中断映射到 IRQ5

EIC 在执行以下操作时不需要软件的介入：

- 根据相关通道的屏蔽位拒绝或接受一个中断请求，
- 将所有未处理 IRQ 请求与存储的当前优先级进行比较。如果当前中断请求优先级高于存储的当前优先级，则为 ARM7 产生一个 IRQ
- 向中断地址寄存器（偏移地址 0x18h）装载最高优先级 IRQ 的地址向量，
- 一旦新的 IRQ 被接受，在硬件优先权堆栈段保存先前中断的优先级，
- 一旦新中断被接受，更新当前中断优先级寄存器

如果多重中断源映射到相同的中断向量，软件需要读中断标志寄存器来确定准确中断源（见表 23 中断标志栏）。

IRQ 中断向量表示于表 23。

表 23 IRQIRQ 中断向量表

Vector	Acronym	Peripheral Interrupt	Peripheral Interrupt Flags
IRQ0	T0.EFTI	Timer 0 global interrupt	5
IRQ1	FLASH	FLASH global interrupt	
IRQ2	PRCCU	PRCCU global interrupt	
IRQ3	RTC	Real Time Clock global interrupt	2
IRQ4	WDG.IRQ	Watchdog timer interrupt	1
IRQ5	XTI.IRQ	XTI external interrupt	16
IRQ6	USB.HPIRQ	USB high priority event interrupt	0-7
IRQ7	I2C0.ITERR	I2C 0 error interrupt	
IRQ8	I2C1.ITERR	I2C 1 error interrupt	
IRQ9	UART0.IRQ	UART 0 global interrupt	9
IRQ10	UART1.IRQ	UART 1 global interrupt	9
IRQ11	UART2.IRQ	UART 2 global interrupt	9
IRQ12	UART3.IRQ	UART 3 global interrupt	9
IRQ13	SPI0.IRQ	BSPI 0 global interrupt	5
IRQ14	SPI1.IRQ	BSPI 1 global interrupt	5
IRQ15	I2C0.IRQ	I2C 0 tx/rx interrupt	
IRQ16	I2C1.IRQ	I2C 1 tx/rx interrupt	
IRQ17	CAN.IRQ	CAN module global interrupt	32
IRQ18	ADC.IRQ	ADC sample ready interrupt	1
IRQ19	T1.GI	Timer 1 global interrupt	5
IRQ20	T2.GI	Timer 2 global interrupt	5
IRQ21	T3.GI	Timer 3 global interrupt	5
IRQ22	Reserved		
IRQ23	Reserved		
IRQ24	Reserved		
IRQ25	HDLC.IRQ	HDLC global interrupt	
IRQ26	USB.LPIRQ	USB low priority event interrupt	7-15
IRQ27	Reserved		
IRQ28	Reserved		
IRQ29	T0.TOI	Timer 0 Overflow interrupt	1
IRQ30	T0.OCMPA	Timer 0 Output Compare A interrupt	1
IRQ31	T0.OCMPB	Timer 0 Output Compare B interrupt	1

两个可屏蔽中断源映射到 FIQ 向量地址，如表 24 所示：

表 24 FIQ 向量表

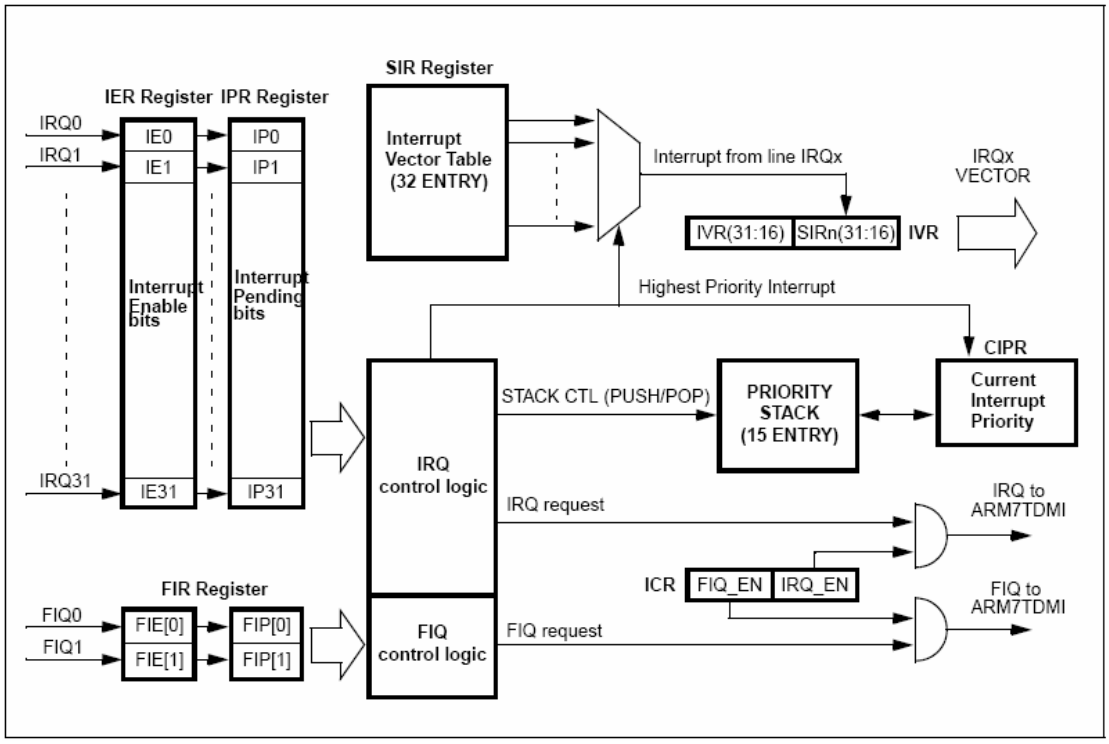
Vector	Interrupt Source
FIQ0	T0.GI - Timer 0 Global Interrupt
FIQ1	WDG.IRQ - Watchdog timer interrupt

这些中断源也可作为一般的 IRQ。在大多数情况下，在应用中只能启用一个 FIQ 源。如果同时采用两个 FIQ 源，可以通过读取 EIC 寄存器里 FIQ 等待位来决定中断源。记住，FIQ 没有优先级机制，所以，如果有两个 FIQ 中断发生，需用软件来控制优先权。

ARM7TDMI CPU 提供 2 级中断，用来适应快速、短延迟中断处理的 FIQ（快速中断请求），和适用于更普遍情况中断的 IRQ（中断请求）。

所以要用硬件控制多中断通道、中断优先级和自动向量化，就需要单独的增强中断控制器（EIC）。

图 26 EIC 框图



- 映射到 ARM 的中断请求线 IRQ 的 32 路可屏蔽中断通道
- 对每个映射到 IRQ 的中断通道提供 16 级可编程优先级
- 硬件支持中断嵌套（15 级）
- 两个映射到 ARM 的中断请求线 FIQ 的可屏蔽中断通道，无优先级也没有向量化。

5.2.1 IRQ 结构

EIC 由一个优先级译码器，有限状态机和一个堆栈组成。

5.2.1.1 优先级译码器

优先译码器是一个不断计算最高优先级未处理 IRQ 的组合电路模块。如果有一个胜出者，将会把刚从仲裁中胜出的 IRQ 中断子程序地址更新到 EIC_IVR（中断向量寄存器），同时发出 nIRQ 内部信号低电平。nIRQ 内部信号与 EIC IRQ 使能位（IRQ_EN）的反相信号进行“或”运算得到 ARM7TDMI 的 nIRQ 信号。

每个通道有一个 4 比特字段，就是在 EIC_SIRn（源中断寄存器 0-31）中的 SIPL（源中断优先级）字段，它定义了该通道的优先级从 0（最低）到 15（最高）。

如果有若干个通道被指定了相同的优先权级别，则将由一个内部硬件菊花链来确定它们的优先权。通道地址越高，优先级越高。如果通道 2 和通道 6 被指定了相同的软件优先级，且同时处在等

待处理状态，则先处理通道 6。

为了在优先裁决中胜出，一个通道必须：

- 处在待处理状态（EIC_IPR0-1，中断等待寄存器，共 32 个等待位，每个通道有一位）。为了进入待处理，通道必须被使能（EIC_IER0-1，中断使能寄存器，共 32 个等待位，每个通道有一位）。
- 拥有最高的优先级，高于当前的优先级（存储在 EIC_CIPR 当前中断优先级寄存器），也高于其他任何等待的中断通道级别。
- 当多个等待中断通道具有相同优先级时，在硬件中断通道链中具有最高位置。

EIC_CIPR 寄存器提供当前正在处理的中断的优先级。在复位后，EIC_CIPR 被清零。在中断处理例程中，其值可能被软件修改，被赋予存储在 EIC_SIRn（源中断寄存器 0-31）中的设定的优先级数值，最高为 15。试图写一个低于在 EIC_SIRn 中的数值将不起作用。

为了操作安全，建议在修改 EIC_CIPR，EIC_SIR 或清除 EIC_IPR 的 IRQ 等待位之前，先关闭总 IRQ，以避免危险的竞争状况。再有，如果 IRQ_EN 在中断服务程序被清零，与当前被处理的 IRQ 关联的等待位不能被清零，否则，将再也不能在弹出堆栈之前恢复 EIC 的状态。

5.2.1.2 有限状态机

有限状态机（FSM）有 2 个状态。准备好（READY）和等待（WAIT）。这两个状态分别对应于 ARM7TDMI 的 nIRQ 信号线的有效（对应 WAIT）或无效（对应 READY）。通过清除 EIC 总使能位 IRQ_EN，可将 nIRQ 的状态无条件地屏蔽掉（高电平无效）。在复位后 FSM 处在准备状态（EIC 的 nIRQ 线为高）。当优先级译码器选出一个新胜出中断后，FSM 从准备状态变到等待状态，同时 EIC 的 nIRQ 变为有效低电平。

为了让 FSM 重回到准备状态，必须要读取 EIC_IVR 寄存器或复位 EIC 单元。可以通过总复位信号对整个器件的复位，或通过清除 APB2_SWRES 寄存器中的位 14 来完成 EIC 的复位。

对 EIC_IVR 进行读取总是将 FSM 从等待移至准备状态，操作假设 FSM 原来处在等待状态，操作同时自动释放 EIC 的 nIRQ 线。

没有标志位表示 FSM 的状态。

5.2.1.3 堆栈

堆栈可达到 15 次事件深度，对应于最大中断嵌套数。堆栈用来压入和弹出先前的 EIC 状态。被压入堆栈的数据 EIC_CICR（当前中断通道寄存器）和 EIC_CIPR（当前中断优先级寄存器）。

当 FSM 处在等待状态时，读取 EIC_IVR 寄存器会建立起一个内部标志。这将引起将先前的 EIC_CICR 和 EIC_CIPR 压入堆栈，这个动作将在读取 EIC_IVR 后的下个内部时钟周期发生。与此同时，内部标志位将被清除。EIC_CICR 和 EIC_CIPR 将被更新为从 EIC_IVR 中读出的中断通道所对应的数值。

若当 FSM 处在准备状态时读取 EIC_IVR，内部标志位将不会建立，EIC 内部堆栈也不会有任何操作。

一个服务例程只能被具有更高优先级的事件所中断。因此，最多可嵌套中断数为 15 个，对应着从 1 到 15 的 15 个优先级别。优先级为 0 的中断是永远不会被执行的。为了使堆栈变满，必须有 15 个嵌套的中断，而且每个中断事件必须从 1 级到 15 级依次出现。主程序优先级必须是 0。

让所有中断源的中断优先级都为 0 的方式可以用在只采用查询处理的应用中。

要弹出堆栈，需要清除 EIC 中对应于 EIC_CICR 寄存器内中断的等待位。清除其他的等待位将不会弹出堆栈。注意不要将仍在栈中的事件对应的等待位清空，否则，当到达这一级取栈时将无法弹出堆栈。当弹出堆栈时，EIC_CICR 和 EIC_CIPR 将恢复为以前中断事件对应的值。

5.2.1.4 EIC 中断向量化

当ARM7TDMI对中断请求进行了译码，0x18这个地址的指令就被执行。此时，EIC_IVR寄存器已经更新为最高级等待中断程序的地址。为了充分利用EIC机制，0x18这个地址的指令可以把放在EIC_IVR中的地址加载到程序计数器。这样，CPU向量就直接指向正确的中断程序而且不需要另外增加软件处理。

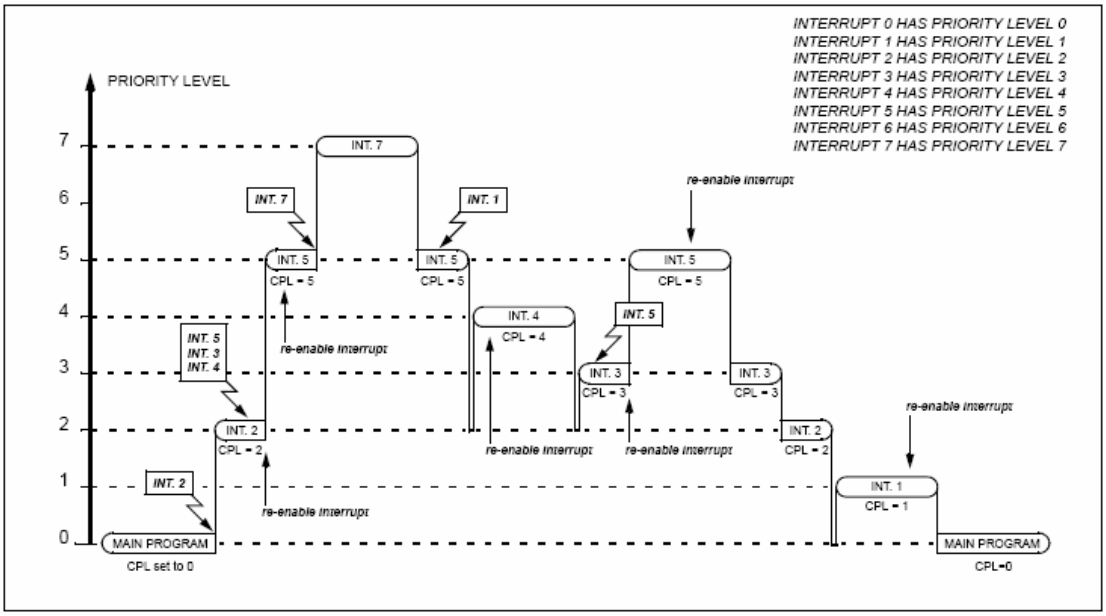
由于优先权解码器总是处于工作状态，仲裁一直在进行。有可能发生的情况是有一个中断事件拉低了ARM7TDMI的nIRQ信号线，假如在nIRQ线被拉低和读EIC_IVR操作之间一个新的最高优先级事件出现了，当读取EIC_IVR时，得到的值将对应最高优先级的等待中断。

读取EIC_IVR并从位于0x18这个地址的指令直接跳转到正确的中断程序并不是强制性的。另一种方法可以是跳转到单一的中断入口，后面再去读EIC_IVR。唯一必须遵守的规则是必须首先读取EIC_IVR一次，此后清除对应的等待位。从EIC的角度来看，当读出EIC_IVR时中断被认可，当对应的等待位被清除时中断就完成了。从ARM7TDMI核的角度来看，当ARM7TDMI的 nIRQ线被拉低时中断被认可，当执行了异常返回时中断就完成了。

EIC的nIRQ线等效于ARM7TMDI的nIRQ信号线，但它还可以被EIC的全局使能位（IRQ_EN）屏蔽。当EIC的nIRQ线被拉低时，如果EIC全局使能位没有设置，则ARM7TDMI的nIRQ线路将不会变为低有效。

EIC_IPR是一个读/清除寄存器，所以当写0时没有任何作用，当写1时将清除相关位。因此，禁止使用读-修改指令以避免EIC_IPR状态的崩溃。大部分的EIC等待位是和外设等待位相关的。外设的等待位必须先于EIC等待位清除，否则EIC等待位将会重置，中断例程将执行两遍。

图27 嵌套中断请求过程举例



5.2.1.5 EIC IRQ注意事项

当FSM处于READY状态时，读取EIC_IVR是无效的，读取的数据是不可预测的。实际上，EIC会给EIC_IVR赋予默认的中断请求程序地址。

程序设计失误的后果，也会造成当FSM处于WAIT 状态时，EIC_IVR可能出现不可预料的值。这种代码显然必须要避免，因为在随后执行中断子程序过程中，CPU将会执行这个程序，然而EIC_IVR寄存器里的数据却不是相应的数据。由于对应的等待位可能被清除，这将导致一个堆栈EIC崩溃。

以下几种情况下将会发生上述错误：

- 当把EIC_SIRn寄存器里的等待通道的优先水平值降低到低于或等于当前程序的优先级时
- 当软件清除某些等待位而没有注意去执行标准的中断程序过程（详见下文）。

在上述情况下，优先权解码器丢失了最高优先者，然而EIC的nIRQ线仍然处在有效状态。只有读EIC_IVR时，EIC的nIRQ线才能释放。

CPU将会执行中断程序而EIC_IVR寄存器中并没有与之相应的数据，这有可能破坏堆栈。如果对应的等待位位被清除，将不可能执行EIC堆栈弹出操作。

正常处理中断事件的方法是在中断例程中只读取EIC_IVR寄存器一次。在退出中断例程之前，必须清除相应外设和EIC的等待位。一旦EIC_IVR被读取时，在EIC_IVR寄存器没有被当作是程序指针时，应用软件就能通过读取EIC_CICR寄存器来知道当前正在执行那个中断例程。

如果没有在中断例程中读取EIC_IVR，nIRQ线将不会释放，中断将会被执行两遍。如果等待位已经被清除了，由于不能完成弹出操作，EIC堆栈将会被破坏。

在一个中断例程内部，清除一个低于当前优先级的等待位并不会造成问题，因为在这种情况下nIRQ线没有被拉为低有效。然而，清除一个具有较高优先级的等待位可能会造成问题，因为nIRQ线在这种情况下已经被拉低成有效，当中断被重新启动时EIC堆栈将会被破坏。

清除已经在堆栈里的中断等待位也会破坏堆栈。

在主程序里，如果全局中断被禁止，则所有的中断源被禁止，所有的等待位都被清除。如果nIRQ已经被拉低，一旦全局中断被允许，CPU就执行一个中断程序。在这种情况下，EIC_IVR读操作将会得到不可预测的值，且破坏EIC堆栈。

只有一种不需要执行中断例程就可以清除相应等待位的安全办法，就是从较高优先级的IRQ例程中清除它们。用这种方法可确保EIC的nIRQ线被释放。

所有的EIC等待位都可以被清除，其中也包括随后用户想要处理的那些。然而，用户代码需要确认的是，对于这些要处理的中断，外设的等待位不被清除。这样一来，与之对应的EIC等待位将会被重新设置。由于所有的EIC等待位都被清除了，EIC堆栈可确保被正确弹出。另一种解决方案是确保与EIC_IVR读操作所对应的EIC等待位被清除。

5.2.2 FIQ 机制

相对于EIC的IRQ机制，EIC的FIQ机制不提供任何自动中断向量和对每个FIQ中断源的软件优先级别。它提供一个全局的FIQ使能位，每一个FIQ通道有一个使能位和等待位。

ARM核CPSR寄存器的全局F位与EIC里的全局FIQ使能位有几点不同。F位在用户模式下不能修改，而EIC里的全局FIQ使能位在任何模式下都可修改。另外，F位不能够修改内部信号nFIQ的电平。当EIC的全局FIQ使能位影响nFIQ信号电平时，F位只是可将通往芯核的信号屏蔽掉。当EIC里的全局FIQ使能位被清除时，nFIQ信号总是无效的。一旦EIC里的全局FIQ使能位被置位，同时至少有一位FIQ等待位被设置时，nFIQ信号即生效。

如果想让FIQ通道源出现中断等待，就必须要设置相应的FIQ使能位。在清除FIQ使能位设置时，如果遇到相应的等待位也被设置，则等待位不能被清除。在等待位被清除前通道一直保持激活状态。

为了使 CPU 指向地址 0x1C（FIQ 异常向量地址），F 位和全局 EIC FIQ 使能位必须被使能，同时至少一位 FIQ 等待位被激活。否则，CPU 将不会进入 FIQ 异常程序。

5.3 寄存器描述

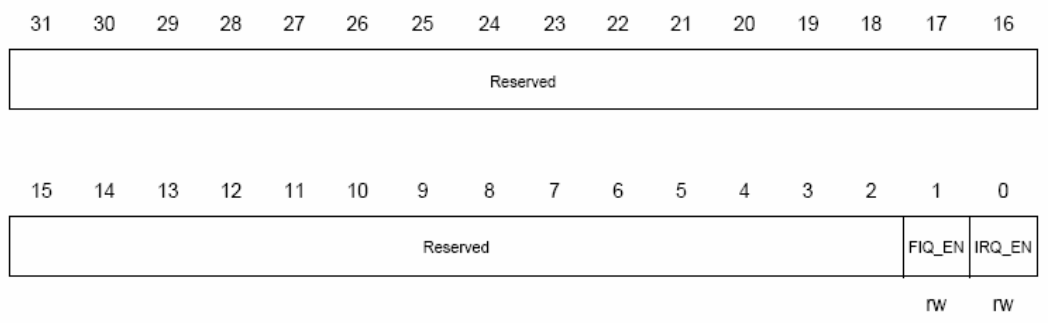
在本节将会使用到以下的缩写:

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
read/clear (rc_w1)	Software can read as well as reset this bit by writing '1'. Writing '0' has no effect on the bit value.

5.3.1 中断控制寄存器(EIC_ICR)

地址偏置:00h

复位值:0000 0000h



Bit 31:2 保留的，始终读为 0

Bit 1 FIQ_EN: FIQ 的输出使能信号

0：增强型中断控制器的 FIQ 输出到 CPU 被禁止，即使 EIC 在其输入端已经检测到有效且被使能的快速中断请求，也不产生输出。

1：增强型中断控制器 FIQ 输出到 CPU 被允许。

软件可以对这位进行读写。

Bit 0 IRQ_EN：IRQ 的输出使能位

0：增强型中断控制器的 IRQ 输出到 CPU 被禁止，即使 EIC 已经在其输入端检测到有效的和被使能的中断请求。

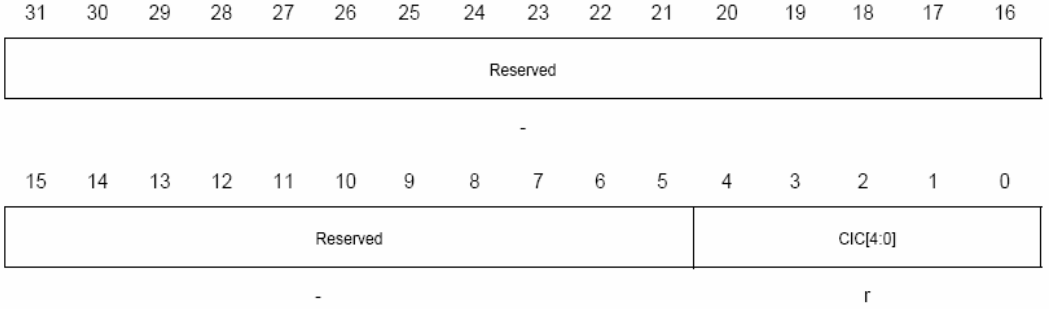
1：增强型中断控制器的 IRQ 输出到 CPU 被允许。

软件可以对这位进行读写。

5.3.2 当前中断通道寄存器（EIC_CICR）

地址偏置：04h

复位值：0000 0000h



EIC_CICR 报告当前被服务的中断通道的编号。因为总共有 32 个可能的通道 ID（0-31），所以有意义的寄存器位只有 5 位（位 4 到位 0）。

在复位之后，EIC_CICR 的值被赋予‘0’，并且只在处理器已经开始服务一个有效的 IRQ 中断请求之后才能由 EIC 更新，比如，在已经读完 IVR 之后一个时钟周期。

为了让这些发生，EIC 寄存器必须进行如下的配置：

- EIC_ICR 的 IRQ_EN bit=1（允许 nIRQ 信号通往 ARM7TDMI）
- EIC_IER0 不能全为 0（至少要保证一个中断通道是有效的）
- 在由 IERx 寄存器控制打开的中断通道中，至少有一个必须使相关 SIR 寄存器的 SIPL 域不被设置为 0，因为 EIC 只有在它检测到一个被使能的中断请求，并且其优先级数值大于 EIC_CIPR（当前中断优先级寄存器）的值的时候才能产生对处理器核的中断请求（使 nIRQ 信号线有效）。

当输入至 ARM7TDMI 的信号 nIRQ 被激活时，软件将会读取 EIC_IVR（中断向量寄存器）。这个读操作将通知 EIC 逻辑，ISR（中断服务程序）已经启动，CICR 可以被更新。

EIC_CICR 的值不能被软件更改（因为它是只读的寄存器）。

Bit 31:5　保留的，始终读为 0。

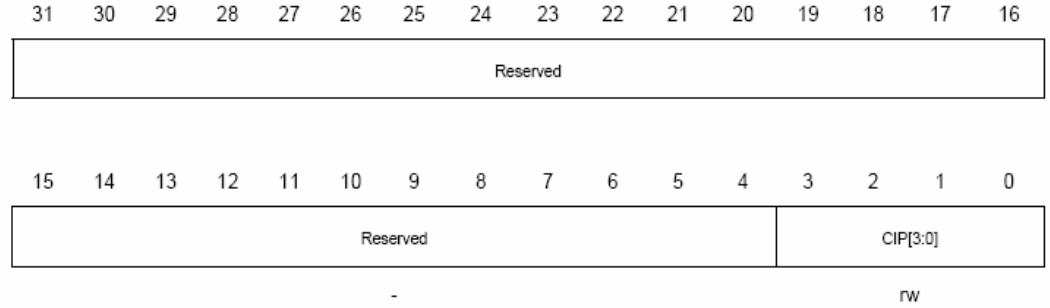
Bit 4:0　CIC[4:0]：当前中断通道

中断编号，该中断的服务程序目前正处在执行状态。这些都是只读位。

5.3.3 当前中断优先级寄存器（EIC_CIPR）

地址偏置：08h

复位值：0000 0000h



EIC_CIPR 寄存器报告当前被服务的中断的优先级。总共有 16 个可能的优先级（0 至 15）。所以有意义的寄存器位只有 4 位（3 至 0）。

复位之后，当前中断优先级（CIP）的值被设置为 0，只在处理核开始服务一个有效的 IRQ 中断请求之后才能由 EIC 更新。

为了让这些发生，EIC 寄存器必须进行如下的配置：

- EIC_ICR 寄存器中的 IRQ_EN 位必须是置位。
- 不能使 EIC_IERO 寄存器中的所有位都是‘0’（至少要有一个中断通道有效）。
- 由 EIC_IER 控制使能的中断通道，至少要有一个必须使相关 EIC_SIR 寄存器的 SIPL 域不被设置为 0，因为 EIC 只有在它检测到一个被使能的中断请求，并且其优先级数值大于 EIC_CIPR（当前中断优先级寄存器）的值的时候才能产生对处理器核的中断请求（使 nIRQ 信号线有效）。

当输入至 ARM7TDMI 的 nIRQ 信号被激活时，处理核将会读取 EIC_IVR。这个读操作将会告诉 EIC，ISR 已经被启动，且 EIC_CIPR 寄存器可以被适当更新。

只有在想要提升运行的 ISR 优先级，且仅当一个中断服务程序正在运行的时候，才能用软件更改 EIC_CIPR 的值。EIC 逻辑允许写入到 CIP 域中任何大于或等于对应于当前正在被服务的中断通

道的优先级的数值。

比如：假设 IRQ 信号处于置位，因为通道#4 有一个被使能的中断请求，它的优先级值为 7；在软件读取 EIC_IVR 寄存器之后，EIC 将 7 加载到 CIP 域。在中断服务程序完成之前，写入数值 7 到 15 都将被允许，而试图用低于 7 的优先级值去修改 CIP 内容将不产生作用。

用户软件必须要避免 FSM 处于等待且 EIC_IVR 产生不可预知值的状态。

Bit 31:0　保留，始终读为 0。

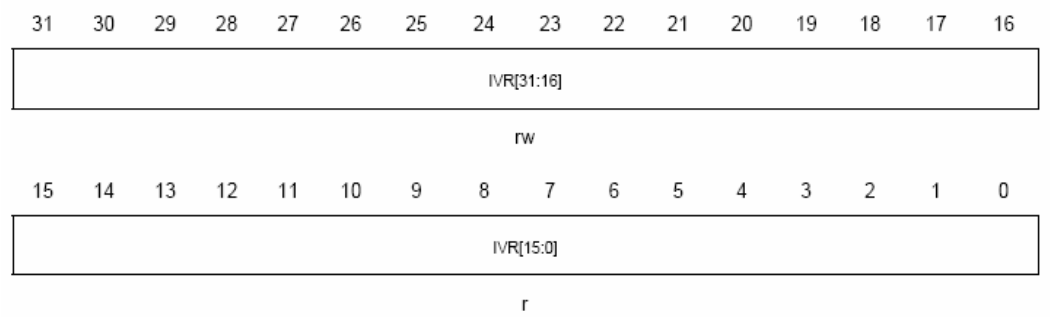
Bit 3:0　CIP[3:0]：当前中断优先级

当前处于执行状态中断的优先级数值。软件可以对这些位进行读写操作。

5.3.4 中断矢量寄存器（EIC_IVR）

地址偏置：18h

复位值：0000 0000h



EIC_IVR 是在检测到 nIRQ 信号有效之后软件必须读取的 EIC 寄存器。

对 EIC_IVR 的读操作告诉 EIC，对应于中断等待请求的中断服务例程（ISR）已经被启动。

这也就意味着：

- IRQ 信号可以被撤除。
- EIC_CIPR 和 EIC_CICR 可以被更新。
- 那些优先级低于或等于当前中断的中断请求不能被处理。

Bit 31:0　IVR(31:16)：中断向量（高位部分）

这个寄存器的值不取决于将要被服务的中断。它需要在初始化的时候由用户编程。它对所有中断通道来说是公共的。软件可以对这些位进行读写操作。

Bit 15:0　IVR(15:0)　中断向量（低位部分）

这个寄存器的值取决于将要被服务的中断（即，具有最高优先级的被使能的中断之一），它是对应于将要被服务的通道的 EIC_SIR 内源中断向量（SIV）的拷贝。它们都是只读位。

注意：EIC 并不关心 IVR 的内容：从控制器的角度来看它只是两个 16 位域的简单串联而已。

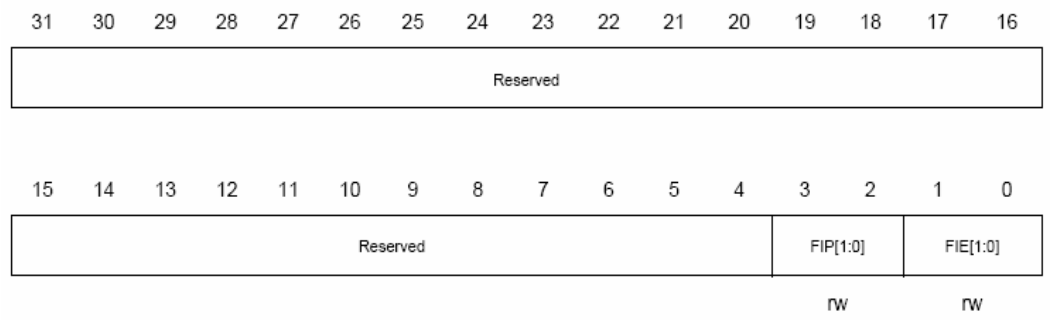
IVR=IVR（31:16）& SIRn(31:16)

需要写入 IVR（31:16）的是指向中断服务程序开始位置的内存地址的高位部分。域 SIRn(31:16)包含与特定通道 ISR 相关的内存地址的低 16 位。

只有在 CPU 处于非调试模式下，且用户代码在 ARMIRQ 模式下执行的时候，读取 IVR 才能被确认。

5.3.5 快速中断寄存器（EIC_FIR）

地址偏置：1Ch
复位值：0000 0000h



为了让控制器对 2 个快速中断（FIQ）通道做出反应，使能位 1 和 0 必须被置位为 1，第 3 和第 2 位表明哪个通通道是中断源。

Bit 31:4　保留，始终读为 0。

Bit 3:2　FIP[1:0]：通道 1 和 0 快速中断等待位

这些位在对应通道的快速中断请求时由硬件置位。这些位只能用软件清除，即，写入'0'没有作用，而写入'1'能清除该位。

0：没有快速中断等待在通道 n 中。

1：有快速中断等待在通道 n 中。

Bit 1:0　FIE[1:0]：FIQ 通道 1 和 0 中断使能位

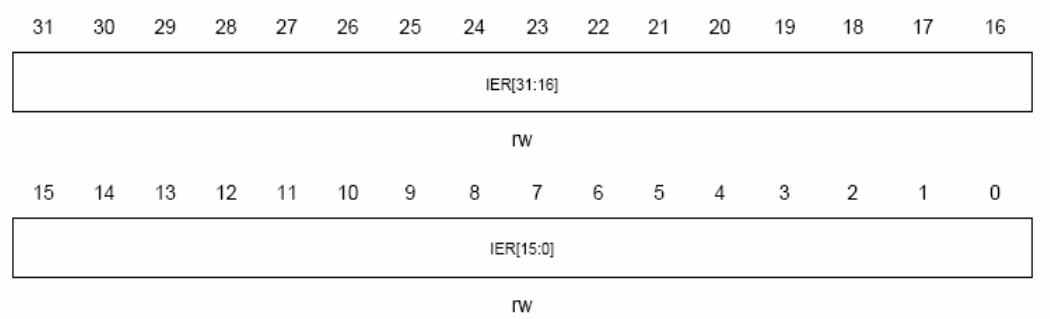
为了让控制器对指定通道上的 FIQ 做出响应，对应的 FIE 位必须被置位。

0：FIQ n 通道上的快速中断请求被禁止。

1：FIQ n 通道上的快速中断请求被使能。

5.3.6 中断使能寄存器 0（EIC_IER0）

地址偏置：20h
复位值：0000 0000h



Bit 31:0　IER[31:0]：通道 31 到 0 的中断使能位

EIC_IER0 是一个 32 位的寄存器，它为 32 个 EIC 中断输入通道中的每一个通道提供了一个使能位。

为了允许对一个特定中断通道的中断响应，在 EIC_IER0 寄存器中的对应位必须置为 1。一个'0'值阻止对应的中断等待位被置位。

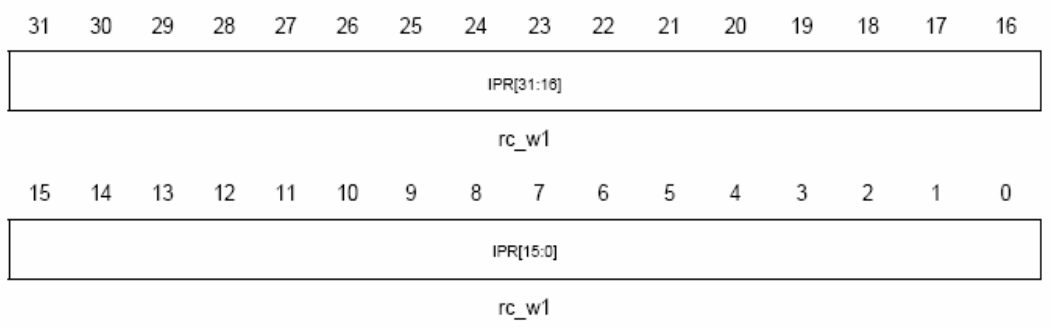
0：输入通道被禁止。

1：输入通道被允许。

5.3.7 中断等待寄存器 0 (EIC_IPR0)

地址偏置:40h

复位值:0000 0000h



Bit 31:0 IPR[31:0]:通道 31 至 0 为中断等待位

EIC_IPR0 是一个 32 位的寄存器，为 32 路 EIC 中断输入通道的每一路都提供一个等待位。

这里保存着通道中断状态的有关信息。如果使能寄存器 EIC_IER0 中的对应位已经被置位，则 EIC_IPR0 中的某位被置位就意味着相关通道已经声明了一个尚未被服务的中断请求。

这些位都是可读和可清零的，即写‘0’无效，而写‘1’就能清除该位。

0：没有中断等待。

1：中断正在等待

注意： 在退出一个 ISR 之前，软件必须已经清除了与被执行的中断服务程序相关联的 EIC_IPR0 位。这一位的清除操作将被 EIC 解释为中断结束（EOI）序列，并将允许中断堆栈弹出和对新中断的处理。

注意： 中断等待位必须小心处理，因为如果意外的等待位清除操作被执行的话，EIC 状态机及其内部优先级硬件堆栈可能被强迫进入不可恢复状态。

举例 1：

假设一个或多个中断通道被使能，并且它们的优先级大于零。一旦有中断请求产生，EIC 状态机就处理新的输入并给出 nIRQ 信号。如果在读取 EIC_IVR 之前，不管出于任何原因软件清除了等待位，nIRQ 信号都将一直保持有效，即使不再有等待的中断。清除 nIRQ 信号线逻辑的唯一方法就是读取 EIC_IVR(0x18)寄存器，或者发送一个软件复位信号给 EIC。

举例 2：

假设一个或多个中断通道被激活，且它们的优先级都大于零。一旦有中断请求产生，EIC 状态机就处理新的输入并给出 nIRQ 信号。如果在读取 EIC_IVR 之后，不管任何原因，软件在完成 ISR 之前清除与正被服务通道关联的等待位，EIC 都将检测到一个中断结束（EOI）命令，并发送一个弹出请求至优先级堆栈，因而一个新的中断，即使其优先级更低，也将得到处理。

为了结束中断处理过程（EOI），中断等待清除操作必须在相关 ISR 的末段执行，且清除的对象必须是与正被服务的通道对应的等待位。另一方面，一旦被服务通道的等待位被软件清除，即便是被错误地清除， EIC 都将进入 EOI（结束中断）过程。

注意： 为了安全地清除一个当前未被服务的 IRQ 的等待位，EIC_ICR 寄存器的 IRQ_EN 必须首先被清除。如果不这么做的话，EIC 状态机将进入一个不可恢复的状态。

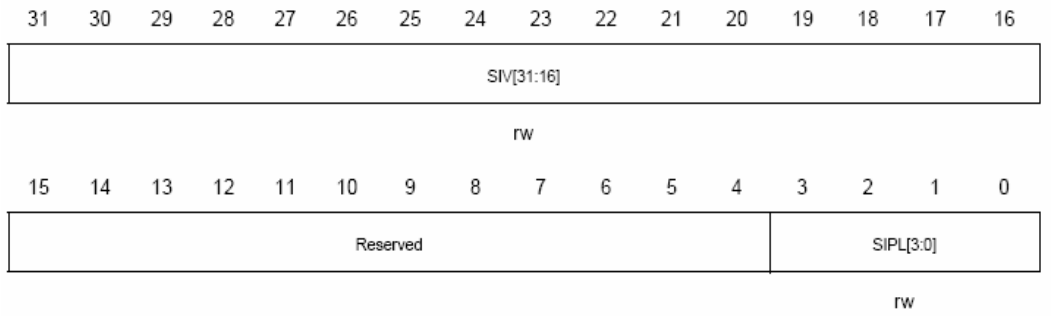
一般来说，在主程序中，清除一个等待位没有什么不好。但是，当这种操作是在一个 IRQ 程序内执行的时候，非常重要一点就是不要错误地清除当前被服务 IRQ 关联的 IPR 位。由于 IRQ_En 位冻结了堆栈，对当前 IRQ 的弹出操作将不会被执行，甚至于以后

当 IRQ 重新被使能时也不可能执行了。

5.3.8 源中断寄存器 - 通道 n(EIC_SIRn)

地址偏置：60h 至 DCh

复位值：0000 0000h



总共有 32 个不同的 EIC_SIRn 寄存器，对应于各个输入中断通道。

Bits 31:16 SIV[31:16]：对应中断通道 n（n=0…31）的源中断向量

这个域包含了中断向量中依赖于中断通道的部分，当 EIC_IVR（地址 0x18）被读取时，将被提供给处理器。

根据处理器核预期得到的不同类型（32 位地址或操作码，见 IVR 描述），SIV 将被加载为中断通道的 ISR 地址偏置，或者加载为第一条 ISR 指令操作码的低位部分（包含跳转偏移量）。

Bit 15:4 保留的，始终读为 0。

Bit 3:0 SIPL[3:0]：用于中断通道 n 的源中断优先级。这 4 位允许将中断通道与优先级值 0 至 15 联系起来。复位值为 0。

注意： 为了得到 EIC 逻辑的处理，一个中断通道必须拥有一个比当前中断优先级（CIP）更高的优先级。最低的 CIP 值是 0，因此，所有优先级值为 0 的中断源都将永远不能产生 IRQ 请求，即使已经被正确地使能。

5.3.9 EIC 寄存器映射

表 25 EIC 寄存器映射

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	EIC_ICR	reserved																														FIQ_EN	IRQ_EN
4	EIC_CICR	reserved																												CIC[4:0]			
8	EIC_CIPR	reserved																												CIP[3:0]			
18	EIC_IVR	Jump Instruction Opcode or Jump Base Address																Jump Offset															
1C	EIC_FIR	reserved																												FIP [2:0]		FIE [1:0]	
20	EIC_IER	IER[31:0]																															
40	EIC_IPR	IPR[31:0]																															
60	EIC_SIR0	SIV0[31:16]																reserved												SIPL0[3:0]			
64	EIC_SIR1	SIV1[31:16]																reserved												SIPL1[3:0]			
68	EIC_SIR2	SIV2[31:16]																reserved												SIPL2[3:0]			
6C	EIC_SIR3	SIV3[31:16]																reserved												SIPL3[3:0]			
70	EIC_SIR4	SIV4[31:16]																reserved												SIPL4[3:0]			
74	EIC_SIR5	SIV5[31:16]																reserved												SIPL5[3:0]			
78	EIC_SIR6	SIV6[31:16]																reserved												SIPL6[3:0]			
7C	EIC_SIR7	SIV7[31:16]																reserved												SIPL7[3:0]			
80	EIC_SIR8	SIV8[31:16]																reserved												SIPL8[3:0]			
84	EIC_SIR9	SIV9[31:16]																reserved												SIPL9[3:0]			
88	EIC_SIR10	SIV10[31:16]																reserved												SIPL10[3:0]			
8C	EIC_SIR11	SIV11[31:16]																reserved												SIPL11[3:0]			
90	EIC_SIR12	SIV12[31:16]																reserved												SIPL12[3:0]			
94	EIC_SIR13	SIV13[31:16]																reserved												SIPL13[3:0]			
98	EIC_SIR14	SIV14[31:16]																reserved												SIPL14[3:0]			
9C	EIC_SIR15	SIV15[31:16]																reserved												SIPL15[3:0]			
A0	EIC_SIR16	SIV16[31:16]																reserved												SIPL16[3:0]			
A4	EIC_SIR17	SIV17[31:16]																reserved												SIPL17[3:0]			
A8	EIC_SIR18	SIV18[31:16]																reserved												SIPL18[3:0]			
AC	EIC_SIR19	SIV19[31:16]																reserved												SIPL19[3:0]			
B0	EIC_SIR20	SIV20[31:16]																reserved												SIPL20[3:0]			
B4	EIC_SIR21	SIV21[31:16]																reserved												SIPL21[3:0]			
B8	EIC_SIR22	SIV22[31:16]																reserved												SIPL22[3:0]			
BC	EIC_SIR23	SIV23[31:16]																reserved												SIPL23[3:0]			
C0	EIC_SIR24	SIV24[31:16]																reserved												SIPL24[3:0]			
C4	EIC_SIR25	SIV25[31:16]																reserved												SIPL25[3:0]			
C8	EIC_SIR26	SIV26[31:16]																reserved												SIPL26[3:0]			
CC	EIC_SIR27	SIV27[31:16]																reserved												SIPL27[3:0]			
D0	EIC_SIR28	SIV28[31:16]																reserved												SIPL28[3:0]			
D4	EIC_SIR29	SIV29[31:16]																reserved												SIPL29[3:0]			
D8	EIC_SIR30	SIV30[31:16]																reserved												SIPL30[3:0]			
DC	EIC_SIR31	SIV31[31:16]																reserved												SIPL31[3:0]			

5.3.10 编程考虑

下面是有关如何对 EIC 寄存器编程以便快速起步的一些指导原则。在接下来的部分里，我们假设要对标准中断进行处理，比如，我们想要检测通道 22 上的中断，它的优先级为 5。

首先，你需要对通道 22 赋予优先级值和跳转地址。因此：

- 在 SIR22 寄存器的 SIPL 部分写入二进制值“0101”，即，优先级值 5（必须是非零才能允许 IRQ 产生）。

使用 2 个寄存器来提供通道中断矢量给 EIC 控制器（IVR[31:16]和 SIR22[31:16]）：

- 将对应中断通道 7 的中断服务程序开始的内存偏置地址（或跳转偏移）写入 SIR22[31:16]，即写入对应通道 22 的 SIR 寄存器的高位部分。
- 在 IVR 寄存器的高半字即 IVR[31:16]中插入跳转基地址（或跳转操作码）。

最后，必须在全局级别和中断通道级别上使能中断。为此，必须执行如下步骤：

- 将 ICR 的 IRQ_EN 位设置为 1
- 将 IER 的 22 位设置为 1

至于 FIQ 中断，因为它们没有向量化和优先级，只需要完成以上的前两步。假设你想让 FIQ 的

通道 1 生效：

- 将 ICR 的 FIQ_EN 位设置为 1。
- 将 FIR 寄存器中 FIE 的位#1 设置为 1。

5.3.11 应用注意事项

每个中断服务程序（ISR）都应该具备下面的代码块。

- 一个进入 ISR 的头部程序，必须具有以下几部分：

```
1) STMFD sp!, {r5,lr}   工作区 r5 和当前返回地址 lr_irq 推入系统堆栈。
2) MRS r5, spsr         将 spsr 存入 r5
3) STMFD sp! , {r5}     保存 r5
4) MSR cpsr_c,#0x1F     重新使能 IRQ，进入系统模式
5) STMFD sp! , {lr}     为系统模式保存 lr_sys
```

注意： r5 是一个本例中从可用的寄存器 r0 到 r12 中选择出来的通用寄存器。因为没有其他方法将 SPSR 直接保存到 ARM 堆栈，上述操作需要两步完成，使用 r5 做为临时寄存器。

- ISR 主体程序
- 一个离开 ISR 的尾部程序，必须包含：

```
1) LDMFD sp! , {lr}     为系统模式恢复 lr_sys
2) MSR cpsr_c , #0xD2   禁止 IRQ，进入 IRQ 模式
3) 清除 EIC 中的等待位（使用合适的 IPRx）
4) LDMFD sp! , {r5}     恢复 r5
5) MSR spsr , r5        恢复状态寄存器 spsr
6) LDMFD sp!，{r5, lr}  恢复状态 lr_irq 和工作空间
7) SUBS pc，lr，#4      从 IRQ 返回并重新使能中断
```

接下来的两部分将给出对上述代码的一些注释，以及从 ISR 中调用子程序的一些忠告。

5.3.11.1 避免 LR_sys 和 r5 寄存器内容丢失

第一个例子涉及到 LR_sys 内容丢失的问题：假设一个头部程序中没有指令 5 的 ISR（也就意味着尾部程序中没有指令 1）刚刚启动了；如下这些将会发生：

- 指令 4) 被执行（因此进入了系统模式）
- 以一个 BL 指令调用了一个子程序，LR_sys 现在包含了返回地址 A：第一条子程序指令应该将 LR_sys 寄存器存储在堆栈中（这条指令的地址叫做 B）
- 但在前一个操作来得及被执行之前，一个更高优先级的中断开始了
- 一个新的 ISR 将地址 B 存储在 LR_irq 中并进入系统模式
- 用一条 BL 指令调用了一个新的子程序：LR_sys 被装入新的返回地址 C（这个操作将覆盖先前的值 A!），地址 C 现在被存储在堆栈中。
- 最高优先级的 ISR 结束，地址 B 被恢复了：现在可以将 LR_sys 值送入堆栈了，但是它的值已经改变为地址 C（而不是 A）了。

避免上述危险状况的发生的方法是在头部程序的最后插入行 5，随之在尾部程序的开始插入行 1。

类似的原因可能导致寄存器 r5 的内容被毁，为了解决这个问题，必须加入头部程序中的行 3 和尾部程序中的行 4。

5.3.11.2 有关从 ISR 中调用子程序的提示

这一部分讨论一个子程序被 ISR 调用的例子。

假设这类程序以如下的指令开始：

```
STMFD SP! , {… , LR}
```

它可能如此结束：

```
LDMFD SP!, {… ,LR}
MOV PC, LR
```

如果一个更高优先级的 IRQ 发生在最后 2 个指令之间，而且新的 ISR 又调用另外一个子程序，LR 的内容将会丢失：如此一来，当最近的 IRQ 结束时，先前被中断的子程序将不能返回到正确的地址。

为了避免这种情况，前面的两个指令必须用单条指令代替：

```
LDMFD SP!，{… ,PC}
```

这条指令将自动将存储的链接寄存器直接移入程序计数器，使得子程序正确返回。

5.4 外部中断（XTI）

外部中断单元（XTI）的主要功能是管理外部中断信号线。XTI 连接到 EIC 模块的 IRQ5 通道。

使用 XTI 寄存器，可以将 14 个 I/O 编程为外部中断线或者唤醒线，可以将 MCU 从停止状态唤醒。

注意：只有 WAKEUP 管脚（P0.15）可以用来从 STANDBY 模式唤醒。

另外两条 XTI 线用于特定的中断源：

- 软件中断
- USB 结束悬挂事件

参考表 26。

一些外部中断信号线可以被映射到 I/O 口，这些 I/O 可以被允许作为 CAN、I^2C、BSPI 或 UART 外设的输入。这就是说你可以对它进行编程，使得任何在一个串行总线上的动作都可以产生一个中断并且将 MCU 从 STOP 状态唤醒。

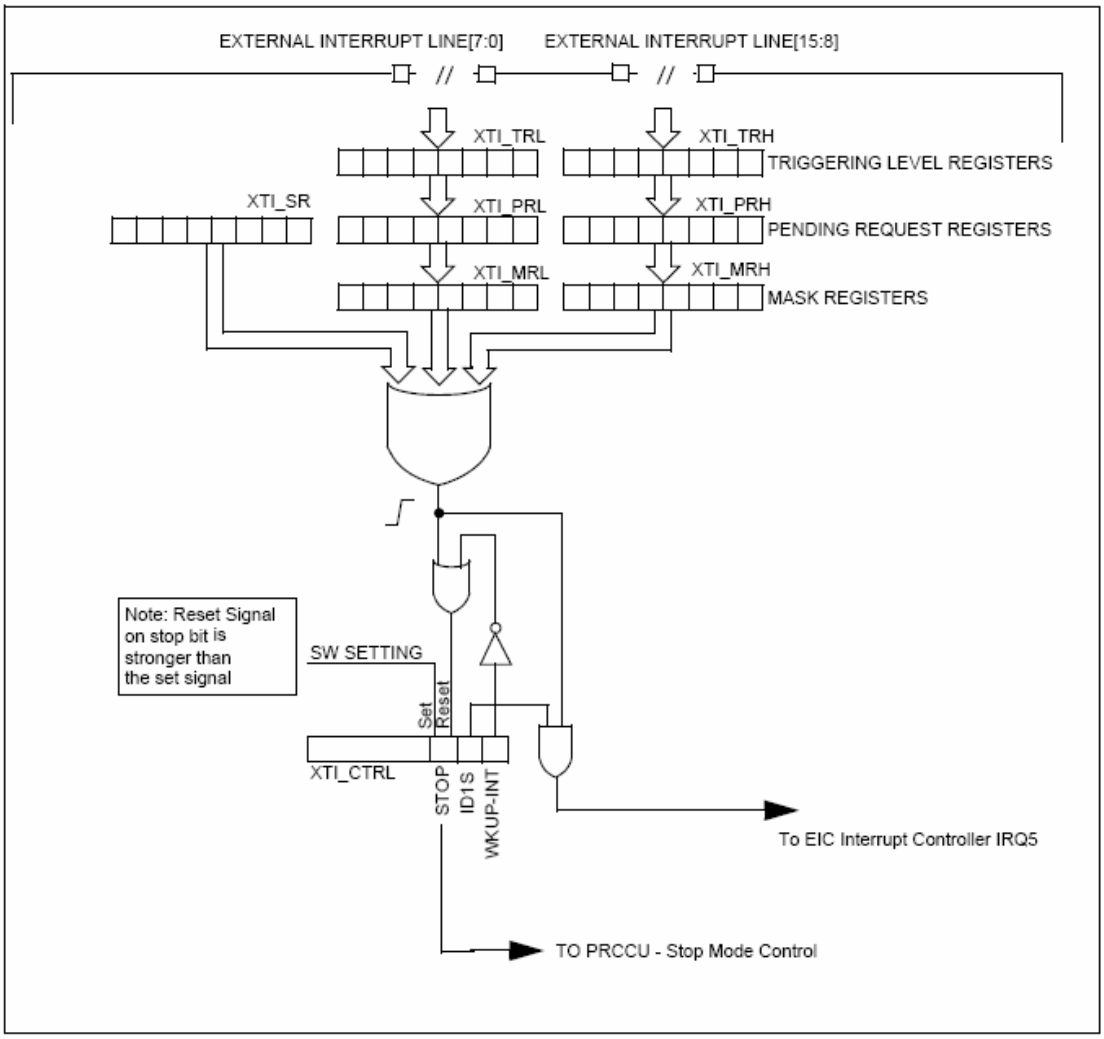
表 26 外部中断线映射

Wake-up line #	Wake-up line source
0	SW interrupt - no HW connection.
1	USB wake-up event: generated while exiting from suspend mode
2	Port 2.8 - External Interrupt
3	Port 2.9 - External Interrupt
4	Port2.10 - External Interrupt
5	Port 2.11 - External Interrupt
6	Port 1.11 - CAN module receive pin (CANRX).
7	Port 1.13 - HDLC clock (HCLK) ***or*** I2C.0 Clock (I0.SCL)
8	Port 1.14 - HDLC receive pin (HRXD) ***or*** I2C.0 Data (SDA)
9	Port 0.1 - BSPI0 Slave Input data (S0.MOSI) ***or*** UART3 Receive Data Input (U3.Rx)
10	Port 0.2 - BSPI0 Slave Input serial clock (S0.SCLK) ***or*** I2C.1 Clock (I1.SCL)
11	Port 0.6 - BSPI1 Slave Input serial clock (S1.SCLK)
12	Port 0.8 - UART0 Receive Data Input (U0.Rx)
13	Port 0.10 - UART1 Receive Data Input (U1.Rx)
14	Port 0.13 - UART2 Receive Data Input (U2.Rx)
15	Port 0.15 - WAKEUP pin ***or*** RTC ALARM

5.4.1 特性

- 外部中断线可以被用来将系统从 STOP 模式唤醒
- 唤醒或中断的可编程选择
- 可编程的唤醒触发沿极性
- 所有的唤醒线都可分别屏蔽
- 由软件可产生唤醒中断

图 28 XTI 框图



5.4.2 功能描述

5.4.2.1 中断模式

使用如下的步骤可将 16 条线设置为中断源：

1. 设置 16 条唤醒线的屏蔽位(XTI_MRL、XTI_MRH)。
2. 设置唤醒线的触发沿寄存器(XTI_PRL、XTI_TRH)。
3. 在 EIC 寄存器中，使能 IRQ5 中断通道，这样来自 16 条唤醒线中的中断就可以被正确识别了。
4. 清除 XTI_CTRL 寄存器中的 WKUP-INT 位，以禁止唤醒模式并允许中断模式；
5. 置位 XTI_CTRL 寄存器中的 ID1S 位，允许将 16 条唤醒线作为外部中断源信号线。

5.4.2.2 唤醒模式选择

使用如下步骤可将 16 线设置为唤醒源：

1. 设置 16 条唤醒线的屏蔽位(XTI_MRL、XTI_MRH)。
2. 设置唤醒线的触发沿寄存器(XTI_PRL、XTI_TRH)。
3. 如果在唤醒事件之后要运行一个中断程序，那么用 EIC 寄存允许 IRQ5 中断通道。否则，如果唤醒事件只是将代码在它被停止的地方重新运行，那么 IRQ5 中断通道必须被屏蔽。
4. 因为从 STOP 模式退出时 PRCCU 可以产生中断请求，所以当唤醒事件只是用于重新启动代码执行时，必须注意屏蔽它。
5. 置位 XTI_CTRL 寄存器中的 WKUP-INT 位，这样做是用来选择唤醒模式。
6. 置位 XTI_CTRL 寄存器中的 ID1S 位，允许将 16 条唤醒线作为外部中断源线。

5.4.3 编程考虑

以下将给出一些设计应用程序的指导原则。

5.4.3.1 进入/退出 STOP 模式的步骤

1. 通过写 XTI_TRH 和 XTI_TRL 寄存器编程外部唤醒线的触发极性。
2. 确保 XTI_MRH 和 XTI_MRL 寄存器中有一个屏蔽位为“1”（这样至少有一个外部唤醒线是不被屏蔽的）。
3. 清除至少一个未屏蔽的等待位：如果未屏蔽的等待位没有被清除，STOP 模式是不能进入的。
4. 置位 XTI_CTRL 寄存器中的 ID1S 和 WKUP-INT 位。
5. 为了从关联通道（IRQ5）中产生中断，置位 EIC 寄存器中相关的允许位、屏蔽位和优先极位。
6. 复位 XTI_CTRL 寄存器中的 STOP 位和 CLK_FLAG 寄存器(PRCCU)中的 STOP_I 位。
7. 为了进入 STOP 模式，向 XTI_CTRL 寄存器中的 STOP 位写入序列 1、0、1。正如前面所说，即使没有严格按顺序运行（允许中间加入指令），这三个写操作仍然是有效的。为了复位这个序列，在 XTI_CTRL 寄存器的 STOP 位写入两次逻辑 0 就足够了。
8. 紧跟在 STOP 序列后执行的代码必须检查 STOP 位和 PRCCU STOP_I 位的状态，用以确定器件是否进入 STOP 模式。如果设备没有进入 STOP 模式，就必须从开头重新执行进入步骤，否则程序将从下一点继续执行。
9. 查询唤醒等待位，用以决定哪条唤醒线将引起从 STOP 状态的退出。
10. 清除被置位的唤醒等待位。

5.4.3.2 多个等待位的同时设置

有可能发生几个同时发生的唤醒事件设置了不同的等待位。为了接受外部唤醒/中断线中的继发事件，一旦第一个中断程序已经被执行，XTI_PRX 寄存器中的相关等待位就必须有一个被清除：这个操作允许在内部线上产生一个上升沿（如果至少还有一个等待位被置位且未被屏蔽)，这样可以再次设置中断控制器中的等待位。中断控制器的同一通道中下一个中断能否得到服务取决于屏蔽位的状态。有两种可能的情况：

1. 用户选择复位所有的等待位：在这个通道中将不会有更多的中断产生。在这种情况下用户必须：
   — 清除中断控制器的屏蔽位（以避免在下一次复位操作中产生中断请求假象)
   — 复位 XTI_PRH 寄存器
   — 复位 XTI_PRL 寄存器
2. 用户选择保持至少一个等待位有效：在中断控制器通道上至少还有一个中断请求将被产生。在

这种情况下，用户必须复位这个想得到的等待位。这个操作将在中断控制器通道上产生一个上升沿，相应的等待位将再次被置位。一个在这个通道上的中断是否被服务取决于相关屏蔽位的状态。

5.4.3.3 STOP 模式进入条件

假设器件处于运行模式：STOP 位设置过程中将有可能发生如下情况：

情况 1：错误的 STOP 位设置过程

如果在 STOP 位设置过程中接受中断请求，这种情况有可能发生。在这种情况下检测 STOP 和 STOP_I 位将得到：

STOP=0, STOP_I=0

这就是说由于错误的 STOP 位设置过程，设备没有进入 STOP 模式：用户必须重新尝试这个过程。

情况 2：正确的 STOP 位设置过程

在这种情况下设备进入 STOP 模式。

为了退出 STOP 模式，必须接受一个唤醒中断。这就意味着：STOP=0, STOP_I=1.

这就是说由于一个外部唤醒事件设备经进入并且退出了 STOP 模式。

情况 3：在 STOP 位设置过程中一个唤醒事件在外部唤醒线上发生

有两种可能情况：

1. 到 CPU 的中断请求被禁止：在这种情况下，器件将不进入 STOP 模式，没有中断服务程序会被运行，并且程序将从 STOP 位设置过程后面的指令继续执行。STOP 位和 STOP_I 位的状态再次成为：

   STOP=0, STOP_I=0

   应用程序可以通过查询外部线的等待位来确定设备为什么没有进入 STOP 状态。
2. 到 CPU 的中断请求被使能：在这种情况下，设备不进入 STOP 模式，中断服务程序将会被运行。STOP 位和 STOP_I 位的状态再次成为：

   STOP=0, STOP_I=0

   通过查询外部线的等待位（至少有一个为 1），中断服务程序可以确定为什么设备没有进入 STOP 模式。

如果设备真的从 STOP 模式退出，PRCCU 的 STOP_I 位仍然被置位，必须用软件复位。否则，如果一个中断请求在 STOP 位设置过程中被接受，PRCCU 的 STOP_I 位将被复位。这就意味着系统过滤了 STOP 模式的进入请求。

中断程序可以用 WKUP-INT 位来检测和鉴别事件是来自中断模式还是唤醒模式，让代码运行不同的程序。

为了退出 STOP 模式，16 条（未屏蔽）唤醒线中的一个产生事件就足够了：时钟在经过振荡器重新启动所需要的延时之后重新开始。

注意：在从 STOP 模式被唤醒之后，软件可以成功地复位等待位（对变化边沿敏感），即使是在相应的唤醒线仍然起作用的情况下(高或低，取决于对事件触发寄存器的编程)；用户必须查询外部引脚的状态以发现并且鉴别来自长事件中的短事件(例如，具有不同按键长度的键盘输入)。

5.4.4 寄存器描述

5.4.4.1 XTI 软件中断寄存器(XTI_SR)

地址偏移：1Ch

复位值：00h

7	6	5	4	3	2	1	0
XTIS7	XTIS6	XTIS5	XTIS4	XTIS3	XTIS2	XTIS1	XTIS0
rw	rw	rw	rw	rw	rw	rw	rw

位 7:0 = XTIS[7:0] ：软件中断等待位。

这些位可以由软件设置以实现软件中断。

中断程序必须清除被置位的等待位。这些中断被全局中断许可（XTI_CTRL 寄存器中的 ID1S 位）屏蔽。

0：没有软件中断等待；

1：有软件中断等待。

5.4.4.2 唤醒控制寄存器

地址偏移：24h

复位值：00h

7	6	5	4	3	2	1	0
reserved					STOP	ID1S	WKUP-INT
-					rw	rw	rw

位 7:3 = 保留

位 2 = STOP：停止位

为了进入 STOP 模式，用三个写操作（不一定要连贯的）向这个位写入序列 1、0、1。当正确的顺序被认可时，STOP 位被置位，并且 PRCCU 使 MCU 进入 STOP 模式。这个软件过程只有在下列条件成立时才能成功：

— WKUP-INT 位为“1”；

— 所有未屏蔽的等待位被复位；

— 至少一个屏蔽位为“1”（至少有一个外部唤醒线没有被屏蔽）。

否则器件无法进入 STOP 模式，程序代码继续执行，并且 STOP 位仍然是清除的。

如果当设备处于 STOP 模式时，出现来自任何一个未屏蔽唤醒线的唤醒中断，则 STOP 位被硬件复位。在下面两种情况下 STOP 位为“1”：

— 顺序写入过程的第一个写入指令之后（“1”被写入 STOP 位）；

— 顺序写入过程成功结束的时候（即，在第三个写入指令之后).

警告：如果中断请求在这个写入过程中被接受，系统将不会进入 STOP 模式（因为这个写入过程没有完成）。在中断服务程序结束之后，建议通过两次向 XTI_CTRL 寄存器的 STOP 位写入逻辑 0 来复位这个过程的状态机。否则，这个不完整的过程将等待补充完整（只有在过程的第三次正确写指令之后 STOP 位才被置位）。

警告：任何时候向设备发出 STOP 请求，系统都需要几个时钟周期才能进入 STOP 模式）（要获得更多的细节请参阅 PRCCU 那一章）。因此，在进入 STOP 模式之前，跟在 STOP 位设置过程指令之后的指令可能会开始执行（也要考虑 ARM7 三级流水线的作用）。为了避免在正确的 STOP 位设置过程之后，但还没有进入 STOP 模式之前运行任何有效的指令，在 STOP 位设置过程之后（在第三次 STOP 位写指令之后）必须要执行几条（至少 6 条）无用的哑指令。另外，如果在退出 STOP 模式时要执行一个中断程序，必须增加另外一组哑指令（至少三条），这样做是为了将潜伏周期考虑在内。这里所考虑的情况是，当进入 STOP 模式时，流水线内容也被冻结；当系统重新启动时，第一条要运行的指令在进入 STOP 模式之前已经完成取指和译码。下面是一些管理 STOP 模式进入/退

出的示例程序代码。

```
LDR R0, = APB0 + APB_XTIP   ; 设置唤醒模块基地址
MOV R1, #3                  ; 设置控制寄存器
STR R1, [R0, #XTI_CTRL]
MOV R3, #0x08               ; 去掉唤醒线3的屏蔽
STR R3, [R0, #XTI_MRL]
MOV R2, #0x07               ; 用R2向STOP位写入'1'
STR R2, [R0, #XTI_CTRL]     ; 第1条序列指令(写'1')
STR R1, [R0, #XTI_CTRL]     ; 第2条序列指令(写'0')
STR R2, [R0, #XTI_CTRL]     ; 第3条序列指令(写'1')
MOV R1, R1                  ; 一组9条哑指令
MOV R1, R1
MOV R1, R1
MOV R1, R1
MOV R1, R1
MOV R1, R1
MOV R1, R1
MOV R1, R1
MOV R1, R1
```

如果你想让系统从 STOP 模式重启，但不进入中断服务程序，而是直接执行在 STOP 序列之后的第一条有效指令，只需要 6 条哑指令就够了。

位 1 = ID1S ：XTI 全局中断屏蔽。

这一位可以通过软件置位和清除。

0：XTI 中断禁止；

1：XTI 中断允许。

警告：为了避免在 EIC 的 IRQ5 通道上产生虚假中断请求，建议在修改 ID1S 位之前清除在 EIC IER 寄存器中对应的允许位。

位 0 = WEUP-INT：唤醒中断。这一位可以通过软件置位和清除。

0：16 条唤醒线可以被用来在 EIC 中断控制器的 IRQ5 通道上产生中断；

1：16 条唤醒线被作为从 STOP 模式退出的唤醒源。

5.4.4.3 XTI 标志寄存器高位(XTI_MRH)

地址偏移：28h

复位值：00h

7	6	5	4	3	2	1	0
XTIM15	XTIM14	XTIM13	XTIM12	XTIM11	XTIM10	XTIM9	XTIM8
rw	rw	rw	rw	rw	rw	rw	rw

位 7:0 = XTIM[15:8]：唤醒屏蔽位。

如果 XTIMx 被置位，当对应的 XTIPx 等待被置位时，中断或唤醒事件（取决于 ID1S 和 WKUP-INT 比特）就会产生。更具体来说，如果 XTIMx=1 并且 XTIPx=1，那么：

— 如果 ID1S=1 并且 WKUP-INT=1，那么产生中断和唤醒事件；

— 如果 ID1S=1 并且 WKUP-INT=0，那么只产生中断事件；

— 如果 ID1S=0 并且 WKUP-INT=1，那么只产生唤醒事件；

— 如果 ID1S=0 并且 WKUP-INT=0，那么既不产生中断也不产生唤醒事件。

如果 XTIMx 被复位，就不会产生唤醒事件。

5.4.4.4 XTI 屏蔽寄存器低位(XTI_MRL)

地址偏移：2Ch

复位值：00h

7	6	5	4	3	2	1	0
XTIM7	XTIM6	XTIM5	XTIM4	XTIM3	XTIM2	XTIM1	XTIM0
rw	rw	rw	rw	rw	rw	rw	rw

位 7:0 = XTIM[7:0]：唤醒屏蔽位。

如果 XTIMx 被置位，当对应的 XTIPx 等待位被置位时，中断或唤醒事件（取决于 ID1S 和 WKUP-INT 比特）就会产生。更具体来说，如果 XTIMx=1 并且 XTIPx=1，那么：

— 如果 ID1S=1 并且 WKUP-INT=1，那么产生中断和唤醒事件；

— 如果 ID1S=1 并且 WKUP-INT=0，那么只产生中断事件；

— 如果 ID1S=0 并且 WKUP-INT=1，那么只产生唤醒事件；

— 如果 ID1S=0 并且 WKUP-INT=0，那么既不产生中断也不产生唤醒事件。

如果 XTIMx 被复位，就不会产生唤醒事件。

5.4.4.5 XTI 触发极性寄存器高位(XTI_TRH)

地址偏移：30h

复位值：00h

7	6	5	4	3	2	1	0
XTIT15	XTIT14	XTIT13	XTIT12	XTIT11	XTIT10	XTIT9	XTIT8
rw	rw	rw	rw	rw	rw	rw	rw

位 7:0 = XTIT[15:8]：XTI 触发极性位。

这些位用软件置位和清除。

0：对应的 XTIPx 等待位在输入唤醒线的下降沿被置位。

1：相应的 XTIPx 等待位在输入唤醒线的上降沿被置位。

5.4.4.6 XTI 触发极性寄存器低位(XTI_TRL)

地址偏移：34h

复位值：00h

7	6	5	4	3	2	1	0
XTIT7	XTIT6	XTIT5	XTIT4	XTIT3	XTIT2	XTIT1	XTIT0
rw	rw	rw	rw	rw	rw	rw	rw

位 7:0 = XTIT[7:0]：XTI 触发极性位。

这些位被软件置位和清除。

0：对应的 XTIPx 等待位在输入唤醒线的下降沿被置位。

1：对应的 XTIPx 等待位在输入唤醒线的上降沿被置位。

警告：

1. 由于外部唤醒线被边沿触发，在这些线上不能有短脉冲干扰。
2. 当正在写入 XTI_TRH 和 XTI_TRL 寄存器时，如果外部唤醒线出现上升沿或下降沿，等待位将不会被置位。

5.4.4.7 XTI 等待寄存器高位(XTI_PRH)

地址偏移：38h
复位值：00h

7	6	5	4	3	2	1	0
XTIP15	XTIP14	XTIP13	XTIP12	XTIP11	XTIP10	XTIP9	XTIP8
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

位 7:0 = XTIP[15:8]：XTI 等待位。
当相应唤醒线上发生触发事件时，这些位被硬件置位。通过软件只能将这些位写成“0”
0：没有唤醒触发事件发生；
1：有唤醒触发事件发生。

5.4.4.8 XTI 等待寄存器低位(XTI_PRL)

地址偏移：3Ch
复位值：00h

7	6	5	4	3	2	1	0
XTIP7	XTIP6	XTIP5	XTIP4	XTIP3	XTIP2	XTIP1	XTIP0
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

位 7:0 = XTIP[7:0]：XTI 等待位。
当对应唤醒线上发生触发事件时，这些位被硬件置位。通过软件只能将这些位写成“0”
0：没有唤醒触发事件发生；
1：有唤醒触发事件发生。

5.4.5 XTI 寄存器映射

表 27. XTI 寄存器映射

Address Offset	Register Name	7	6	5	4	3	2	1	0
1C	XTI_SR	XTIS(7:0)							
24	XTI_CTRL	reserved					STOP	ID1S	WKUP-INT
28	XTI_MRH	XTIM(15:8)							
2C	XTI_MRL	XTIM(7:0)							
30	XTI_TRH	XTIT(15:8)							
34	XTI_TRL	XTIT(7:0)							
38	XTI_PRH	XTIP(15:8)							
3C	XTI_PRL	XTIP(7:0)							

要获得基地址，请参考表 3“APB2 存储器映射”。

6 实时时钟（RTC）

6.1 简介

实时时钟提供了一组连续运转的计数器，辅以适当的软件，可用来实现时钟日历功能。可以写入计数器的值以设置系统当前的时间/日期。

实时时钟包含一个 APB 的从接口，用来提供对内部 32bit 寄存器按字的读写；当主电源供电停止时，这个接口与 APB 总线断开的。

6.2 主要特征

- 可编程的预分频器：外部时钟可被分到2^{20}倍。
- 用于长时期测量的32位可编程计数器。
- 外部时钟输入（必须比PCLK2时钟至少慢4倍，通常是32kHz）。
- 单独的供电电源。
- 4个专用的可屏蔽中断线：

  —告警中断，用于产生一个软件可编程警告中断。

  —秒中断，用于产生一个可编程周期长度时间（最大到1秒）的周期性的中断信号。

  —溢出中断，用于检测什么时候内部可编程计数器计数溢出至0。

  —全局中断：对上述中断的逻辑“或”功能，允许一个单独的中断通道来管理所有的中断源。

6.3 功能描述

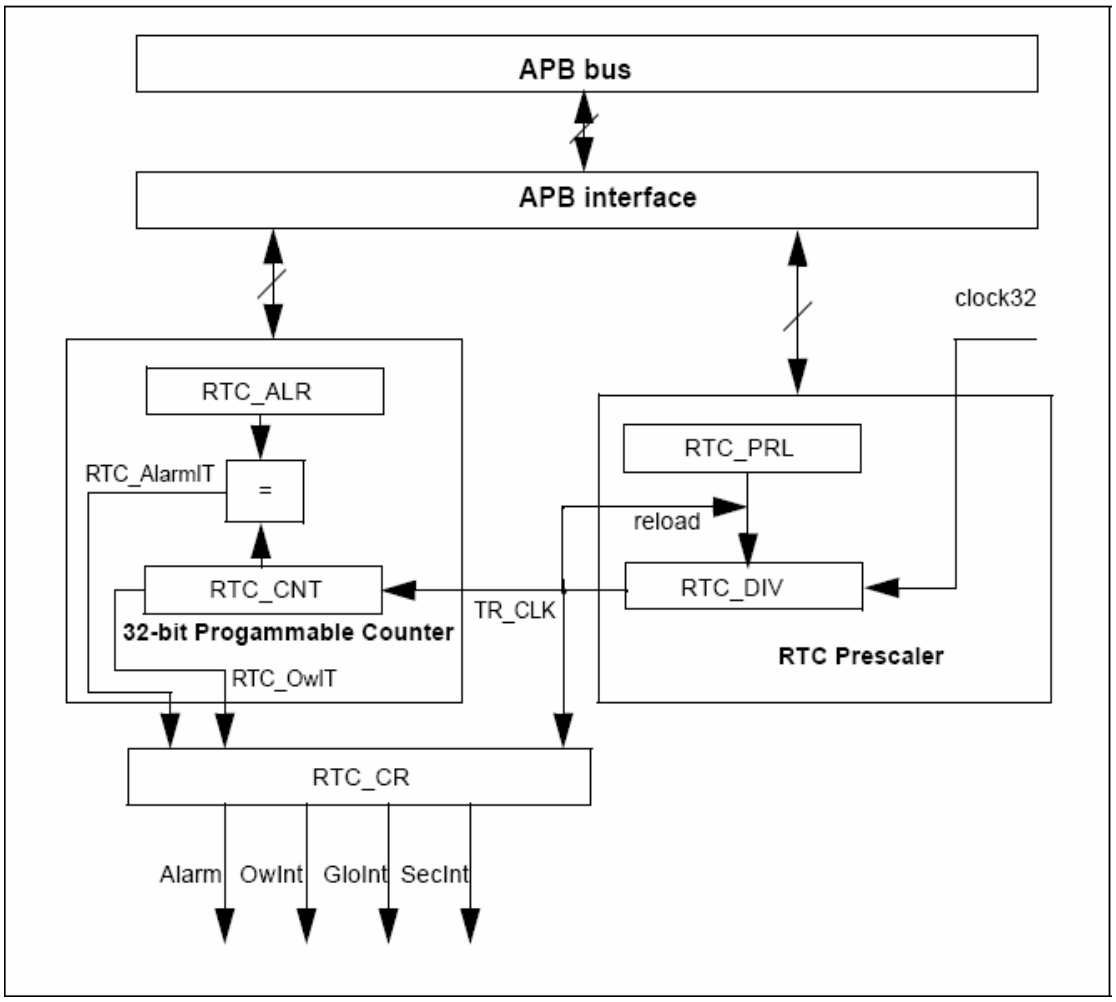
6.3.1 概述

实时时钟包括两个主要单元（参见图 29），第一个单元（APB 接口）起到与 APB 总线的接口作用。这个单元也包含一组 16 位的寄存器，与 PCLK2 时钟同步，从 APB 总线可用读或写模式访问（更详细内容请参考寄存器说明部分）。APB 接口由 PCLK2 时钟驱动。

另一个单元（RTC 核心）包含一串由 2 个主要模块构成的可编程计数器。第一个模块是 RTC 预分频器模块，用来形成 RTC 的时基 TR_CLK，TR_CLK 可编程设定最大周期达 1 秒。这个模块包含了一个 20 比特的可编程分频器（RTC 预分频器）。在通过 RTC_CR 寄存器设定使能的情况下，在每一个 TR_CLK 周期，RTC 都将产生一个中断（SecInt）。第二个模块是一个 32 比特的可编程计数器，它能被初始化为当前的系统时间。系统时间以 TR_CLK 的速率增长，并且与一个可编程的日期（存储在 RTC_ALR 寄存器）相比较，在 RTC_CR 控制寄存器设定为允许时，用来产生一个告警中断。

重要提示：由于 RTC 有两个不同的时钟域，在从停止模式唤醒之后，APB 接口（由 PCLK2 控制）不能匹配 RTC 域（由 32kHz 振荡器始终驱动）的寄存器 / 计数器的值，因为当系统处于停止模式时，RTC 域仍继续运行。因此为了避免从 APB 接口读到错误的值，在退出停止模式之后和读 RTC 寄存器之前．应用程序应等候至少 1 个 RTC 周期或者 31.25us。相同的防范也适用于 WFI 模式，因为 APB 接口在此模式下关闭，之后又开启。

图 29 RTC 简化方块图



6.3.2 复位过程

系统所有寄存器都通过系统复位或软件复位来异步地复位，但 RTC_ALR，RTC_CNT，RTC_DIV 除外。这些寄存器和实时时钟计数器只能用低电压检测器（上电复位）来复位．它们不受任何其他复位源的影响，也不受待机模式影响。

6.3.3 自由运行模式

在上电复位后，该外围设备进入自由运行模式。在这个工作模式下，RTC 预分频器和可编程计数器开始计数。中断标志也被启动，但是因为中断信号是被屏蔽的，所以没有中断产生。中断信号必须通过设定 RTC_CR 寄存器中适当的位才能被允许。为了避免虚假的中断形成，推荐在使能中断信号之前清除之前的中断请求。

6.3.4 配置模式

为了写入 RTC_PRL，RTC_CNT，RTC_ALR 寄存器，该外设必须进入配置模式。这一模式通过设定 RTC_CRL 寄存器中的 CNF 位来进入。

此外，任何写入 RTC 寄存器的操作只有在之前的写入操作完成后才被允许。为了让软件能检测这种情况，RTC_CR 寄存器提供了 RTOFF 位用来指示寄存器的更新正在进行中。只有当这一位的值为“1”时，RTC 计数器才可以写入新的值。

配置过程：

1. 查询 RTOFF，等候直到它的值变为“1”。
2. 置位 CNF 位，进入配置模式。
3. 写入一个或多个 RTC 寄存器。
4. 清零 CNF 位，退出配置模式。

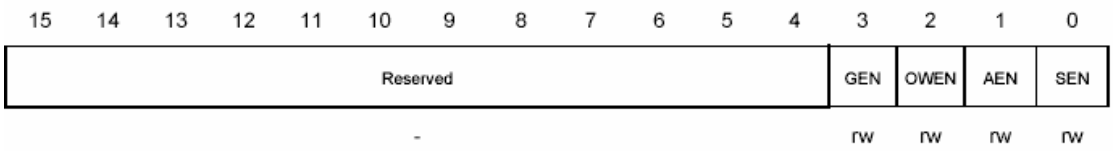
当 CNF 位清零时，写操作才能执行，并且写操作需要至少 2 个 Clock32 周期才能完成。

6.4 寄存器描述

RTC 寄存器不能按字节访问。保留位不能写入，且它们的值永远读为“0”。

6.4.1 RTC 控制寄存器高位部分（RTC_CRH）

地址偏移量：00h
复位值：0000h



这些位用于屏蔽中断请求。注意在复位时所有中断都被禁止，所以保证初始化后，可以对 RTC 寄存器写入而不用担心会有中断请求在等待。当外围电路正在完成一个之前的写操作（标志位 RTOFF=0，参考“配置模式”描述）时，RTC_CRH 寄存器不能写入。

RTC 的功能被这个控制寄存器控制。某些位必须通过特殊的配置过程来写入（参见“配置模式”的描述）。

Bit15：4 = 保留，始终读为“0”

Bit3=GEN：全局中断使能
0：屏蔽全局中断
1：允许全局中断

Bit2=OWEN：溢出中断使能
0：屏蔽溢出中断
1：允许溢出中断

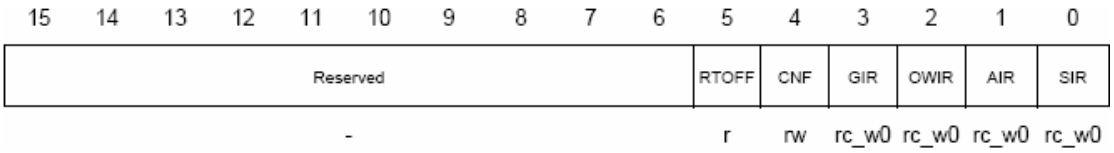
Bit1=AEN：告警中断使能
0：屏蔽告警中断
1：允许告警中断

Bit0=SEN：秒中断使能
0：屏蔽秒中断
1：允许秒中断

6.4.2 RTC 控制寄存器低位部分（RTC_CRL）

地址偏移量：04h

复位值：0020h



RTC 的功能被这个控制寄存器控制。当外围电路正在完成一个之前的写操作（标志位 RTOFF=0，参考“配置模式”的描述）时，RTC_CR 寄存器不能写入。

Bit15：6=保留，始终读为“0”

Bit5=RTOFF：RTC 操作停止

用这一位，RTC 报告了最后一个对其寄存器的写操作的执行状态，指出它是否已经完成。当它的值为“0”时，不能写入任何 RTC 寄存器。这一位只能读。

0：对 RTC 寄存器的最后写操作仍在进行中。

1：对 RTC 寄存器上的最后写操作已经结束。

Bit4=CNF：配置标志

软件必须设置这一位才能进入配置模式，以便允许新的值写入 RTC_CNT，RTC_ALR 或 RTC_PRL 寄存器。只有已经被置 1 的 CNF 位被软件置“0”时，写操作才能执行。

0：退出配置模式（开始 RTC 寄存器的更新）。

1：进入配置模式。

Bit3=GIR：全局中断请求

这一位包含全局中断请求信号的状态，当至少一个其他中断有效时，这一位变为高电平值。当这一位被置位时，只有 GEN 位置位时，对应的中断才能产生。GIR 位只能通过硬件置位，通过软件清零，而写入“1”时它不会变化。

0：GloInt 全局中断条件不满足。

1：GloInt 全局中断请求在等待。

Bit2=OWIR：溢出中断请求

这一位存储了周期中断请求信号（RTC_OwIT）的状态，这一信号是由 32 位可编程计数器的溢出形成的。如果系统时间必须保持更新（与日期同步），这个中断可能用于将系统从长期的待机状态中唤醒。当这一位是“1”时，只有 OWEN 位设置为“1”时，相应的中断才能产生。OWEN 位只能通过硬件设置为“1”，只能通过软件清零，写入“1”时它不会变化。

0：溢出中断条件不满足。

1：溢出中断请求在等待。

Bit1=AIR：告警中断请求

这一位包含了周期中断请求信号（RTC_AlarmIt）的状态，这一信号是当到达了在 RTC_ALR 寄存器中设定的门限时，由 32 位可编程计数器产生的。当这一位是“1”时，只有 AEN 位设置为“1”时，相应的中断才能产生。AIR 位只能被硬件设置为“1”，只能通过软件清零，而写入“1”时它不会变化。

0：告警中断条件不满足

1：告警中断请求正在等待。

Bit0=SIR：秒中断请求

这一位包含了由 20 位可编程预分频器的溢出产生的秒中断信号（RTC_SecIt）的状态，这一预分频

器用来增加 RTC 计数器的值。因此这一中断提供了一个周期信号，其周期对应着编程确定的 RTC 计数器分辨率（通常是 1 秒）。当这一位是“1”时，只有 SEN 位设置为“1”时，相应的中断才能形成。SIR 位只能被硬件设置为“1”，只能通过软件清零，而写入“1”时它不会变化。。
0：“秒”中断条件不满足。
1：“秒”中断请求正在等待。

任何中断请求都会一直保持等待状态，直到适当的 RTC_CR 请求位被软件置位，通知中断请求已经被接受。

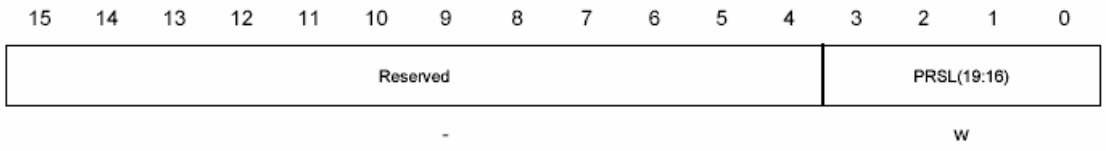
注意复位时中断是被禁止的，可以写入 RTC 寄存器，没有中断请求会发生等待。

6.4.3 RTC 预分频器加载寄存器高位（RTC_PRLH）

地址偏移量：08h
只能写入（参见 117 页的“配置模式”）
复位值：0000h

预分频器加载寄存器保持了 RTC 预分频器的计数周期值。它们由 RTC_CR 寄存器的 RTOFF 位提供写保护，写操作只有在 RTOFF 位的值为“1”时才被允许。

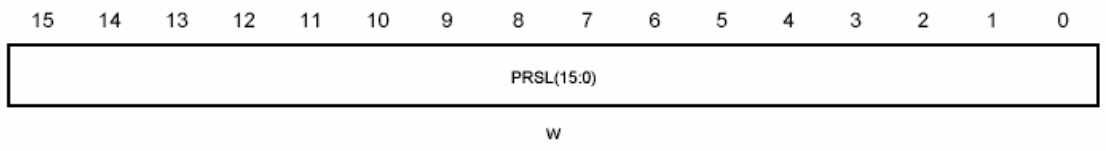
Bit15:4=保留，永远读为“0”



Bit3:0=PRSL[19:16]：RTC 预分频器再装入值高位部分

6.4.4 RTC 预分频器加载寄存器低位（RTC_PRLL）

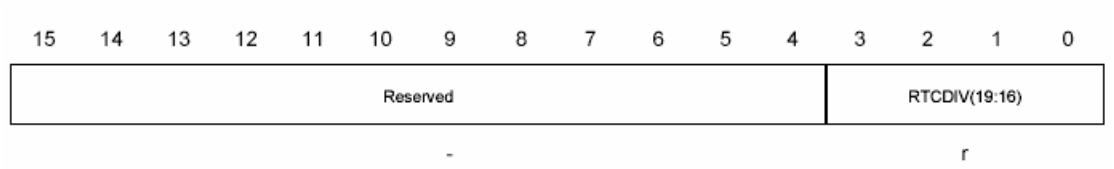
地址偏移量：0Ch
只能写入（参见 117 页的“配置模式”）
复位值：8000h



Bit15:0=PRSL[15:0]：RTC 预分频器再装入值低位部分
注意：复位值设定对 32kHz 的振荡器的 TR_CLK 信号周期为 1 秒。

6.4.5 RTC 预分频器分频寄存器高位（RTC_DIVH）

地址偏移量：10h
复位值：0000h
Bit15:4=保留，永远读为“0”

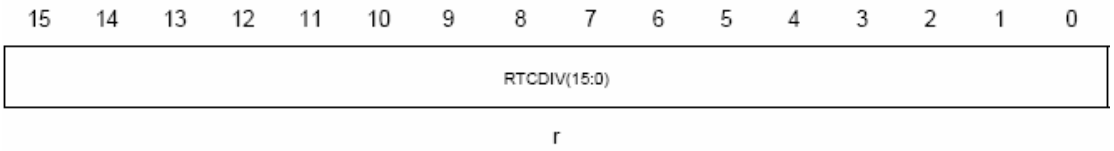


每一个 TR_CLK 周期中，RTC_PRL 寄存器中存储的值重装入 RTC 预分频器内的计数器。为了得到一个精确的时间度量，可以读出预分频器计数器的当前值而不停止其计数，这一值存储在 RTC_DIV 寄存器中。这个寄存器的值只能读出，并且在 RTC_PRL 或 RTC_CNT 寄存器有任何改变后被硬件重装。

Bit3:0=RTCDIV[19:16]：RTC 时钟分频器高位部分

6.4.6 RTC 预分频器分频寄存器低位（RTC_DIVL）

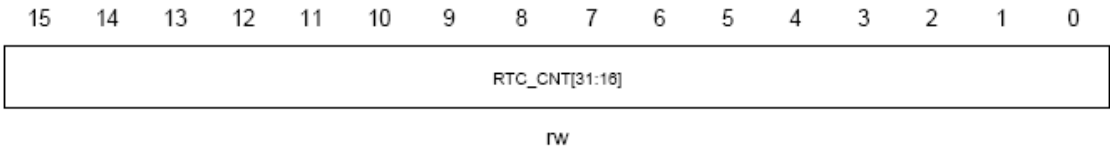
地址偏移量：14h
复位值：8000h



Bit3:0=RTCDIV[15:0]：RTC 时钟分配频器低位部分

6.4.7 RTC 计数寄存器高位（RTC_CNTH）

地址偏移量：18h
复位值：0000h

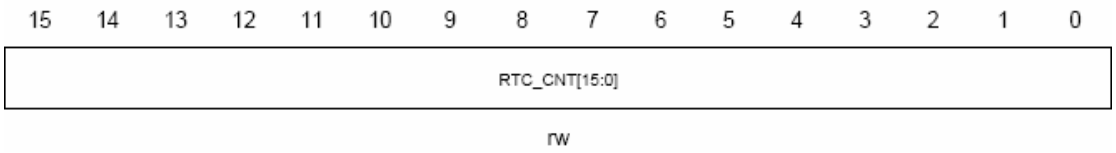


RTC 核有一个 32 位的可编程计数器，它通过两个 16 位的寄存器进行访问；计数器的速率是基于由预分频器产生的 TR_Clock 参考时钟。RTC_CNT 寄存器保存了这个计数器的计数值。它们由 RTC_CR 寄存器中的 RTOFF 位提供写保护，写入操作只有在 RTOFF 位的值为“1”时才允许。对高位（RTC_CNTH）或低位（RTC_CNTL）寄存器中的写操作直接加载相应的可编程计数器，并且重装入 RTC 预分频器。当被读取时，计数器的当前值（系统日期时间）被返回。当外部时钟振荡器工作时，计数器会保持运行状态，即使主系统的电源关闭（待机状态）时也如此。

Bit15:0 = RTC_CNT[31:16]：RTC 计数器高位部分
读 RTC_CNTH 寄存器时，RTC 计数寄存器高位部分的当前值被返回。要写入这个寄存器时，需要使用 RTC_CR 寄存器的 RTOFF 位来进入配置模式。

6.4.8 RTC 计数寄存器低位（RTC_CNTL）

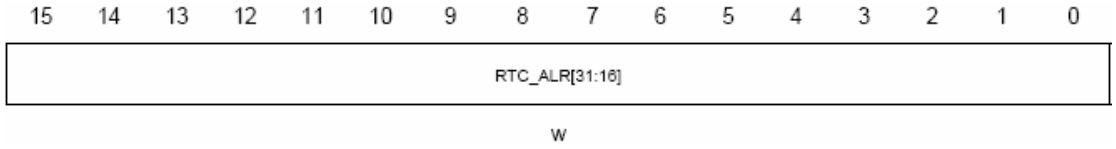
地址偏移量：1Ch
复位值：0000h



Bit15:0 = RTC_CNT[15:0]：RTC 计数器低位部分
读 RTC_CNTL 寄存器时，RTC 计数寄存器低位部分的当前值被返回。要写入这个寄存器时，需要使用 RTC_CR 寄存器的 RTOFF 位来进入配置模式。

6.4.9 RTC 告警寄存器高位（RTC_ALRH）

地址偏移量：20h
只能写入（参见 117 页的“配置模式”）
复位值：FFFFh

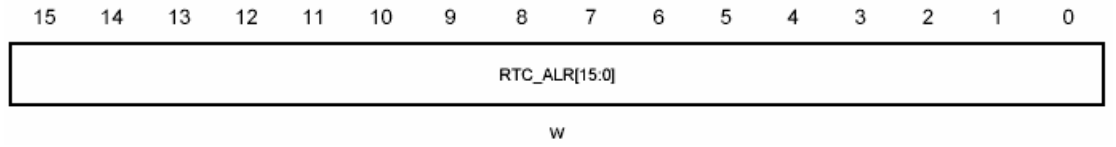


当可编程计数器计数到存储在 RTC_ALR 寄存器中的 32 位值时，会触发一个告警，产生 RTC_alarmIT 中断请求。本寄存器由 RTC_CR 寄存器的 RTOFF 位提供写保护，当 RTOFF 位的值为“1”时，写操作被允许。

Bit15:0 = RTC_ALR[31:16]：RTC 告警高位部分
告警时间的高位部分通过软件写入这个寄存器。要写入这个寄存器时，需要使用 RTC_CR 寄存器的 RTOFF 位来进入配置模式。

6.4.10 RTC 告警寄存器低位（RTC_ALRL）

地址偏移量：24h
复位值：FFFFh



Bit15:0=RTC_ALR[15:0]：RTC 告警低位部分
告警时间的低位部分通过软件写入这个寄存器。要写入这个寄存器时，需要使用 RTC_CR 寄存器的 RTOFF 位来进入配置模式。

6.5 RTC 寄存器影射

RTC 的寄存器映射为 16 位的可寻址寄存器，如下表所描述：

表 28 RTC 寄存器映射

Address Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RTC_CRH	---												GEN	OWEN	AEN	SEN
4	RTC_CRL	---										RTOFF	CNF	GIR	OWIR	AIR	SIR
8	RTC_PRLH	---												PRSL			
Ch	RTC_PRLL	PRSL															
10h	RTC_DIVH	---												DIV			
14h	RTC_DIVL	DIV															
18h	RTC_CNTH	CNTH															
1Ch	RTC_CNTL	CNTL															
20h	RTC_ALRH	ALARMH															
24h	RTC_ALRL	ALARML															

基地址请参考 18 页表 3 “APB2 存储器映射”。

7 看门狗定时器（WDG）

7.1 介绍

看门狗定时器外设可用来作为自由运行定时器，或作为看门狗用于解决由于硬件或软件失效造成的处理器故障。

7.2 主要特性

- 十六位减法计数器
- 八位时钟预分频器
- 可靠的再装入序列
- 自由运行定时器模式
- 计数结束产生中断

7.3 功能描述

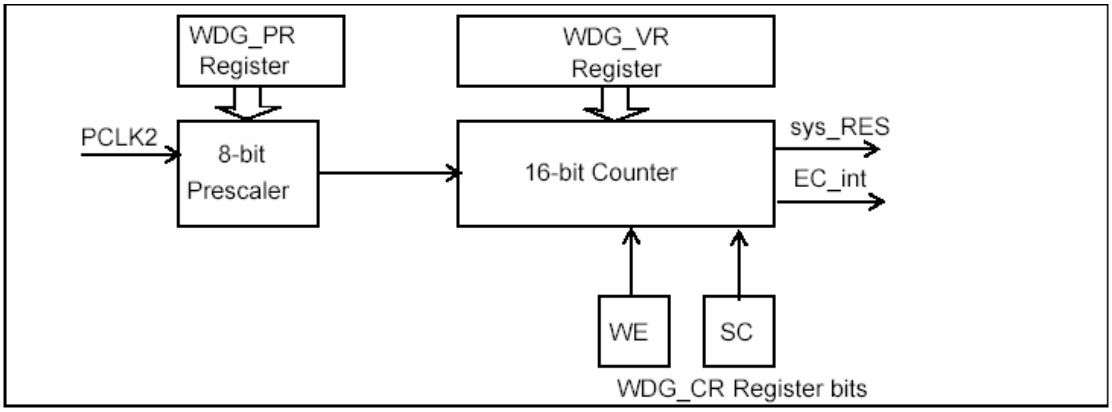
图30显示了看门狗定时器模块的功能模块。这个模块可以用作看门狗或是自由运行定时器。在这两种工作模式下，都可以通过对WDG_CNT寄存器的读操作来获得16位计数器的值。

7.3.1 自由运行定时器模式

如果WDG_CR寄存器的WE位没有被软件置为“1”，那么设备将进入自由运行定时器模式。

在这种运行模式下，当WDG_CR寄存器的SC位被置为“1”时， WDG_VR的值被装入计数器，计数器开始向下计数。

图 30　看门狗定时器功能模块



当计数值到达计数终值（0000h）时，将产生一个中止计数中断（EC_int）， WDG_VR的值将被重新装入。只有SC位被清零，计数器才能停止运行。

7.3.2 看门狗模式

如果 WDG_CR 寄存器的 WE 位被软件置为“1”，那么该外设将进入看门狗模式。软件不能改变这种运行模式（SC 位无效，WE 位不能被清零）。

当外设进入这种运行模式下， WDG_VR的值被装入计数器，计数器开始向下计数。当计数

值到达计数终端（000h）时，将产生一个系统复位信号（(sys_RES)。

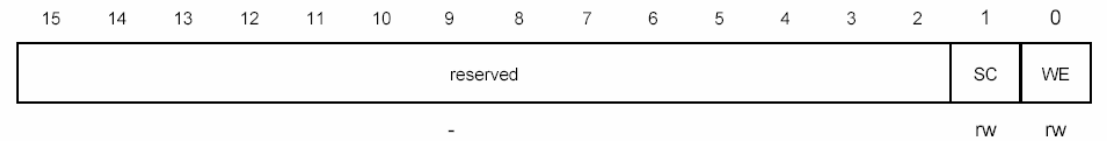
如果两个值（0xA55A和0x5AA5）连续按顺序被写入WDG_KR寄存器（见7.4），WDG_VR的值被重新装入计数器，因此避免了计数终值端的出现。

7.4 寄存器描述

看门狗定时器的寄存器不能按字节访问。
保留位总是被读为“0”，不能被改写。

7.4.1 看门狗控制寄存器（WDG_CR）

地址偏移量：00h
复位值：0000h



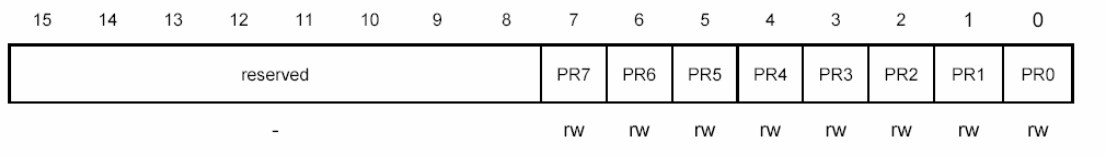
位 15：2=保留，必须被清零

位 1=SC：开始计数位
1：计数器装入定时器预先设置的值并开始计数。
0：计数器被停止。
这些功能只在定时器模式下有效（WE =0）。

位 0=WE：看门狗使能位
1：看门狗模式被使能
0：定时器模式被使能
这个位不能通过软件清除。
当 WE 位为 1 时，SC 位无效

7.4.2 看门狗预分频器寄存器（WDG_PR）

地址偏移量：04h
复位值：00FFh



位 15：8 = 保留
位7：0= **PR[7:0]**:预分频值
定时器计数时钟被PR[7:0]+1倍分频。
当看门狗模式被使能（WE位为1），或重装次序出现，或定时器模式下计数器开始位（SC）被置为1时，此预分频值有效。

7.4.3 看门狗预加载值寄存器(WDG_VR)

地址偏移量：08h
复位值：FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TV15	TV14	TV13	TV12	TV11	TV10	TV9	TV8	TV7	TV6	TV5	TV4	TV3	TV2	TV1	TV0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15：0= **TV[15:0]**:定时器预加载值。
当定时器计数器开始计数，或是重加载序列出现，或到达计数终值时，此数值被装入定时器计数器。到达计数终止的时间（us）由下式给出：

$$(PR[7:0]+1)*(TV[15:0]+1)*t_{PCLK2}/1000\ (\mu s)$$

其中t_{PCLK2}是以ns为单位的时钟周期。
例如，如果PCLK2=20MHZ，系统复位后的缺省超时设定为256*65535*50/1000=838800us。

7.4.4 看门狗计数器寄存器（WDG_CNT）

地址偏移量：0ch
复位值：FFFFh
位15：0= **CNT[15:0]:** 定时器计数值
16位计数器的当前计数值可通过对这个寄存器的读操作得到。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

7.4.5 看门狗状态寄存器(WDG_SR)

地址偏移量：10h
复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															EC
-															r-c

位15:1 =保留
位0=EC：计数终止等待位
1：计数终止发生
0：计数终止没有发生
在看门狗模式下（WE=1），此位无效。
此位只能通过硬件置位，且必须由软件复位。

7.4.6 看门狗屏蔽寄存器（WDG_MR）

地址偏移量：14h
复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															ECM
-															rw

位15:1 = 保留

位0=ECM：计数终止屏蔽位

1：计数终止中断请求被允许

0：计数终止中断请求被禁止

7.4.7 看门狗键值寄存器（WDG_KR）

地址偏移量：18h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
K15	K14	K13	K12	K11	K10	K9	K8	K7	K6	K5	K4	K3	K2	K1	K0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0 = **K[15:0]:**键值

在看门狗模式被使能时，顺序写入本寄存器两个值（见器件说明），计数器被初始化为TV[15:0]值，WTDPR寄存器中的预分频器值生效。在这两个值的写入操作之间可以执行任意数量的指令。

如果看门狗模式被禁止（WE=0），对此寄存器的写入无效。

从本寄存器读到的值为0000h。

7.5 WDG寄存器映射

表29. 看门狗定时器寄存器映射

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**WDG_CR**	reserved														SC	WE
4	**WDG_PR**	reserved								PR(7:0)							
8	**WDG_VR**	TV(15:0)															
C	**WDG_CNT**	TV(15:0)															
10	**WDG_SR**	reserved															EC
14	**WDG_MR**	reserved															MEC
18	**WDG_KR**	K[15:0]															

有关基地址，请见18页的表3“APB2存储器映射”。

8 定时器

8.1简介

一个定时器是由一个可编程预分频器所驱动的十六位计数器构成的。

定时器有多种用途，包括多至两个输入信号的脉冲宽度测量（输入捕获），或多达两路输出波形的生成（输出比较和脉宽调制）。

利用定时器预分频器，可以在非常宽的范围内对脉冲宽度和波形周期进行调制。

8.2 主要特性

- 可编程预分频器： f_{PCLK2} 可被分频 1 到 256 倍，即：预分频寄存器（0-255）值+1。
- 溢出状态标志和可屏蔽中断。
- 可选择有效边沿的外部时钟输入（必须比 f_{PCLK2} 时钟速度慢四倍以上）。
- 输出比较功能，具有
  - 2 个专用 16 位寄存器
  - 2 个专用可编程信号
  - 2 个专用状态标志
  - 2 个专用中断标志
- 输入捕获功能，具有
  - 2 个专用 16 位寄存器
  - 2 个专用有效沿选择信号
  - 2 个专用状态标志
  - 2 个专用中断标志
- 脉宽调制模式（PWM）
- 单脉冲模式（OPM）
- 脉宽调制输入模式
- 定时器全局中断（5 个内部“或”运算的或分离的中断源，取决于器件）
  - ICIA：定时器输入捕获 A 中断
  - ICIB：定时器输入捕获 B 中断
  - OCIA：定时器输出比较 A 中断
  - OCIB：定时器输出比较 B 中断
  - TOL：定时器溢出中断

方框图如图 31 所示。

8.3 特殊功能

定时器 0：

- T0.EXTCLK 输入是通过预分频器直接连到 CK 时钟引脚上，该预分频器将输入频率进行 8 分频。
- T0.ICAP_B 连接到实时钟警报信号（RTC ALARM）上；这使得定时器 T0 可与实时钟同步。

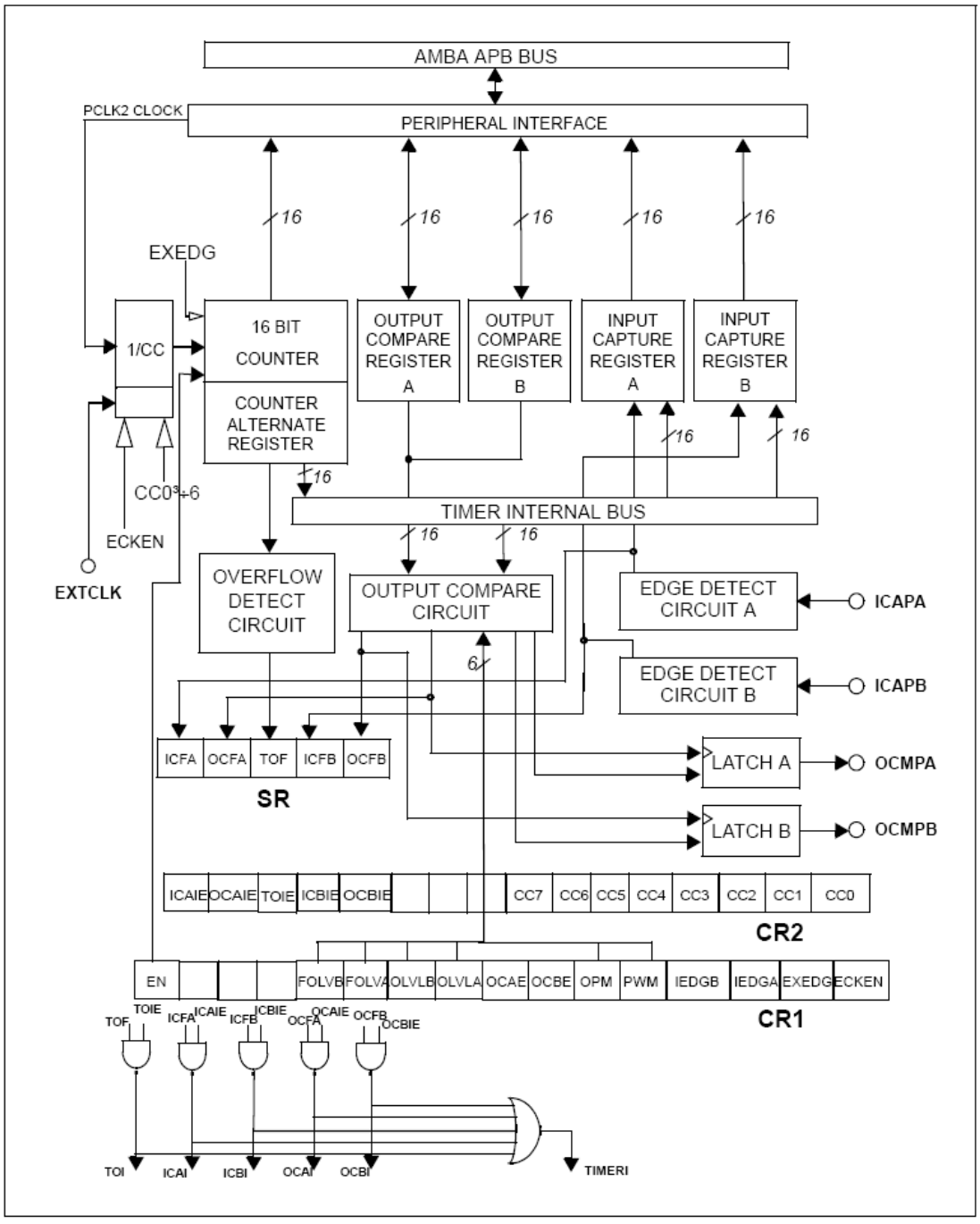
定时器 2：

- T2.ICAP_B（输入捕获 B）连接到来自 HDLC 模块的 HDLC_HRMC（HDLC 接收消息完成）信

号上，该信号是一个正确数据帧完成后产生的中断请求。

- T2.OCMP_B（输出比较 B）被连接到 HDLC_HTEN（传输使能端）。T2.OCMP_B 的上升沿可由硬件允许触发传输的开始，使得可对该事件精确定时，或（与上述 ICAP_B 结合起来）实现在接收一个输入帧与开始回应之间的精确延时。

图31 定时器方框图



8.4 功能描述

8.4.1 计数器

可编程定时器的主要模块是一个十六位计数器和与其相关联的多个十六位寄存器。

对计数寄存器（CNTR）的写入操作将重置计数器为FFFCh值。

定时器的时钟源既可以来自内部又可来自外部，取决于CR1寄存器的ECKREN位的选择。当ECKEN=0时，频率取决于CR2寄存器的预分频器的分频位（CC7-CC0）。

当计数器从FFFFh翻转至0000h时发生上溢，此时SR寄存器的TOF位被置位。如果CR2寄存器的TOIE位被置位，就会产生一个中断；若此条件不满足，则中断仍保持在等待发布状态，直到条件达到。

当TOF字节被置位时，可以通过对SR寄存器的一次写操作来清除溢出中断请求，写操作中，数据总线13-位为0，而所有其它位写为“1”（SR寄存器只能被清除，所以在一个位上写一个“1”没有作用：这使得在清除一个等待位时，不存在错误地清除来自其它源的新的中断请求的风险）。

8.4.2 外部时钟

在CR1寄存器的ECKEN=1时选择外部时钟。

EXEDG位的状态决定了将触发计数的外部时钟引脚EXTCLK上的电平变化的类型。

计数器与来自PCLK2模块的内部时钟的上升沿同步。

在外部时钟的两个连续有效沿之间，至少应该发生4个PCLK2时钟的上升沿；因此外部时钟频率必须低于PCLK2时钟频率的四分之一。

注意：外部时钟不可用于定时器2。

8.4.3 内部时钟

图 32. 计数器时序图，内部时钟 2 分频

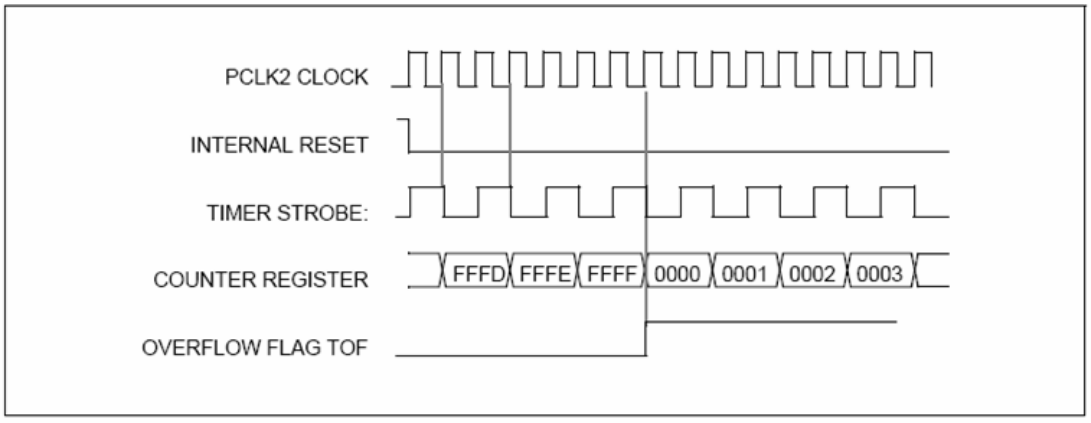


图 33. 计数器时序图，内部时钟 4 分频

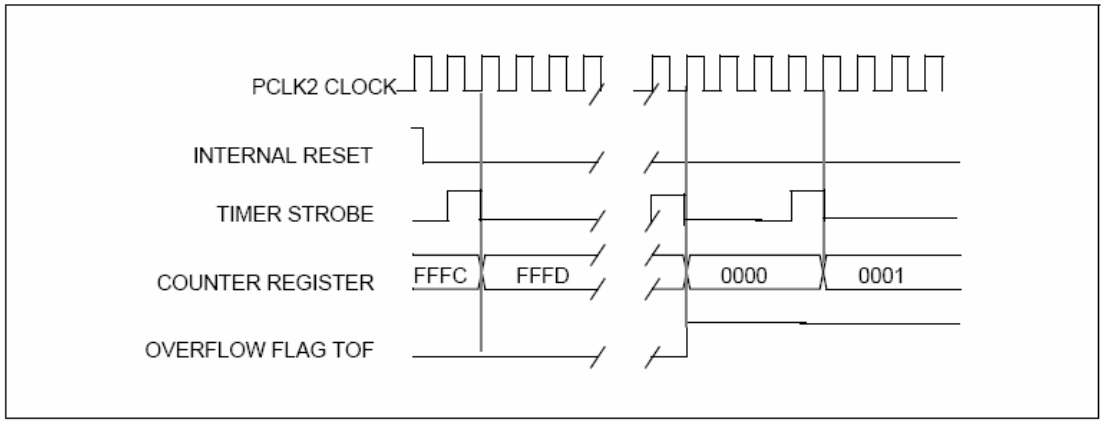
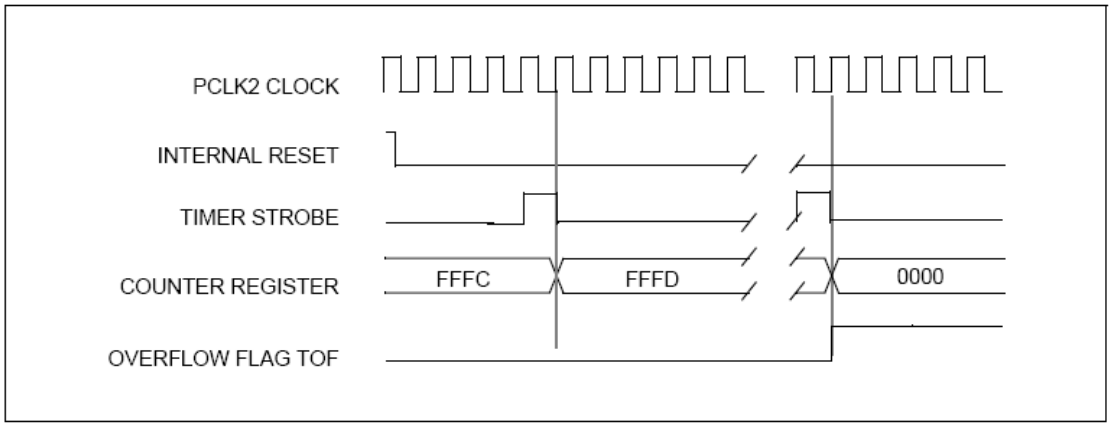


图 34 计数器时序图，内部时钟 n 分频



8.4.4 输入捕获

在这部分，标记 “*i*”，可代表 A 或 B。

当 ICAP*i* 引脚检测到电平变化之后，这两个输入捕获 16 位寄存器（ICAR 和 ICBR）用来锁存计数器的值（如图 35）。

IC*i*R 寄存器是只读寄存器。

有效电平变化可以通过控制寄存器（CR1）的 IEDG*i* 位进行软件编程。

定时分辨率是计数器的一次计数：（$f_{PCLK2/(CC7 \div CC0+1)}$）。

8.4.4.1 步骤

为了使用输入捕获功能，在 CPR1 和 CR2 寄存器中做如下选择：

- 选择定时器时钟源（ECKEN）。
- 在选用了内部时钟的情况下，选择定时器时钟预分频因子（CC7÷CC0）。
- 如果 ICAPA 有效，用 IEDGA 位选择 ICAPA 引脚上的有效变化沿。
- 如果 ICAPB 有效，用 IEDGB 位选择 ICAPB 引脚上的有效变化沿。
- 当 ICAPA（或 ICAPB）有效时，选择 ICAIE（或 ICBIE），以便在一个输入捕获之后产生中断。

当一个输入捕获发生时

- ICF*i* 位被设定。
- IC*i*R 寄存器包含 ICAP*i* 管脚在有效电平变化时计数器的值（见图 36）。
- 如果 ICAIE 被设定（如果只有 ICAPA 有效）或如果 ICBIE 被设定（如果只有 ICAPB 有效），则会产生一个定时器中断；否则，中断保持在等待状态直到有关的能使位被设定。

用下列操作可清除输入捕获中断请求：

1. 当 ICiF 为被清除时，对 SR 寄存器的一次写入操作，对 ICAPA 将位 15 写‘0’，对 ICAPB 将位 12 写‘0’。

图 35 输入捕获框图

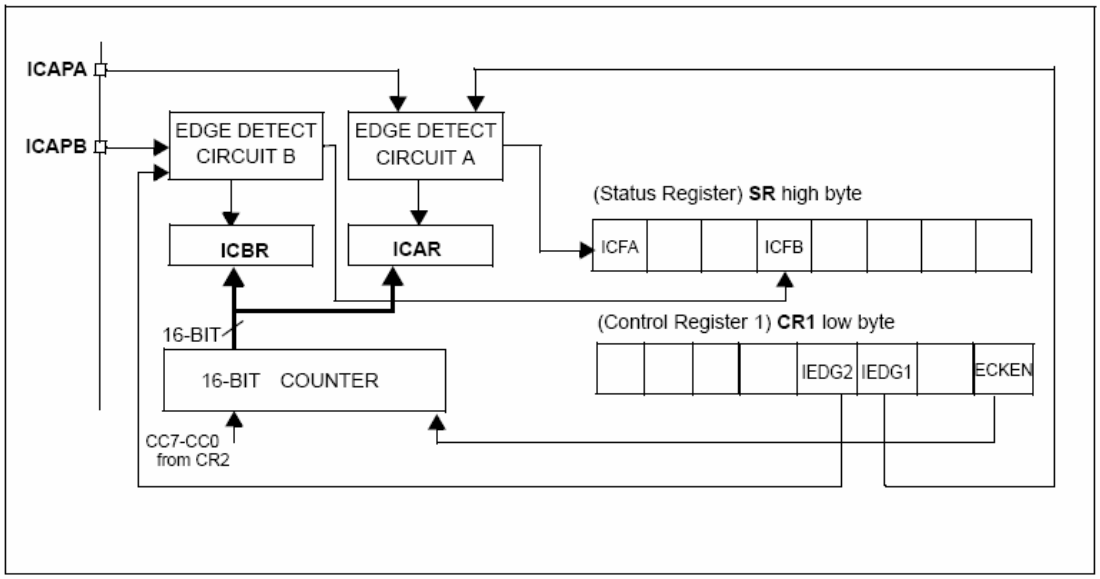
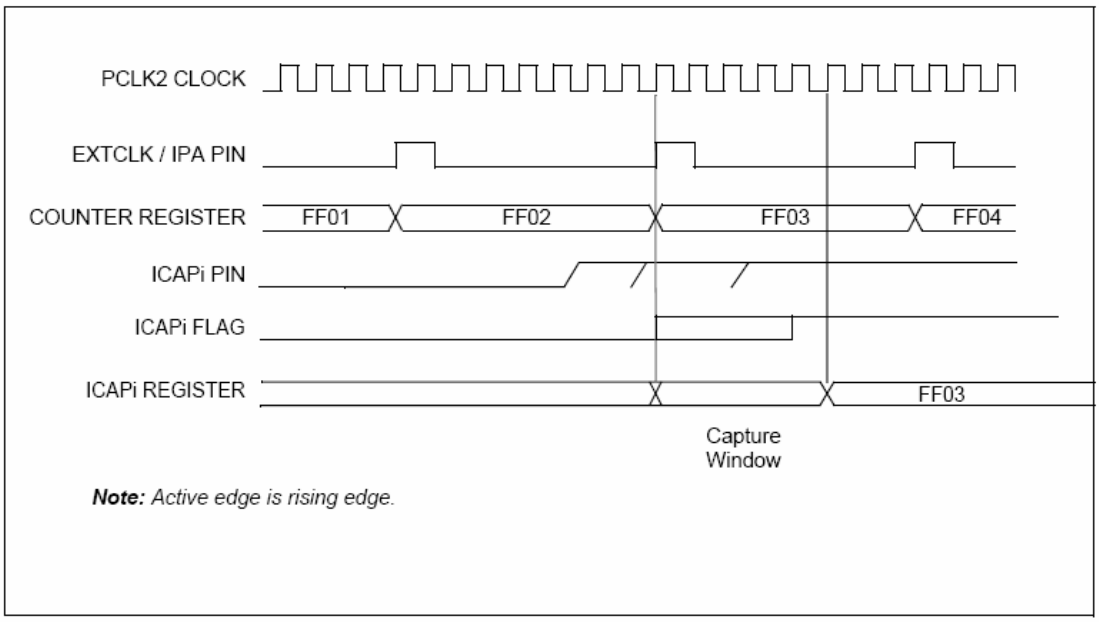


图 36 输入捕获时序图



8.4.5 输出比较

在这一节里，标记“*i*”，可代表 A 或 B。

该功能可以用来控制一个输出波形，或用来指出何时一段时间结束。

当在输出比较寄存器和计数器之间达到匹配时，输出比较功能将会：

- 若 OCiE 位被设定时，赋予引脚一个可编程电平值。
- 在状态寄存器中设置标志。
- 如果被允许则产生一个中断。

两个 16 位寄存器——输出比较寄存器 A（OCAR）和输出比较寄存器 B（OCBA）——保存了要在每个时钟周期与计数器进行比较的数值。

这些寄存器是可读和可写的，且不受定时器硬件的影响。一个复位事件将把 OCiR 值改变为 8000h。

定时分辨率是计数器的一次计数：（$f_{PCLK2/(CC7 \div CC0+1)}$）。

8.4.5.1 步骤

要使用输出比较功能，在 CR1/CR2 寄存器中做如下选择：

- 如果需要一个输出信号，就置位 OCiE，此时 OCMPi 管脚专用于输出比较 i 功能。
- 选择定时器时钟（ECKGEN）和预分频器分频因子（CC7 ÷ CC0）。

在 CR1/CR2 寄存器中选择如下：

- 选定在匹配发生后要应用于 OCMPi 管脚的 OLVLi 位。
- 如比较 A（比较 B）需要产生一个中断，则设置 OCAIE（OCBIE）。

当检测到匹配后：

- OCFi 位被置位。
- OCMPi 引脚获得 OLVLi 位的值（OCMPi 引脚锁存器在复位期间被强制为低电平，并且一直保持低电平，直到有效的比较将其改为 OLVLi 电平）。
- 如果在 CR2 寄存器中 OCAIE（或 OCBIE）位被置位，OCAR（或 OCBR）匹配了定时器计数器（即 OCFA 或 OCFB 被置位）时，则产生一个定时器中断。

当 ICiF 为被清除时，通过向 SR 寄存器的写入操作可清除输出比较中断请求，对于 OCAR 将位 14 设为‘0’，对 OCBR 将位 12 设为‘0’。

如果 OCiE 位未被置位，达到匹配时 OCMPi 引脚为‘0’且 OLVLi 位不显现。

在每次成功比较后，为了控制输出波形或建立新的超时时间，16 位 OCiR 寄存器的值和 OLVLi 位应予改变。

对于一个特定的定时应用，所需的 OCiR 寄存器值可用下列公式计算：

$$\Delta OCiR = \frac{\Delta t * f_{PCLK2}}{(CC7 \div CC0 + 1)}$$

其中：Δt = 希望的输出比较时间长度（按秒计）

f_{PCLK2} = 内部时钟频率

CC7 ÷ CC0 = 定时器时钟预分频器

图 37 输出比较模块示意图

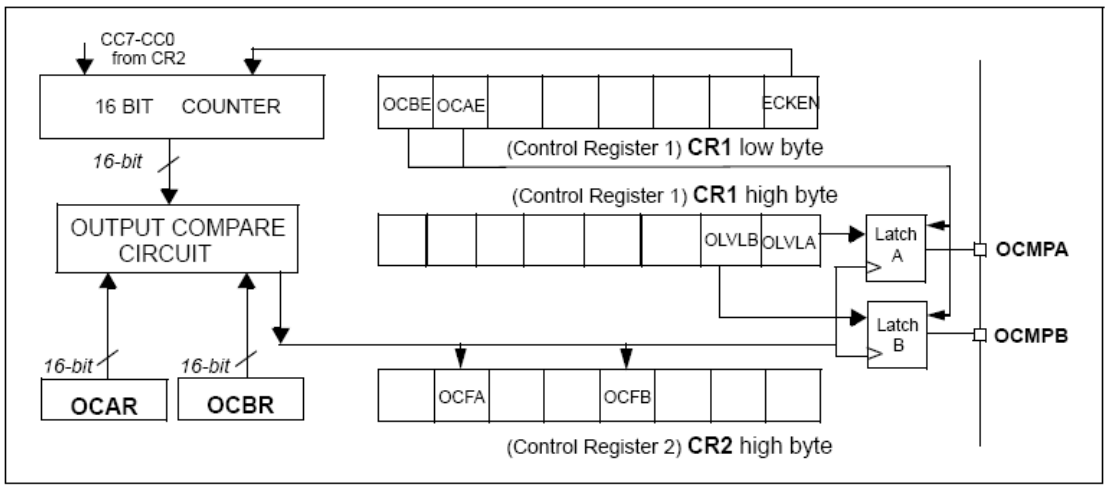
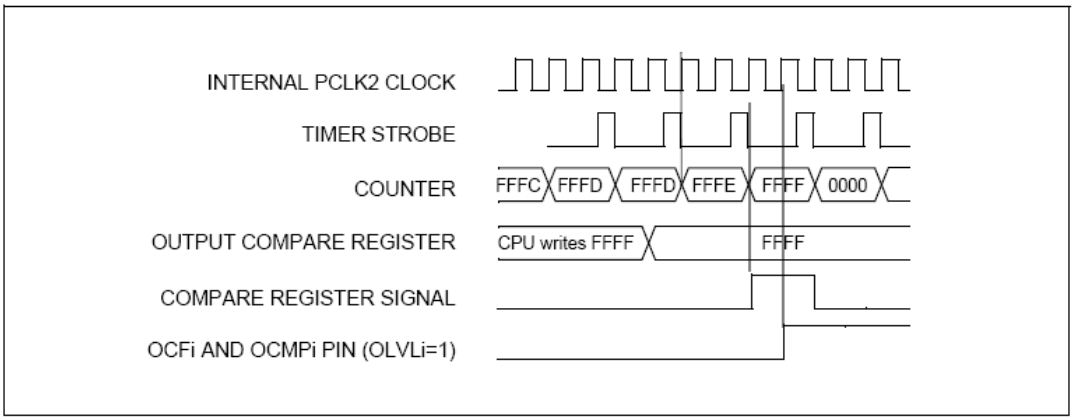


图 38 输出比较时序图，内部时钟 2 分频



8.4.6 强迫比较模式

在这部分，标号“i”可表示 A 或 B。

CR1 寄存器的比特比 11：8 和 CR2 的比特比 7:0 被采用（有关详细的寄存器描述请参见 8.6 节）。

当 FOLVA 位被置位时，如果 PWM 和 OPM 都被清除，则 OLVLA 位被复制到 OCMPA 引脚。当 FOLVB 位被置位时，OLVLB 位被复制到 OCMPB 引脚。

当 OCMPi 管脚被激活（OCiE 位=1）时，为了翻转其电平，OLVLi 位必须被翻转。

注意：

- 当 FOLVi 被置位时，不产生中断请求。
- 然而如果 OCiR=计数器值，OCFi 位可被设定，如果被使能了将产生一个中断。
- 在强迫比较模式下输入捕获功能生效。

8.4.7 单脉冲模式

当外部事件产生时，单脉冲模式能产生一个脉冲。这种模式通过 CR1 寄存器的 OPM 位来选择。

单脉冲模式使用输入捕获 A 功能（触发事件）和输出比较 A 功能。

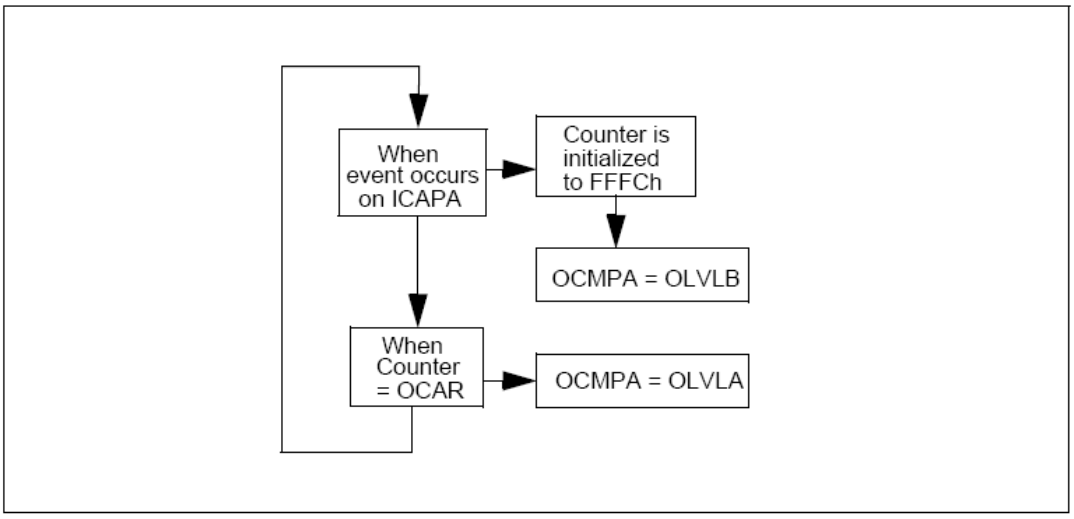
8.4.7.1 步骤

要使用单脉冲模式，在 CR1 寄存器中选择如下：

- 用 OLVLA 位选择在脉冲之后作用到 OCMPA 引脚的电平。
- 用 OLVLB 位选择在脉冲期间作用到 OCMPA 引脚的电平。
- 用 IEDGA 位选择 ICAPA 引脚的有效变化沿。
- 置位 OCAE 位，让 OCMPA 管脚专用于输出比较 A 功能。
- 置位 OPM 位。
- 选择定时器时钟（ECKGEN）和预分频器分频因子（CC7-CC0）。

将对应于脉冲宽度的数值（参考 8.4.8.1 小节的公式）加载到 OCAR 寄存器。

图 39 单脉冲模式循环

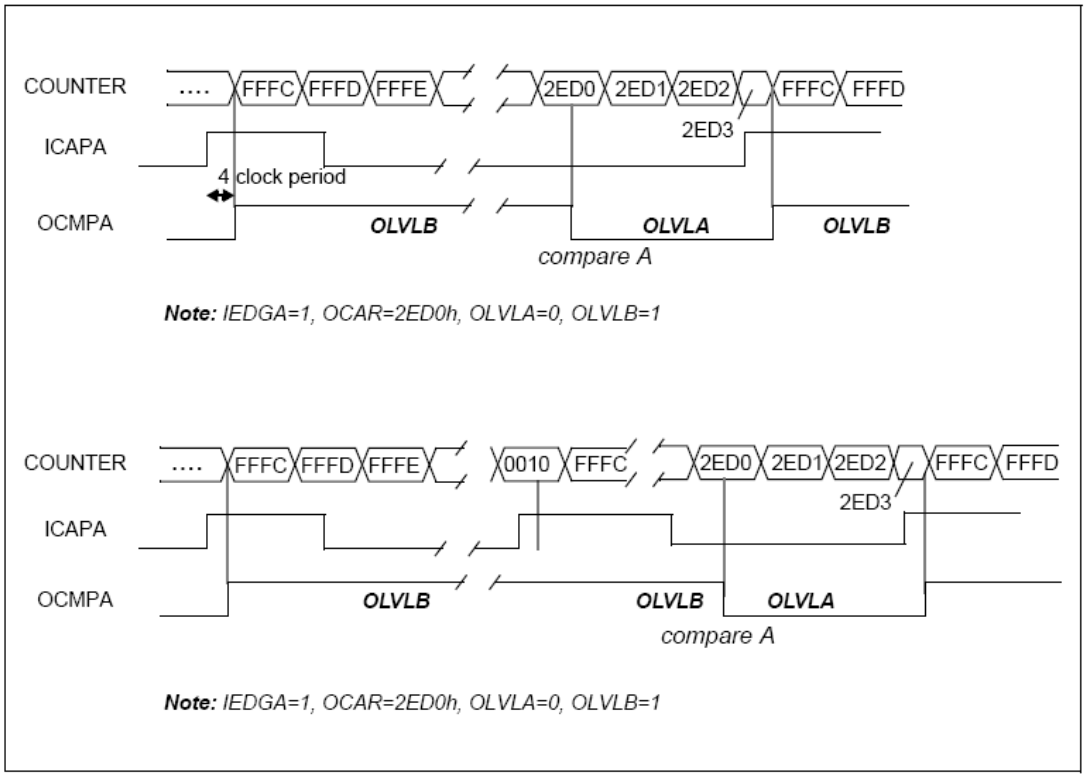


而后， 当 ICAPA 管脚的一个有效事件发生时，计数器初始化为 FFFCH 且 OLVLB 位在四个时钟周期之后被加载到 OCMPA 引脚。当计数器的值等于 OCAR 寄存器包含的值时，OLVLA 位在 OCMPA 管脚输出（参见图 40）。

注意：

- OCFA 位在单脉冲模式下不能被硬件置位，但是 OCFB 位可以产生一个输出比较中断。
- 当有效沿发生时，ICFA 位被置位，且若 ICAIE 位为 1 时会产生一个中断。ICAR 寄存器值将为 FFFCh。
- 当脉宽调制（PWM）和单脉冲模式（OPM）位都置位时，若 FOLVA=1，单脉冲模式是唯一生效的一个；否则 PWM 为唯一生效的模式。
- 在 OPM 模式下强迫比较 B 模式生效。
- 在 OPM 模式下输入捕获 B 功能生效。
- 在 OPM 模式下当 OCAR=FFFBh 时，产生宽度为 FFFFh 的脉冲。
- 在计数器达到 OCAR 值之前如果在 ICAPA 上事件再次发生，则计数器将再次被复位，且产生的脉冲比预计的要长，如图 40 所示。
- 如果在计数器达到 OCAR 值之前对计数器寄存器执行写操作，则计数器将会再次被复位，且产生的脉冲可能比预想的长。
- 在计数器达到 OCAR 值之后如果在计数寄存器上执行一个写入操作，对波形不会产生影响。

图 40 单脉冲模式时序



8.4.8 脉冲宽度调制模式

脉冲宽度调制模式能够产生一个信号，其频率和脉冲宽度由 OCAR 和 OCBR 寄存器的值确定。
脉冲宽度调制模式使用了完整的输出比较 A 功能和 OCBR 寄存器。

8.4.8.1 步骤

要使用脉冲宽度调置模式，在 CR1 寄存器中做如下选择：

- 使用 OLVLA 位选定在与 OCAR 寄存器成功比较之后 OCMPA 引脚上的电平。
- 使用 OLVLB 位选定在与 OCBR 寄存器成功比较之后 OCMPA 引脚上的电平。
- 置位 OCAE 位：让 OCMPA 引脚专用于输出比较 A 功能。
- 置位 PWM 位。
- 选定定时器时钟（ECKGEN）和预分频器分频因子（CC7-CC0）。

根据信号周期值加载 OCBR 寄存器。
如果（OLVLA=0 且 OLVLB=1）用脉冲宽度值加载 OCAR 寄存器。
如果 OLVLA=1 且 OLVLB=0 ，则脉冲宽度是 OCBR 和 OCAR 寄存器值之间的差。
对于特定定时应用，所需 OCiR 寄存器值可用以下公式计算：

$$OCiR \quad Value = \frac{t * f_{PCLK2}}{t_{PRESC}} - 5$$

其中：

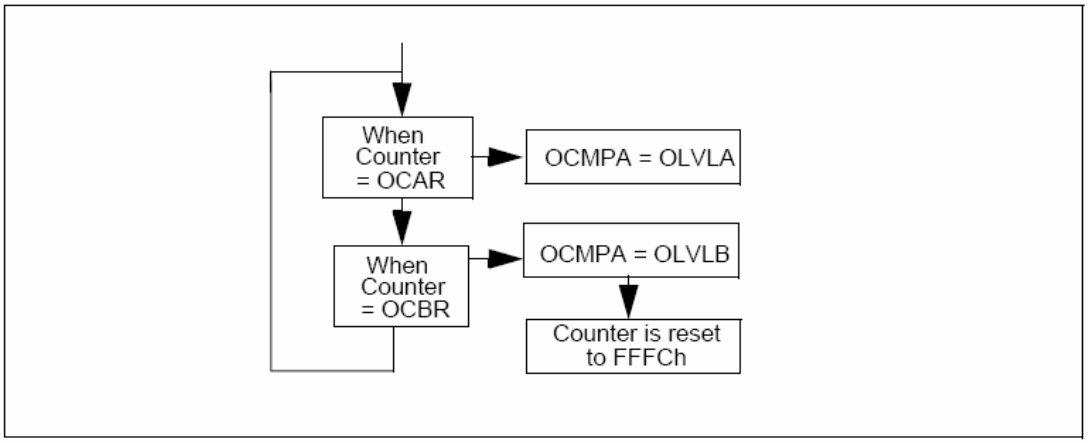
t = 希望的输出比较周期（秒）

f_{PCLK2} = 内部时钟频率（赫兹）

t_{PRESC} = 定时时钟前置分频（1，2…,256）

输出比较 B 事件使计数器初始化为 FFFCh（如图 42）。

图 41 脉冲宽度调制模式循环



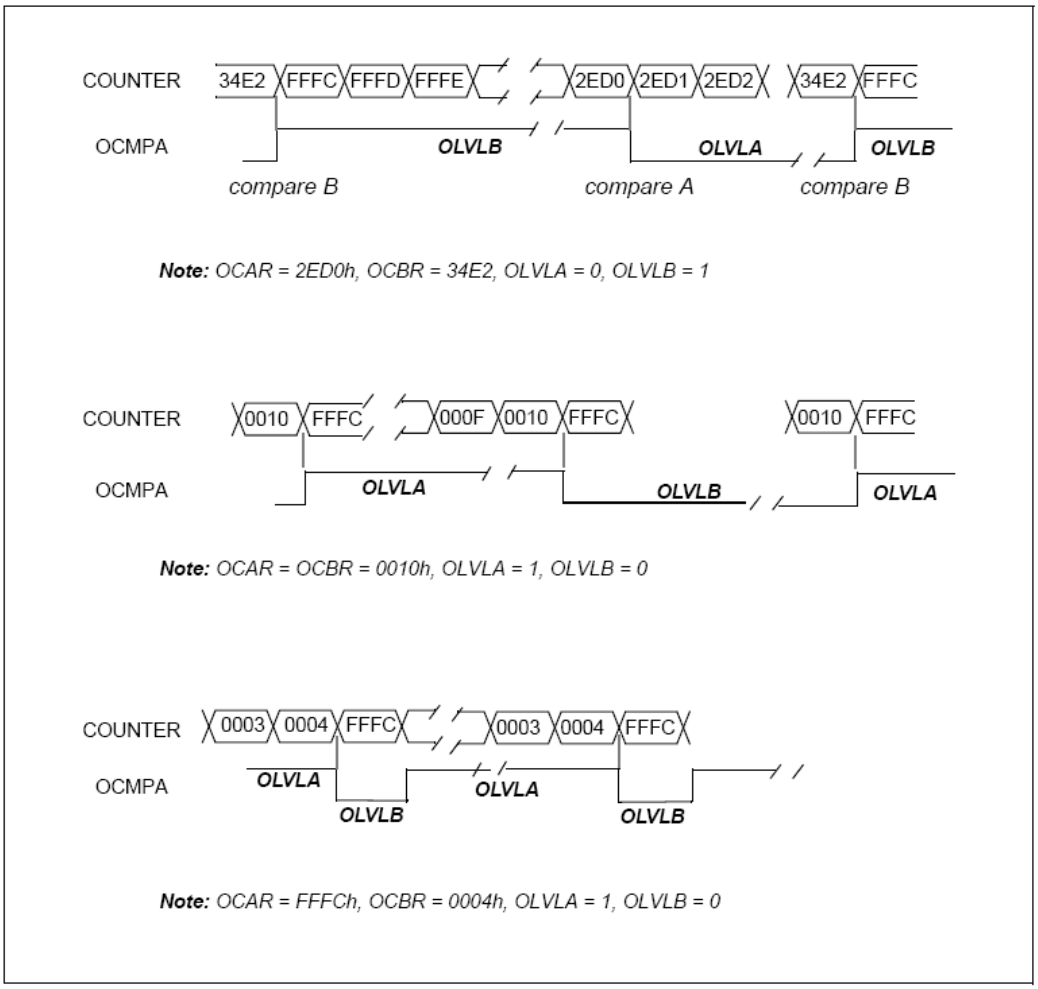
注意：

- 在 PWM 模式下 OCFA 为不能被硬件置位，但是每当计数器与 OCBR 匹配时 OCFB 被置位。
- 在 PWM 模式下输入捕获功能可用。
- 当计数器=OCBR 时 OCFB 位被置位。当 OCBIE 被置位时，这会产生一个中断。这个中断可以为应用程序交互地改变脉冲宽度或周期提供帮助。
- 当脉冲宽度调制（PWM）和单脉冲模式（OPM）位均被置位时，若 FOLVE=0，PWM 是唯一起作用的模式；否则 OPM 为唯一起作用的模式。
- 为了产生有效的波形，加载在 OCBR 上的值必须比在 OCAR 上的大。注意到 0000h 被认为比 FFFCh 或 FFFDh 或 FFFEh 或 FFFFh 要大。
- 当 OCAR >OCBR 时，将不会产生波形。
- 当 OCBR = OCAR 时，将会产生 50%占空比的方波，如图 42。
- 当 OCBR 和 OCAR 加载为 FFFCh（计数器复位值）时，将产生一个方波而计数器将保持 FFFCh 不变。周期将用以下公式计算：

$$period = t_{APB2} \times (PRESC + 1) \times (OCBR + 1)$$

- 当 OCAR 加载为 FFFCh（计数器复位值）时，将会产生如图 42 的波形。
- 当 FOLVA 位和 PWM 位都被置位时，PWM 模式有效。但是如果 FOLVB 位被置位则 OLVLB 位将呈现在 OCMPB 上（当 OCBE bit=1）。
- PWM 模式下在 CNTR 寄存器上执行一个写入操作时，计数器将被复位，脉冲宽度/周期将不会为预期的值。

图 42 脉冲宽度调制模式时序



8.4.9 脉冲宽度调制输入

PWM 输入功能可让我们测量一个外部波形的周期和脉冲宽度。初始沿是可编程的。

它使用两个输入捕获寄存器和输入捕获 A 模块的输入信号。

8.4.9.1 步骤

CR2 寄存器必须被编程为可以产生中断。为了使用脉冲宽度调制模式，在 CR1 寄存器中选择如下：

- 置位 PWM1 位
- 在 IEDGA 中选择第一个沿
- 选择 IEDGB 中第二沿为 IEDGA 的反极性
- 按需要编程时钟源和预分频器
- 设定 EN 位使计数器工作

为了得到一致的测量结果，中断必须连接到输入捕获 A 中断，从 ICAR 读出周期值，从 ICBR 读出脉冲宽度。

时间值按如下方法得到：

$$Period = \frac{t_{PRESC} * ICAR}{f_{PCLK2}}$$

$$Pulse = \frac{t_{PRESC} * ICBR}{f_{PCLK2}}$$

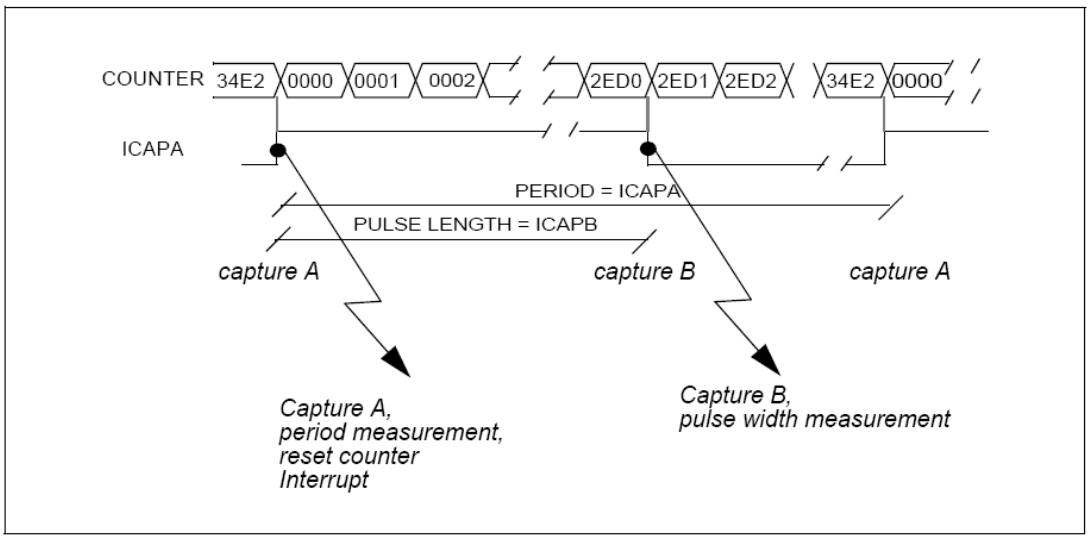
其中

fPCLK2 = 内部时钟频率

tPRESC= 定时器时钟前置分频

输入捕获 A 事件引发计数器初始化为 0000h，允许一个新的测量启动。在 ICAPA 上的第一个输入捕获不会产生相应的中断请求。

图43 脉宽调制输入模式时序



8.5 中断管理

这三种中断源可以映射在五种不同中断通道，也可以映射在相同通道中。

8.5.0.1 中断通道的使用

为了使用中断功能，对每一个被使用的中断通道，按如下顺序完成：

- 置位CR2寄存器的OC*i*IE和（或）IC*i*IE和（或）TOIE位，使能外围设备对希望的事件执行中断请求。

在一个系统中的定时器外围电路被实例化时，通过把预期的一个或多个中断线连接到中断控制器上，就实现了对五个或单一中断通道的选择。

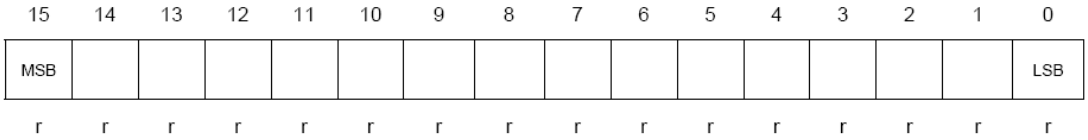
8.6 寄存器描述

每个定时器都有两个控制寄存器，一个状态寄存器，与两个输入捕获、两个输出比较和计数器对应的六对数据寄存器（16位），每个寄存器都仅有一个16位的读写口，这就是说不能读或写入一个字节。

8.6.1 输入捕获A寄存器（TIMn_ICAR）

地址偏移量：00h

复位值：xxxxh

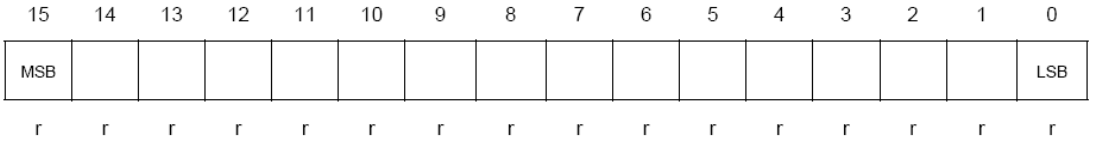


这是一个16位只读寄存器，包含了由输入捕获A传递的计数值。

8.6.2 输入捕获B寄存器（TIMn_ICBR）

地址偏移量：04h

复位值：xxxxh

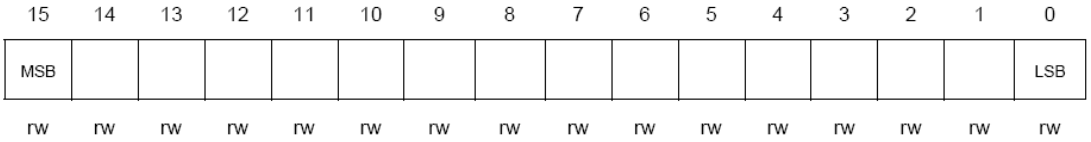


这是一个16位只读寄存器，包含了由输入捕获B传递的计数器值。

8.6.3 输出比较A寄存器(TIMn_OCAR)

地址偏移量：08h

复位值：8000h

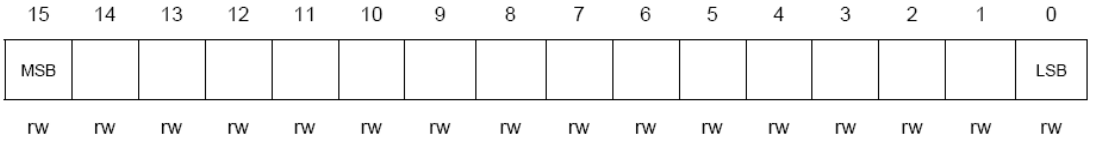


这是一个16位寄存器，包含的值用来与CNTR寄存器比较，在OCMPA输出上表示。

8.6.4 输出比较B寄存器(TIMn_OCBR)

地址偏移量：0Ch

复位值：8000h

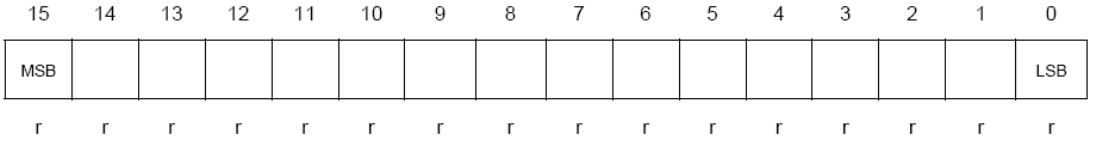


这是一个16位寄存器，包含的值用来与CNTR寄存器比较，在OCMPB输出上表示。

8.6.5 计数器寄存器(TIMn_CNTR)

地址偏移量：10h

复位值：FFFCh



这是一个包含计数器值的16位寄存器。通过写入该寄存器，计数器被复位到FFFCh。

8.6.6 控制寄存器1(TIMn_CR1)

地址偏移量：14h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	PWMI	Reserved		FOLVB	FOLVA	OLVLB	OLVLA	OCBE	OCAE	OPM	PWM	IEDGB	IEDGA	EXEDG	ECKEN
rw	rw	-		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15=**EN**：定时器计数使能

0：定时器计数停止

1：定时器计数开始

位14= **PWMI**:脉宽调制输入

0：PWM输入无效

1：PWM输入有效

位13：12=保留。这些位必须写为0。

位11= **FOLVB**:强制输出比较B

0：无效

1：强制把OLVLB值复制到OCMPB引脚

位10=**FOLVA**: 强制输出比较A

0：无效

1：强制把OLVLA值复制到OCMPA引脚

位9=**OLVLB**:输出电平B

只要和OCBR寄存器完成比较并且CR2寄存器中的OCBE被置位，这位的值就被输出到OCMPB引脚。在单脉冲和脉宽调制模式下这位值被复制到OCMPA引脚。

位8=**OLVLA**:输出电平A

只要和OCAR寄存器完成比较并且CR2寄存器中的OCAE被置位，这位的值就被输出到OCMPA引脚。

位7= **OCBE**:输出比较B使能

0：输出比较B功能启动，但是OCMPB引脚是通用的I/O。

1：输出比较B功能启动，OCMPB引脚专用于执行定时器的输出比较B功能。

位6= **OCAE**:输出比较A使能

0：输出比较A功能启动，但是OCMA引脚是通用的I/O。

1：输出比较A功能启动，OCMPA引脚专用于执行定时器的输出比较A功能。

位5= **OPM**:单脉冲模式

0：单脉冲模式无效

1：单脉冲模式有效，用ICAPA引脚来触发OCMPA引脚上的一个脉冲；有效边沿由IEDGA位给定。产生脉冲的长度取决于OCAR寄存器的内容。

位4=**PWM**：脉宽调制

0：脉宽调制无效

1：脉宽调制有效，OCMPA引脚输出一个可编程周期信号；脉冲宽度取决于OCAR寄存器的值，周期取决于OCBR寄存器的值。

位3 = **IEDGB**:输入沿B

这位决定ICAPB引脚上用哪种电平转换触发输入捕获。

0：下降沿触发输入捕获。

1：上升沿触发输入捕获。

位2= **IEDGA**: 输入沿A

这位决定ICAPA引脚上用哪种电平转换触发输入捕获。

0：下降沿触发输入捕获。

1：上升沿触发输入捕获。

位1= **EXEDG**:外部时钟沿

这位决定外部时钟引脚EXTCLK（或内部信号）的哪种电平转换将用来触发计数器。

0：下降沿触发计数器。

1：上升沿触发计数器。

位0=**ECKEN**：外部时钟使能

0：内部时钟，经预分频器分频后提供定时器时钟。

1：外部时钟源作为定时器时钟。

8.6.7 控制寄存器2(TIMn_CR2)

地址偏移量：18h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICAIE	OCAIE	TOE	ICBIE	OCBIE	reserved			CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0
rw	rw	rw	rw	rw	-			rw	rw	rw	rw	rw	rw	rw	rw

位15= **ICAIE**：输入捕获A中断使能

0：输入捕获A上没有中断

1：如果置位ICFA则产生中断。

位14= **OCAIE**：输出比较A中断使能

0：OCFA置位时没有中断

1：如果置位OCFA则产生中断。

位13= **TOIE**：定时器溢出中断使能

0：中断被禁止

1：只要SR寄存器的TOF位被置位，定时器中断打开。

位12= **ICBIE**：输入捕获B中断使能

0：输入捕获B上没有中断

1：如果置位ICFB则产生中断。

位11= **OCBIE**：输出比较B中断使能

0：OCFB置位时没有中断

1：如果置位OCFB则产生中断。

位10：8=保留。这些位必须写为0。

位7：0=**CC7-CC0**：预分频器分频因子

这8位数是预分频器用来分频内部时钟的因子。定时器时钟等于$f_{PCLK2 / (CC^{7} \div CC0 + 1)}$。

8.6.8 状态寄存器(TIMn_SR)

地址偏移量：1Ch

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICFA	OCFA	TOF	ICFB	OCFB	reserved										
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	-										

位15= **ICFA**:输入捕获标志A

0：无输入捕获（复位值）

1：输入捕获出现，要清除这一位，写入SR寄存器，位15写0（其它位全写为“1”，避免误清除其它等待位）。

位14= **OCFA**:输出比较标记A

0：没有匹配成功（复位值）
1：计数器的值和OCAR寄存器的值相同。在PWM模式下，即使计数器的值匹配OCAR寄存器的值，这位也不被置位。要清除这一位，向SR寄存器的位14写“0”（其它的位都写1，避免误把别的等待位清零）。

位13=**TOF**：定时器溢出
0：无定时器溢出（复位值）。
1：定时器从FFFFh翻转到0000h。要清除这一位，向SR寄存器的位13写0（其它的位都写1，避免误把别的等待位清零）。

位12= **ICFB**：输入捕获标志B
0：无输入捕获（复位值）
1：输入捕获出现。要清除这一位，向SR寄存器的位12写0（其它的位都写1，避免误把别的等待位清零）。

位11= **OCFB**：输出比较标志B
0：没有达到匹配（复位值）
1：计数器的值和OCBR寄存器的值匹配。在PWM模式下也能被置位。要清除这一位，向SR寄存器的位11写0（其它的位都写1，避免误把别的等待位清零）。

8.7 定时器寄存器映射

表30 定时器寄存器映射

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	TIMn_ICAR	Input Capture A															
4	TIMn_ICBR	Input Capture B															
8	TIMn_OCAR	Output Compare A															
C	TIMn_OCBR	Output Compare B															
10	TIMn_CNTR	Counter Value															
14	TIMn_CR1	EN	PWMI	reserved		FOLVB	FOLVA	OLVLB	OLVLA	OCBE	OCAE	OPM	PWM	IEDGB	IEDGA	EXEDG	ECKEN
18	TIMn_CR2	ICAIE	OCAIE	TOE	ICBIE	OCBIE	reserved			CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0
1C	TIMn_SR	ICFA	OCFA	TOF	ICFB	OCFB	reserved										

关于基地址请参见表3“APB2 存储器映射”。

9 CAN 总线

9.1 介绍

C_CAN 模块包含有 CAN 总线内核（CAN Core），报文 RAM 存储器，报文处理器，控制寄存器和模块接口（见图 44）。

CAN 总线内核按照 CAN 总线协议 2.0 版 A、B 部分来实现通信。位速率可以被编程，最高达到 1MBit/s。为了和物理层连接，还需要另外的收发器硬件。

为了实现在CAN总线上的通信，需要设定特定的报文对象（Message Object）。报文对象和用于接收报文过滤的标识符掩码被储存在报文存储器（Message RAM）中。

报文处理器（Message Handler）实现了有关报文处理的所有功能。这些功能包括接收过滤、CAN总线内核和报文存储器间的报文传递、处理发送请求和模块中断的产生。

C_CAN总线的寄存器组可以由CPU直接通过模块接口来访问。这些寄存器被用来控制或者设置CAN内核和报文处理器，也可以来访问报文存储器。

9.2 主要特征

- 支持CAN总线协议2.0A和B部分
- 位速率最大为1 M Bit/s
- 32个报文对象
- 每个报文对象都有自己的标识符掩码
- 可编程的FIFO模式（报文对象的串接）
- 可屏蔽中断
- 对于时间触发的CAN总线应用可以关闭自动重传模式
- 可编程回环模式可供测试操作
- 8位非复用的与Motorola HC08 兼容的模块接口
- 两个16位对AMBA APB的模块接口

9.3 功能框图

C_CAN 与AMBA(Advanced Microcontroller Bus Architecture) 的APB(AMBA外设总线)接口。图44给出了C_CAN的功能框图。

CAN核（CAN Core）

用于报文的串行/并行转换的CAN协议控制器和读写移位寄存器。

报文存储器（Message RAM）

存储报文对象和标识符掩码。

寄存器

所有用来控制和设置C_CAN的寄存器。

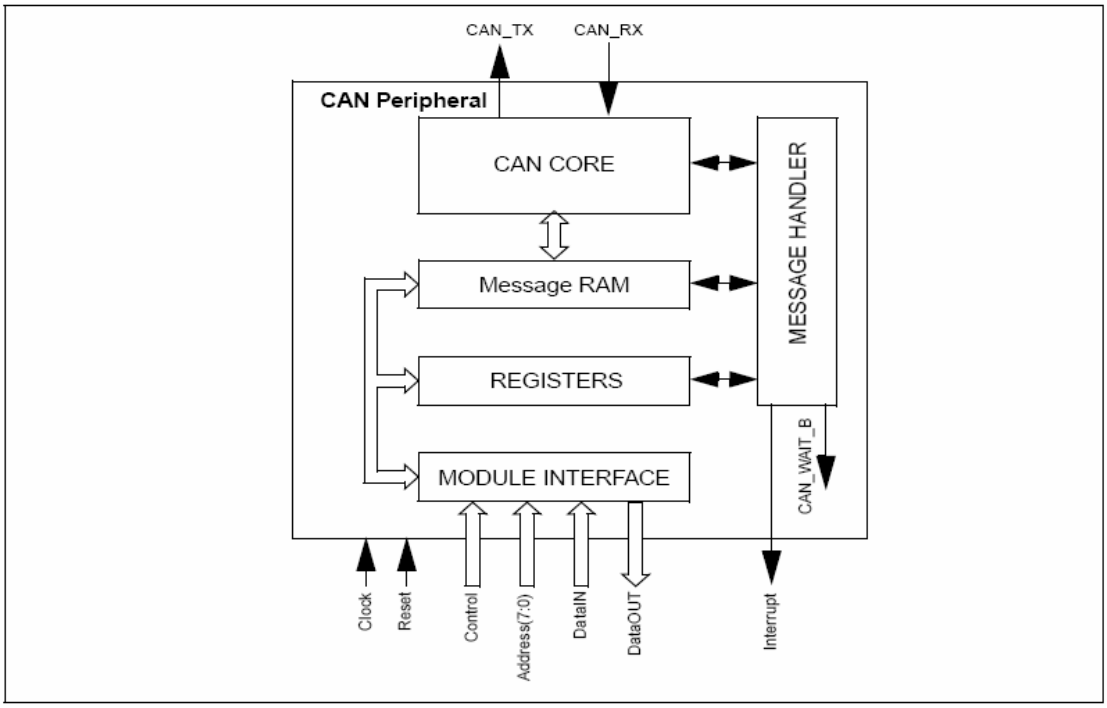
报文控制器：

控制CAN内核中的读写（Rx/Tx）移位寄存器和报文存储器之间的数据传输的状态机。状态机还控制按照对控制和配置寄存器的编程控制中断的产生。

模块接口

C_CAN与来自ARM的16位AMBA APB总线的接口。

图44 CAN外设框图



9.4 功能描述

9.4.1 软件初始化

软件初始化过程通过对CAN控制寄存器的Init位的置位来启动。软件或硬件复位，或进入总线关闭状态可以将该位置位。

一旦Init位被置位，所有CAN总线发送和接收的报文传输均被停止，且CAN_TX发送输出脚的状态呈现隐性（高电平）。错误管理逻辑（EML）计数器不变。设置Init位变化不会改变任何配置寄存器。

为了初始化该CAN控制器，软件必须设定位定时寄存器和每一个报文对象。如果不需要某个报文对象，那么对应的 MsgVal 位应该被清零。换句话说，整个报文对象必须初始化。

当CAN控制寄存器的Init位和设置变化使能（CCE）位都被置位的时候，允许访问位定时寄存器和波特率预分频扩展寄存器来设定位定时参数。

对Init位的清除（只能由CPU进行）完成了软件初始化。之后，比特流处理器（BSP）（见 9.8.10：设置位定时器）使自己与CAN总线上的数据传输同步。它在参与到总线活动并且开始报文传输以前，需要等待有11个连续隐性位（总线空闲）的位序列的出现，以此实现同步。

对报文对象的初始化独立于Init位，可以在任何时刻进行。但在BSP开始报文传输之前，所有报文对象都应该被设定到特定的标识符，或被设定为无效。

为了能够在正常工作时改变对报文对象的设置，软件首先需要清除相应的MsgVal位。设置一旦完成，MsgVal位再次重新置位。

9.4.2 CAN报文传输

一旦C_CAN完成初始化且初始位被清零， C_CAN核会使自己与总线同步并开始报文传输。

收到的报文若通过了报文处理器的接收过滤，就被存储到它们对应的报文对象中。整个报文，包括所有的仲裁位，DLC和8字节数据都会被存储到报文对象中。如果标识符掩码被使用，报文中被设置为“不关心”的仲裁位可能会被覆盖。

软件可以在任何时刻通过接口寄存器来读或者写每一个报文消息，且报文控制器在同时访问的发生时可保证数据的一致性。

需要发送的报文由应用软件来更新。如果对于某个报文存在一个永久的报文对象（仲裁和控制位在配置时被设定），那么仅更新数据字节，且设置NewDat的TxRqst位来开始传送。（当报文对象不够用的时候）如果几个传输的报文被指定在同一个报文对象下，在传输这些报文以前不得不重新设置整个的报文对象。

在同一时刻可以请求发送任意数量的报文对象。报文对象按照它们的内部优先级来相继传送。在任意时刻都可以更新报文，或将其设定为无效，即使是在其请求发送等待过程中也可以进行。在等待的发送没有启动之前报文被更新时，旧的数据就会被丢弃。

根据对报文对象的设置，通过对一个匹配了标识符的远程帧的接收，可以自动产生报文发送请求。

9.4.3 自动重传关闭模式

按照CAN总线的规范（见ISO11898，6.3.3恢复管理），C_CAN对于传输过程中已经丢失仲裁的或者是被错误打断的帧提供自动重传手段。在传输成功结束之前，使用者将不会得到帧传输服务的确认。这就是说，在默认状态下，自动重发是使能的。此功能可以被取消，使得C_CAN能工作在时间触发CAN（TTCAN，见ISO11898-1）环境下。

自动重传关闭模式的设定，是通过将CAN控制器中的关闭自动重传（DAR）位置1来实现。在这种工作模式下，编程者必须要考虑报文缓冲器中控制寄存器的TxRqst和NewDat位的不同现象：

- 传输开始的时候，对应接收报文缓冲器的TxRqst位被清零，而NewDat位仍保持置位。
- 当传输成功结束后，NewDat位被清零。
- 当传输失败时（由于丢失仲裁或者错误），NewDat位仍旧保持置位。

要重新开始传输，CPU应该再次设置TxRqst位。

9.5 测试模式

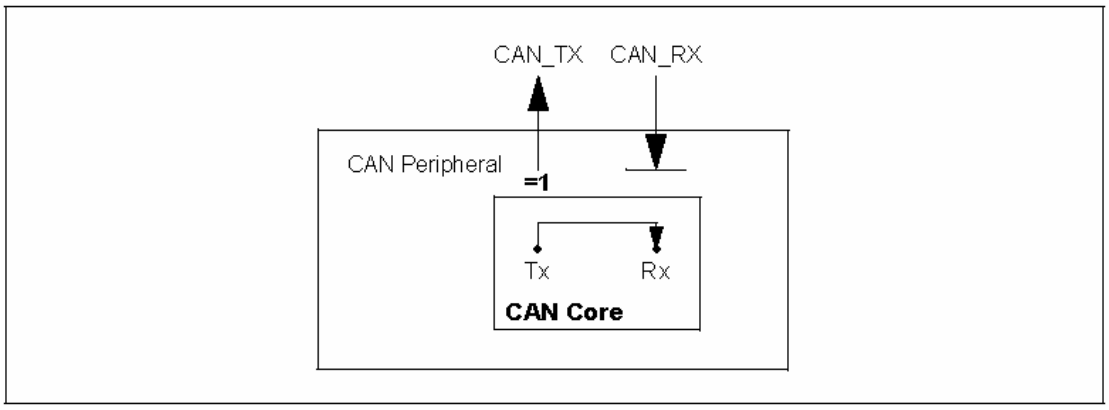
设置CAN控制器中的Test位可以进入测试模式。在测试模式中，测试寄存器中的Tx1, Tx0, LBack, Silent 以及Basic位是可写的。Rx位监视CAN_RX引脚的状态，因此是只读的。当测试位Test被清零时，所有的测试寄存器功能将无效。

9.5.1 静默模式

将测试寄存器中的静默位Silent置1可以让CAN核进入静默（Silent）模式。

在此模式下，C_CAN可以接收到有效的数据帧和有效的远程帧，但是，它仅仅在CAN总线上发送隐性的数据位，并且它不能启动一个传送过程。如果需要CAN核传输一个显性位（例如ACK位，错误帧），该位会在内部变更路径，让CAN核能监视这个显性位，而CAN总线可能仍保持在隐性状态。静默模式可被用来分析CAN总线上的数据流动状况，而不会因传送显性位影响到总线。图45给出了静默模式下CAN_TX和CAN_RX信号与CAN核的连接情况。

图45 静默模式下的CAN内核

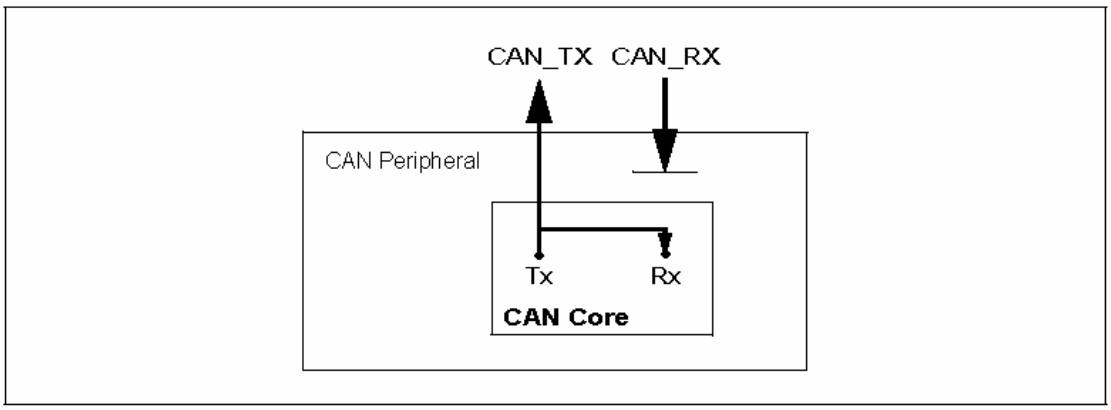


在ISO11898-1规范中，静默模式叫做总线监视模式。

9.5.2 回环模式

将测试寄存器中的LBack位编程为1可让CAN内核进入回环模式。在此模式下，CAN内核把自己传送的报文作为接收到的报文并存储在接收缓冲器中（如果报文通过了接收过滤）。图46给出了在此模式下CAN_TX和CAN_RX信号与CAN核的连接情况。

图46 在回环模式下的CAN内核

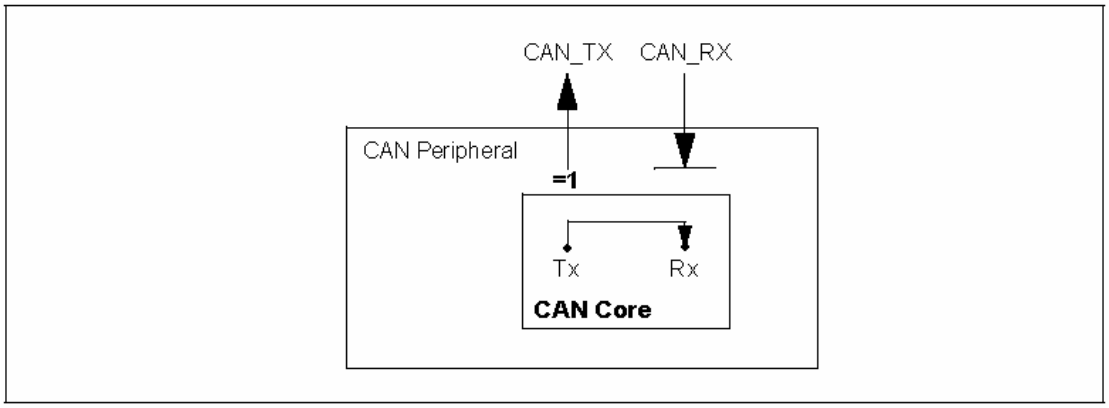


这种模式用来进行自测。为了不受到外部的影响，该回环模式下CAN内核会忽略确认错误（一个数据帧或者远程帧的确认段位里采样到隐性位）。此模式下，CAN核实现一个从Tx 输出到它Rx输入的内部反馈。CAN_RX输入引脚的真实值被CAN核忽略。被传送的报文可以在CAN_TX引脚上监视到。

9.5.3 回环模式与静默模式的结合

同时对LBack和Silent位编程可以将回环模式与静默模式结合在一起。这种模式可以被用来做一个“热测试”（Hot Selftest），这就意味着可以对C_CAN进行测试，而不会影响到连接在CAN_TX和CAN_RX引脚的正在运行的CAN系统。这种模式下，CAN_RX引脚与CAN核的连接被断开，CAN_TX管脚被保持隐性。图47给出了在这种结合模式下CAN_TX和CAN_RX信号与CAN内核的连接情况。

图47 在回环模式与静默模式结合模式下的CAN内核



9.5.4 基本模式

将测试寄存器的Baisc位编程为“1”，CAN核可以进入到基本（Basic）模式。该模式下C_CAN的工作不用报文随机存储器（Message RAM）。

IF1寄存器作为发送缓冲器使用。通过对IF1命令请求寄存器中的Busy位写“1”来请求IF1寄存器中的内容的传送。当Busy位被置位时，IF1寄存器会被锁定。Busy位用来表明传输任务正在等待。

一旦CAN总线空闲下来，IF1寄存器会被载入到CAN内核的移位寄存器，传送开始。当传送结束后，Busy位被清除，同时锁定的IF1寄存器会被释放。

当IF1寄存器被锁定的时候，一个正在等待的发送任务可以因为IF1命令请求寄存器中的Busy位被清除而被取消。如果CPU已经将Busy位清除，那么可能由于丢失仲裁或者出错而需要进行的重发操作会被取消掉。

IF2寄存器被用作接收缓冲器。在接收到一条报文后，移位寄存器的内容会存储到IF2该寄存器中，且不做任何接收过滤。

另外，移位寄存器中的实际内容在报文传输过程中可以被监视。每次当一个对IF2命令请求寄存器中的Busy位写入1时，读报文对象会被初始化，移位寄存器中的内容被存储到IF2寄存器中。

在基本模式下，对与所有报文对象相关的控制和状态位的运算，以及对IFn命令掩码寄存器的控制位的运算都会被关闭。不会计算命令请求寄存器中的报文编号。IF2报文控制寄存器中的NewDat 和 MsgLst 位会保留它们的功能，DLC3-0表示收到的DLC，其他控制位按“0”被读出。

9.5.5 CAN_TX引脚的软件控制

CAN的传送引脚CAN_TX有4个输出功能。除了它的默认功能（串行数据输出）外，这个CAN传输引脚可以驱动CAN的取样点信号来监视CAN内核的位定时。它也能驱动发出连续的显性或者隐性值。与CAN的CAN_RX的接收引脚结合起来，上述后两个功能可以被用来检查CAN总线的物理层。

通过对CAN测试寄存器中TX1和TX0位的编程可以选定CAN_TX管脚的输出模式。

CAN_TX管脚的三种测试功能会对所有的CAN协议功能造成干扰。当选定CAN报文传送或者任何一个测试模式（回送模式、静默模式或者基本模式）时， CAN_TX必须使用它的默认功能。

9.6 寄存器描述

C_CAN分配了一个256个字节的地址空间。寄存器都是被按照16位的寄存器来组织的。

IF1和IF2两组接口寄存器可以控制CPU对报文随机存储器的访问。它们对要传往或者来自RAM的数据进行缓冲，这就消除了CPU的读写与报文的接收/传送之间的冲突。

在这部分，使用了如下的缩写：

read/write (rw)　　软件可对这些位进行读写

read-only (r)　　软件只能读这些位
write-only (w)　　软件只应当写这些位

CAN的寄存器列表如表**31**所示。

表31 CAN寄存器

Register Name	Address Offset	Reset Value
CAN Control Register (CAN_CR)	00h	0001h
Status Register (CAN_SR)	04h	0000h
Error Counter (CAN_ERR)	08h	0000h
Bit Timing Register (CAN_BTR)	0Ch	2301h
Test Register (CAN_TESTR)	14h	0000 0000 R000 0000 b
BRP Extension Register (CAN_BRPR)	18h	0000h
IFn Command Request Registers (CAN_IFn_CRR)	20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)	
IFn Command Mask Registers (CAN_IFn_CMR)	24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)	0000h
IFn Mask 1 Register (CAN_IFn_M1R)	28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)	FFFFh
IFn Mask 2 Register (CAN_IFn_M2R)	2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)	FFFFh
IFn Message Arbitration 1 Register (CAN_IFn_A1R)	30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)	0000h
IFn Message Arbitration 2 Register (CAN_IFn_A2R)	34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)	0000h
IFn Message Control Registers (CAN_IFn_MCR)	38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)	0000h
IFn Data A/B Registers (CAN_IFn_DAnR and CAN_IFn_DBnR)		

Register Name	Address Offset	Reset Value
Interrupt Identifier Register (CAN_IDR)	10h	0000h
Transmission Request Registers 1 & 2 (CAN_TxRnR)	100h (CAN_TxR1R), 104h (CAN_TxR2R)	0000 0000h
New Data Registers 1 & 2 (CAN_NDnR)	120h (CAN_ND1R), 124h (CAN_ND2R)	0000 0000h
Interrupt Pending Registers 1 & 2 (CAN_IPnR)	140h (CAN_IP1R), 144h (CAN_IP2R)	0000 0000h
Message Valid Registers 1 & 2 (CAN_MVnR)	160h (CAN_MV1R), 164h (CAN_MV2R)	0000 0000h

9.6.1 CAN总线接口复位状态

在硬件被复位以后，C_CAN寄存器保持着上表中给定的复位值。

另外，Busoff状态被复位，且输出口CAN_TX被设定为隐性（高电平）。CAN控制寄存器的值0X0001（Init位='1'）允许进行软件初始化。在CPU复位Init位为"0"之前，C-CAN不会影响到CAN总线。

报文随机存储器中存储的数据不会受到硬件复位的影响。上电后，该存储器中的内容不确定。

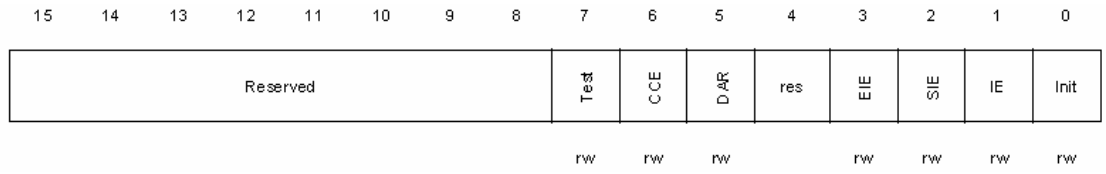
9.6.2 与CAN总线协议相关的寄存器

这些寄存器与CAN内核中的CAN协议控制器相关。它们控制着工作模式和CAN位定时的设置，

还提供各种状态信息。

9.6.2.1 CAN控制寄存器(CAN_CR)

地址偏移：00h，
复位值：0001h



位15-8： 保留的。
这些位是保留位，永远读为“0”，必须写入“0”。

位7： Test：测试模式使能
0：正常操作
1：测试模式

位6： CCE：设置变化使能。
0：不能对位定时寄存器进行写访问
1：允许对位定时寄存器中写操作（当Init位= 1 时）。

位5： DAR:：关闭自动
0：允许对收到干扰的报文进行
1：自动重传被禁止

位4： 保留，
该位为保留位，永远读为“0”，必须写入“0”。

位3： EIE： 出错中断使能
0：禁止，不会产生错误状态中断
1：允许，状态寄存器中Boff或Ewarn位的变化将会产生一个中断

位2： SIE：状态改变中断使能
0：禁止- 不会产生状态改变中断
1：允许- 当一次报文传输成功完成或检测出一个CAN总线错误时产生中断。

位1： IE：模块中断使能
0：禁止
1：允许

位0： Init位，初始化
0：正常工作
1：启动了初始化。

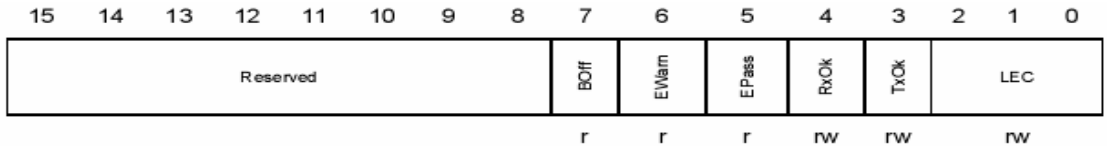
注意： busoff恢复序列（见CAN规范2.0）不能被初始位的置位或复位缩短。如果器件进入了总线

关闭（busoff）状态，它将会自己置位Init，从而使总线的所有活动停止。一旦Init位被CPU清零，器件将会等待129个总线空闲（129*11个连续隐性位）然后再回到正常操作。在**busoff**恢复序列结束时，错误管理计数器会被清除。

在清除Init位后的等待时间里，当每有11个连续隐性位序列被监测到，就有一个Bit0Error码被写入到状态寄存器中，这就让CPU能都很容易地检验CAN总线是否在显性处滞留，或者一直在被打乱，并且还能让CPU监视busoff恢复序列的进行情况。

9.6.2.2 状态寄存器(CAN_SR)

地址偏移：04h
复位值：0000h。



位15-8： 保留
这些位永远读为“0”，必须写入“0”。

位7： BOff：Busoff状态
0：CAN模块没有处在busoff状态
1：CAN模块正处在busoff状态。

位6： EWarn:：告警状态
0：两个出错计数器都在告警限96以下
1：至少有一个EML中的出错计数器达到了告警限96。

位5： EPass：出错认可状态（Error Passive）
0：CAN内核处于出错活动状态（error active）
1：CAN内核处在CAB规范定义的出错认可状态。

位4： RxOk：成功接收到一个报文
0：自从这一位最后一次被CPU清除以后，还没有成功接收到一个报文。该位从不会被CAN内核清除。
1：从这一位最后一次被CPU清除以后已经成功接收到一个报文（与接收过滤的结果无关）。

位3： TxOk：成功发送了一个报文
0：从这一位最后一次被CPU清除以后，还没有成功发送一个报文。该位从不会被CAN内核清除。
1：从这一位最后一次被CPU清除以后已经成功发送了一个报文（没有出错，且得到至少一个节点的确认）。

位2-0： LEC[2:0]：最后出错代码（最后出现在CAN总线上的错误类型）
LEC域中保留着一个代码，这个代码表示了总线上发生的最后一个错误的类型。当报文无错传输（接收或发送）后，这个域将被清零。无用的代码“7”可由CPU写入，用来检查更新。表32 给出了这些出错码：

表32 出错代码

Error Code	Meaning
0	No Error
1	Stuff Error: More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
2	Form Error: A fixed format part of a received frame has the wrong format.
3	AckError: The message this CAN Core transmitted was not acknowledged by another node.
4	Bit1Error: During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.
5	Bit0Error: During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), though the device wanted to send a dominant level (data or identifier bit logical value '0'), but the monitored Bus value was recessive. During busoff recovery, this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceedings of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continuously disturbed).
6	CRCError: The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.
7	Unused: When the LEC shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC.

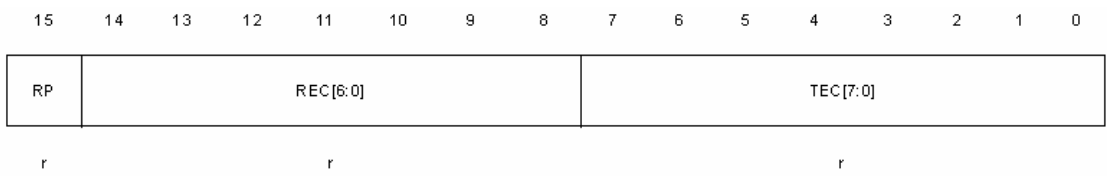
状态中断：

一个状态中断是由BOff和EWarn (出错中断)或者是RxOk，TxOk，和LEC (状态变化中断)产生的，前提是CAN控制寄存器中的对应使能位被置位了。EPass位的一个变化，或者对RxOk，TxOk，或LEC的写操作都不会产生一个状态中断。

如果状态寄存器在等待，读状态寄存器将会把中断寄存器中状态中断值(8000h)清零。

9.6.2.3 出错计数器(CAN_ERR)

地址偏移：08h
复位值：0000h



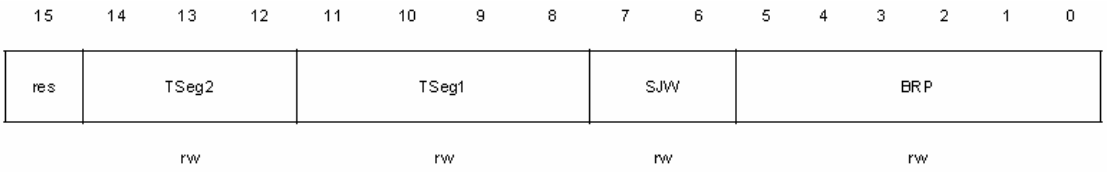
位15： RP：接收出错认可
0：出错计数器在出错认可等级以下；
1：出错计数器已经达到了CAN总线规范定义的出错认可等级。

位14-8： REC[6:0]：接收出错计数器
接收出错计数器的实际状态，它的值是从0到127。

位7-0： TEC[7:0]：发送出错计数器
发送出错计数器的实际状态，它的值是从0到255。

9.6.2.4 位定时寄存器(CAN_BTR)

地址偏移：0Ch
复位值：2301h



位15： 保留
该位为保留位，永远读为“0”，必须写入“0”。

位14-12： TSeg2：采样点后时间间隔
0x0-0x7：TSeg2的有效值是[0…7]。硬件的实际采用值是这里的编程值加1。

位11-8： TSeg1：采样点前时间间隔减去Sync_Seg值
0x01-0x0F：TSeg1的有效值是[1…15]。 硬件的实际采用值是这里的编程值加1。

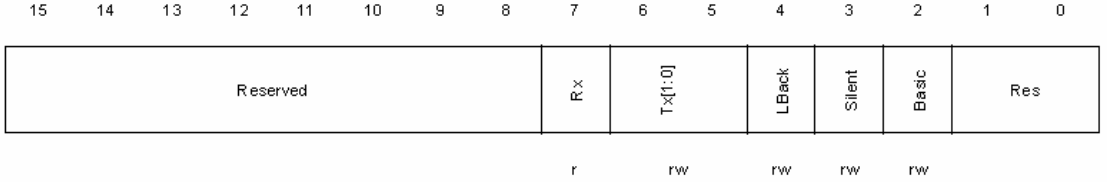
位7-6位： SJW：(重)同步跳跃宽度
0x0-0x3：有效的编程值是[0…3] 。硬件的实际采用值是这里的编程值加1。

位5-0： BRP：波特率预分频因子
0x01-0x3F：振荡器的频率被这个值分频用来产生位时间份额。位时间由这样的量子的倍数组成。波特率预分频因子的有效取值为[0…63]。硬件的实际采用值是这里的编程值加1。

注意： 使用8MHz的模块时钟APB_CLK时，复位值0x2301设定了C_CAN的速率为500 kBit/s。只有在CAN控制寄存器中的CCE位和Init位被置位时才能对这些寄存器写入。

9.6.2.5 测试寄存器(CAN_TESTR)

地址偏移：14h
复位值：0000 0000 R000 0000 b (R: RX引脚的当前值)



位15-8： 保留
这些位为保留位，永远读为“0”，必须写入“0”。

位7： Rx：CAN_RX管脚的当前值
0：CAN总线是显性的(CAN_RX = ‘0’)
1：CAN总线是隐性的(CAN_RX = ‘1’)

位6-5： Tx[1:0]：对CAN_TX引脚控制
00：复位值，CAN_TX由CAN核控制
01：在CAN_TX引脚处可以监视采样点
10：CAN_TX管脚驱动一个显性值(‘0’)
11：CAN_TX管脚驱动一个隐性值(‘1’)

位4： LBack：回环模式
0：回环模式被禁止
1：回环模式被允许

位3： Silent：静默模式
0：正常操作
1：模块处于静默模式

位2： Basic：基本模式
0：基本模式被禁止
1：IF1 寄存器被用作Tx Buffer缓冲器, IF2 寄存器被用作Rx缓冲器

位1:0： 保留
这些位为保留位，永远读为“0”，必须写入“0”。

通过对CAN控制寄存器中的测试位Test置位，可允许对测试寄存器写操作。不同的测试功能可以结合，但是，Tx1-0 ≠ “00”会打乱报文的传输。

9.6.2.6 BRP扩展寄存器(CAN_BRPR)

地址偏移：18h
复位值：0000h

位15-4位：保留
这些位为保留位，永远读为“0”，必须写入“0”。

位3-0： BRPE：波特率预分频因子扩展
0x00-0x0F：通过对BRPE编程，波特率的预分频值可以扩展到1023。实际硬件实现中，硬件采用的值是对BRPE (高位)和BRP (低位)的编程值加1。

9.6.3 报文接口寄存器组

有两组接口寄存器，用来控制CPU对报文RAM的访问。通过对要传送的报文进行缓冲，接口寄存器避免了CPU对报文RAM访问与CAN报文的接收和发送之间的冲突。在一个传送过程中，一个完

整的报文对象（见 9.6.3.10）或者报文对象的某些部分可能会在报文RAM与IFn报文缓冲寄存器之间传送。

除了在基本模式下，这两组接口寄存器的功能是相同的。它们的使用方式可以是用一组寄存器将报文传送到报文RAM，而用另一组寄存器将报文从报文RAM中取出，允许这两个过程可以相互中断。表**33 IF1和IF2报文接口寄存器组**给出了这两个接口寄存器组的概要。

每一组接口寄存器由报文缓冲寄存器组成，它们由各自的命令寄存器控制。命令掩码寄存器说明了数据传送的方向以及报文对象的哪个部分将被传送。命令请求寄存器用来在报文RAM中选择一个报文对象作为目标或者源用于传送，并用来启动命令掩码寄存器中指定的操作。

表33 IF1和IF2报文接口寄存器组

Address	IF1 Register Set	Address	IF2 Register Set
CAN Base + 0x20	IF1 Command Request	CAN Base + 0x80	IF2 Command Request
CAN Base + 0x24	IF1 Command Mask	CAN Base + 0x84	IF2 Command Mask
CAN Base + 0x28	IF1 Mask 1	CAN Base + 0x88	IF2 Mask 1
CAN Base + 0x2C	IF1 Mask 2	CAN Base + 0x8C	IF2 Mask 2
CAN Base + 0x30	IF1 Arbitration 1	CAN Base + 0x90	IF2 Arbitration 1
CAN Base + 0x34	IF1 Arbitration 2	CAN Base + 0x94	IF2 Arbitration 2
CAN Base + 0x38	IF1 Message Control	CAN Base + 0x98	IF2 Message Control
CAN Base + 0x3C	IF1 Data A 1	CAN Base + 0x9C	IF2 Data A 1
CAN Base + 0x40	IF1 Data A 2	CAN Base + 0xA0	IF2 Data A 2
CAN Base + 0x44	IF1 Data B 1	CAN Base + 0xA4	IF2 Data B 1
CAN Base + 0x48	IF1 Data B 2	CAN Base + 0xA8	IF2 Data B 2

9.6.3.1 IFn 命令请求寄存器(CAN_IFn_CRR)

地址偏移：20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)

初始值：0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Busy	Reserved									Message Number					
r										rw					

一旦应用程序将报文编号写入命令请求寄存器，一个报文的传送就开始了。由于这个写操作，Busy位被自动地置位，用以通知CPU传送正在进行。在等待了3到6个APB_CLK时钟周期后，在接口寄存器和报文RAM之间的传送过程完成，Busy位被清零。

位15 Busy：忙标志位

0：读/写操作已经完成

1：正在对IFn命令请求寄存器进行写操作。

这个位只能被软件读。

位14:6 保留的

这些位为保留位，永远读为“0”，必须写入“0”。

位5:0 报文号

0x01-0x20：有效的报文号，报文RAM中的报文对象被选定用于数据传送

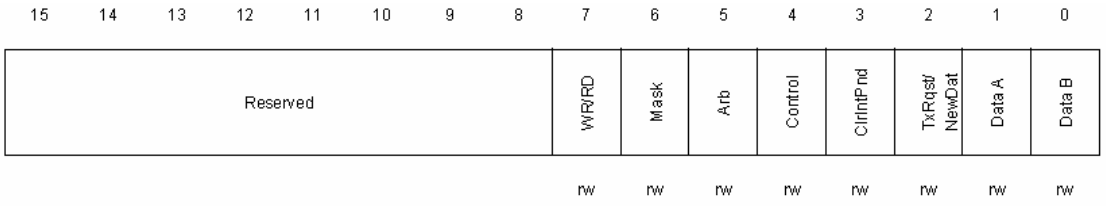
0x00：无效的报文号，被解释为0x20

0x21-0x3F：无效的报文号，被解释为0x01-0x1F。

注意： 当一个无效的报文号被写入命令请求寄存器时，报文编号将被变换成有效值，然后报文对象将被传送。

9.6.3.2 IFn命令掩码寄存器(CAN_IFn_CMR)

地址偏移：24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)
复位值：0000h



IFn命令掩码寄存器的控制位指定了传送方向，并选定IFn报文缓冲寄存器中哪一个作为数据传送的源或目的。

位15:8 保留
这些位为保留位，永远读为“0”，必须写入“0”。

位7 WR/RD：写/读
0：读：从命令请求寄存器寻址的报文对象传送数据到选定的报文缓冲寄存器
1：写：从选定的报文缓冲寄存器传送数据到命令请求寄存器寻址的报文对象

位6:0 根据传送方向的不同，这些IFn命令掩码寄存器的位有不同的功能：

方向=写

位6 = 掩码 访问掩码位
0：掩码位无变化
1：传送标识符掩码+MDir+MXtd到报文对象

位5 = Arb 访问仲裁位
0：仲裁位无变化
1：传送标识符+Dir+Xtd+MsgVal到报文对象

位4 = Control 访问控制位
0：控制位无变化
1：传送控制位到报文对象

位3 = ClrIntPnd 清除中断等待位
当写入到报文对象时，这一位被忽略。

位2 = TxRqst/NewDat 访问发送请求位
0：TxRqst位无变化
1：置位TxRqst位

如果通过编程IFn命令掩码寄存器的TxRqst/NewDat位来请求发送，保温控制寄存器中的

TxRqst位被忽略。

位1 = Data A 访问数据字节3:0
0：数据字节3:0不变
1：传送数据字节3:0到报文对象

位0 = = Data B 访问数据字节7:4
0：数据字节7:4不变
1：传送数据字节7:4到报文对象

位6:0 **方向=读**

位6 = 掩码 访问掩码位
0：掩码位无变化
1：传送标识符掩码+MDir+MXtd到IFn报文缓冲寄存器

位5 = Arb 访问仲裁位
0：仲裁位无变化
1：传送标识符+Dir+Xtd+MsgVal到IFn报文缓冲寄存器

位4 = Control 访问控制位
0：控制位无变化
1：传送控制位到IFn报文缓冲寄存器

位3 = ClrIntPnd 清除中断等待位
0：IntPnd位保持不变
1：清除报文对象中的IntPnd位。

位2 = TxRqst/NewDat 访问发送请求位
0：NewDat位无变化
1：清除报文对象中的NewDat位。

注意：对报文对象的读访问可以和对控制位IntPnd和NewDat的清除结合起来。传送到IFn报文控制寄存器的这些位的值总是反映了清除这些位之前的状态。

位1 = Data A 访问数据字节3:0
0：数据字节3:0不变
1：传送数据字节3:0到IFn报文缓冲寄存器

位0 = = Data B 访问数据字节7:4
0：数据字节7:4不变
1：传送数据字节7:4到IFn报文缓冲寄存器

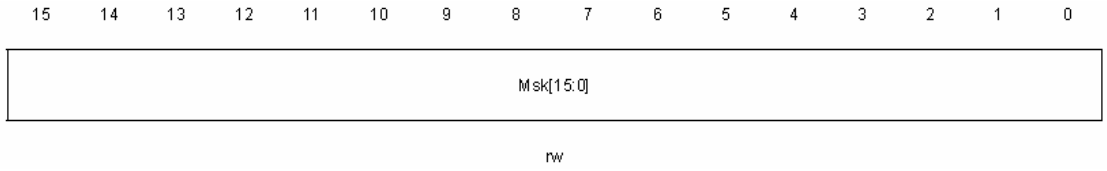
9.6.3.3 IFn报文缓冲寄存器

报文缓冲寄存器中的位映射了报文RAM中报文对象。报文对象位的功能描述见9.6.3.10：在报

文存储器中的报文对象。

9.6.3.4 IFn掩码1寄存器(CAN_IFn_M1R)

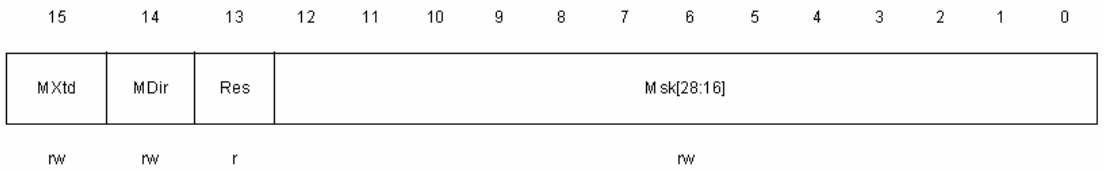
地址偏移：28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)
复位值：FFFFh



本寄存器中Msk位的功能描述见9.6.3.10: 在报文存储器中的报文对象。

9.6.3.5 IFn掩码2寄存器(CAN_IFn_M2R)

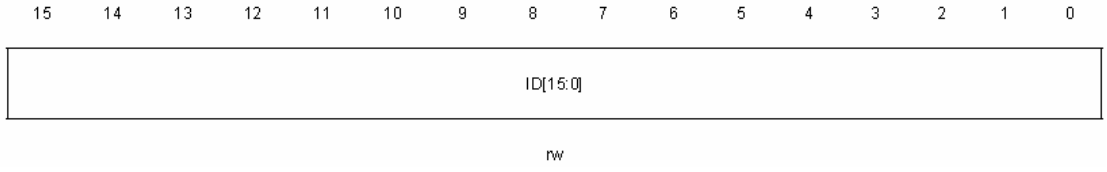
地址偏移：2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)
复位值：FFFFh



本寄存器中报文对象位的功能描述见9.6.3.10: 在报文存储器中的报文对象。

9.6.3.6 IFn报文仲裁1寄存器(CAN_IFn_A1R)

地址偏移：30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)
复位值：0000h



本寄存器中报文对象位的功能描述见9.6.3.10: 在报文存储器中的报文对象。

9.6.3.7 IFn报文仲裁2寄存器(CAN_IFn_A2R)

地址偏移：34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)
复位值：0000h

本寄存器中报文对象位的功能描述见9.6.3.10: 在报文存储器中的报文对象。

9.6.3.8 IFn报文控制寄存器(CAN_IFn_MCR)

地址偏移：38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)
复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw				rw			

本寄存器中报文对象位的功能描述见9.6.3.10: 在报文存储器中的报文对象。

9.6.3.9 IFn数据A/B寄存器(CAN_IFn_DAnR and CAN_IFn_DBnR)

CAN报文中的数据字节以下列顺序存储在IFn报文缓冲寄存器中：

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Message Data A1 (address 0x3C)	Data(1)								Data(0)							
IF1 Message Data A2 (address 0x40)	Data(3)								Data(2)							

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Message Data B1 (address 0x44)	Data(5)								Data(4)							
IF1 Message Data B2 (address 0x48)	Data(7)								Data(6)							
IF2 Message Data A1 (address 0x9C)	Data(1)								Data(0)							
IF2 Message Data A2 (address 0xA0)	Data(3)								Data(2)							
IF2 Message Data B1 (address 0xA4)	Data(5)								Data(4)							
IF2 Message Data B2 (address 0xA8)	Data(7)								Data(6)							
	rw								rw							

在一个CAN数据帧中，Data(0)是第一个，Data(7)是最后一个被发送或被接收的。在CAN总线的串行位流中，每个字节的最高位将被首先发送。

9.6.3.10: 在报文存储器中的报文对象

在报文RAM中有32个报文对象。为了避免CPU对报文RAM的访问与CAN报文的接收和发送之间的冲突，CPU不能直接访问报文对象，这些访问要通过IFn接口寄存器来处理。

表34给出了报文对象结构的概要。

表34 **报文存储器中一个报文对象的结构**

Message Object												
UMask	Msk 28-0	MXtd	MDir	EoB	NewDat		MsgLst	RxIE	TxIE	Int Pnd	RmtEn	TxRqst
MsgVal	ID28-0	Xtd	Dir	DLC 3-0	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

仲裁寄存器ID28-0，Xtd和Dir用于定义送出报文的标识符和类型，还被用来（与掩码寄存器Msk28-0, MXtd和Mdir一起）对输入报文的接收过滤。收到的报文存储在有效的报文对象中，该报文对象匹配了标识符和设定的接收（数据帧）或发送（远程帧）方向。扩展帧只能保存在设定了Xtd的报文对象中，标准帧保存在清除了Xtd的报文对象中。如果一个收到的报文（数据帧或远程帧）匹配了多个有效的报文对象，它被存储在具有最低报文编号的报文对象中。细节请见9.8.2.3“接收报文的接收过滤”。

MsgVal　报文无效

1：该报文对象已经配置，应该被报文处理器处理。

0：该报文对象被报文处理器忽略。

注意：在初始化过程中，应用软件在清除CAN控制器的Init位之前，必须清除所有没有用到的报文对象的MsgVal位。在标识符Id28-0, 控制位Xtd，Dir，或数据长度码DLC3-0被修改之前，或者该报文对象不再需要时，也必须清除该位。

Umask　使用接收掩码

1：使用掩码(Msk28-0, MXtd, and MDir)用于接收过滤。

0：忽略掩码。

注意：如果Umask位被设置为1，在将MsgVal位设置为1之前，必须在报文对象初始化时对报文对象的掩码位进行编程。

Msk28-0　标识符掩码

1：对应的标识符位被用于接收过滤

0：报文对象的标识符中对应的位不能禁止接收过滤时的匹配

Xtd　扩展标识符

1：29位（“扩展的”）标识符将用于该报文对象

0：11位（“标准的”）标识符将用于该报文对象

MXtd　掩码扩展标识符

1：扩展标识符位（IDE）用于接收过滤

0：扩展标识符位（IDE）不影响接收过滤

注意：当11位（“标准的”）标识符用于一个报文对象时，收到的数据帧的标识符被写入ID28-ID18位。对于接收过滤，只考虑这些位和掩码位中的Msk28-Msk18。

Dir　报文方向

1：方向=发送：当TxRqst设置时，相应的报文对象作为数据帧发送。当接收到具有匹配标识符的远程帧时，这个报文对象的TxRqst位被置位（若RmtEn=1）。

0：方向=接收：当TxRqst设置时，具有本报文对象标识符的远程帧被发送。当接收到具有匹配标识符的数据帧时，该报文被存储在这个报文对象中。

Mdir　掩码报文方向

1：报文方向位（Dir）用于接收过滤

0：报文方向位（Dir）不影响接收过滤

EoB　缓冲区结束

1：单个报文对象或一个FIFO缓冲区中最后一个报文对象

0：报文对象属于一个FIFO缓冲区，且不是该FIFO缓冲区中最后一个报文对象。

注意：该位用来连接两个或多个报文对象（最多32个）形成FIFO缓冲区。对于单个报文对象（不属于一个FIFO缓冲区），该位必须永远设置为1。有关报文对象连接的细节请见9.8.7“设置一个FIFO缓冲区”。

NewDat 新数据

1：报文处理器或应用软件已经向该报文对象的数据部分写入了新数据。

0：从应用软件最近一次清除该标志以后，还没有由报文处理器向该报文对象的数据部分写入新数据。

MsgLst 报文丢失（只当报文对象的方向=接收时有效）

1：当NewDat仍然置位时，报文处理器将一个新报文存储到这个报文对象中，CPU丢失了一个报文。

0：从最近一次本位被CPU清除以来没有报文丢失。

RxIE 接收中断使能

1：在成功接收到一帧后将置位IntPnd位。

0：在成功接收到一帧后IntPnd位保持不变。

IntPnd 中断等待

1：本报文对象是中断来源。如果没有其他更高优先级的中断，在中断寄存器中的中断标识符将指向这个报文对象。

0：本报文对象不是中断来源。

RmtEn 远程使能

1：在接收到一个远程帧时，TxRqst被置位。

0：在接收到一个远程帧时，TxRqst不改变。

TxRqst 发送请求

1：已经请求了本报文对象的发送，发送尚未完成。

0：本报文对象没有等待发送。

DLC3-0 数据长度码

0-8：数据帧有0-8字节

9-15：数据帧有8字节

注意：一个报文对象的数据长度码的定义必须与其他节点上具有相同标识符的所有相应对象的长度码相同。当报文处理器存储一个数据帧时，它将DLC写为接受到报文给定的值。

Data 0：CAN数据帧的第1个字节

Data 1：CAN数据帧的第2个字节

Data 2：CAN数据帧的第3个字节

Data 3：CAN数据帧的第4个字节

Data 4：CAN数据帧的第5个字节

Data 5：CAN数据帧的第6个字节

Data 6：CAN数据帧的第7个字节

Data 7：CAN数据帧的第8个字节

注意：在接收过程中，Data 0字节是移入CAN内核移位寄存器的第1个字节，而Data 7是最后一个字节。当报文处理器存储一个数据帧时，它将所有8个字节写入一个报文对象。如果数据长度码小于8，报文对象的余下字节将被不确定的数值覆盖。

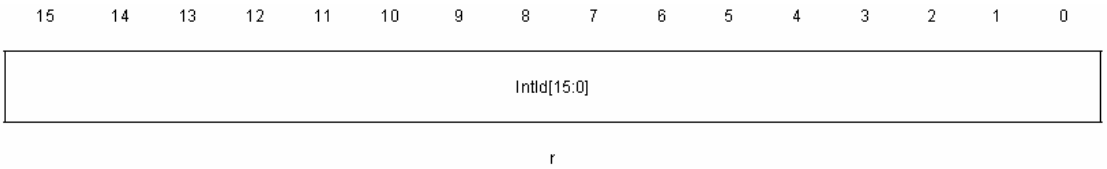
9.6.4 报文处理器寄存器

所有的报文处理器寄存器都是只读的。其内容，每个报文对象的TxRqst, NewDat, IntPnd,和

MsgVal位，以及中断标识符都是由报文处理器状态机提供的状态信息。

9.6.4.1 中断标识符寄存器

地址偏移：10h
复位值：0000h



位15:0　IntId15:0 中断标识符（表35指出了中断来源）

如果有多个中段在等待，CAN中断寄存器将指向具有最高优先级的等待中断，而不理会它们的时间顺序。在应用软件对其清除之前，中断一直保持等待。如果IntId不是0x0000且IE为1，输入到EIC的IRQ中断信号有效。中断一直保持有效，直到IntId变回0x0000（中断的起因被复位）或IE被清除。

状态中断具有最高优先级。在报文中断中，报文对象的中断优先级随着报文编号的增大而降低。

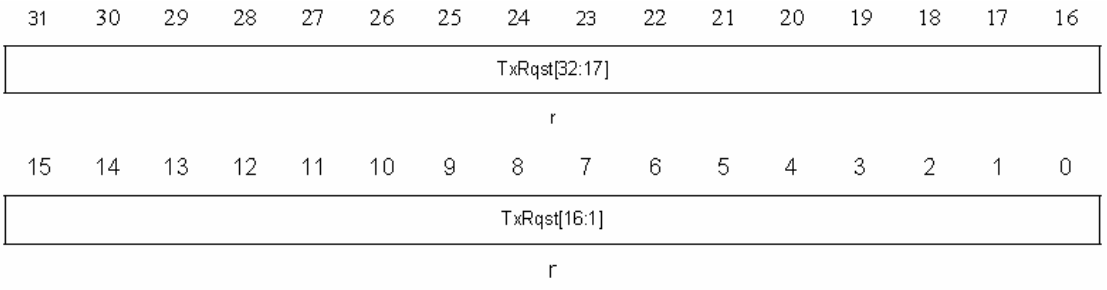
通过清除报文对象的IntPnd位来清除报文中断。状态中断的清除是通过对状态寄存器的读操作来完成。

表35 中断来源

0x0000	No Interrupt is Pending
0x0001-0x0020	Number of Message Object which caused the interrupt.
0x0001-0x0020	Number of Message Object which caused the interrupt.
0x0021-0x7FFF	unused
0x8000	Status Interrupt
0x8001-0xFFFF	unused

9.6.4.2 发送请求寄存器1,2(CAN_TxRnR)

地址偏移：100h (CAN_TxR1R), 104h (CAN_TxR2R)
复位值：00000000h



这些寄存器保存了32个报文对象的TxRqst位。通过读取TxRqst位，CPU可以检查在一个发送请求中哪一个报文对象正在等待。一个特定报文对象的TxRqst位可以由应用软件通过IFn报文接口寄存器来置位/复位，也可以由报文处理器在接收到一个远程帧之后或在一次成功发送之后来置位或复位。

位31:16　TxRqst32-17 （所有报文对象的）发送请求位

0：该报文对象没有等待发送

1：该报文对象请求了发送但还没有完成。

这些位是只读的。

位15:0 TxRqst16-1 （所有报文对象的）发送请求位

0：该报文对象没有等待发送

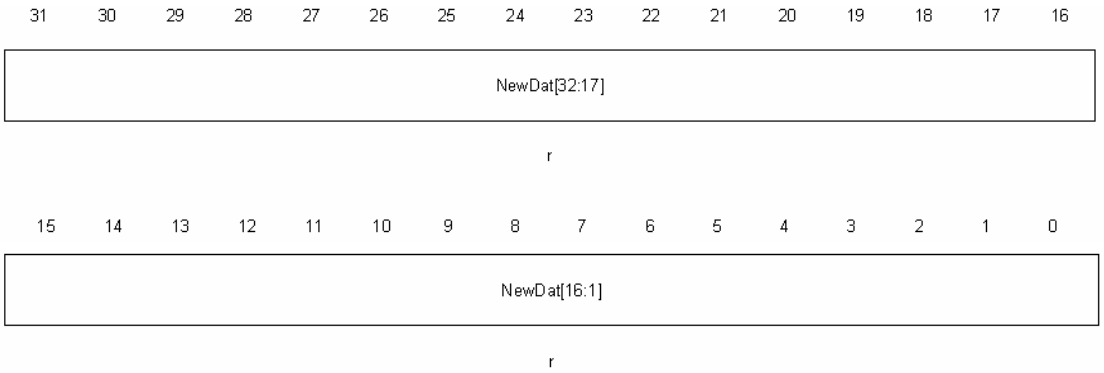
1：该报文对象请求了发送但还没有完成。

这些位是只读的。

9.6.4.3 新数据寄存器1, 2 (CAN_NDnR)

地址偏移：120h (CAN_ND1R), 124h (CAN_ND2R)

复位值：0000 0000h



这些寄存器保存了32个报文对象的NewDat位。通过读取NewDat位，CPU可以检查哪一个报文对象的数据部分被更新了。一个特定报文对象的NewDat位可以由CPU通过IFn报文接口寄存器来置位/复位，也可以由报文处理器在接收到一个数据帧之后或在一次成功发送之后来置位或复位。

位31:16 NewDat32:17 （所有报文对象的）新数据位

0：从应用软件最后一次清除该位以来，还没有任何新数据由报文处理器写入报文对象的数据部分。

1：报文处理器或应用软件已经将新数据写入到这个报文对象的数据部分。

位15:0 NewDat16:1 （所有报文对象的）新数据位

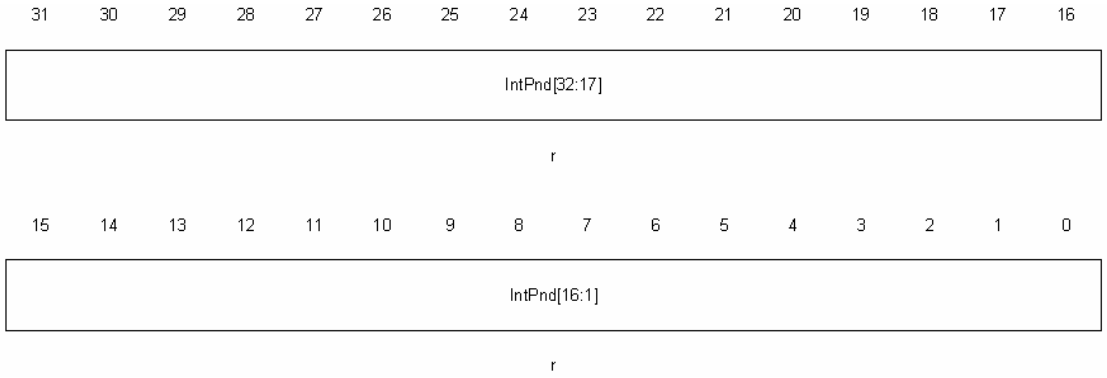
0：从应用软件最后一次清除该位以来，还没有任何新数据由报文处理器写入报文对象的数据部分。

1：报文处理器或应用软件已经将新数据写入到这个报文对象的数据部分。

9.6.4.4 中断等待寄存器1，2 (CAN_IPnR)

地址偏移：140h (CAN_IP1R), 144h (CAN_IP2R)

复位值：0000 0000h



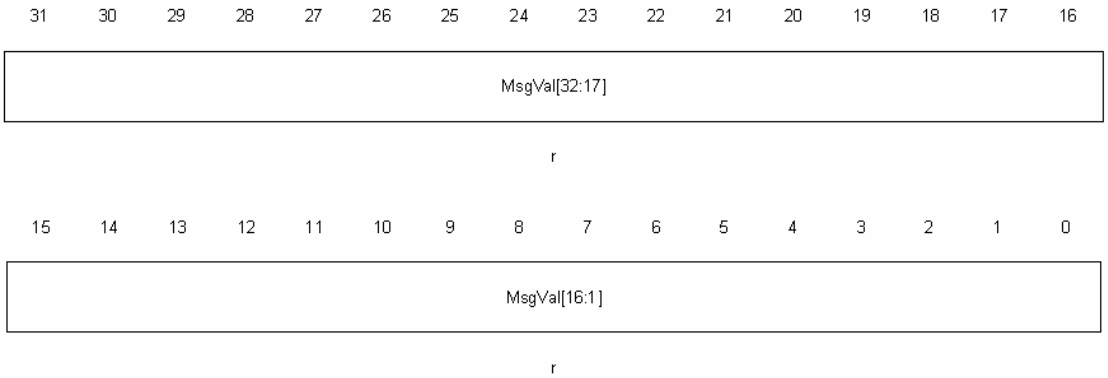
这些寄存器包含了32个报文对象的IntPnd位。通过读取IntPnd位，CPU可以检查哪一个报文对象引起了中断等待。一个特定报文对象的IntPnd位可以由应用软件通过IFn报文接口寄存器来置位/复位，也可以由报文处理器在接收到或成功发送一个帧之后来置位或复位。这将会影响到中断寄存器中IntId位的值。

位31:16　IntPnd32-17 （所有报文对象的）中断等待位
0：该报文对象不是中断源
1：该报文对象是一个中断的源

位15:0　IntPnd16-1 （所有报文对象的）中断等待位
0：该报文对象不是中断源
1：该报文对象是一个中断的源

9.6.4.5 报文有效寄存器1，2 (CAN_MVnR)

地址偏移：160h (CAN_MV1R), 164h (CAN_MV2R)
复位值：0000 0000h



这些寄存器保存了32个报文对象的MsgVal位。通过读取MsgVal位，应用软件可以检查哪一个报文对象是有效的。一个特定报文对象的MsgVal位可以由应用软件通过IFn报文接口寄存器来置位/复位。

位31:16　MsgVal32-17 （所有报文对象的）报文有效位
0：该报文对象被报文处理器忽略
1：该报文对象已经配置，应该由报文处理器处理。

位15:0　MsgVal16-1 （所有报文对象的）报文有效位
0：该报文对象被报文处理器忽略

1：该报文对象已经配置，应该由报文处理器处理。

9.7 寄存器映射

表36 CAN寄存器映射

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	CAN_CR	Reserved								Test	CCE	DAR	res	EIE	SIE	IE	Init
04h	CAN_SR	Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
08h	CAN_ERR	RP	REC6-0							TEC7-0							
0Ch	CAN_BTR	res	TSeg2			TSeg1				SJW		BRP					
10h	CAN_IDR	Intld15-8								Intld7-0							
14h	CAN_TESTR	Reserved								Rx	Tx1	Tx0	LBack	Silent	Basic	Reserved	
18h	CAN_BRPR	Reserved												BRPE			
20h	CAN_IF1_CRR	Busy	Reserved							Message Number							
24h	CAN_IF1_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/	Data A	Data B
28h	CAN_IF1_M1R	Msk15-0															
2Ch	CAN_IF1_M2R	MXtd	MDir	res	Msk28-16												
30h	CAN_IF1_A1R	ID15-0															
34h	CAN_IF1_A2R	MsgVal	Xtd	Dir	ID28-16												

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
38h	CAN_IF1_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
3Ch	CAN_IF1_DA1R	Data(1)								Data(0)							
40h	CAN_IF1_DA2R	Data(3)								Data(2)							
44h	CAN_IF1_DB1R	Data(5)								Data(4)							
48h	CAN_IF1_DB2R	Data(7)								Data(6)							
80h	CAN_IF2_CRR	Busy	Reserved									Message Number					
84h	CAN_IF2_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/	Data A	Data B
88h	CAN_IF2_M1R	Msk15-0															
8Ch	CAN_IF2_M2R	MXtd	MDir	res	Msk28-16												
90h	CAN_IF2_A1R	ID15-0															
94h	CAN_IF2_A2R	MsgVal	Xtd	Dir	ID28-16												
98h	CAN_IF2_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
9Ch	CAN_IF2_DA1R	Data(1)								Data(0)							

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A0h	CAN_IF2_DA2R	Data(3)								Data(2)							
A4h	CAN_IF2_DB1R	Data(5)								Data(4)							
A8h	CAN_IF2_DB2R	Data(7)								Data(6)							
100h	CAN_TxR1R	TxRqst16-1															
104h	CAN_TxR2R	TxRqst32-17															
120h	CAN_ND1R	NewDat16-1															
124h	CAN_ND2R	NewDat32-17															
140h	CAN_IP1R	IntPnd16-1															
144h	CAN_IP2R	IntPnd32-17															
160h	CAN_MV1R	MsgVal16-1															
164h	CAN_MV2R	MsgVal32-17															

注意 保留位读为’0’，但IFn Mask 2寄存器除外，该寄存器的保留位读为’1’。
基地址请见表2 “APB存储器影射”。

9.8 CAN 通信

9.8.1 管理报文对象

报文 RAM 中报文对象的配置不受芯片复位的影响（MsgVal, NewDat, IntPnd, 和 TxRqst 位除外）。在应用软件清除 CAN 控制寄存器中 Init 位操作之前，所有的报文对象必须由应用软件初始化，或者必须是“无效”（MSgVal=0）的，且位定时必须被配置。

报文对象的配置是通过对两个接口之一的屏蔽、仲裁和控制域进行编程来实现的。通过写入相应的 IFn 命令请求寄存器，IFn 报文缓冲器寄存器被载入报文 RAM 的被寻址报文对象中。

当清除 CAN 控制寄存器中的 Init 位时，CAN 内核的 CAN 协议控制器状态机和报文处理器状态机控制 C_CAN 内部的数据流。通过了接收过滤波的接收到的报文被存储在报文 RAM 中，等待传输请求的报文被载入到 CAN 内核的移位寄存器中，并通过 CAN 总线被发送。

应用软件读取接收到的报文，并通过 IFn 接口寄存器更新要发送的报文。根据配置，CPU 被特定的 CAN 报文或 CAN 出错事件中断。

9.8.2 报文处理器状态机

报文处理器控制在 CAN 内核 Rx/Tx 移位寄存器、报文 RAM 和 IFn 寄存器之间的数据传送。报

文处理器状态机 FSM 控制如下功能：

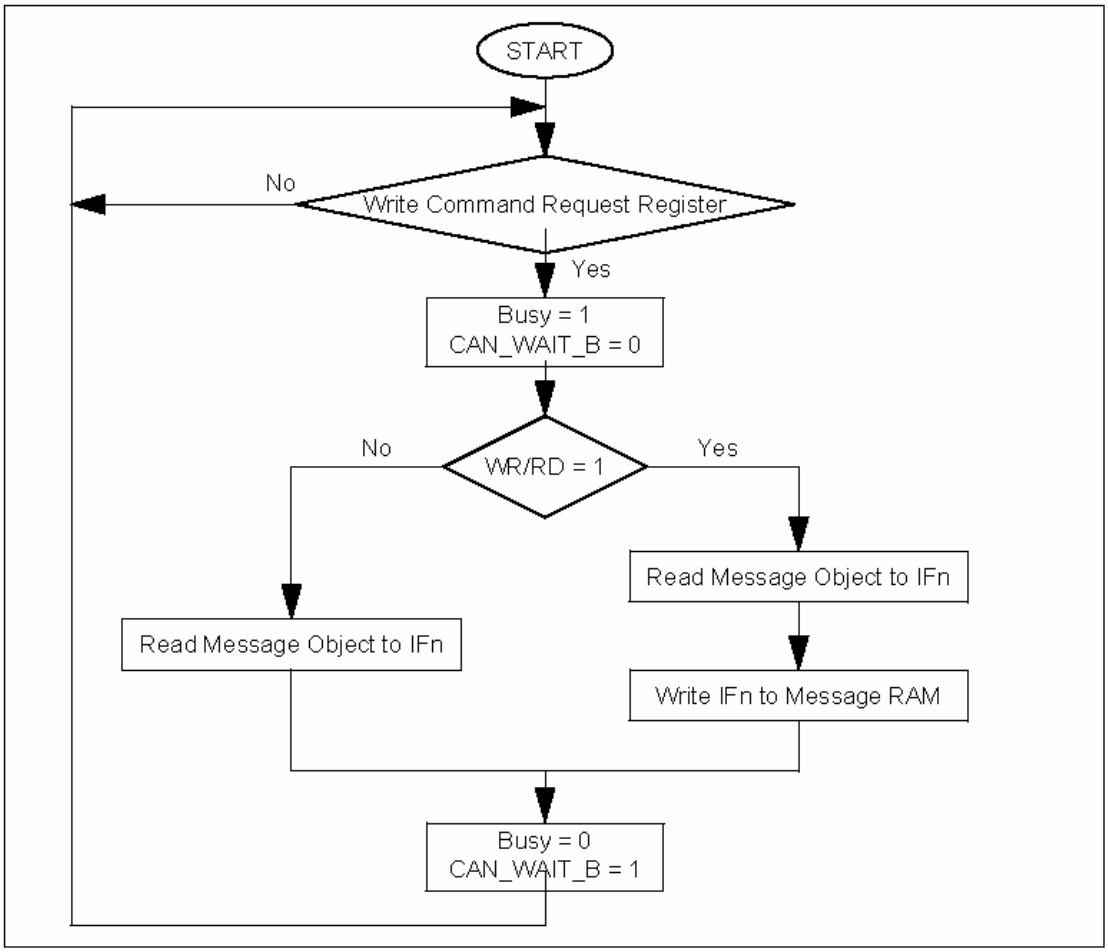
- 从 IFn 寄存器到报文 RAM 的数据传送；
- 从报文 RAM 到 IFn 寄存器的数据传送；
- 从移位寄存器到报文 RAM 的数据传送；
- 从报文 RAM 到移位寄存器的数据传送；
- 从移位寄存器到接收过滤单元的数据传送；
- 为匹配报文对象扫描报文 RAM；
- 处理 TxRqst 标志位；
- 处理中断

9.8.2.1 报文 RAM 的数据传送

当 CPU 初始化 IFn 寄存器和报文 RAM 间的数据传送时，报文处理器则设置相应的命令请求寄存器(CAN_IFn_CRR)的 Busy 位。当转换完成后，Busy 位再次被清除（见图 48）。

对应的命令掩码寄存器指定了是传送完整的报文对象，还是其中的某些部分。由于报文 RAM 的结构，不能对一个报文对象的某个字节或位写数据。必须总是将一个完整的报文对象写入报文 RAM。因此，从 IFn 寄存器到报文 RAM 的数据传送需要一个读-修改-写周期。首先，报文对象的那些不变的部分会从报文 RAM 中被读出，然后报文缓冲寄存器中的全部内容将被写入报文对象。

图 48. IFn 寄存器与报文 RAM 间的数据传送



完成报文对象的局部写入操作后，没有在命令掩码寄存器中选中的报文缓冲寄存器将设置选中的报文对象的实际内容。

完成报文对象的局部读出操作后，没有在命令掩码寄存器中选中的报文缓冲寄存器的内容不变。

9.8.2.2 报文发送

如果 CAN 内核中的移位寄存器已经准备好装载数据，并且在 IFn 寄存器和报文 RAM 间没有数据传送，那么将会检查报文有效寄存器中的 MsgVal 位和发送请求寄存器中的 TxRqst 位。具有最高优先级等待发送请求的有效报文对象由报文处理器载入移位寄存器，开始发送。这时报文对象的 NewDat 为会被清除。

在一次发送成功完成之后，如果自发送开始后没有新数据新数据写入报文对象(NewDat = '0')，那么报文控制寄存器(CAN_IFn_MCR)中的 TxRqst 位会被清除。如果报文控制寄存器的 TxIE 位被置位，中断标识寄存器(CAN_IDR)的 IntPnd 位也会在一次成功发送后被置位。如果 C_CAN 失去仲裁，或是在发送中过程中发生了错误，那么报文会在 CAN 总线再次空闲时被重新发送。同时，如果一个更高优先级报文的发送被请求，那么报文会根据它们的优先级顺序进行发送。

9.8.2.3 接收报文的接收过滤

当一个输入报文的仲裁和控制域(标识符 + IDE + RTR + DLC)被完全移位到 CAN 核的 Rx/Tx 移位寄存器中时，报文处理器 FSM 开始扫描报文 RAM，寻找匹配的有效报文对象。

要在报文 RAM 中扫描一个匹配的报文对象，接收过滤单元从 CAN 内核的移位寄存器加载仲裁位。报文对象 1 的仲裁和掩码位(包括 MsgVal, UMask, NewDat,和 EoB)接着被载入到接收过滤单元，并与来自移位寄存器的仲裁域进行比较。这个过程对接下来的报文对象一直重复进行，直到找到一个匹配报文对象，或是到达了报文 RAM 的末端。

如果匹配发生，扫描停止，报文处理器 FSM 将根据接收到的数据帧的类型（数据帧或远程帧）做下一步处理。

数据帧的接收

报文处理器 FSM 将从 CAN 内核移位寄存器收到的报文存储到报文 RAM 内对应的报文对象中。不只数据字节，还有所有仲裁位及数据长度码都被存入相应的报文对象。这样做是为了保持数据字节与标识符相关联，即使是使用了仲裁屏蔽寄存器。

NewDat 位被置位，表示（还没有被 CPU 看到的）新数据被接收。应用软件应该在读取报文对象以后清除设置 NewDat 位。如果在接收的时候该位已经被置位，那么 MsgLst 会被置位，表示以前的数据（假定还没有被 CPU 看到）已丢失。如果 RxIE 被置位，则 IntPnd 置位，使得中断寄存器指向该报文对象。

报文对象的 TxRqst 位被清除是为了防止在请求的数据帧刚被接收的情况下，还发送远程帧。

远帧帧的接收

当接收了一个远程帧时，要考虑匹配报文对象的三种不同的组合：

1) Dir = '1'(方向=发送), RmtEn = '1', UMask = '1' 或 '0'

收到一个匹配远程帧时，该报文对象的 TxRqst 位会被置位，其它位保持不变。

2) Dir = '1' (方向=发送), RmtEn = '0', UMask = '0'

收到一个匹配远程帧时，该报文对象的 TxRqst 位保持不变，该远程帧被忽略。

3) Dir = '1' (方向=发送), RmtEn = '0', UMask = '1'

收到一个匹配远程帧时，该报文对象的 TxRqst 位会被清除。来自移位寄存器的仲裁和控制域(标识符 + IDE + RTR + DLC)会被存储到报文 RAM 的报文对象中，且该报文对象的 NewDat 位被置位。报文对象的数据域保持不变，该远程帧会象被类似接收数据帧一般被处理。

9.8.2.4 接收/发送优先级

报文对象的接收/发送优先级依赖于其报文编号。报文对象 1 的优先级最高，而报文对象 32 的优先级最低。如果有一个以上的发送请求发生时，会根据它们所对应报文对象的优先级顺序进行服务。

9.8.3 设置一个发送对象

图 37 说明应如何初始化一个发送对象。

图 37 发送对象的初始化

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	1	0	0	0	appl.	0	appl.	0

仲裁寄存器的值(ID28-0 和 Xtd 位)由应用程序提供。这些值定义了标识符和输出报文的类型。如果使用 11bit 位（“标准帧”）的标识符，那么在 ID28-ID18 位编程，ID17-ID0 被忽略。

如果设置了 TxIE 位，那么在该报文对象成功发送后，会设置 IntPnd 位。

如果设置了 RmtEn 位，一个匹配的远程帧接收会引起 TxRqst 位置位；该远程帧会自动被一个数据帧响应。

数据寄存器的值(DLC3-0, Data0-7)也由应用程序提供，在数据有效前 TxRqst 和 RmtEn 位不能被置位。

可以利用掩码寄存器(Msk28-0, UMask, MXtd,和 Mdir 位) （让 UMask=’1’）来允许一组具有相似标识符的远程帧置位 TxRqst 位。这时 Dir 位应被屏蔽。

9.8.4 更新一个发送对象

CPU 可以随时通过 LFn 接口寄存器更新一个发送对象的数据字节，在更新前不必置位 MsgVal 和 TxRqst 位。即使是只更新一部分数据字节，相关的 IFn Data A 和 IFn Data B 寄存器中的所有 4 个字节数据都要在寄存器中的数据被传送到报文对象之前被设为有效内容。为此，或者用 CPU 将所有四位数据写入 IFn 数据寄存器，或者在 CPU 写入部分新数据之前，将完整报文对象传送到 IFn 数据寄存器。

当只更新（8 个）数据字节时，首先将 0x0087 写入请求掩码寄存器，然后将报文对象的编号写入命令请求寄存器，升级数据字节同时设置 TxRqst 位。

为了防止在发送的结束时清除 TxRqst 位，而造成在数据更新时该发送已经在进行中，NewDat 位要和 TxRqst 位一起置位。细节请见 9.8.2.2 “报文发送”。

当 NewDat 位和 TxRqst 位一起置位时，一旦新的发送开始的时候，NewDat 会被清除。

9.8.5 设置一个接收对象

表 38 说明了怎样初始化一个接收对象。

表 38　接收对象的初始化

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	0	0	0	appl.	0	0	0	0

仲裁寄存器的值(ID28-0 和 Xtd 位)由应用程序提供。这些定义了标识符和可接收的输入报文的类型。如果使用一个 11 位（“标准帧”）的标识符，那么应对 ID28-ID18 位编程，ID17-ID0 可以忽略。当接收到一个具有 11 位标识符的数据帧时，ID17-ID0 被置 0。

如果设置了 RxIE 位，在接收的数据帧被接受并存储在报文对象时，IntPnd 位会被置位。

数据长度码(DLC3-0)由应用程序提供。当报文处理器存储一个数据帧在报文对象时，将存储收到的数据长度码和 8 字节数据。如果数据码长度小于 8 ，那么报文对象的余下字节将被不确定的值覆盖。

掩码寄存器(Msk28-0, UMask, MXtd,和 Mdir 位) 可以用来 （让 UMask=’1’）允许一组具有相似标识符的数据帧被接受。在典型的应用中 Dir 位不应被屏蔽。

9.8.6 处理接收报文

CPU 可以随时通过 IFn 接口寄存器读取一个收到的报文。数据的一致性由报文处理器状态机来保证。

通常，CPU 会先将 0x007F 写入命令掩码寄存器，然后将报文对象的编号写入命令请求寄存器。这样可以将整个接收报文从报文 RAM 传送到报文缓冲寄存器。此外，报文 RAM 中（不是报文缓冲器中）的 NewDat 和 IntPnd 位会被清零。

如果报文对象采用了接收过滤的掩码，那么仲裁位会显示接收了哪个匹配报文。

NewDat 的实际值表示自上一次本报文对象被读取后，是否收到了新的报文。MsgLst 的实际值表示了自上一次本报文对象被读取后，是否收到了一个以上的新报文。MsgLst 位不会被自动清除。

借助于远程帧，CPU 可以请求另一个 CAN 节点为一个接收对象提供新数据。设置 TxRqst 位会引起发送一个具有接收对象的标识符的远程帧。该远程帧触发另一个 CAN 节点开始发送与之匹配的数据帧。如果在远程帧被发出之前收到了一个匹配的数据帧，那么 TxRqst 位会自动清除。

9.8.7 设置 FIFO 缓冲区

除了 EoB 位以外，设置属于一个 FIFO 缓冲区的接收对象与设置一个单独的接收对象方式相同。请见 9.8.5: “设置一个接收对象”。

为了将两个或更多个报文对象连接成一个 FIFO 缓冲区，这些报文对象的标识符和屏蔽位（如果使用了）必须编程为匹配数值。由于报文对象的隐含优先级，具有最低编号的报文对象会成为缓冲区的第一个对象。除了最后一个报文对象外，FIFO 缓冲区中所有报文对象的 EoB 必须被设为 0。一个 FIFO 缓冲区的最后一个报文对象的 EoB 位被设为 1，将其配置为块的结尾。

9.8.8 接收带有 FIFO 缓冲区的报文

具有与 FIFO 缓冲匹配的标识符的报文被接收后，被存入该 FIFO 缓冲区的报文对象中，该缓冲区的开头是具有最低报文标号的报文对象。

当一个报文被存入 FIFO 缓冲区中的一个报文对象时，该报文对象的 NewDat 位被置位。通过在 EoB 为 0 时置位 NewDat 位，报文对象被锁定，以便报文处理器后面的写访问，直到应用软件将 NewDat 位再写为 0。

多个报文被存入 FIFO 缓冲区，一直到存入该 FIFO 缓冲区的最后一个报文对象。如果前面没有任何报文对象被清除 NewDat 位而得到释放，随后所有送到这个 FIFO 缓冲区的报文都将被写入该 FIFO 缓冲区的最后一个报文对象中，因而就会覆盖掉以前的报文。

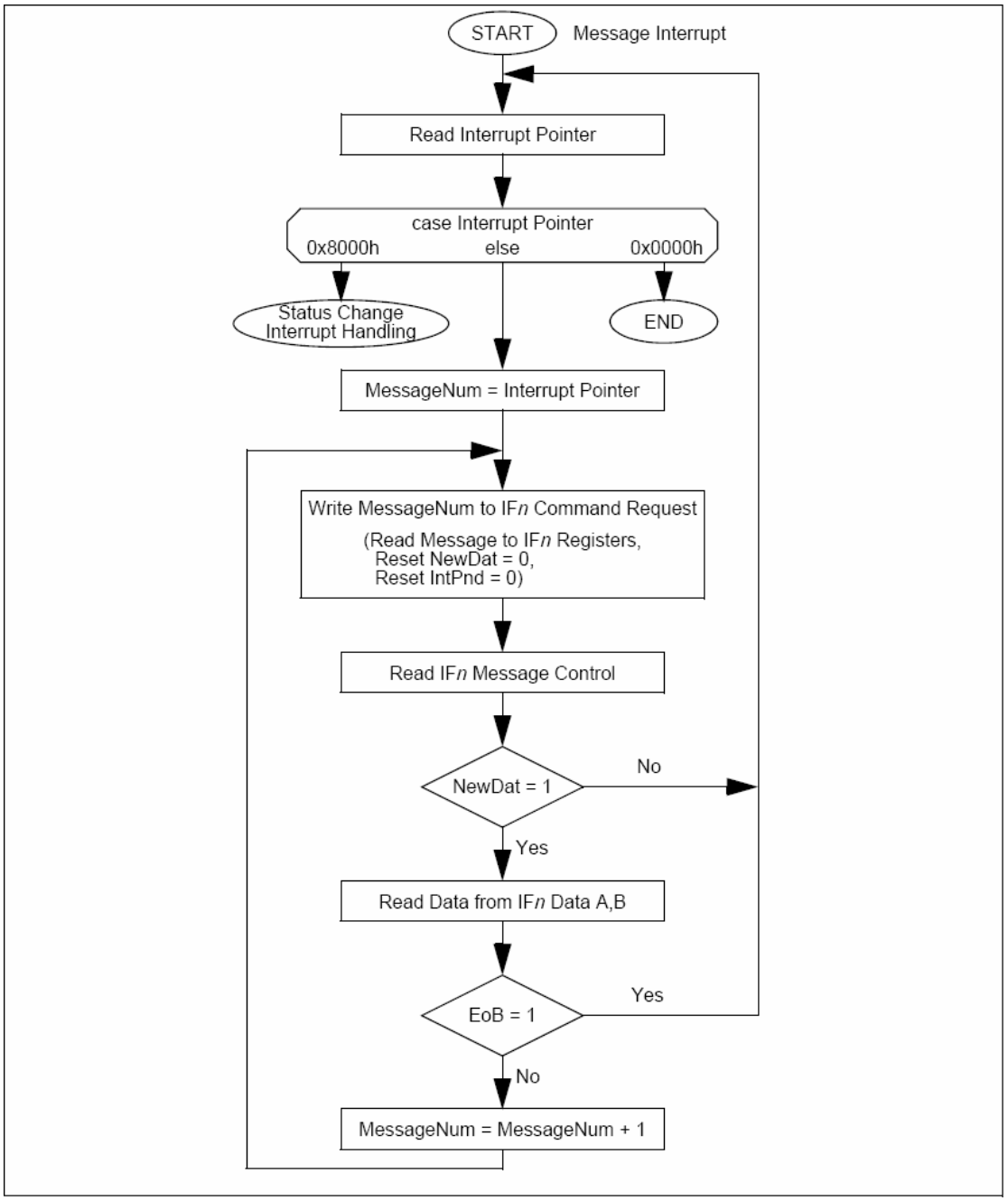
9.8.8.1 从 FIFO 缓冲器读取数据

当CPU通过将一个报文对象的编号写入IFn命令请求寄存器，从而将报文对象的内容传送到IFn报文缓冲寄存器时，对相应的命令掩码寄存器的编程方式应该使得 NewDat 和 IntPnd 位复位为 0 (TxRqst/NewDat = ‘1’ 且 ClrIntPnd = ‘1’)。在报文控制寄存器中，这些位的值总是反映了对这些位复位之前的状态。

为了保证 FIFO 缓冲区的正确功能，CPU 需要从具有最低编号的 FIFO 对象开始读取报文对象。

图 49 说明 CPU 如何处理一组连接成 FIFO 缓冲区的报文对象。

图 49 CPU 处理一个 FIFO 缓冲区



9.8.9 处理中断

如果有多个中断都在等待，那么 CAN 中断寄存器会指定具有最高优先级的中断，而忽略它们发生的时间顺序。一个中断在应用软件清除它之前一直处于等待状态。

状态中断具有最高的优先级。在所有的报文中断中，报文对象的中断优先级随着报文标号的增大而降低。

通过清除一个报文对象的 IntPnd 位可清除报文中断。读取状态寄存器可以清除状态中断。

在中断寄存器中的中断标识 IntId 指出了中断产生的原因。当没有中断等待时，寄存器的值为零。当中断寄存器的值为非零时，就有中断在等待，此时如果 IE 被置位，到 EIC 的 IRQ 中断信号会被激活。该中断一直处于有效状态，直到中断寄存器返回到零（引起中断的原因被清除了）或是 IE 被复位。

值 0x8000 表示一个由于 CAN 内核更新（不一定改变）了状态寄存器（出错中断或状态中断）而产生的中断。该中断具有最高的优先级。CPU 可以更新状态位 RxOk，TxOk 和 LEC，但是 CPU 对状态寄存器的写操作从不会产生或是清除一个中断。

其它的值说明中断源是一个报文对象。IntId 指向正在等待的具有最高优先级的报文中断。

CPU 可以控制状态寄存器的改变是否会引起一个中断(CAN 控制寄存器中的 EIE 和 SIE 位)，以及当中断寄存器的值不是 0 时中断线是否会被激活（CAN 控制寄存器中的 IE 位）。即使是 IE 被复位，中断寄存器仍然会被更新。

CPU 有两种可能的方法来追踪报文中断源。首先可以追踪中断寄存器中的 IntId，其次可以查询中断等待寄存器（见“中断等待寄存器 1，2(CAN_IPnR)”）。

读取引起中断报文的中断服务程序，可以在读取报文的同时清除报文对象的 IntPnd 位（在命令掩码寄存器中的 ClrIntPnd 位）。当 IntPnd 清零时，中断寄存器将指向下一个等待中断的报文对象。

9.8.10 设置位定时

在 CAN 位定时配置过程中，即使是一个小错误不导致立即出现故障，但 CAN 网络的性能会明显地下降。

在很多情况下，CAN 位同步过程将会弥补一个 CAN 位定时设置错误，达到只会偶尔产生一个错误帧的水平。然而，在进行仲裁的情况下，如果有两个或更多 CAN 节点同时尝试发送一个帧，那么一个错位的采样点会导致其中一个发送器变为出错认可。

对这种零星错误的分析要求仔细了解 CAN 总线上 CAN 节点内部和 CAN 节点间的位同步机制。

9.8.10.1 位时间和位速率

CAN 支持的位速率范围为 1kBit/s 到 1000kBit/s。CAN 网络中的每一个成员都有自己的时钟发生器，通常是一个晶振。位时间（即位速率的倒数）的定时参数可以对每个 CAN 节点单独地配置，这样即使各 CAN 节点的振荡器周期（或 f_{OSC}）可能不同，仍可以建立一个相同的位速率。

这些振荡器的频率并不是绝对稳定的，微小的变化可能由温度或电压的改变引起，或是由元件质量变坏引起。只要变化幅度保持在给定的振荡器容差范围（df）内，那么 CAN 节点就能通过重新同步到位流的机制对不同位速率进行补偿。

根据 CAN 规范，位时间被分割为 4 段：同步段，传播时间段，相位缓冲段 1 和相位缓冲段 2。每个段都由规定的、可编程数目的时间份额（time quantum）组成（见表 39）。时间份额的长度 tq 是构成位时间的基本时间单元，由 CAN 控制器的系统时钟 f_{APB} 和位定时寄存器（CAN_BTR）的 BRP 位确定: $t_q=BRP/f_{APB}$。

同步段 sync_seg 是位时间的一部分，在这部分中会有 CAN 总线电平的变化沿发生。发生在

sync_seg 之外的一个沿与 sync_seg 之间的距离，称为该沿的相位误差。传播时间段 Prop_seg 是用来补偿 CAN 网络内部的物理延迟时间。相位缓冲段 Phase_seg1 和 Phase_seg2 围绕着采样点。（重）同步跳转宽度（SJW）定义了重同步可以将采样点在相位缓冲段定义的限定范围内移动多远，以补偿边沿的相位误差。

图 50 位定时

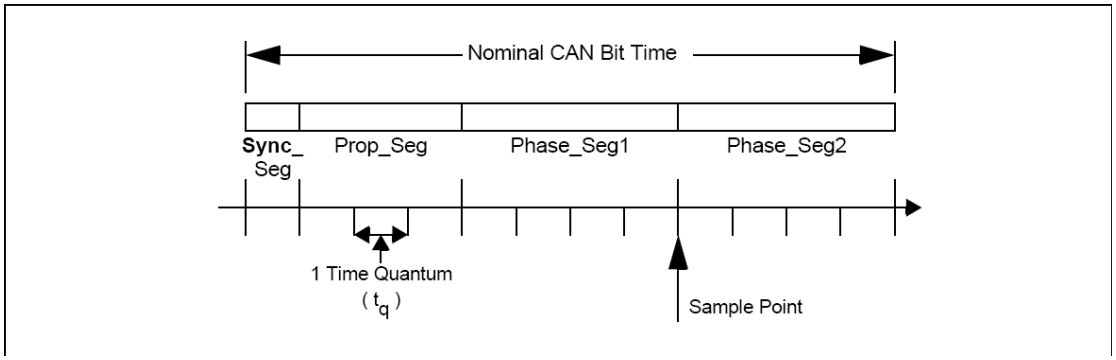


表 39 CAN 位定时参数

Parameter	Range	Remark
BRP	[1 .. 32]	defines the length of the time quantum t_q
Sync_Seg	1 t_q	fixed length, synchronization of bus input to system clock
Prop_Seg	[1.. 8] t_q	compensates for the physical delay times
Phase_Seg1	[1..8] t_q	may be lengthened temporarily by synchronization
Phase_Seg2	[1.. 8] t_q	may be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	may not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

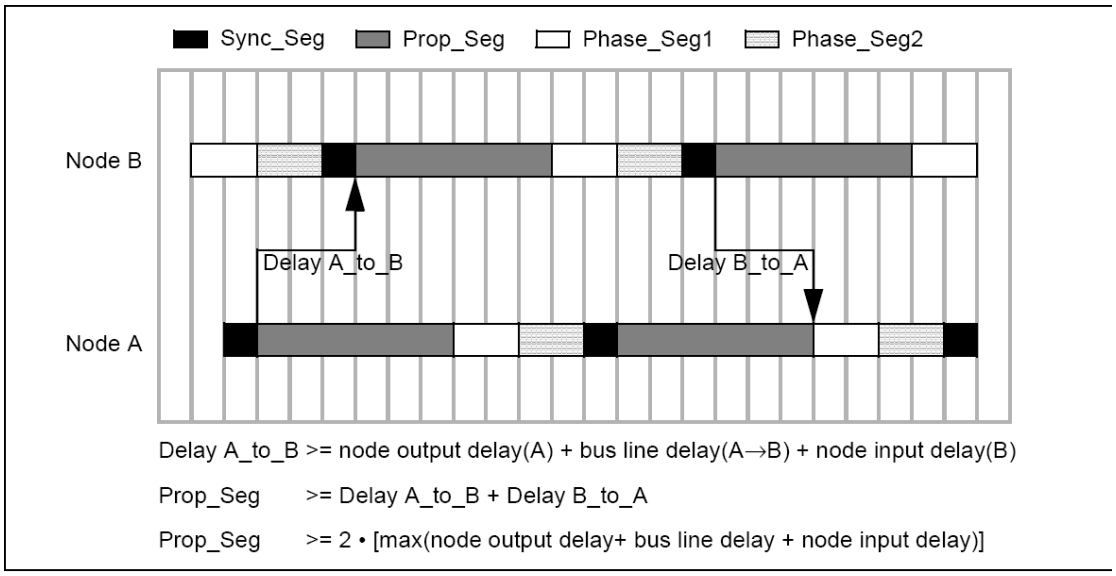
一个给定位速率可以用不同的位时间结构来实现，但是为了 CAN 网络的功能正常，必须要考虑物理延迟时间和振荡器容差范围。

9.8.10.2 传播时间段

位时间的这个部分用于补偿网络内部的物理延迟时间。这些延迟时间由总线上的信号传播时间和 CAN 节点的内部延迟时间组成。

任何与 CAN 总线上的位流同步的 CAN 节点，都会与该位流的发送器之间有相位差，这是由两个节点间的信号传播时间引起的。CAN 协议使用非破坏性按位仲裁和由 CAN 报文接收机提供显性认可位的方法，这就要求 CAN 节点在发送位流的同时，必须能够接收由同步于该位流的其它 CAN 节点发送的显性位。图 51 中的例子说明相位漂移和两个节点间的传播时间。

图 51 传播时间段



在这个例子中，节点 A 和节点 B 都是发送器，参与 CAN 总线仲裁。节点 A 发送了一个帧起始位（Start of Frame），它比节点 B 提早了一个位时间，因此节点 B 将自己同步到接收到的从隐性到显性的变化沿。因为节点 B 在其发送后接收了这个边的延迟（A 到 B），故 B 的定时段相对于 A 进行了移动。节点 B 发送一个较高优先级的标识符，因此，在某个特定的标识符位上它将赢得仲裁，在该位处它传送一个显性位，而节点 A 发送了一个隐性位。由节点 B 发送的显性位在经延迟（B 到 A）后到达节点 A。

因为存在振荡器容差，节点 A 采样点的实际位置不确定，可能处在其相位缓冲段定义范围内的任意位置。因此节点 B 发送的位必须在 Phase-Seg1 开始前到达节点 A。这个条件限制了 Prop-Seg 的长度。

如果由节点 B 发送的隐性到显性的跳变沿在 Phase-Seg1 开始后到达节点 A，有可能发生节点 A 采样到一个隐性位而不是一个显性位，这会导致一个位错误，用一个出错帧标志破坏掉当前的帧。

这种错误只有在两个节点都参与总线仲裁、两节点的振荡器分别处于容差的相反两端、且被很长的总线分开时才可能发生。这是一个在位定时结构中的一个较小的错误（Prop_Seg 太短）引起偶发总线错误的例子。

某些 CAN 硬件实现提供了可选择的 3 次采样模式，但 C-CAN 不提供这种模式。在这种模式中，CAN 总线输入信号通过一个数字低通滤波器，应用三个采样和一个多数逻辑去决定有效的比特值。这会导致额外 1 个 tq 的输入延迟，要求一个更长的 Prop-Seg。

9.8.10.3 相位缓冲段和同步

相位缓冲段(Phase_Seg1 和 Phase_Seg2)和同步跳转宽度（SJW）用于补偿振荡器容差。相位缓冲段可以由同步过程来延长或缩短。

同步发生在从隐性到显性的边沿，其目的是控制边沿与采样点间的距离。

在每个时间份额对实际的总线电平采样，并将其与以前的采样点总线电平对比，可以检测到边沿。只有在前一个采样点得到隐性位，且在当前实际时间份额的总线电平为显性时才可以施行同步。

如果一个边沿发生在 Sunc-Seg 内则它是同步的，否则，该边沿与 Sunc-Seg 的结尾之间的距离就是边沿相位误差，误差的衡量以时间份额为单位。如果边沿发生在 Sunc-Seg 之前，那么相位误差是负的；否则是正的。

有两种同步类型：硬同步和重同步。

在帧起始时，总线会进行一次硬同步。只有当发生重同步时才在帧内进行。

- 硬同步

在一个硬同步以后，Sunc-Seg 的结尾会重起位时间，而不理会边沿相位误差。因此，硬同步促使边沿发生，这会使得硬同步处于重新启动的位时间的同步段之内。

- 位重同步

  重同步可导致位时间的延长或缩短，让采样点的位置相对于跳变沿发生移动。

  当引起重同步的边沿的相位误差是正的，那么相位 Phase-Seg1 会被延长。如果相位误差的幅度小于 SJW，则 Phase-Seg1 延长的幅度等于相位误差幅度；否则，延长的幅度等于 SJW。

  当引起重同步的边沿的相位误差是负的，那么 Phase-Seg2 被缩短。若相位误差幅度小于 SJW，Phase-Seg1 缩短的幅度等于相位误差幅度；否则，缩短的幅度等于 SJW。

当跳变沿的相位误差幅度小于或等于编程确定的 SJW 值，那么硬同步和重同步的结果是一样的。如果相位误差幅度大于 SJW，那么重同步不能完全补偿相位误差，仍然有一个误差（相位误差-SJW）存在。

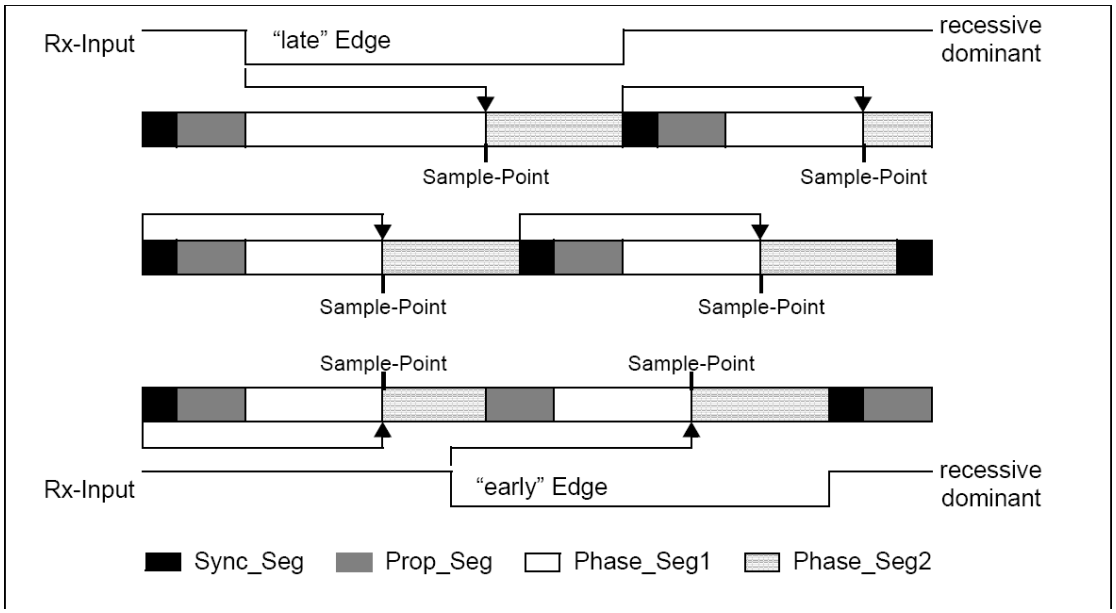
在两个采样点之间只能有一种同步发生。同步机制在跳变沿与采样点之间维持一个最小距离，提供时间让总线电平稳定并和滤除长度短于(Prop_seg+Phase_seg1)的噪声毛刺。

除了毛刺噪声以外，大多数同步是由仲裁引起的。所有的节点会“硬”同步于由首先开始发送的“领先”发送器发送的跳变沿，不过因为传播延迟时间的存在，这些节点不能达到理想的同步。“领先”的发送器不一定赢得仲裁，因此各接收器必须让他们自己再与“领先”的发送器同步，而这个发送器与先前“领先”的同步有差别。同样的情况也发生在确认域，这时，发送器和一些接收器必须要与在发送显性确认位时“夺得领先”的接收器取得同步。

在仲裁结束后发生的同步由振荡器容差引起。这时，在两次同步之间发送器与接收器振荡器时钟周期的差别累积起来（最多达 10 位）。这个累计的差额不能超过 SJW，由此限定了振荡器的容差范围。

图 52 中的例子说明相位缓冲段是如何用于相位误差的补偿。有三个关于两个连续位定时的图。上面的是图说明同步在“迟到”跳变沿，下面的图说明同步在“早到”跳变沿，中间的图是没有同步发生的参考。

图 52　对“迟到”和“早到”边沿的同步



第一个例子中，从隐性到显性的跳变沿发生在传播段 Prop_Seg 结尾处。该边沿“迟到”是因为它发生在在同步段 Sync_Seg 之后。作为对这样一个迟到沿的反应，相位段 Phase_Seg1 被延长，使得从沿到采样点的距离就等同于假若没有发生跳变沿时从同步段 Sync_Seg 到采样点间的距离。这个“迟到”边沿的相位误差小于 SJW，因此能够被完全补偿，结果在该位（一个显性位时间长度）

结束时从显性到隐性的跳变沿发生在同步段 Sync_Seg 内。

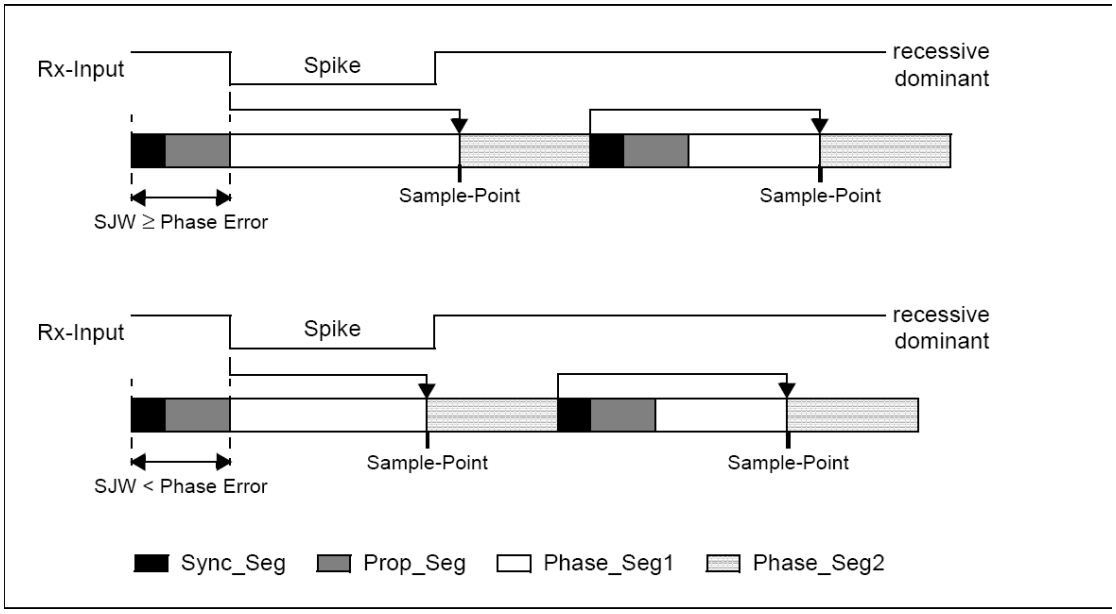
第二个例子中，一个从隐性到显性的跳变沿发生在 Phase_Seg2 段内。该边沿“早到”是因为它发生在在同步段 Sync_Seg 之前。作为对这样一个早到沿的反应，相位段 Phase_Seg2 被缩短，同时同步段 Sync_Seg 被忽略。使得从该边沿到采样点的距离就等同于假若没有发生跳变沿时从同步段 Sync_Seg 到采样点间的距离。像上例一样，这个“早到”边沿的相位误差幅度小于 SJW，所以能够被完全补偿。

相位缓冲段的长度只是被暂时延长或缩短，在下一个位时间内，段长度恢复到它们原被编程的长度。

以上的例子中，位定时是从 CAN 状态机的角度定义的，其中位时间在采样点处开始和结束。这个状态机在与一个“早到”沿同步时忽略了同步段 Sync_Seg，因为它不能根据“早到”沿将 Phase_Seg2 内发生跳变沿处的时间份额重新定义为同步段 Sync_Seg。

图 53 中的例子显示出短的显性噪声毛刺是怎么样被同步机制滤除的。这两个例子中的毛刺在传播段 Prop_Seg 的结尾处开始，长度为传播段+相位段 1（Prop_Seg + Phase_Seg1）。

图 53 短显性毛刺的滤除



第一个例子中，同步跳转宽度大于或等于从隐性到显性的毛刺边沿的相位误差。因此采样点移动到毛刺结尾之后，采样到隐性总线电平。

第二个例子中，SJW 短于相位误差，因此采样点被移动的不够远，显性的毛刺被当作实际的总线电平进行了采样。

9.8.10.4 振荡器容差范围

CAN 协议的版本从 1.1 升级到 1.2 后振荡器容差范围增加了(版本 1.0 从来没有用芯片实现过)。从显性到隐性的跳变沿处进行同步的选项已经废弃了，现在只考虑从隐性到显性的跳变处进行同步。曾经实现了 1.1 协议版本的 CAN 控制器只有 Intel82526 和 Philips82C200，这两种都已经被后继的产品取代了。协议升级到版本 2.0（A 和 B）对振荡器容差没有影响。

一个振荡器频率 f_{osc} 在围绕标称频率 f_{nom} 的振荡器容差范围 df 为：

$(1-df) \cdot f_{nom} \leq f_{osc} \leq (1+df) \cdot f_{nom}$

它依赖于 Phase-seg1, Phase-seg2, SJW 和位时间的比例。最大容差 df 受以下两个条件限制：

$$\text{I:}\quad df \le \frac{min(\text{Phase\_Seg1}, \text{Phase\_Seg2})}{2 \cdot (13 \cdot \text{bit\_time} - \text{Phase\_Seg2})}$$

$$\text{II:}\quad df \le \frac{\text{SJW}}{20 \cdot \text{bit\_time}}$$

必须要考虑到，SJW 的取值不能长于两个相位缓冲段中较小者，传播时间段限制了可能用于相位缓冲段的那部分位时间。

设置组合 Prop_seg1=1，Phase_Seg1=Phase_Seg2=SJW=4 可得到最大可能的振荡器容差 1.58%。本组合中传播时间段只占位时间的 10%，不适用于短的位时间；它可用于在长 40 米的总线上达到最高 125Kbit/s（位时间=8μs）的位速率。

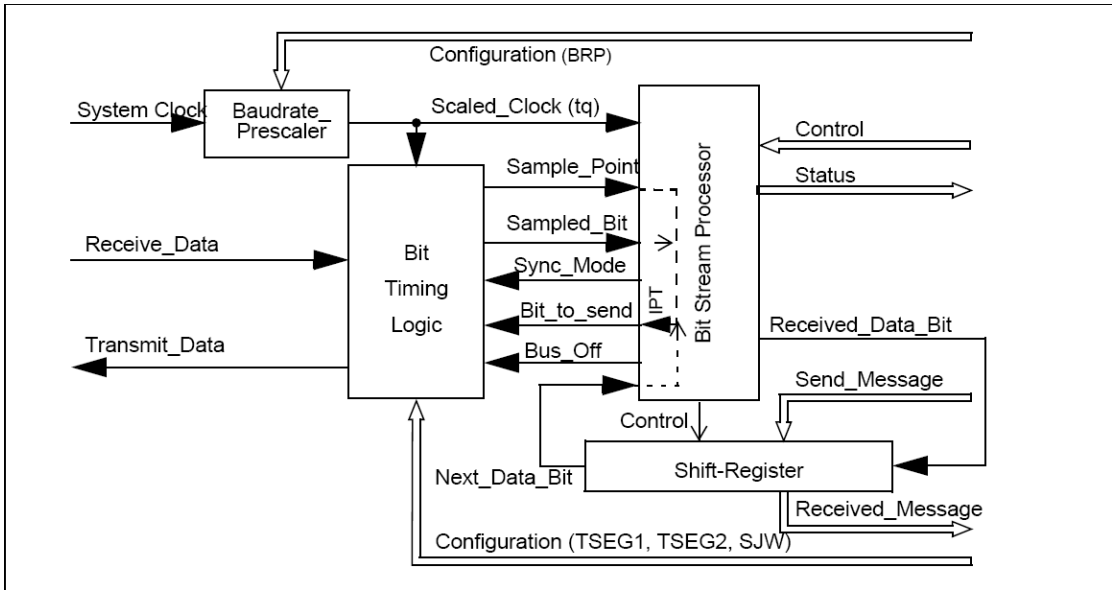
9.8.10.5 配置 CAN 协议控制器

在 C_CAN 和多数 CAN 实现中，位时间配置是通过对两个寄存器字节的编程。Prop_Seg 与 Phase_Seg1 的和（作为 TSEG1）同 Phase_Seg2 组合为一个字节，SJW 和 BRP 组合在另一个字节中（见图 54）。

在这些位定时寄存器中，四个部分 TSEG1，TSEG2，SJW 和 BRP 需要被设定为一个数值，这个值比其实际起作用的值小 1。因此，编程值范围是[0…(n-1)]而不是[1…n]。例如，这样可以将 SJW(实际起作用范围[1…4])用两个比特表示。

因此位时间的长度（编程值）为[TSEG1+TSEG2+3]t_q，或 (作用值)为 [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_seg2] t_q。

图 54 CAN 内核中 CAN 协议控制器的结构



位定时寄存器中的数据是 CAN 协议控制器的设置输入。波特率预分频器（用 BRP 设置）定义了时间份额的长度——位时间的基本时间单元；位定时逻辑（由 TSEG1，TSEG2 和 SJW 设定）定义了位时间中的时间份额数。

位时间的处理，采样点位置的计算，以及偶发的同步都受控于位定时逻辑（BTL）状态机，该状态机在每个时间份额被计算一次。其它的 CAN 协议控制器，位流处理器（BSP）状态机每个位时间在采样点处被计算一次。

移位寄存器以串行方式发送报文，并行方式的接收报文。其装载和移位受控于位流处理器 BSP。

BSP 完成报文与帧的相互转换。它负责产生和丢弃用于封装数据的固定格式的位，插入和抽取

填充位，计算并检测 CRC 码，完成错误管理，还决定要采用哪种类型的同步。BSP 在采样点计算，处理采样到的总线输入位。计算采样点后要发送的下一个比特（例如数据位、CRC 位、填充位、出错标志、或空闲）所需的时间称为信息处理时间（IPT）。

IPT 与应用有关，但不能长于 $2t_q$，对 C_CAN 来说 IPT 是 $0t_q$。IPT 的长度是 Phase_Seg2 段长度取值的下限。在发生同步的情况下，Phase_Seg2 可能被缩短为小于 IPT 的值，但这不影响总线定时。

9.8.10.6 计算位定时参数

通常位定时结构的计算从要求的位速率或位时间开始。得到的位时间（1/位速率）必须是系统时钟周期的整数倍。

位时间可以包含 4 到 25 个时间份额，时间份额 t_q 的长度由波特率预分频因子 BRP 确定 $t_q=BRP/f_{sys}$ 。可能有几种组合都能得到要求的位时间，使得下面的步骤可以反复进行。

首先要确定的部分位时间是传播段 Prop_Seg。其长度取决于根据系统测量的延迟时间。对于可扩展的 CAN 总线系统，必须确定其最大的总线长度和最大的节点延迟。得到的 Prop_Seg 时间要转换为时间份额（舍入到 t_q 的最接近的整数倍）。

同步段 Sync_Seg 是固定的 $1t_q$ 长度，余下时间(位时间-Prop_Seg-1)tq 用于两个相位缓冲段。如果余下的 t_q 数量为偶数，两个相位缓冲段取相同长度 Phase_Seg2 = Phase_Seg1，否则 Phase_Seg2 = Phase_Seg1 + 1。

Phase_Seg2 的最小规定长度也要考虑进来。Phase_Seg2 不能比 CAN 控制器的 IPT 短，而 IPT 的范围是[0..2]tq，取决于控制器的实际硬件。

同步跳转宽度 SJW 的长度设置为其最大值，取 4 和 Phase_Seg1 中的较小值。

计算得到的设置结构所需要振荡器容差范围根据“9.8.10.4: 振荡器容差范围”中给出的公式来计算。

如果由多种可能的设置结构，应选择允许最大振荡器容差范围的那个结构。

不同系统时钟下的 CAN 节点要求不同的结构来适合同一个位速率。CAN 网络中的传播时间计算是基于最长延迟时间的节点，该计算对整个网络中只做一次。

CAN 系统的振荡容差范围受限于具有最低容差范围的节点。

根据计算结果，可能会要求减少总线长度或位速率，或提高振荡器频率的稳定性，目的是找到 CAN 位定时的满足协议的结构。

得到的配置结构被写入位时间寄存器：

```
(Phase_Seg2-1)&(Phase_Seg1+Prop_Seg-1)&
(SynchronisationJumpWidth-1)&(Prescaler-1)
```

高波特率下位定时计算的示例

本例中，APB_CLK 频率为 10MHz，BRP 为 0，位速率为 1MBit/s。

t_q	100	ns	= t_{APB_CLK}
delay of bus driver	50	ns	
delay of receiver circuit	30	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	600	ns	= 6 • t_q
t_{SJW}	100	ns	= 1 • t_q
t_{TSeg1}	700	ns	= t_{Prop} + t_{SJW}
t_{TSeg2}	200	ns	= Information Processing Time + 1 • t_q
$t_{Sync-Seg}$	100	ns	= 1 • t_q
bit time	1000	ns	= $t_{Sync-Seg}$ + t_{TSeg1} + t_{TSeg2}
tolerance for APB_CLK	0.39	%	= $\frac{min(PB1, PB2)}{2x(13x(\text{bit time} - \text{PB2}))}$
			= $\frac{0.1\mu s}{2x(13x(1\mu s - 0.2\mu s))}$

此例中，连接起来的位时间参数为$(2\text{-}1)_3$&$(7\text{-}1)_4$&$(1\text{-}1)_2$&$(1\text{-}1)_6$，为定时寄存器编程为=0x1600。

低波特率下位定时计算的示例

本例中 APB_CLK 频率为 2MHz，BRP 为 1，位速率为 100KBit/s。

t_q	1	µs	= 2 •t_{APB_CLK}
delay of bus driver	200	ns	
delay of receiver circuit	80	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	1	µs	= 1 • t_q
t_{SJW}	4	µs	= 4 • t_q
t_{TSeg1}	5	µs	= t_{Prop} + t_{SJW}
t_{TSeg2}	4	µs	= Information Processing Time + 3 • t_q
$t_{Sync-Seg}$	1	µs	= 1 • t_q
bit time	10	µs	= $t_{Sync-Seg}$ + t_{TSeg1} + t_{TSeg2}
tolerance for APB_CLK	1.58	%	= $\frac{min(PB1, PB2)}{2x(13x(\text{bit time} - \text{PB2}))}$
			= $\frac{4\mu s}{2x(13x(10\mu s - 4\mu s))}$

此例中，连接后位时间参数为$(4\text{-}1)_3$&$(5\text{-}1)_4$&$(4\text{-}1)_2$&$(2\text{-}1)_6$，位定时寄存器编程为=0x34C1。

10 I²C 接口模块（I2C）

I²C总线接口模块是微处理器和串行I²C总线之间的接口。它既提供多主功能又提供从功能，并控制所有针对I²C总线的串行化、协议、仲裁和定时。它支持快速I²C模式（400KHz）。

10.1 主要特征

- 并行总线/I²C协议转换器
- 多主机功能
- 7位/10位寻址
- 发送器/接收器标志
- 字节结束发送标志
- 传输问题检测
- 标准/快速I²C模式

I2C主机特性

- 时钟产生
- I²C总线忙标志
- 仲裁丢失标志
- 字节结束发送标志
- 发送器/接收器标志
- 开始位检测标志
- 开始位和结束位产生

I2C从机特性

- 开始位检测标志
- 结束位检测标志
- I²C总线忙标志
- 开始位或者结束位错误状态检测
- 可编程I²C地址检测
- 传输问题检测
- 字节结束发送标志
- 发送器/接收器标志

10.2 一般描述

除了接收和发送数据外，这个接口还可以对数据进行串并格式间的转换，转换使用中断或者查询握手方式。软件可以允许或禁止中断。接口通过数据引脚(SDA)和时钟引脚(SCL)连接到I²C总线上，它可以连接到标准的I²C总线或者快速I²C总线上，由软件进行选择。

10.2.1 模式选择

I²C接口可以工作在以下4种模式：

- 从发送器/接收器

- 主发送器/接收器

缺省时接口工作在从模式。

当接口产生了开始（START）状态后，它自动地由从模式转换到主模式；当仲裁丢失或者产生停止(STOP)信号时，接口就从主模式转换到从模式，以实现允许多主机功能。

10.2.2 通信流程

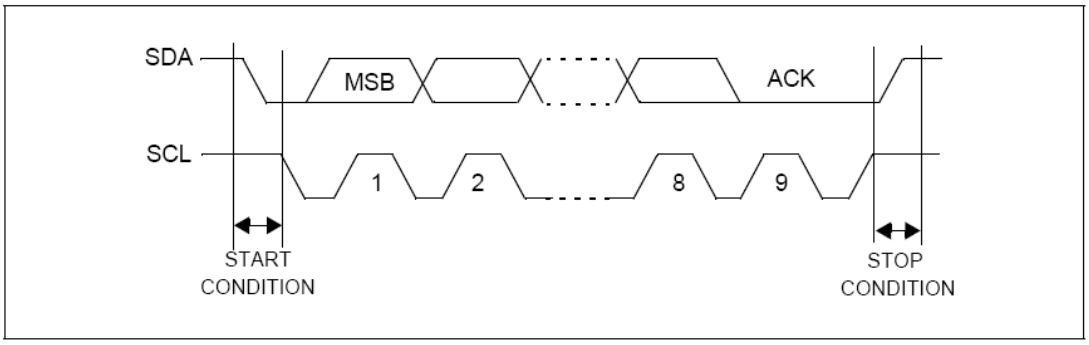
在主模式下，首先启动一个数据传输并产生时钟信号。串行数据传输总是始于一个开始状态，终止于停止状态。当选定主模式后，开始和停止状态就都在主模式下通过硬件产生。

在从模式下，接口能够识别自己的地址（7 或者10位）和公共呼叫（General Call）地址。对公共呼叫地址的检测可以通过软件开启或关闭。

数据和地址是以8位字节传输的，高位在前。开始状态信号后的第一个（或两个）字节包含了地址（7位模式下是一个字节是，10位模式下是两个字节）。地址总是在主模式下发送的。

第9个时钟脉冲跟在8个时钟周期的字节传输的后面，在第9个脉冲时间里，接收器必须向发送器发送一个确认位。参考图55。

图55 I^2C总线协议



确认信号的开与关可通过软件设定。

I^2C接口地址和/或公共呼叫地址可以通过软件进行选择。

I^2C接口的速度可以在标准（0-100KHz）和快速I^2C（100-400KHz）之间进行选择。

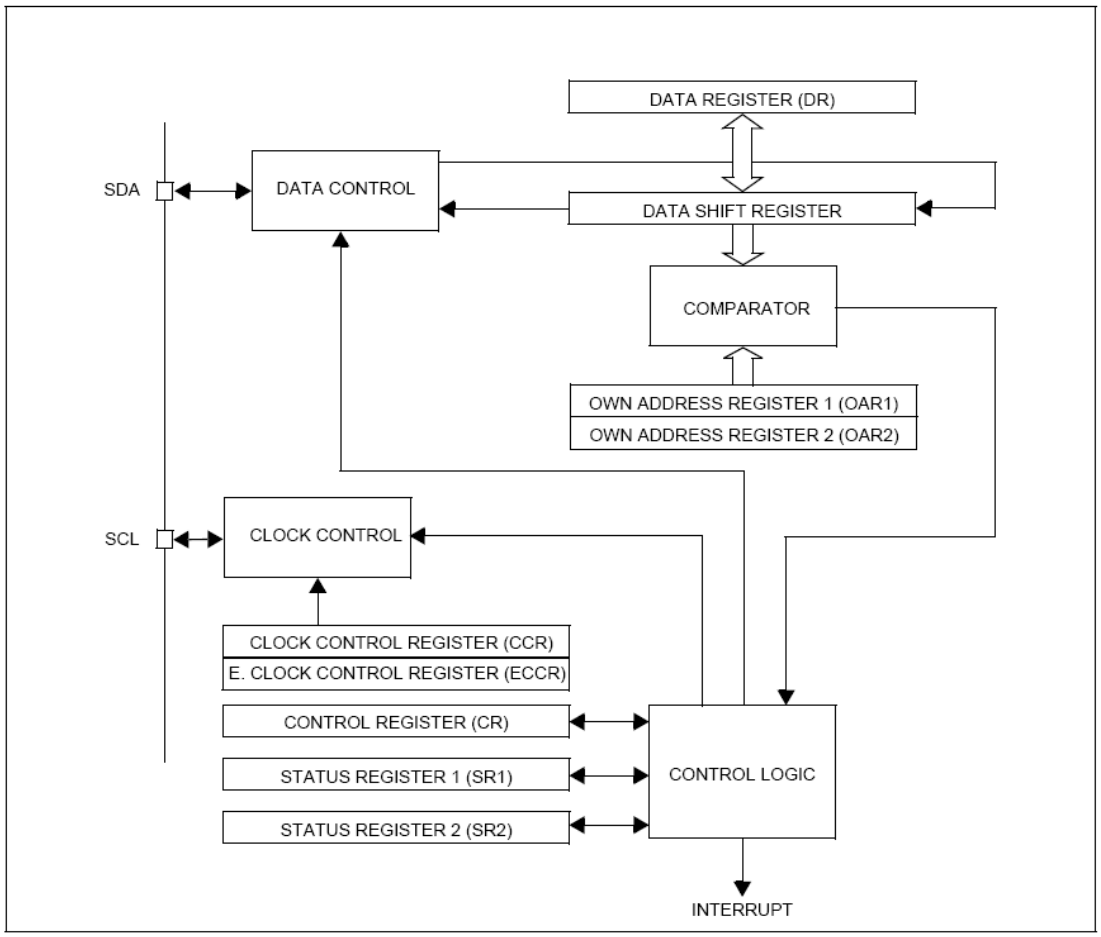
10.2.3 SDA/SCL信号线控制

发送器模式：接口在发送之前保持时钟线低电平以等待微控制器将字节写入数据寄存器。

接收器模式：接口在接收之后保持时钟线的低电平以等待微控制器将数据寄存器中的字节读出。

SCL信号频率（f_{SCL}）通过可编程时钟分频器来控制，这依赖于I^2C总线模式。

图56 I^2C接口框图



10.3 功能描述

参考10.5节关于I2Cn_CR,I2Cn_SR1和I2Cn_SR2位的定义。

除启动发送或接收过程外， I^2C接口工作在默认的从模式（M/SL位被清零）。

首先，接口频率必须利用I2Cn_OAR2寄存器的Fri位设定。

10.3.1 从模式

一旦开始状态被检测到，就从SDA线接收地址并且送到移位寄存器中；然后与接口地址或者公共呼叫（如果已通过软件选择）地址进行比较。

注意：在10位的地址模式中，比较包括头序列（11110xx0）和两个最高位地址。

头匹配（只在10位模式下）：如果ACK位被设定，接口产生一个确认脉冲。

地址不匹配： 接口忽略它，并且等待另一个开始状态。

地址匹配： 接口按序列产生：

- 确认脉冲（如果设定了ASK位）
- 如果设定了ITE位，以一个中断设定EVF和ADSL位

然后接口等待着对I2Cn_DR寄存器的读取，同时将SCL线保持在低电平。（见图57传输时序EV1）。

在7位模式下，下一步要读取I2Cn_DR寄存器，根据最低位（数据方向位）确定从机应该进入接收还是发送模式。

在10比特模式下，在接受地址序列以后，从模式总处于接收模式下。一旦收到重复的开始状态及其后面跟着的匹配地址位和低字节位组头序列（11110xx1），就进入发送模式。

10.3.1.1 从接收器

在获取了地址和读取了I2Cn_SR1寄存器后，从机通过内部移位寄存器将从SDA线收到的字节存入I2Cn_DR寄存器。在接收到每个字节后，接口按顺序产生：

- 确认脉冲（如果ASK位被置位）
- 将EVF和BTF置位，且如果ITE位被置位则产生一个中断。

然后接口将SCL线保持在低电平，等待对I2Cn_SR1的读取，以及其后对I2Cn_DR寄存器的读取（见图57传输时序EV2）。

10.3.1.2 从发送器

在接收了地址和读取了I2Cn_SR1寄存器后，从发送器通过内部移位寄存器将I2Cn_DR寄存器的字节发送到SDA线。

从发送器将SCL线保持在低电平，等待对I2Cn_SR1的读取，紧接是写入I2Cn_DR寄存器（见图57传输时序列EV3）。

当接收到确认信号时：

- 由硬件将EVF和BTF置位，且伴随一个中断（如果ITE位被置位）。

10.3.1.3 关闭从通信

在最后一个字节被传送以后，由主控器产生一个停止状态。接口检测这个信号，将EVF和STOPF位置位，并产生一个中断（如果ITE位被置位）。

然后接口等待对I2Cn_SR2寄存器的读取（见图57传输时序EV4）。

10.3.1.4 错误情况

- **BERR**：在对一个字节的传输过程中检测到一个停止状态或开始状态。在这种情况下，EVF和BERR位被置位，如果ITE位被置位则产生一个中断。

  如果是一个停止状态，那么接口就放弃数据，释放总线，等待另一个开始状态。

  如果是一个开始状态，那么接口就放弃数据，等待总线上的下一个从地址。
- **AF**：非确认位的检测。在这种情况下，EVF和AF位被置位，如果ITE位被置位则产生一个中断。

注意：在以上两种情况下，SCL没有被保持在低电平；但SDA线由于最后可能传送的"0"比特而保持低电平，那么就必须通过软件释放两条信号线。

10.3.1.5 如何释放SDA/SCL线

当BTF被置位时，设置然后清除STOP位。在传输当前字节后SDA/SCL线被释放。

10.3.2 主控模式

从缺省的从模式转换到主控模式，需要产生一个开始状态信号。

10.3.2.1 开始状态

在BUSY位被清零期间，设定开始（START）位会引起接口转换到主控模式（M/SL位被置位）

并产生一个开始状态。

一旦开始状态被发送了，EVF和SB位被硬件置位，如果ITE位被置位则产生一个中断。

然后，主控器将SCL线保持在低电平，等待对I2Cn_SR1的读取，紧接着在I2Cn_DR寄存器中写入从地址（见图57 传输时序EV5）。

10.3.2.2 从地址发送

然后通过内部移位寄存器将从地址发送到SDA线。

在7位寻址模式中，发送一个字节的地址。

在10位寻址模式中，发送包含头序列的第一个字节引起以下的事件：

- EVF位被硬件置位，如果ITE位被置位则产生一个中断。

然后，主控器保持SCL线低电平，等待对I2Cn_SR1的读取，紧接着对I2Cn_DR寄存器的写操作（见图57传输时序EV9）。

然后接口发送第二个地址字节。

在传输完成以后（如果ASK位被置位，还要得到来自从器件的确认信号后）：

- EVF位被硬件置位，如果ITE位被置位则伴随一个中断。

然后，主控器将SCL线保持在低电平，等待着对I2Cn_SR1的读取，紧接着是对I2Cn_DR寄存器的写信号（例如设置了PE位），（见图57 传输时序EV6）。

随后主控器必须进入接收或者发送模式。

注意： 在10位寻址模式下，为了将主控器转换到接收器模式，软件必须产生一个重复的开始状态，并且重新发送最低位被置位(11110xx1）的头序列。

10.3.2.3 主接收器

跟随着地址发送并且在I2Cn_SR1和I2Cn_CR寄存器被访问后，主接收器通过内部的移位寄存器将从SDA线接收到的字节存入I2Cn_DR寄存器。在每个字节后，接口按顺序产生：

- 确认脉冲（如果ASK位被置位）
- EVF和BTF被硬件置位，如果ITE位被置位则产生一个中断。

然后，接口保持SCL线在低电平，等待对SR1寄存器的读取，紧接着是对I2Cn_DR寄存器的读取（见图57 传输时序EV7）。

要关闭通信：在读取I2Cn_DR寄存器的最后一个字节之前，置位停止位（STOP）以产生停止状态，接口自动地返回到从模式（M/SL位被清零）。

注意：为了在最后一个接收到的数据字节以后产生一个非确认脉冲，ASK位必须在刚刚读取倒数第二个字节前被清零。

10.3.2.4 主发送器

紧跟在地址发送之后，并且在读取了I2Cn_SR1寄存器之后，主控器通过内部的移位寄存器发送I2Cn_DR寄存器中的字节到SDA线。

主控器将SCL线保持在低电平，等待对I2Cn_SR1寄存器的读取及其后对I2Cn_DR寄存器的写操作。（见图57 传输时序EV8）

当收到了确认比特，接口就置位EVF和BTF位，如果ITE位被置位则产生中断。

要关闭通信：在将最后一个字节写入I2Cn_DR寄存器之后，置位停止位（STOP）以产生停止状态。接口自动地返回到从模式（M/SL位被清零）

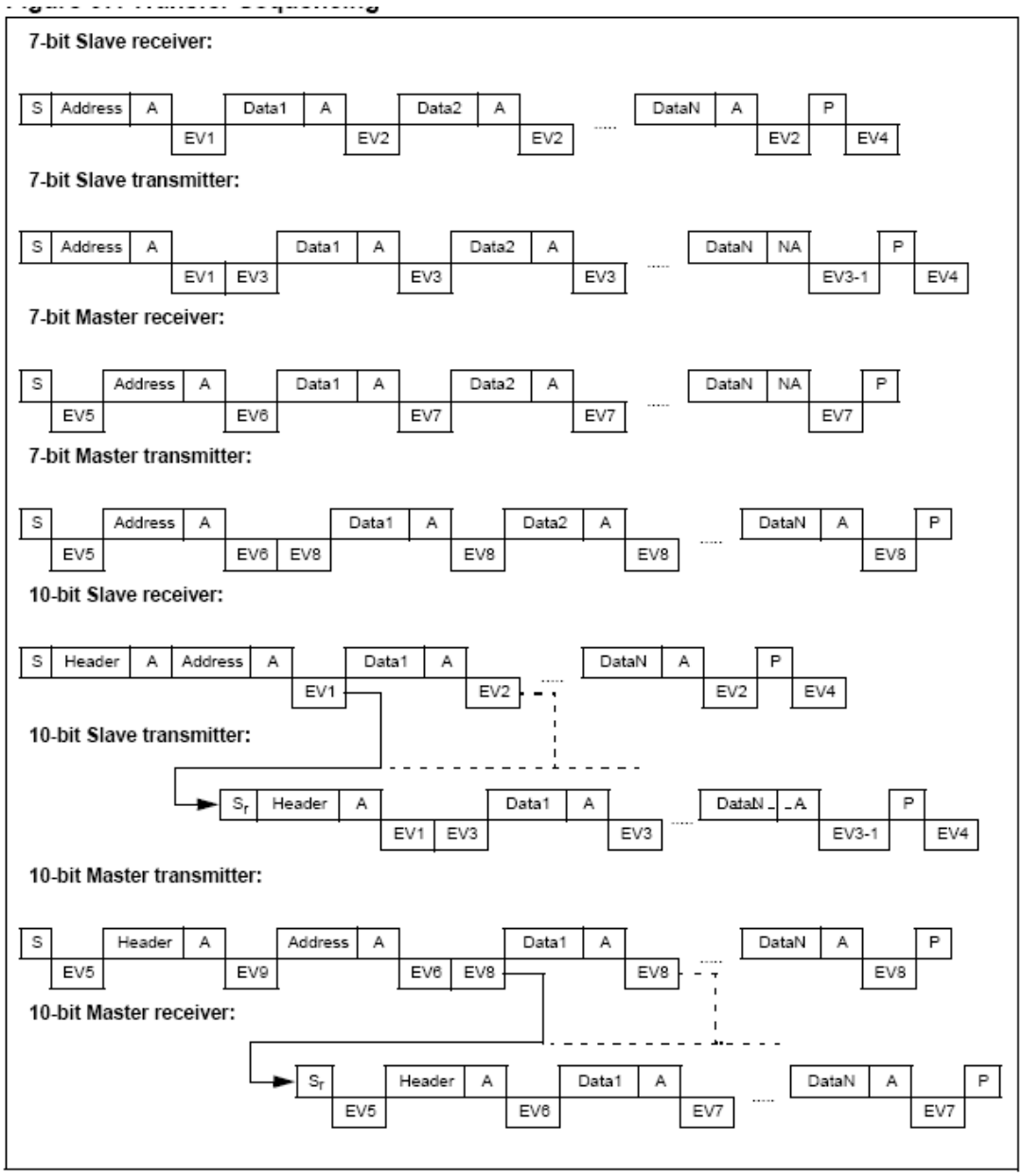
10.3.2.5 出错情况

- BERR：在一个字节传输过程中检测到停止或开始状态。在这种情况下，EVF和BERR位被硬件置位，如果ITE位被置位则产生中断。
- AF：检测到非确认位。在这种情况下，EVF和AF位被硬件置位，如果ITE位被置位则产生中断。为了从错误中恢复，应设置开始或停止位。
- ARLO: 检测到仲裁丢失状态

在这种情况下，ARLO位被硬件置位。如果ITE被置位，则伴随一个中断，且接口自动地返回到从模式（M/SL位被清零）。

注意：在所有这几种情况下，SCL没有被拉成低电平；但是SDA线由于最后可能传送的"0"比特而保持低电平，那么就需要由软件去释放两条信号线。

图57 传输时序



图例说明：

S=开始，S_r=重复开始，P=停止，A=确认，NA=非确认，EVx=事件（如果ITE=1，带有中断）

EV1：EVF=1,ADSL=1,通过读取I2Cn_SR1寄存器来清除。

EV2：EVF=1,BTF=1,通过读取I2Cn_DR寄存器来清除。
EV3：EVF=1,ADSL=1,通过读取I2Cn_SR1寄存器接着写入DR寄存器而来清除。
EV3-1：EVF=1,AF=1,BTF=1；通过读取SR2寄存器清除AF。BTF的清除是通过释放数据线（STOP=1,STOP=0），或者通过写入I2Cn_DR寄存器（DR=FFh）。
注意：如果总线通过STOP=1,STOP=0被释放，紧接着的EV4不可见。
EV4：EVF=1,STOPF =1,通过读取SR2寄存器来清除.
EV5：EVF=1,SB=1,通过读取I2Cn_SR1寄存器接着写入I2Cn_DR寄存器而被清除。
EV6：EVF=1,ENDAD=1,通过读取I2Cn_SR2寄存器接着写入I2Cn_CR寄存器(如PE=1)而被清除。
EV7：EVF=1,BTF=1,通过读取I2Cn_DR寄存器来清除。
EV8：EVF=1,BTF=1,通过向I2Cn_DR寄存器写入来清除。
EV9：EVF=1,ADD10=1,通过读取I2Cn_SR1寄存器接着写入I2Cn_DR寄存器来清除。

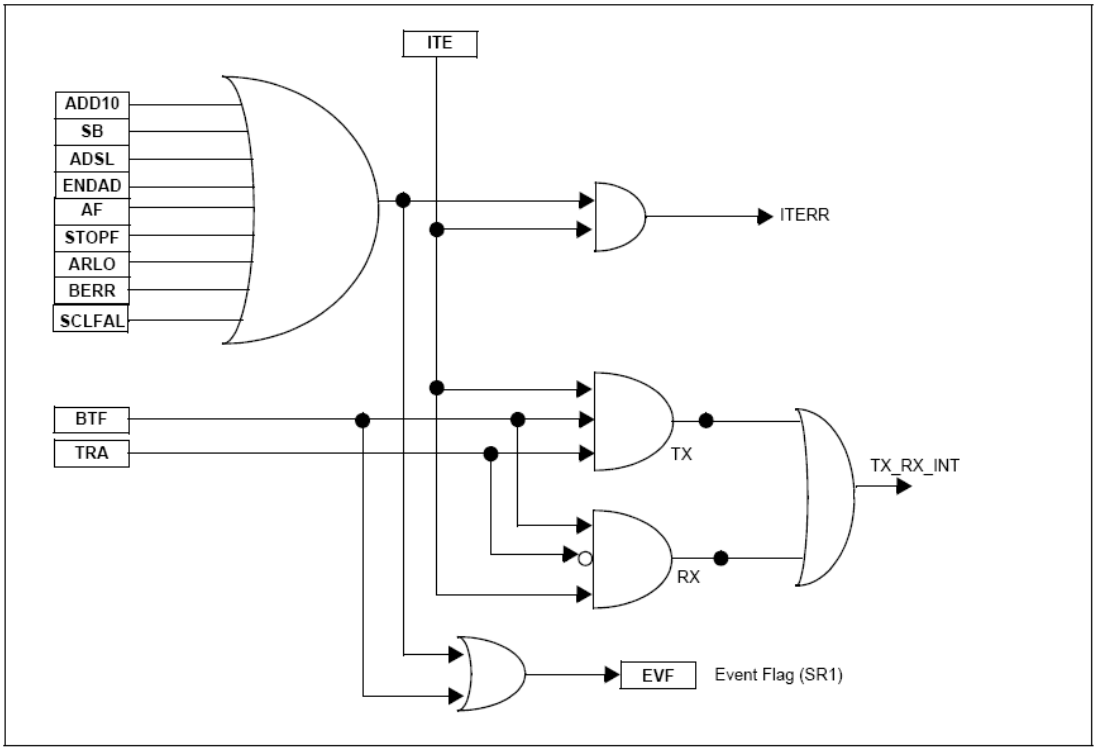
10.4 中断

几种中断可以由模块标志出：

- 与总线事件相关的请求，例如开始和停止事件，仲裁丢失等等
- 与数据发送和/或接收相关的请求。

这些请求通过图58所描述的两种不同的信号线发布给中断控制器。不同的标志识别不同的事件，可以通过软件查询（中断服务程序）。

图58 事件标志和中断产生



10.5 寄存器描述

10.5.1 I^2C控制寄存器（I2Cn_CR）

地址偏移：00h

复位值：00h

7	6	5	4	3	2	1	0
reserved		PE	ENGC	START	ACK	STOP	ITE
-		rw	rw	rw	rw	rw	rw

位7:6 = 保留位，永远读为0

位5 = PE：外围设备使能。
该位通过软件置位和清零
0：外围设备关闭
1: 打开主/从功能
注意：

- 0：除了STOP位外，I2Cn_CR和I2Cn_SR寄存器所有位都被复位，当PE=0时所有的输出被释放。
- 1：相应的I/O引脚由硬件选择为替换功能。

为了使能I^2C接口，设PE=1，向I2Cn_CR寄存器**写入两次**，因为第一次写只启动了接口（只有PE置位）。

位4 = ENGC：使能公共呼叫。
该位通过软件进行置位和清零。当接口被禁止(PE=0)时，也被硬件清零。公共呼叫地址00h被确认（01h被忽略）。
0：公共呼叫被禁止
1: 公共呼叫被使能。

位3 = START：产生一个开始状态。
该位通过软件进行置位和清零。当接口被关闭(PE=0)时，或开始状态被发送（如果ITE=1则产生中断）时，也被硬件清零。

- 在主模式下
  0：不产生开始状态；
  1：产生重复的开始状态
- 在从模式下
  0：不产生开始状态；
  1：当总线空闲，产生开始状态

位2 = ACK：确认使能
该位通过软件进行置位和清零。当接口被关闭(PE=0)时，也被硬件清零。
0：没有确认返回
1：当收到地址字节或者数据字节以后，返回确认信号

位1 = STOP：产生停止状态
该位通过软件进行置位和清零，在主模式下，也被硬件清零。注意：当接口被关闭时(PE=0)这位不被清零。

- 在主模式下
  0：不产生停止状态；
  1：在当前字节传输后，或者在当前开始状态被发送后产生停止。当停止状态被发送后，STOP位被硬件清零。

- 在从模式下

0：不产生停止状态；

1：在当前字节传输完（BTF=1）后，释放SCL和SDA线。在这种模式下，STOP位必须得通过软件清零。

位0 = ITE：中断使能

该位通过软件进行置位和清零。当接口被关闭(PE=0)时，也被硬件清零

0：中断被禁止

1：中断被使能

参见图58中关于事件与中断的关系。

当检测到ADD10，SB，BTF或ADSL标志或者EV6事件时，SCL线被拉成低电平。

10.5.2. I2C状态寄存器（I2Cn_SR1）

偏移地址: 04h

复位值: 00h

7	6	5	4	3	2	1	0
EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB
r	r	r	r	r	r	r	r

位7 = **EVF:** 事件标志

当事件发生时该位由硬件置位。 在发生错误事件或者像图57所描述的事件时，由软件读取寄存器I2Cn_SR2而清零。当接口关闭时(PE=0)它也被硬件清零。

0: 无事件

1: 发生了以下事件中的一个：

• BTF=1 （接收或发送了字节）

• ADSL=1 （当 ACK=1时，在从模式下地址匹配了）

• SB=1 （在主模式下产生了开始状态）

• AF=1 （字节发送后没有接收到确认）

• STOPF=1 （从模式下检测到停止状态）

• ARLO=1 （主模式下仲裁丢失）

• BERR=1 （总线错误，检测到位置错误的开始或停止状态)

• ADD10=1 （主机已发送了头字节）

• ENDAD=1 （主模式下成功发送了地址字节)

位6 = **ADD10**: *主模式下十位寻址*

在十位地址模式下当主机发送首字节后，该位被硬件置位。通过软件读取寄存器I2Cn_SR2后再将第二地址字节写入寄存器I2Cn_SR2的后，它被清零。当I^2C外设关闭时(PE=0)它也被硬件清零。

0: 没有 ADD10事件发生

1: 主控器已发送首地址字节(或头字节)。

位 5 = **TRA**: *发送/接收.*

当 BTF 置位时，如果一个数据字节已被发送则TRA=1。当BTF清零时它被自动清零。当检测到停止状态时(STOPF=1)，总线仲裁丢失后 (ARLO=1) ，或者当接口被禁止时(PE=0)它被硬件清零。

0：收到数据 (当BTF=1时).
1：发送了数据

位 4 = BUSY：*总线忙*.
检测到开始状态时该位被硬件置位，当检测到停止状态时它被硬件清零。它指示在总线上一次通信在进行中。当接口被禁止时(PE=0)该信息仍然被更新。
0：总线上无通信
1：总线上正在进行通信

Bit 3 = **BTF**:*字节传送结束*
一旦一个字节被正确地接收或发送了，该位马上被硬件置位，若ITE=1则产生一个中断。 在软件读取寄存器I2Cn_SR1接着再读/写I2Cn_DR寄存器后，它被清零。当接口被关闭时(PE=0)它也被硬件清零。
- 在一个字节发送之后，接收到确认时钟脉冲后该位被置位。在送出了一个地址时，只有在EV6 事件发生后该位才被置位(见图57)。将下一个字节写入I2Cn_DR寄存器可以清除 BTF。
- 跟随在字节接收之后，若ACK=1，在发送确认时钟脉冲之后该位被置位。在读取寄存器I2Cn_SR1接着读取寄存器I2Cn_DR ，则BTF被清零。

在BTF=1时SCL线被保持在低电平。
0: 字节传送未完成
1: 字节传送成功

位2 = **ADSL**：*地址匹配了 (从模式)*.
一旦接收到的从地址与I2Cn_OAR寄存器内容匹配，或者识别出一个公共呼叫，则该位被硬件置位。当ITE=1时产生一个中断。通过软件读取寄存器I2Cn_SR1清零，或当接口关闭时(PE=0)它也被硬件清零。
当ADSL=1时SCL线被保持为低电平。
0: 地址不匹配或没有接收到。
1: 接收的地址匹配。

位1 = **M/SL**: *主/从*.
一旦接口处在主模式下 (写入START=1)该位就被硬件置位。当在总线上检测到停止状态或失去仲裁(ARLO=1)时，它被硬件清零。当接口关闭时(PE=0)它也被清零。
0: 从模式.
1: 主模式.

位0 = **SB**: *起始位 (主模式)*.
当产生了开始状态(随着写入START=1)时，该位被硬件置位。当ITE=1时产生一个中断。通过读取寄存器 I2Cn_SR1接着再将地址字节写入I2Cn_DR寄存器，它被软件清零。当接口关闭时(PE=0)它也被硬件清零。
0: 无起始状态.
1: 产生了起始状态。

10.5.3 I2C状态寄存器2 (I2Cn_SR2)

地址偏移: 08h
复位值: 00h

7	6	5	4	3	2	1	0
reserved		ENDAD	AF	STOPF	ARLO	BERR	GCAL
-		r	r	r	r	r	r

位7:6 = 保留位，始终被读为0。

位5 = **ENDAD**: *地址发送结束.*

下列情况下该位被硬件设置：

• 7位地址模式：该地址字节已经被发送;

• 10位地址模式：高位和低位部分已经在寻址阶段被发送了。

当主控器需要从从器件接收数据时，它只需要再次发送从地址的高位部分；因此 ENDAD标志被置位，不需要等待地址的低位。在读取SR2接着再写入CR后它被软件清零，当接口关闭时(PE=0)它也被硬件清零。

0: 没有地址传输结束

1: 地址传输结束

位4 = **AF**：*确认失败.*

当确认没有返回时该位被硬件置位。当ITE=1时产生一个中断。通过读取寄存器I2Cn_SR2可被软件清零，或者接口被关闭时(PE=0)被硬件清零。

当AF=1时SCL线路不保持低电平。

0: 无确认失败。

1: 确认失败。

位3 = **STOPF**：*检测到停止状态(从模式).*

在确认之后(如果 ACK=1)当在总线上检测到停止状态时，该位被硬件置位。如果ITE=1则产生一个中断。通过读取I2Cn_SR2寄存器可被软件清零，当接口关闭时(PE=0)被硬件清零。

当STOPF=1时SCL线不保持低电平。

0: 没有检测到停止状态

1: 检测到停止状态

位2 = **ARLO**： 失去仲裁

当接口在总线仲裁输给另一个主机的时候，该位由硬件置位。如果ITE=1会产生一个中断。通过读取I2Cn_SR2寄存器可被软件清除，当接口关闭时(PE=0)被硬件清零。

在ARLO事件后接口自动转回到从模式 (M/SL=0)。

当ARLO=1时SCL 线不保持低电平。

0: 没有检测到失去仲裁

1: 检测到失去仲裁

位1 = **BERR**：总线错误 .

当接口检测到位置错误的开始或停止状态时，该位被硬件置位。如果ITE=1则产生一个中断。通过读取I2Cn_SR2寄存器被软件清除，当接口关闭时(PE=0) 被硬件清除。

当 BERR=1时SCL线不保持低电平。

0: 无位置错误的开始或停止状态

1: 有位置错误的开始或停止状态

位 0 = **GCAL** *公共呼叫(从模式)*

在ENGC=1时，当一个公共呼叫在总线上被检测到，则被硬件置位。当检测到一个停止状态时(STOPF=1)，或者当接口关闭时 (PE=0)它被硬件清零。

0：在总线上没有检测到

1：在总线上检测到公共呼叫地址

10.5.4 I2C时钟控制寄存器(I2Cn_CCR)

地址偏移：0Ch
复位值: 00h

7	6	5	4	3	2	1	0
FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0
rw	rw	rw	rw	rw	rw	rw	rw

位 7 = **FM/SM**: *快速/标准 I²C 模式.*

这些位通过软件置位和清零。当接口关闭时(PE=0)他们不会被清除。

0: 标准 I²C 模式

1: 快速 I²C 模式

位 6:0 = **CC6-CC0**：*12位时钟分频因子.*

这些位同扩展时钟控制寄存器的CC11-CC7位一起，根据I²C模式选择总线速度 (f_{SCL})。 当接口关闭时(PE=0)他们不会被清除。

- 标准模式 (FM/SM=0)： f_{SCL} ≤100kHz

  $f_{SCL} = f_{PCLK1} / (2 \times [CC11...CC0]+7)$

  给定某一 f_{PCLK1}后，就会容易获得正确的分频因子：

  $[CC11...CC0] = ((f_{PCLK1} / f_{SCL}) - 7) / 2 = ((t_{SCL} / t_{PCLK1}) - 7) / 2$
- 快速模式 (FM/SM=1)： 100kHz < f_{SCL}< 400kHz

  $f_{SCL} = f_{PCLK1} / (3 \times [CC11...CC0]+9)$

  给定某一f_{PCLK1}后，就会容易获得正确的分频因子：

  $[CC11...CC0] = ((f_{PCLK1} / f_{SCL}) - 9) /3 = ((t_{SCL} / t_{PCLK1}) - 9) / 3$

注意 编程的 f_{SCL}假定在 SCL and SDA 线上无负载。.

注意 为了正确使用分频因子，[CC11...CC0]必须大于等于0x002 (000000000010b)。
[CC11...CC0] 等于 0x001 (000000000001b)不被承认。

注意 取得的速度含有 ~2% 容差。

10.5.5 I2C 扩展时钟控制寄存器 (I2Cn_ECCR)

地址偏移：1Ch
复位值： 00h

7	6	5	4	3	2	1	0
reserved			CC11	CC10	CC9	CC8	CC7
			rw	rw	rw	rw	rw

Bit 7-5 =保留位，总读为0。

Bit 6:0 = **CC11-CC7**：*12位时钟分频.*

这些位连同时钟控制寄存器一起，根据I²C的模式选择总线的速度 (f_{SCL})。当接口关闭时(PE=0)他们不会被清除。

10.5.6 I2C自身地址寄存器1 (I2Cn_OAR1)

地址偏移：10h
复位值: 00h

7	6	5	4	3	2	1	0
ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
rw	rw	rw	rw	rw	rw	rw	rw

7位寻址模式

比特7:1 = **ADD7-ADD1**: *接口地址.*

这些位定义I^2C 接口总线地址。当接口关闭时(PE=0) 他们不会被清零。

比特 0 = **ADD0**：*地址方向位.*

这一位无关紧要，对于0或1接口均认可。当接口关闭时(PE=0)不会被清零。

注意：地址01h 永远被忽略.

10位地址模式

比特7:0 = **ADD7-ADD0**：*接口地址*

这些位是I^2C 接口总线地址的低8位。当接口关闭时(PE=0) 他们不会被清零。

10.5.7 I2C 自身地址寄存器2 (I2Cn_OAR2)

地址偏移：14h

复位值： 20h

7	6	5	4	3	2	1	0
FR2	FR1	FR0	reserved		ADD9	ADD8	res.
rw	rw	rw	-		rw	rw	-

Bit 7:5 = **FR2-FR0**: *频率位*

只有在接口关闭时(PE=0)这些位才能通过软件设置。为了将接口设置成与I^2C规定的延迟符合，应选择与系统频率f_{PCLK1}对应的值。

f_{PCLK1} Range (MHz)	FR2	FR1	FR0
5 - 10	0	0	0
10 - 16.67	0	0	1
16.67 - 26.67	0	1	0
26.67 - 40	0	1	1
40 - 53.33	1	0	0
53.33 -66	1	0	1
66 - 80	1	1	0
80 - 100	1	1	1

Bit 4:3 = 保留位，永远读为0。

Bit 2:1 = **ADD[9:8]**: *接口地址.*

这些位是接口的I²C总线地址的最高有效位 (仅用于10位模式)。当接口关闭时(PE=0) 他们不会被清零。

Bit 0 = 保留位，永远读为0。

10.5.8 I2C数据寄存器(I2Cn_DR)

地址偏移：18h

复位值：00h

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7:0 = **D7-D0**：*8位数据寄存器.*

这些位包含在总线上发送或接收的字节。

• 发送模式：当软件对寄存器I2Cn_DR写入时，字节发送自动开始。

• 接收模式：利用地址的最低位，第一个字节数据被自动接收在I2Cn_DR寄存器中。然后，在读取I2Cn_DR寄存器后，跟着的字节一个接一个的被接收。

10.6 I²C 寄存器映射

表40 I²C接口寄存器映射

Address Offset	Register Name	7	6	5	4	3	2	1	0
00	I2Cn_CR	reserved		PE	ENGC	START	ACK	STOP	ITE
04	I2Cn_SR1	EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB
08	I2Cn_SR2	reserved		ENDAD	AF	STOPF	ARLO	BERR	GCAL
0C	I2Cn_CCR	FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0
10	I2Cn_OAR1	ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
14	I2Cn_OAR2	FR2	FR1	FR0	reserved		ADD9	ADD8	res.
18	I2Cn_DR	D7	D6	D5	D4	D3	D2	D1	D0
1C	I2Cn_ECCR	reserved			CC11	CC10	CC9	CC8	CC7

基地址请见表 2, “APB1 内存映射”。

11 带缓冲器的 SPI（BSPI）

11.1简介

BSPI 模块是一个标准的用于 IC 之间控制通信的四引脚串行外设接口。它的一边接 SPI 总线，另一边有一个标准的寄存器数据和中断接口。

BSPI 包含两个 10 字×16 位的 FIFO，一个用于接收另一个用于发送。它可以直接操作 8 位和 16 位长的字，并且能为接收和发送事件分别产生向量中断。

11.2主要特征

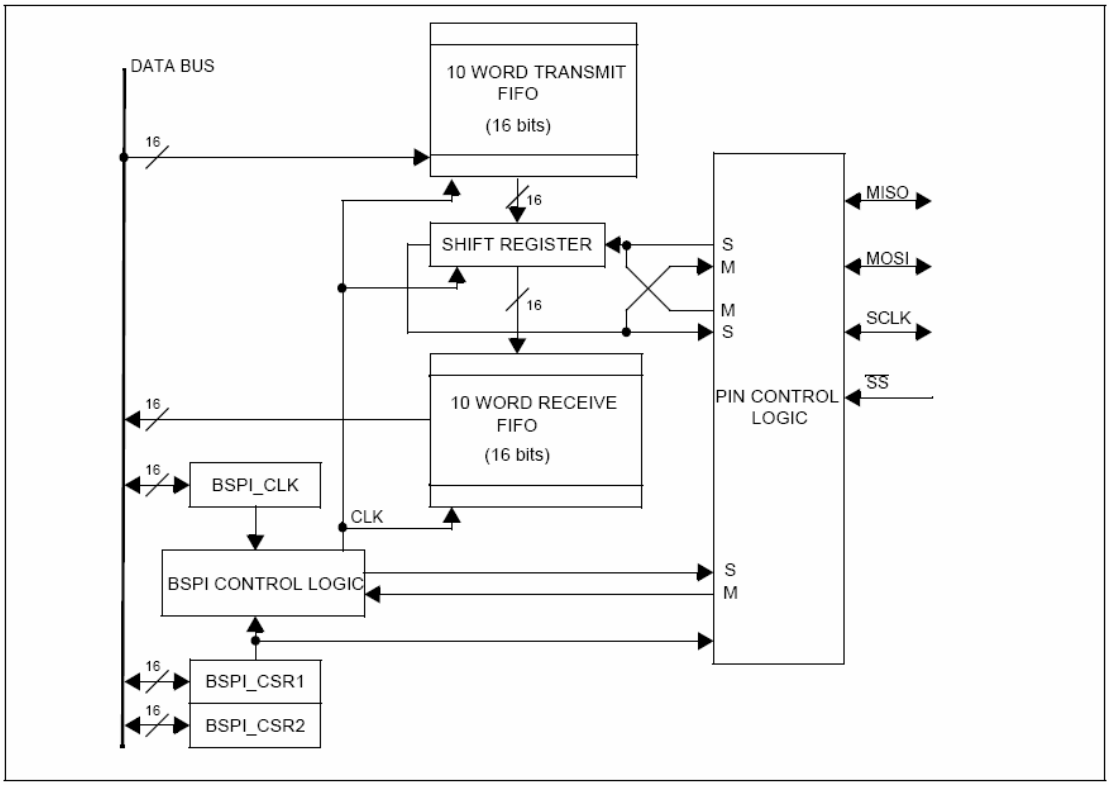
- 可编程的接收 FIFO 深度
- 最大10个字的接收FIFO
- 可编程深度的发送FIFO
- 最大10个字的发送FIFO
- 主从模式支持
- 内置时钟与分频

11.3体系结构

处理器把 BSPI 看作是一个存储器映射的外设，它可以用标准的查询或中断编程技术来使用。存储映射意味着处理器通信能够利用标准指令和寻址模式完成。

当一个 SPI 传输发生时，数据就会同时发送和接收。一个串行时钟信号线对两条串行数据线上的信息的移位和采样进行同步。一个从器件选择线用来单独选择一个从器件。BSPI 系统中的核心单元是一个 16 位移位寄存器和一个 10 字×16 位的读数据缓冲器。BSPI 的方框图如图 59 所示。

图 59 BSPI 方框图



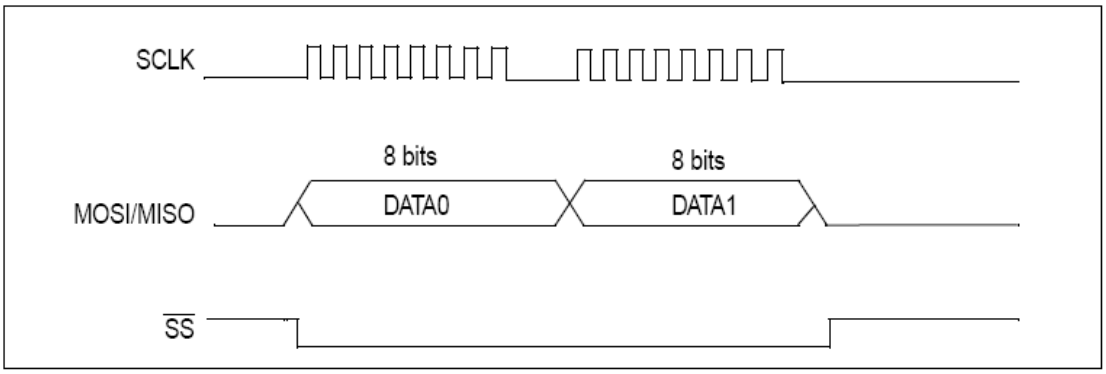
BSPI是一种四线、双向的总线。数据路径由选择的操作模式所确定。模块提供了主和从两种模式和一些相关的移动控制信号来控制移动方向。这些引脚的描述见表41：BSPI引脚。

表41 BSPI引脚

Pin Name	Description
SCLK	The bit clock for all data transfers. When the BSPI is a master the SCLK is output from the chip. When configured as a slave the SCLK is input from the external source.
MISO	Master Input/Slave Output serial data line.
MOSI	Master Output/Slave Input serial data line.
$\overline{SS}$	Slave Select. The $\overline{SS}$ input pin is used to select a slave device. Must be pulled low after the SCLK is stable and held low for the duration of the data transfer. The $\overline{SS}$ on the master must be deasserted high.

11.4 BSPI操作

图60 BSPI总线传输



在BSPI传输过程中（见图60），数据是被同时移出和移进(发送和接收)的。SCLK线使信息的移动和采样信号同步。SCLK在BSPI配置成主控器时作为输出，而配置成从器件时作为输入。用从选择线来选择一个单独的从BSPI设备，没有被选择的从器件不与BSPI总线接口。

BSPIn_CSR1寄存器的CPOL（时钟极性）和CPHA（时钟相位）位用来选择串行时钟四种组合之一（见图61，62，63，64），这些位对于主从BSPI器件必须相同。时钟极性位可选择高电平有效或是低电平有效的时钟，这并不影响传送格式。时钟相位位用来选择传送格式。

有一个16位的移位寄存器直接连接BSPI总线。随着发送的数据从寄存器发出，接收的数据就会填入该寄存器。

注意： 当BSPI单元配置成从模式时，与系统时钟（APB时钟）相比，SCLK_IN时钟频率必须为其8倍分频或更低。

图61 BSPI时钟图解（CPOL=0，CPHA=0）

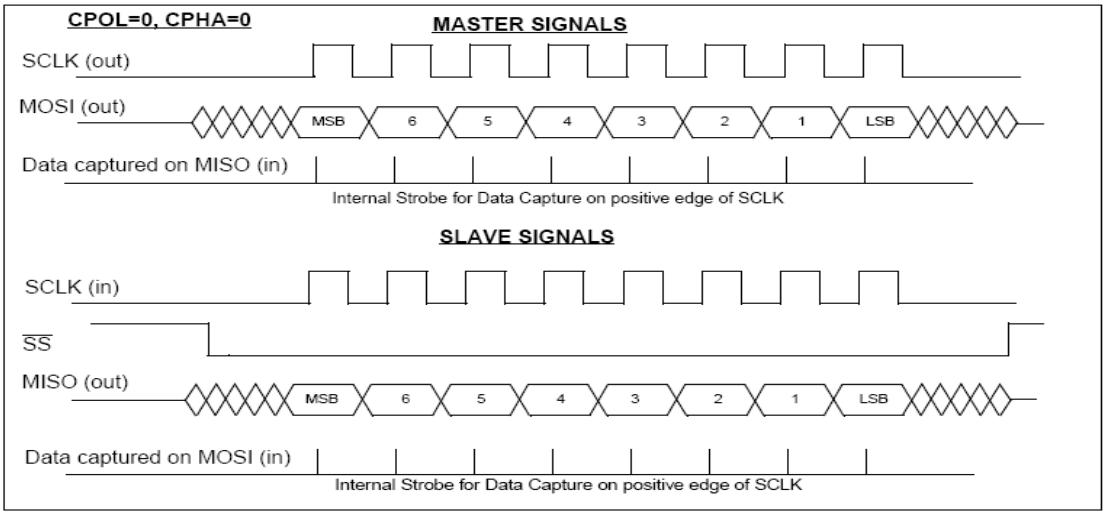


图62 BSPI时钟图解（CPOL=0，CPHA=1）

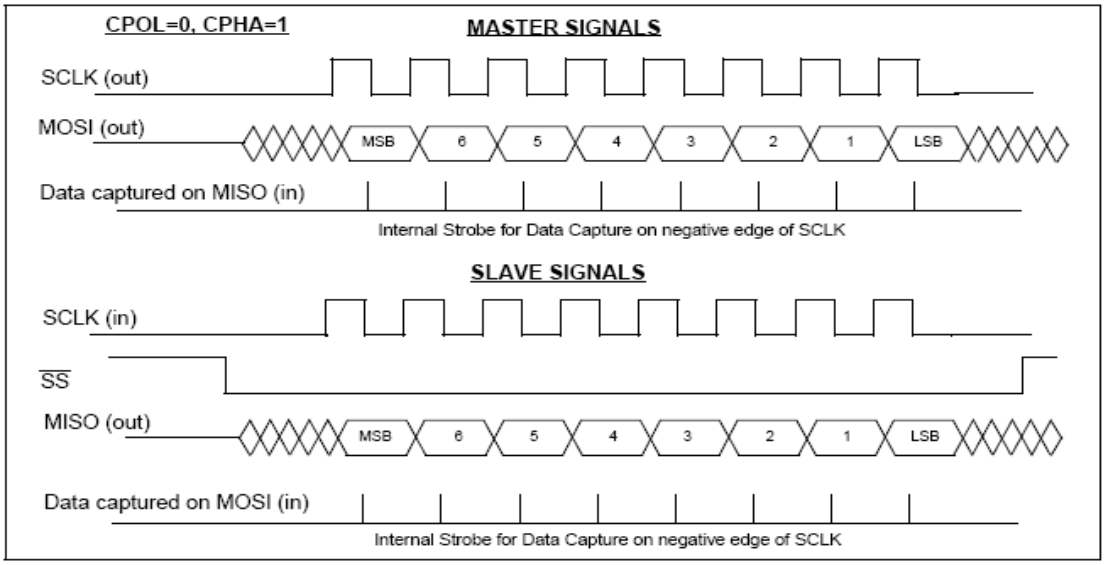


图63 BSPI时钟图解（CPOL=1,CPHA=0）

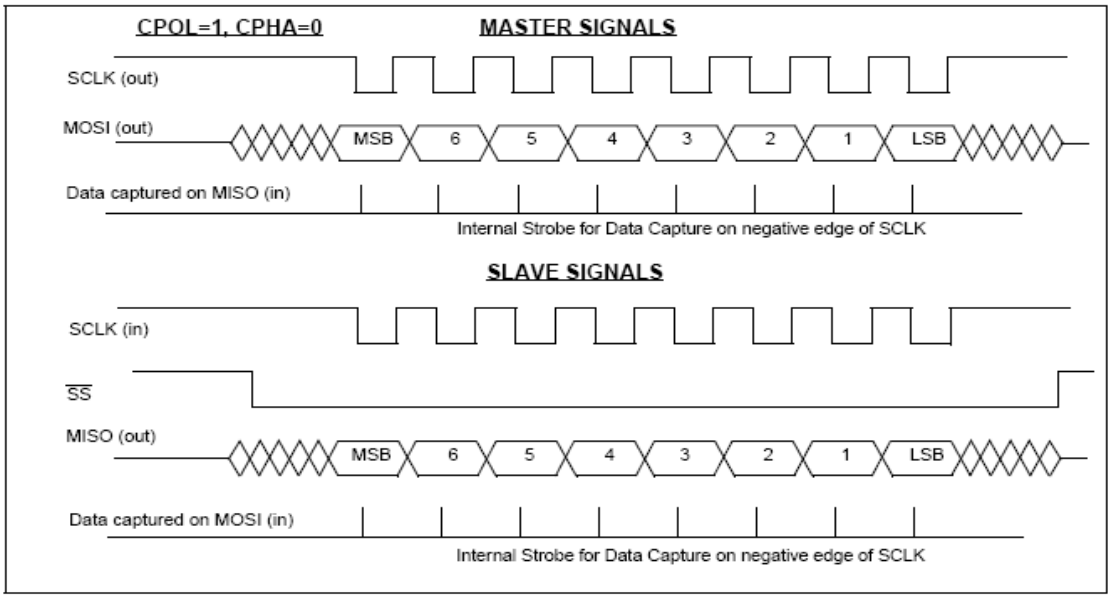
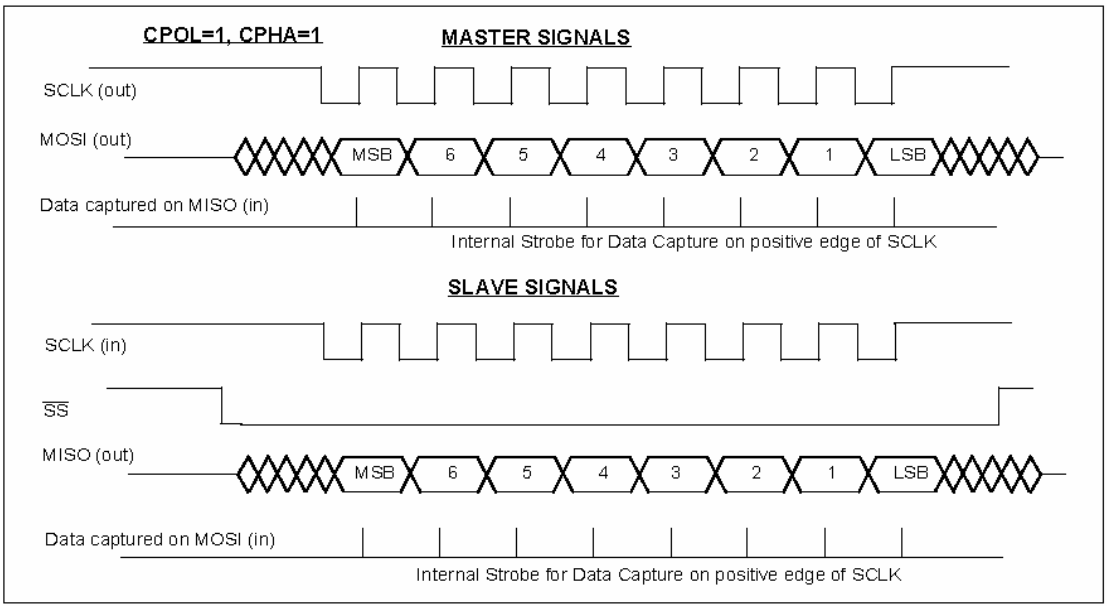


图64 BSPI时钟图解（CPOL=1,CPHA=1）



11.5 发送FIFO

发送FIFO包含一个10×16位的寄存器组，它能工作于8或16位模式下，由BSPIn_CSR1的字长控制位(WL[1:0])进行配置。如果使用8位模式，数据被向左对齐，表示只有一个字的高位的部分被发送。在一次发送完成之后，下一个数据字从发送FIFO中载入。

使用者可以设置FIFO深度，从默认值的1个位置，一直到最大值10个位置。也可以动态改变设置，但改变只有在当前发送完成之后才起作用。状态标志位给出报告包括：FIFO已满（TFF），FIFO不空(TFNE)，FIFO为空（TFE）和发送缓冲器下溢出（TUFL）。BSPIn_CSR2的发送中断使能控制位（TIE[1:0]）确定发送中断源。若中断源被激活，一个高有效的中断将通知给处理器。

如果TUFL标志被设置了，随后对发送FIFO中的写操作将清除这个标志。如果开放了中断，则中断将撤销。在处理器写周期结束和每次发送结束后TFF和TFNE标志就会被更新。

注意： 只有在模块被使能时数据才能写进FIFO（见BSPI控制寄存器的BSPI系统使能位）。在BSPI被使能之前，如果将一个数据字写入发送FIFO，则不会有数据发送。

11.6 接收FIFO

BSPI接收FIFO是一个10字×16位的FIFO，用来缓冲从BSPI总线接收来的数据字。

FIFO能够工作在8位和16位模式下，由控制／状态寄存器BSPIn_CSR1的WL[1:0]位控制。与在FIFO中的字深无关，如果工作在8位模式下，数据将占据每个FIFO位置处的高字节（数据是左对齐的）。接收FIFO使能位RFE[3:0]确定了对所有的传送FIFO为多少字的深度。FIFO默认深度是一个字。只要在FIFO中有至少一个数据，在控制／状态寄存器BSPIn_CSR2中的RFNE位就置位，即：至少在一个位置上有数据。而直到在所有FIFO位置都包含数据之后，RFF标志才被置位，即：当FIFO深度被填满且没有任何数据被读取时，RFF被置位。

如果FIFO是一个字的深度，则一旦有数据写入，RFNE和RFF标志都将被设置。当数据读取时两个标志都被清除。RFF没置位时，对FIFO的写和读是相互独立的。但如果RFF已经被置位，读操作必须在下一次写之前进行，否则将发生溢出（ROFL）。

11.7 启动状态

如果BSPI工作在主模式下，它必须先被使能，然后MSTR位必须设为高电平。TFE标志位将被设置，指示发送FIFO为空；如果TIE设置了，则产生TFE中断。要发送的数据必须写进发送数据寄存器BSPIn_TXR；TFE中断将被清除，然后将根据BSPIn_CLK寄存器的值产生BSPI时钟，数据发送开始。此时第二个TFE中断发生，它使得外设在下一个发送开始之前有充足的数据传送时间来请求数据。

如果BSPI要工作在从模式，器件也必须使能。在主控器的SCLK稳定后，SS线必须设为低电平。TFE标志将被置位，通知发送数据寄存器为空；该标志将被写入发送数据寄存器BSPIn_TXR的操作所清除。第二个TFE中断发生，以便为下一次发送请求数据。

11.8 时钟问题和移位寄存器的清除

如果一个时钟问题导致BSPI移位寄存器中的数据没有对齐，可以用关闭BSPI使能的方法清除它。其效果是置位TFE，请求为下一次发送将数据写入发送寄存器。清除BSPI使能也将清除接收位计数器。下一个收到的数据块将写进FIFO中最后一次成功传送的下一个位置。如果FIFO只有一个字深，它将写进这个唯一可用的位置。

11.9 中断控制

BSPI根据监视发送接收逻辑的状态位来产生一个中断。中断由随后的读写操作来确认或清除，这些读写操作将去掉错误或状态更新的条件。要由程序员来确定中断源，然后移除错误，或者改变BSPI的状态。在发生多个错误的情况下，中断将保持有效状态，直到所有的中断源都被清除。

例如，在TFE状态下，每当最后一个字传送到发送缓冲器，TFE标志就会被设置。如果允许中断，则中断将施加到处理器。要想清除中断，使用者必须写至少一个字的数据到FIFO，或者，如果状态是合法的话关闭中断。

11.10 寄存器描述

11.10.1 BSPI控制/状态寄存器1（BSPIn_CSR1）

地址偏移：08h
复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RFE[3:0]				WL[1:0]		CPHA	CPOL	BEIE	res.	res.	REIE	RIE[1:0]		MSTR	BSPE
rw				rw		rw	rw	rw	-	-	rw	rw		rw	rw

位15:12 = **RFE[3:0]**：接收FIFO使能

接收FIFO能被编程工作在最多10个字的深度。接收FIFO使能位声明了所有的传送使用多少字深。FIFO默认为一个字深度，即类似于单个数据寄存器。下表说明了FIFO是怎样被控制的。

RFE[3:0]	Depth of FIFO
0000	1st word enabled
0001	1st & 2nd words enabled
0010	1-3 words enabled
0011	1-4 words enabled
0100	1-5 words enabled
0101	1-6 words enabled
0110	1-7 words enabled
0111	1-8 words enabled
1000	1-9 words enabled
1001	1-10 words enabled
1010	Default: 1st word enabled
1011	Default: 1st word enabled
1100	Default: 1st word enabled
1101	Default: 1st word enabled
1110	Default: 1st word enabled
1111	Default: 1st word enabled

位11:10 = **WL[1:0]**：字长
这两位设定接收FIFO和发送数据寄存器的操作字长，如下所示：

WL[1:0]	Word Length
00	8-bit
01	16-bit
10	Reserved
11	Reserved

位9 = **CPHA**：时钟相位选择

使用CPOL位确定主-从时钟关系。当CPHA=0时，一旦SS变为低电平，第一个数据采样在SCLK第一个边沿处捕捉。当CPHA=1时，数据从第二个边沿处采样。

位8=**CPOL**：时钟极性选择。

当清除了这一位并且数据没有在传送时，SCLK引脚就会出现一个稳定的低电平值。如果这一位被置位，SCLK引脚闲置时为高电平。这一位和CPHA位一起用来确定主-从时钟关系。

0：选择高有效时钟；SCLK空闲为低电平。

1：选择低有效时钟；SCLK空闲为高电平。

位7 = **BEIE**：总线错误中断使能。

当这一位设定为1时，每当一个总线错误状态发生，中断将输出到处理器。

位6:5=**保留位**，必须保持在复位值（0）。

位4 = **REIE**：接收错误中断使能。

当这一位设定为1并且接收器溢出错误状态发生时，一个接收错误中断将送到处理器。

位3:2 = **RIE[1:0]**：BSPI接收中断使能位。

RIE1:0位是中断使能位，它们设定在接收数据时，何时中断处理器。可有如下设置：

RIE1	RIE0	Interrupted on
0	0	Disabled
0	1	Receive FIFO Not Empty
1	0	Reserved
1	1	Receive FIFO Full

位1 = **MSTR**：主/从选择。

0：BSPI设为从器件。

1：BSPI设为主控器。

位0 = **BSPE**：BSPI系统使能。

0：BSPI系统使能。

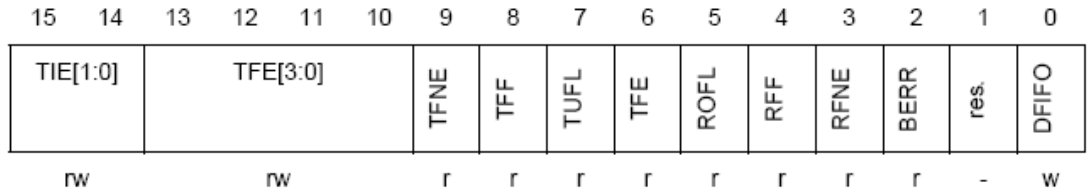
1：BSPI系统关闭。

注意： 在选择中断之前外围设备应该被使能。用这种方法可以避免中断请求信号的不可预测行为。

11.10.2 BSPI控制/状态寄存器2（BSPIn_CSR2）

地址偏移：0Ch

复位值：0040h



位15:14=**TIE**[1:0]：BSPI发送中断使能。

这些位控制发送中断源。

TIE1	TIE0	Interrupted on
0	0	Disabled
0	1	Transmit FIFO Empty
1	0	Transmit underflow.
1	1	Transmit FIFO Full

位13:10=**TFE**[3:0]：发送FIFO使能。

这些位控制发送FIFO深度。下表列出了所有有效的设置：

TFE[3:0]	Depth of FIFO
0000	1st word enabled
0001	1st & 2nd words enabled
0010	1-3 words enabled
0011	1-4 words enabled

TFE[3:0]	Depth of FIFO
0100	1-5 words enabled
0101	1-6 words enabled
0110	1-7 words enabled
0111	1-8 words enabled
1000	1-9 words enabled
1001	1-10 words enabled
1010	Default: 1st word enabled
1011	Default: 1st word enabled
1100	Default: 1st word enabled
1101	Default: 1st word enabled
1110	Default: 1st word enabled
1111	Default: 1st word enabled

位9=**TFNE**：发送FIFO非空。

当FIFO包含至少一个字时这一位被设定。

位8=**TFF**: 发送FIFO满。

每当写入发送FIFO的字数等于由TFE[3:0]规定的FIFO位置个数时，TFF被置位。在数据写完之后，标志就会立即被设定。

位7=**TUFL**：发送下溢

如果TFE位已被置位，并且在发送数据寄存器为下一次发送要把内容移入移位寄存器时，处理器还没有把要发送的数据放入发送寄存器，则该状态位被置位。

当CPHA=1时，TUFL在第一个时钟边沿处置位；当CPHA=0时，在SS有效时TUFL置位。如果TIE[1:0]设为”10”，当TUFL被设置时，中断将送到处理器。

注意： 从应用角度看，意识到在下溢事件发生后取得的第一个字应该被忽略是非常重要的，因为这个数据是在下溢标志设定之前装入移位寄存器的。

位6=**TFE**：发送FIFO空。

每当发送FIFO把它的最后一个字传给发送缓冲器时，这一位就会被置位。如果中断被允许，则当最后一个字送入发送缓冲器时将会触发一个中断。

位5=**ROFL**：接收器溢出

如果FIFO接收器已满，并且在另一个收到的字达到时，FIFO还没有被处理器读取，这一位就会被设置。如果REIE设置了，则这一位被置位时，一个中断将送到处理器。当CSR寄存器和FIFO被读取时，这一位就会清零。

位4=**RFF**：接收FIFO满

这一状态位指示由RFE[3:0]位定义个数的FIFO位置已满。即，如果FIFO是4字深度，4个位置都已收到数据。如果RIE[1:0]设为‘11’，则当本状态位被设置时，有一个中断将送到处理器。在至少读取了一个字后，这一位清零。

位3=**RFNE**：接收FIFO非空

这一状态位表示在接收FIFO中有数据。每当在FIFO中至少有一个数据块，即对8位模式有8比特，16位模式时有16比特，这一位就会被设置。如果RIE[1:0]位设为’01’，一旦该位设置了，有一个中断将送到处理器。当所有的有效数据都从FIFO中读出时，这一位清零。

位2=**BERR**：总线错误

这一位指示发生了总线错误，即在BSPI总线上同时有多个器件作为主机。总线错误状态定义为：当模块被设置为主机时，其从选择线变为低有效。这表明在BSPI总线上有多个节点试图作主机，造成了冲突。

位1=保留位，必须设为复位值（0）

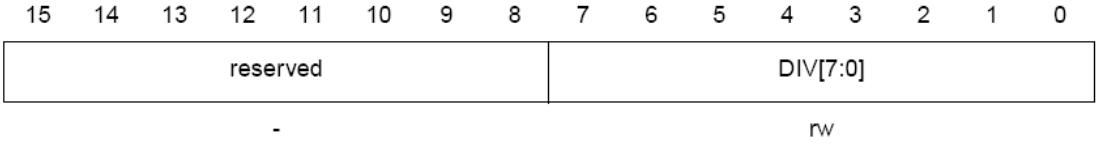
位0=**DFIFO**：使FIFO无效

当这一位有效时，FIFO指针全部设为0，RFE位设为0，因此BSPI设定为一个位置。在FIFO中的数据会丢失。在一个时钟周期后这一位复位为0。

11.10.3 BSPI主时钟分频寄存器（BSPIn_CLK）

地址偏移量:10h

复位值：0006h



位15:8=保留，必须保持为复位值(0)。

位7:0=**DIV[7:0]**：分频因子位

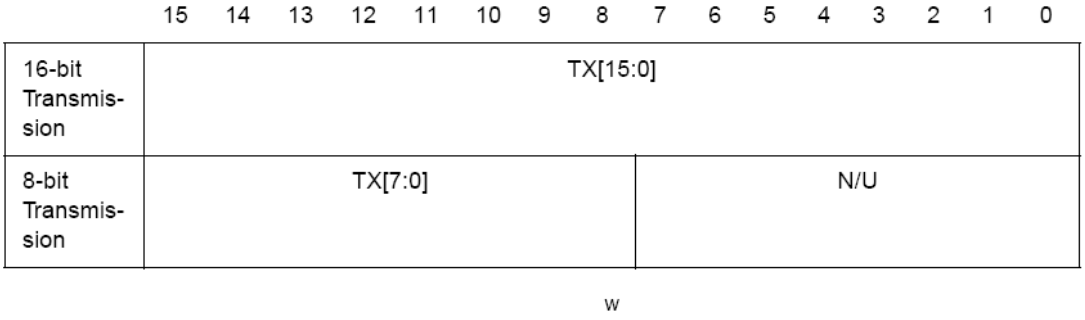
这几位用来控制BSPI串行时钟频率与APB1时钟的关系。在主模式下这一位必须是大于5的偶数，即6是最低的分频因子。在从模式下这一位必须是大于7的偶数，即8是最低的分频因子。

这些位必须在BSPE或MSTR位之前设置，即在BSPI被设置为主模式之前设置。

11.10.4 BSPI发送寄存器（BSPIn_TXR）

偏移地址：04h

复位值：n/a



位15:0=**TX[15:0]**：发送数据

这个寄存器用来把要发送的数据写进BSPI。如果FIFO有效，写进这个寄存器的数据在发送之前将传送给FIFO。如果FIFO无效，寄存器内容将直接传送给移位寄存器。在16位模式下，寄存器的所有位都被使用。在8位模式下，只使用寄存器的高8位。在这两种情况下，数据都是左对齐的，即，Bit[15]=最高位；Bit[0]或Bit[8]=最低位，取决于操作模式。

11.10.5 BSPI接收寄存器（BSPIn_RXR）

地址偏移量：00h

复位值：0000h

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-bit Transmis-sion	RX[15:0]															
8-bit Transmis-sion	RX[7:0]								N/U							

r

位15:0=**RX[15:0]**：接收到的数据

这个寄存器包含从BSPI总线接收到的数据。如果FIFO无效，移位寄存器中的数据直接放入接收寄存器。如果FIFO有效，接收到的数据送到FIFO。在16位模式下，所有寄存器位是可用的。在8位发送模式下，只使用寄存器的高8位。在寄存器两种发送模式下数据都是左对齐的，即Bit[15]=最高位；Bit[0]或Bit[8]=最低位，由操作模式确定。

11.11 BSPI寄存器映射

下表给出BSPI寄存器的一个概要。

Addr. Offset	Reg. Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	BSPIn_ RXR	RX[15:0] (*)															
04h	BSPIn_ TXR	TX[15:0](*)															
08h	BSPIn_ CSR1	RFE[3:0]				WL[1:0]		CPHA	CPOL	BEIE	res.	res.	REIE	RIE[1:0]		MSTR	BSPE
0Ch	BSPIn_ CSR2	TIE[1:0]		TFE[3:0]				TFNE	TFF	TUFL	TFE	ROFL	RFF	RFNE	BERR	res.	DFIFO
10h	BSPIn_ CLK	Unused								DIV[7:0]							

注释：*数据根据发送模式进行左对齐，BIT[15]=MSB。

基地址请见表2“APB1存储器映射”。

12 UART

12.1 介绍

UART 接口提供 STR71x 与其他微控制器、微处理器和外设进行串行通信的手段。

UART 支持全双工异步通信。8 位或 9 位数据传输、奇偶校验的产生和停止位的位数是可以编程的。奇偶校验，成帧和过载错误检测可以用来提高数据传输的可靠性。数据的发送和接收能简单地实现双缓冲，也可以使用 16 字深度的 FIFO。对多处理器通信应用，它提供一种机制来区分地址和数据字节。支持回环测试选项。16 位的波特率发生器为 UART 提供单独的串行时钟信号。

12.2 主要特征

- 全双工异步通信
- 两个内部 FIFO（16 字深度）用于数据发送和接收
- 16 位波特率发生器
- 数据帧长可为 8 位和 9 位
- 奇偶校验位（偶或奇）和停止位

12.3 功能描述

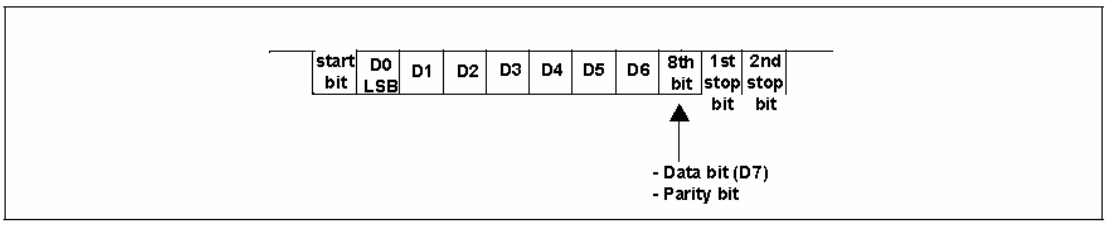
UART 支持全双工异步通信，其中发送器和接收器使用同样的数据帧格式和波特率。数据通过 TXD 引脚发送，通过 RXD 引脚接收。

8 位数据帧（见图 65）的组成可以是：

- 8 个数据位 D0-D7（通过设置 Mode 位域为 001）；
- 7 位数据 D0-D6 加一位自动产生的奇偶校验位（通过设置 Mode 位域为 011）。

奇偶校验可以是奇检验或偶校验，这取决于 UARTn_CR 寄存器的 PArityOdd 位。如果这 7 位数据的模 2 和为 1，则偶校验位置 1，而在奇校验时则对校验位清零。

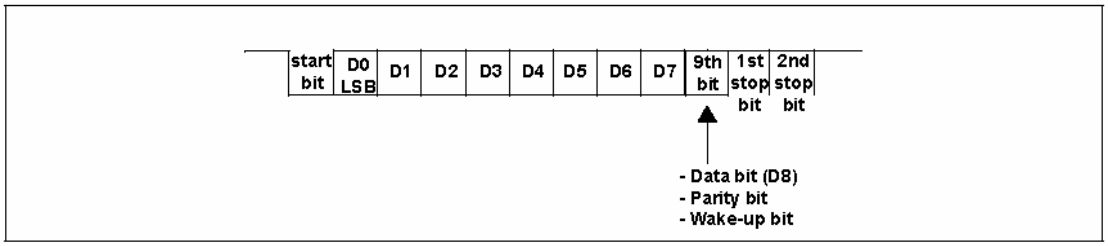
图 65 8 位数据帧



9 位数据帧（见图 66）的组成可以是：

- 9 位数据位 D0-D8（通过设置 Mode 位域为 100）；
- 8 位数据位 D0-D7 加一位自动产生的奇偶校验位（通过设置 Mode 位域为 111）；
- 8 位数据位 D0-D7 加唤醒位（通过设置位 Mode 域模式为 101）。

图 66 9 位数据帧



校验可以是奇校验或偶校验，取决于 UARTn_CR 寄存器的 ParityOdd 位。如果这 8 位数据计算模 2 和的值是 1，则偶校验时校验位置 1，若为奇校验则校验位清零。

在唤醒模式下，如果第 9 位（唤醒位）是 1 时，接收到的数据帧才被送到接收缓存寄存器。如果唤醒位是 0，不会触发接收中断请求，也不会传送数据。

这种功能可以用来控制多处理器系统中的通信。当主处理器想发送一组数据给其中一个从设备时，它首先发送一个地址字节来辨别目标从设备。地址字节与数据字节不同之处在于，这个额外的第九位对地址字节来说是 1，而对数据字节来说是 0。所以不会有从设备被一个数据字节所中断。一个地址字节会中断所有的从设备（工作在 8 位数据+唤醒位模式），所以每一个从设备可以检查接收到字符（地址）的低 8 位。那个被寻址的从设备将会切换到 9 位数据模式，使得从设备能够接收即将到来的数据字节（其中唤醒位被清零）。目前不被寻址的从设备仍保持在 8 位数据+唤醒位的模式，忽略随后的数据字节。

12.3.1 发送

通过向 UARTn_TxBUFR 写入，可将要发送的值写到发送 FIFO 缓冲器 TxFIFO 中去。TxFIFO 作为一个具有 16 个元素的 9 位向量阵列。

如果 FIFO 被使能（UARTn_CR 中 FifoEnable 被置位），当它包含了 16 个字符时，TxFIFO 被认为已满。此时再向 UARTn_TxBUFR 写入将不能覆盖 TxFIFO 中最近输入的字符。如果 FIFO 设为无效，当 TxFIFO 包含 1 个字符时，它就被认为是满的（UARTn_SR 的 TxFull 被置位），在这种情况下，对 UARTn_TxBUFR 的写操作就会覆盖里面的内容。

如果 FIFO 被使能，当 TxFIFO 包含 8 个或更少的字符时 UARTn_SR 中 TxHalfEmpty 会被置位。如果 FIFO 没有被使能，当 TxFIFO 是空时， UARTn_SR 中 TxHalfEmpty 会被置位。

对 UARTn_TxRSTR 写任何数都会使 TxFIFO 清空。

为了发送一个数据，该数据会被移出 TxFIFO 的底部进入一个 9 位的发送移位寄存器。如果这个发送器是空闲的（发送移位寄存器是空的），若有数据被写入 UARTn_TxBUFR 使得 TxFIFO 变为不空，发送移位寄存器会立即从 TxFIFO 加载，移位寄存器中数据的发送会在下一个波特率时间单元开始。

在发送器将要发送停止位的时候，如果 TxFIFO 不是空的，发送移位寄存器会立即从 TxFIFO 装载数据，一旦当前停止位区间结束，这个新数据的发送就会立即开始（即，下一个开始位的发送会紧跟在当前的停止位区间的后面）。因此背靠背的数据发送就能实现。如果此时 TxFIFO 是空的，那么发送移位寄存器会变空。UARTn_SR 中的 TxEmpty 位指出发送移位寄存器是否为空。

在改变了 FifoEnable 位后，复位 FIFO（通过写 UARTn_TxRSTR 寄存器）是很重要的，因为 FIFO 指针的状态可能是任意遗留数值。

回环选项（通过 UARTn_CR 中 LoopBack 位选择）在内部连接了发送移位寄存器的输出和接收移位寄存器的输入。这可以用来在早期测试串行通信，而不需要提供外部网络。

12.3.2 接收

接收由数据输入（RXD）引脚上的下降沿启动，前提是 UARTn_CR 中 Run 和 UARTn_CR 中 RxEnable 位已经被置位。RXD 引脚以当前选定波特率的 16 倍被采样。开始位的第一、第二、第三

采样的多数裁决可确定有效位值，这避免了可能由噪声引起的错误结果。

当开始位被采样时如果检测值不是一个 0，接收电路被复位，等待 RXD 引脚上的下一个下降沿跳变。如果开始位是有效的，接收电路继续采样并且把到来的数据帧移进接收移位寄存器。对于随后的数据和校验位，在每一位时间的第七、第八、第九采样的多数裁决用来决定有效的位值。

注意：当数据输入引脚（RXD）被一直拉低在“0”电平时，如果接收被启动，会报告一个帧错误，因为接收阶段采样初始值为一个下降沿。

对 0.5 停止位，在停止位期间第 3、第 4、第 5 采样的多数裁决用来确定有效的停止位。

对 1 个和 2 个停止位，在停止位期间的第 7、第 8、第 9 采样的多数裁决用来确定有效的停止位。

对 1.5 停止位，在停止位期间第 15、第 16、第 17 采样的多数裁决用来确定有效的停止位。

在 RXD 引脚接收到的有效位值被移进一个 10 位的接收寄存器中。

接收 FIFO 缓冲 RxFIFO，是一个具有 16 个元素的 10 位向量阵列（每个从 9 到 0）。如果 RxFIFO 是空的，UARTn_SR 中 RxBufNotEmpty 被置为 0。如果 RxFIFO 不是空的，读取 UARTn_RxBUFR 会得到 RxFIFO 中最早的值。如果 FIFO 不被使能，则当 RxFIFO 包含一个字符时，它被认为是满的。当 RxFIFO 包含多于 8 个字符时，UARTn_SR 中 RxHalfFull 被置位。对 UARTn_RxRSTR 写任何东西都可以使 RxFIFO 清空。

一旦上个停止位的有效值被决定，接收移位寄存器的内容就被传送到 RxFIFO（在唤醒模式下除外，这种模式下只有当唤醒位是“1”时，传送才会发生）。接收电路然后等待 RXD 引脚的下一个开始位（下降沿跳变）。

当 RxFIFO 已经是满的，且有一个字符从接收移位寄存器装载到 RxFIFO 时，UARTn_SR 中 OverrunError 位被置位。当读取 UARTn_RxBUFR 寄存器时，该位被清零。

每个 RxFIFO 数据项的最高位（RxFIFO[x][9]）记录了当该数据向被收到时是否有帧错误（即有效停止位中的一个值是‘0’）。当 RxFIFO 中的有效数据项至少有一个的最高位置位时，UARTn_SR 中 FrameError 被置位。

如果使用的模式要求校验位，则 RxFIFO[x][8]位（如果选择的是 8 位数据加校验模式）或 RxFIFO[x][7]（如果选择的是 7 位数据加校验模式）记录了当数据项被收到时是否有校验错误发生。

注意：收到数据中不包含校验位。当 RxFIFO 中的有效数据中至少有一个的位 8 被置位（如果选择的是 8 位数据+校验位模式）或者位 7 被置位（如果选择的是 7 位数据+校验位模式）时，UARTn_SR 中 ParityError 被置位。

在改变了 FIFO 的使能位后，对 FIFO 复位使之为空（通过对 UARTn_RxRSTR 寄存器进行写操作）是非常重要的，因为 FIFO 指针的状态可能是无效的。

通过对 UARTn_CR 的 RxEnable 位清零，可以停止接收。当前接收的帧会完成并产生接收状态标志。此帧后面的开始位不会被识别。

12.3.3 超时机制

UART 包括一个 8 位超时计数器。当下列条件至少有一个成立时，它的值会重新从 UARTn_RxBBUFR 加载。

- 对 UARTn_RxBBUFR 进行读操作
- UART 开始接收一个字符
- 对 UARTn_TOR 进行写操作

如果上面条件都不成立，计数器按波特率时钟做减法计数直到 0。

每当 RxFIFO 不空，且超时计数器恰好是零时，UARTn_SR 中 TimeoutNotEmpty 为“1”。

每当 RxFIFO 为空，且超时计数器恰好是零，UARTn_SR 中 Timeoutdle 为“1”。

这种逻辑的效果是，每当 RxFIFO 中已经有一些东西，超时计数器就会递减，直至 RxFIFO 有某种事件发生。如果什么也没发生，且超时计数器递减到达零，则 UARTn_SR 中 TimeoutNotEmpty

标志位会被置位。

当软件清空了 RxFIFO 时，超时计数器会复位并且重新开始递减。如果没有新字符到来，当计数器递减到零时，UARTn_SR 中 Timeoutdle 标志位会被置位。

12.3.4 波特率的产生

波特率发生器提供了一个 16 倍于波特率的时钟，称为过采样时钟。只有当 UARTn_CR 中 Run 被置‘1’时，这个时钟才运转。把这个位清零会立即冻结 UART 发送器和接收器的状态。仅在 UART 空闲才能这样做。

波特率和对给定波特率所需的重载值是由下面的公式决定：

波特率＝PCLK1/(16*<UART_BaudRate>)

<UART_BaudRate>=PCLK/(16 x 波特率)

其中：<UART_BaudRate>代表 UARTn_BR 寄存器的内容，取整为一个无符号的 16 位整型，PCLK1 是 APB1 系统外设的时钟频率。

表 43 和表 44 列举了各种常用的波特率和它们所需的重载值，以及对于两种不同 PCLK1 时钟频率（分别为 16Mhz 和 20Mhz）的偏离误差。

表 43 PCLK1=16Mhz 的波特率

Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
625K	1.6	2	0002	20%
38.4K	26.042	26	001A	0.160%
19.2K	52.083	52	0034	0.160%
9600	104.167	104	0068	0.160%
4800	208.333	208	00D0	0.160%
2400	416.667	417	01A1	0.080%
1200	833.333	833	0341	0.040%
600	1666.667	1667	0683	0.020%
300	3333.333	3333	0D05	0.010%
75	13333.333	13333	3415	0.003%

表 44 PCLK1=20Mhz 的波特率

Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
625K	2	2	0002	0%
38.4K	32.552	33	0021	1.358%
19.2K	65.104	65	0041	0.160%
9600	130.208	130	0082	0.160%
4800	260.417	260	0104	0.160%
2400	520.833	521	0209	0.032%
1200	1041.667	1042	0412	0.032%
600	2083.333	2083	0823	0.016%
300	4166.667	4167	1047	0.008%

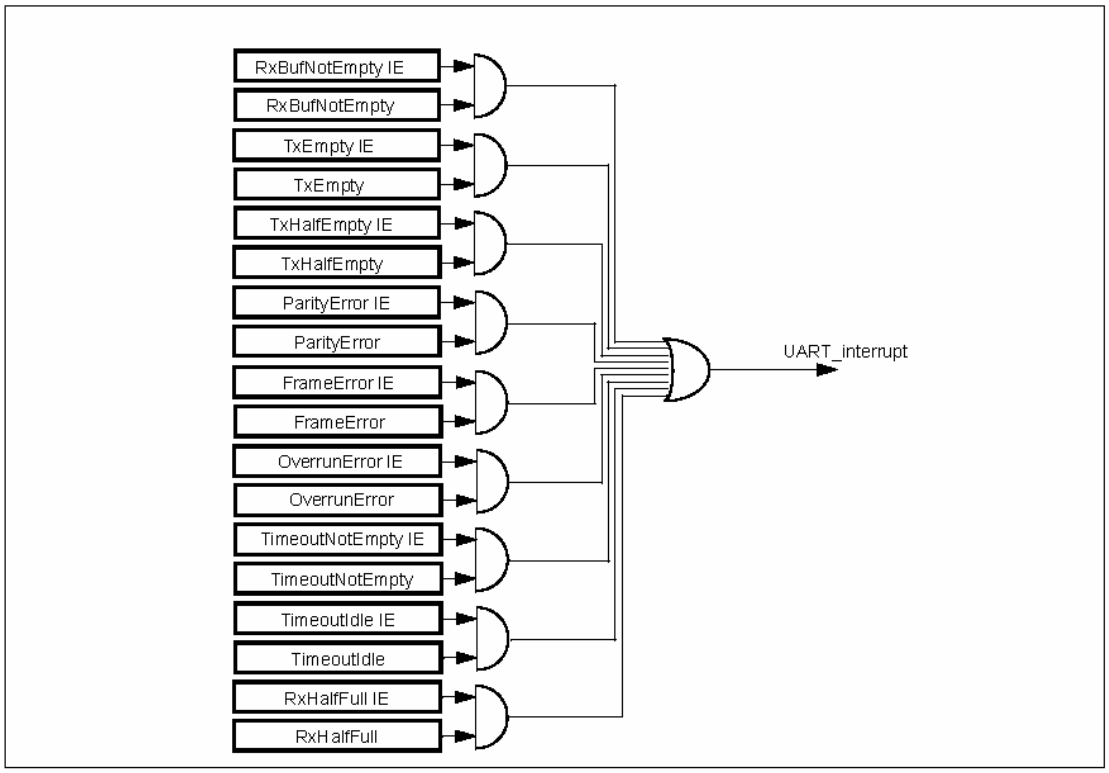
Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
75	16666.667	16667	411B	0.002%

12.3.5 中断控制

UART 有一个单独的中断请求线，叫作 UARTn_interrupt。UARTn_SR 寄存器的状态位确定了引起中断的原因。当一个状态位是 1（高）并且 UARTn_IER 寄存器中相应的位是 1 时，UARTn_interrupt 会变成高电平（见图 67）。

注意：UARTn_Status 寄存器是只读的。UARTn_Status 位只能通过对 FIFO 的操作进行清零。通过对 UARTn_RxReset 和 UARTn_TxReset 寄存器的写操作可以对 RxFIFO 和 TxFIFO 进行复位。

图 67 UART 中断请求



12.3.6 当 FIFO 无效时使用 UART 中断

当 FIFO 没有被使能时，UART 提供三个中断请求来控制通过串行通道进行的数据交换。

- 当数据从 UARTn_TxBUFR 移到发送移位寄存器时，TxHalfEmpty 被触发。
- 在停止位被发送之前，TxHalfEmpty 被触发。
- 当接收的数据帧被移到 UARTn_TxBUFR 时，TxBufNotEmpty 被触发。

对一个单独的传送，使用发送器中断（TxEmpty）已经足够，它指示除了停止位，先前加载的数据已经发送了。

对于多个背靠背传送，使用 TxEmpty 会仅仅留下一个停止位的时间让处理程序对中断作出反映并启动另一次发送。使用发送缓冲中断（TxHalfEmpty）来重新加载发送数据，可有发送一个完整的帧那么长的时间留给中断服务程序，因为当先前数据仍发送的同时，UARTn_TxBUFR 就可以重加载。

TxHalfEmpty 是对重载程序的早触发，而 TxEmpty 表明了一帧内的数据域已完成发送。因此，使用握手协议的软件应该依靠在数据块的结尾的 TxEmpty 来确保所有的数据确实已经发送了。

12.3.7 当 FIFO 使能时使用 UART 中断

为背靠背地发送大量字符，驱动程序会对 UARTn_TxBUFR 写入 16 个字符，然后每次 TxHalfEmpty 中断触发时，它会再写 8 个。当不再有数据要发送时，一个 TxEmpty 中断将告知所有数据已经发送完。

当接收时，驱动可以使用 RxBufNotEmpty，在每次字符到来时都中断。另一种方法是，如果数据一个接一个地到来，它可以用 RxHalfFull，在 RxFIFO 中有多于 8 个字符时进行中断。在数据溢出之前，它可有最长可接收 8 个字符的时间来响应这个中断。如果有少于 8 个字符流入，且至少在一个超时段内没有再接收到数据，这个驱动程序可能被两个超时中断之一唤醒，这两个中断为 TimeoutNotEmpty 或者 Timeoutdle。

12.3.8 智能卡(SmartCard)模式下特有操作

为和 ISO SmartCard 规范一致，下面的模式会在 UART 的智能卡模式被支持。

当 SmartCard 模式位为 0 时，进行正常的 UART 操作。当 SmartCard 模式位为 1 时，进行下面的操作：

- 发送移位寄存器的数据的发送保证最小有 1/2 波特时钟的延迟。在正常操作中，一个满的发送移位寄存器将在下个波特时钟沿处开始移位。在 SmartCard 模式下，这个发送被进一步延迟至少 1/2 波特时钟。
- 如果在一个编程为有 1/2 停止位的数据帧的接收过程中出现校验错误，在接收帧完成后，即在 1/2 停止位结束时，发送线会被拉低一个波特时钟周期。这用来向 SmartCard 表明发送到 UART1 的数据还没被正确接收。
- 通过对 UART1_GTR 寄存器的编程，TxEmpty 标志位的设置可以被延迟。在正常操作时，当发送移位寄存器是空的，且没有更进一步的发送请求时，TxEmpty 就被设置。

  在 Smartcard 模式下，发送移位寄存器变空事件可触发保护时间计数器开始向上计数，一直计数到在 UART1_GTR 寄存器中的可编程的值为止。在这个时间段内 TxEmpty 被强迫为低。当保护时间计数器到达到编程值时，TxEmpty 声明为高。

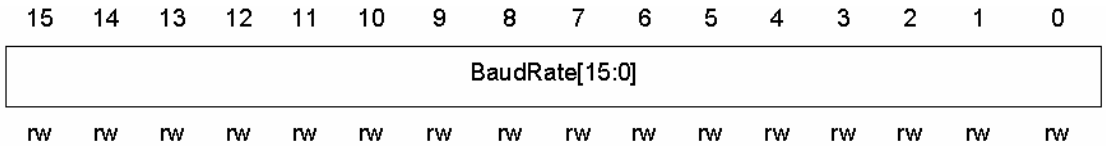
  TxEmpty 的撤除不受 Smartcard 模式的影响。

在收到一个字符后，接收器使能位被清除。这就避免了在 UART 驱动软件处理了目前的字符之前，如果出现智能卡将 RXD 线驱动为低的情况，会使接收器去检测另一个开始位。

12.4 寄存器描述

12.4.1 UART 波特率寄存器（UARTn_BR）

地址偏移：00h
复位值:0001h



UARTn_BR 寄存器是个双功能的波特率发生器兼重载寄存器。

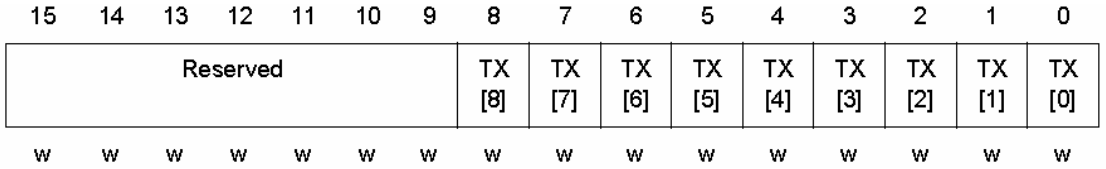
对这个寄存器进行读操作返回定时器的内容，写操作更新重载寄存器。

每次对 UARTn_BR 寄存器进行写操作时，重载寄存器的内容会重载到定时器。然而，如果当对 UARTn_BR 寄存器进行写操作时，UARTn_CR 寄存器的 Run 位为 0，则在 Run 位置 1 后的第一个 PCLK1 时钟周期之前，定时器不会重载。

位 15:0 = BaudRate[15:0]　*UART 波特率*
写功能：16 位重载值
读功能：16 位计数值

12.4.2 UART 发送缓冲寄存器（UARTn_TxBUFR）

偏移地址：04h
复位值：0000h



对发送缓冲寄存器进行写操作会启动数据发送。

位 15:9=保留值，必须保持在复位值（0）。

位 8＝TX[8]：发送缓冲数据的 D8 位。

发送缓冲数据位 D8，或校验位，或唤醒位，或未定义——取决于操作模式（通过 UARTn_CR 寄存器的 Mode 域来设定）。

注意：如果用 Mode 域选择了一个 8 位数据帧模式，那么这个位就应当写 0。

注意：如果用 Mode 域选择了带有校验位的数据帧，TX[8]位将包含校验位（通过 UART 自动产生）。对这个位写'0'或'1'对发送的帧都没有影响。

位 7＝TX[7]：发送缓冲数据 D7 位。

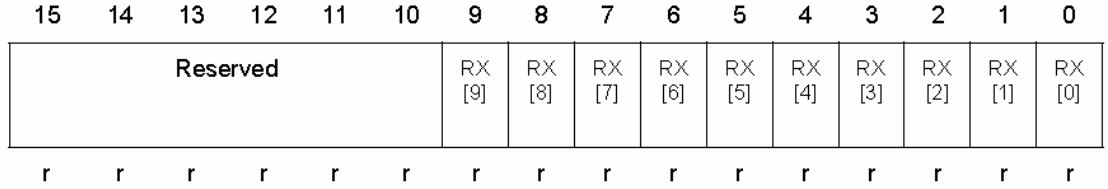
发送缓冲数据位 D7 或者校验位——取决于操作模式（通过 UARTn_CR 寄存器的 Mode 域来设定）。

注意：如果选择了带有校验位的工作模式，TX[7]位包含校验位（通过 UART 自动产生）。对这个位写'0'或'1'都对发送的数据帧没有影响。

位 6:0=TX[6:0]:发送缓冲数据位 D(6:0)

12.4.3 UART 接收缓冲寄存器（UARTn_RxBUFR）

地址偏移：08h
复位值：0000h



接收到的数据（如果选择了校验模式，也包含校验位）可从接收缓冲寄存器读出。

位 15:10=保留位，进行读操作时总读作 0。

位 9=RX[9]：帧错误

如果被置位，表示存储在 RX[8:0]的数据出现了帧错误（即，当接收到数据时，有效停止位之一为'0'）。

位 8＝RX[8]：接收缓冲数据 D8。

接收缓冲数据位 D8，或校验位，或唤醒位——取决于操作模式的选择（通过 UARTn_CR 寄存器 Mode 域选择）。

注意 ：如果 Mode 域选择了 7 位或 8 位的数据帧，则该位无定义。当读 7 位或 8 位数据帧时，软件应忽略这个位。

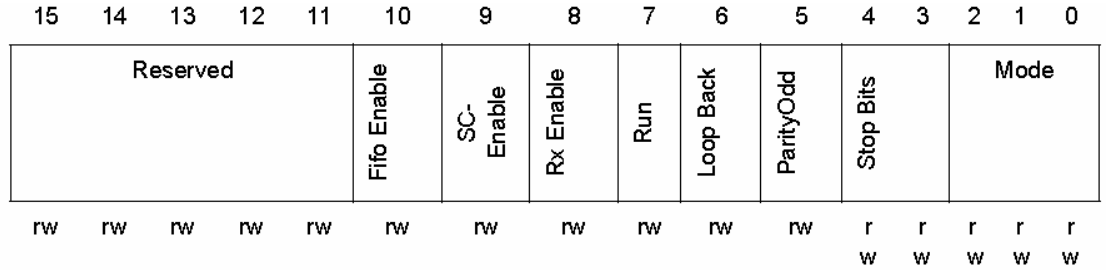
位 7＝RX[7]：接收缓冲数据位 D7。

接收缓冲数据位 D7，或是校验位——取决于操作模式的选择（通过 UARTn_CR 寄存器 Mode 域选择）。

位 6:0=RX[6:0]：:接收缓冲数据位 D(6:0)。

12.4.4 UART 控制寄存器（UARTn_CR）

地址偏移：0Ch
复位值：0000h



这个寄存器控制了 UART 的操作模式，它包含对工作模式和错误校验选择的的控制位、错误辨认的状态标志位。

注意：将模式控制域（Mode）编程为保留的结合，会导致不可预测的行为。

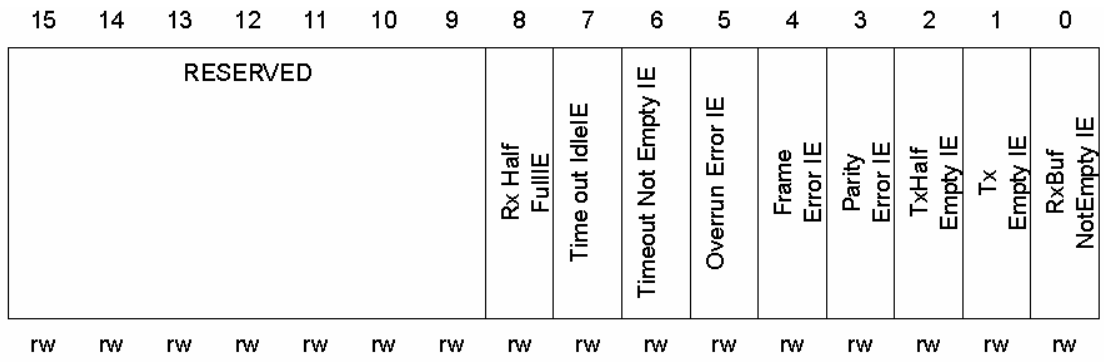
注意：仅当波特率发生器控制位（Run）被置 1 时，串行数据发送和接收才可以进行。当 Run 位清零，TXD 会置 1。对 Run 位清零会立即冻结当前发送器和接收器的状态。当 UART 不是空闲

时，不应这样做。

位 15:11=保留位，进行读操作时总作为 0。
位 10=FifoEnable：FIFO 使能
0：FIFO 模式无效
1：FIFO 模式使能
位 9＝SCEnable 保留用于 SmartCard：模式使能位
0: SmartCard 模式无效
1: SmartCard 模式使能
如果不使用 SmartCard 模式，必须设为 0。
位 8：RxEnable：接收器使能
0：接收器关闭
1：接收器使能
位 7＝Run：波特率发生器运行位
0：波特率发生器关闭（UART 不工作）
1：波特率被使能
位 6＝LoopBack：回环模式使能
0：标准发送/接收模式
1：回环模式使能
注意：只有当 UART 空闲时，才能修改这个位。
位 5=ParityOdd：校验选择
0: 偶校验（当数据中‘1’的个数为奇数个时校验位被置位）
1： 奇校验（当数据中‘1’的个数为偶数个时校验位被置位）
位 4:3=Stop Bit：停止位个数选择
这两位用来选择停止位的个数
00：0.5 个停止位
01：1 个停止位
10：1.5 个停止位
11： 2 个停止位
位 2:0=Mode：UART 工作模式控制
000：保留
001：8 位数据位
010：保留
011：7 位数据+校验位
100：9 位数据位
101：8 位数据位+唤醒位
110：保留
111：8 位数据位+校验位

12.4.5 UART 中断使能寄存器（UARTn_IER）

地址偏移：10h
复位值：0000h



UARTn_IE 寄存器用来使能中断源。

当 UARTn_SR 寄存器的一个状态位是'1'时，会发生中断，且对应的 UARTn_IER 寄存器位会置 1。

位 15:9=保留位，进行读操作时总读为 0。

位 8=RxHalfFullIE：接收器缓冲区半满中断使能

0：禁止中断。

1：允许使能。

位 7=TimeoutidleIE：超时空闲中断使能

0：禁止中断。

1：允许中断。

位 6=TimeoutNotEmptyIE：超时非空中断使能

0：禁止中断。

1：允许中断。

位 5= OverrunErrorIE：过载错误中断使能

0：禁止中断。

1：允许中断。

位 4=FrameErrorIE：成帧错误中断使能

0：禁止中断。

1：允许中断。

位 3=ParityErrorIE：校验错误中断使能

0：禁止中断。

1：允许中断。

位 2=TxHalfEmptyIE：发送缓冲半空中断使能

0：禁止中断。

1：允许中断。

位 1= TxEmptyIE：发送器空中断使能

0：禁止中断。

1：允许中断。

位 0= RxBufNotEmptyIE：接收器缓冲不空中断使能

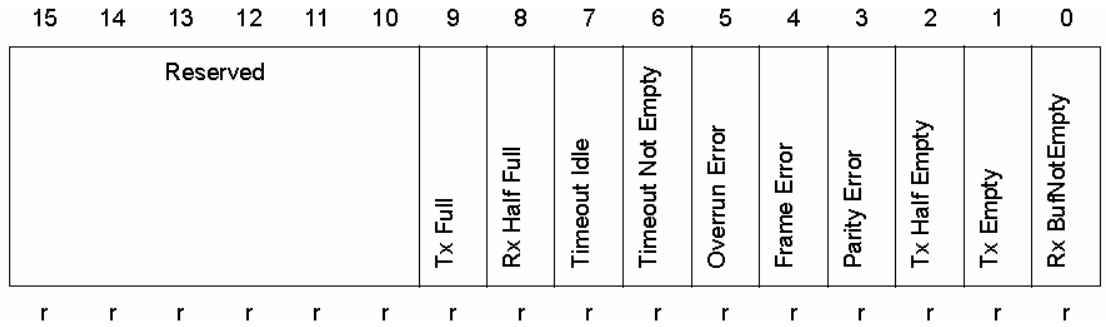
0：禁止中断。

1：允许中断。

12.4.6 UART 状态寄存器（UARTn_SR）

地址偏移：14h

复位值：0006h



UARTn_SR 寄存器确定中断的原因。

位 15:10=保留值，总是读为 0 值。

位 9=TxFull：发送缓冲 TxFIFO 满

当 TxFIFO 包含 16 个字符时置位。

位 8=RxHalffull：接收缓冲 RxFIFO 半满

当 RxFIFO 包含多于 8 个字符时置位。

位 7=Timeoutdle：超时空闲

当有一个超时且 RxFIFO 是空时，置位。

位 6=TimeoutNotEmpty：超时非空

当有一个超时且 RxFIFO 非空时，置位。

位 5=OverrunError: 过载错误

当接收到数据且 RxFIFO 是满时，置位。

位 4=FrameError：成帧错误

当 RxFIFO 包含了错误帧时，置位。

位 3=ParityError：校验错误

当 RxFIFO 包含了有校验错误的数据时，置位。

位 2=TxHalfEmpty： 发送缓冲 TxFIFO 半空

当 TxFIFO 至少有一半为空时，置位。

位 1=TxEmpty：发送缓冲 TxFIFO 空

当发送移位寄存器为空时，置位。

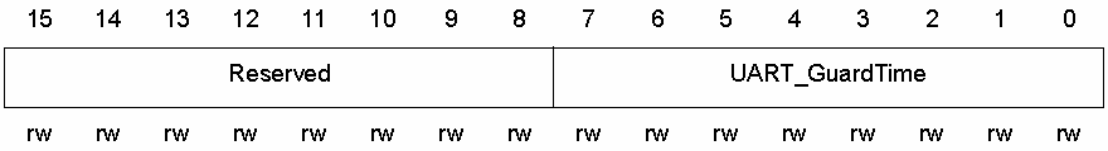
位 0＝RxBuNotEmpty：接收缓冲非空

当 RxFIFO 非空（RxFIFO 包含至少一个数据）时置位

12.4.7 UART 保护时间寄存器（UARTn_GTR）

地址偏移：18h

复位值：0000h



UARTn_GTR 寄存器让使用者可以定义可编程数目的波特率时钟来延迟 TxEmpty 的置位。

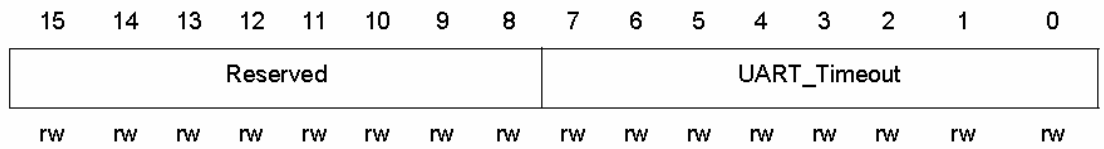
位 15:8 =保留值，总读为 0。

位 7:0 = UART_GuardTime.

用来延时 TxEmpty 的置位的波特率时钟的个数。

12.4.8 UART 超时寄存器（UARTn_TOR）

地址偏移：1Ch
复位值：0000h



这个寄存器的目的是提供一个超时系统，用来保证在前后两个的接收字符之间不相隔太长的时间。

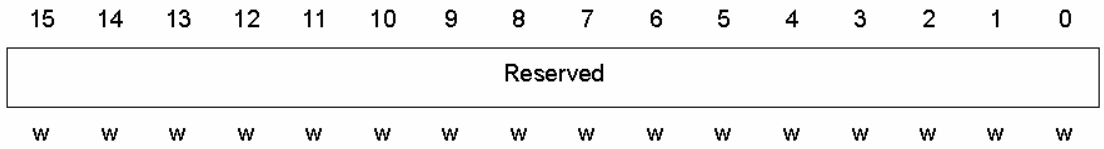
位 15:8=保留值，总作为 0 读

位 7:0＝UART_Timeout：超时

超时计数频率与波特率相同。

12.4.9 UART 发送复位寄存器 (UARTn_TxRSTR)

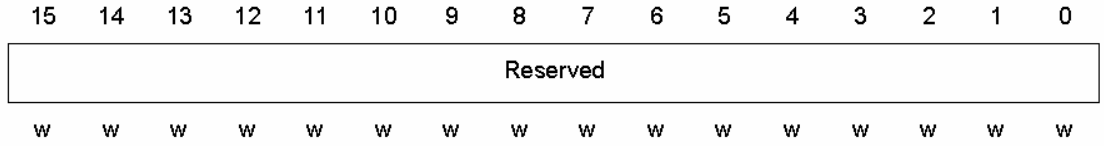
地址偏移：20h
复位值：保留



对这个寄存器的写操作可以清空 TxFIFO。

12.4.10 UART 接收复位寄存器(UARTn_RxRSTR)

地址偏移：24h
复位值：保留



对这个寄存器的写操作可以清空 RxFIFO。

12.5 UART 寄存器映射

下列的表格总结了 UART 实现的寄存器。

表 68 UART 外设寄存器的映射

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	UARTn_BR	UART_BaudRate															
4	UARTn_TxBUFR	Reserved							UART_TxBuffer								
8	UARTn_RxBUFR	Reserved						UART_RxBuffer									
C	UARTn_CR	Reserved					FifoEnable	Reserved	RxEnable	Run	LoopBack	ParityOdd	Stop Bits		Mode		
10	UARTn_IER	Reserved							UART_IntEnable								
14	UARTn_SR	Reserved						UART_Status									
18	UARTn_GTR	Reserved											UART_GuardTime				
1C	UARTn_TOR	Reserved								UART_Timeout							
20	UARTn_TxRSTR	UART_TxReset															
24	UARTn_RxRSTR	UART_RxReset															

基地址请见表 2，“APB1 存储器映射”。

13 智能卡接口（SC）

13.1 介绍

智能卡(SmartCard)接口是对 UART1 的扩展，UART 寄存器描述请见第 12 章。智能卡接口是为了支持在 ISO7816-3 标准中定义的智能卡异步协议而设计的。UART1 配置为 8 位数据位加上校验位，0.5 或 1.5 个停止位，并使能智能卡模式，就可提供智能卡接口的 UART 功能。一个 16 位计数器作为智能卡时钟发生器，将 PCLK1 时钟分频后为智能卡提供时钟。通用 IO 配合软件用来提供智能卡接口需要的其余功能。ISO7816-3 中定义的包括数据反转和高位在先的信号反转协议，由软件来处理。

13.2 外部接口

智能卡需要的信号由表 45 给出：

表 45 智能卡引脚

引脚	功能
SCClk	智能卡时钟
I/O	输入输出串行数据，两端都是开漏驱动。
RST	智能卡复位输入
Vcc	供电电压
Vpp	编程电压

STR71x 提供的信号见表 46：

引脚	名称	输入/输出	功能
Port 0.12(真开漏兼容 5V)	ScClk	输出，用于 5V 智能卡的开漏输出	提供智能卡时钟
	ScDataOut	输出，开漏驱动器	串行数据输出，开漏驱动器
	ScDataIn	输入	串行数据输入
	ScRST	输出，开漏	卡复位输入
	ScCmdVcc	输出	电源电压使能/禁止
	ScCmdVpp	输出	编程电压使能/禁止
	ScDetect	输入	智能卡检测

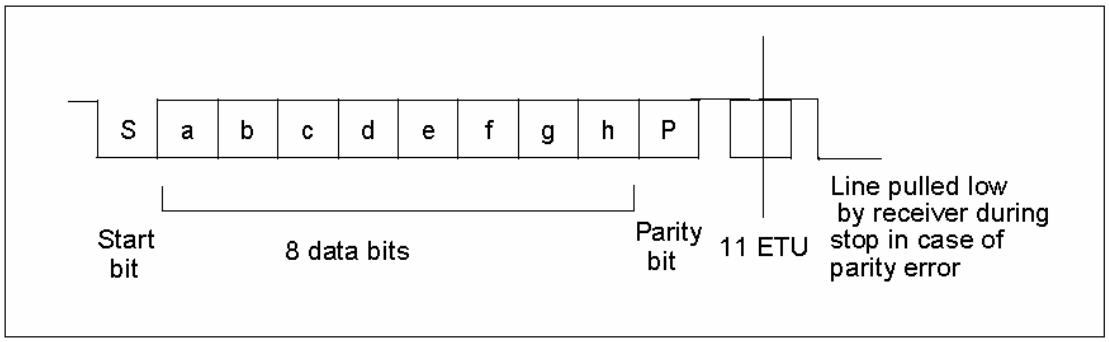
ScRST，ScCmdVpp，ScCmdVcc 和 ScDetect 信号都是用软件控制 GPIO 位来实现的。将 I/O 口的 GPIO 位编程在替换功能模式下，可将 UART 的 TXD 数据信号用正确的驱动类型连接到 ScDataOut 信号引脚，将时钟发生器连接至 ScClk 引脚。在 STR71x 输出引脚表中详细介绍了 GPIO 位可以指定的替换功能引脚位。

注意：STR71x 的 I/O 与 3V 的智能卡兼容。如果是 5V 智能卡，STR71x 的 I/O 线可以为智能卡正确地驱动 5V 输入端口。唯一存在的问题就是 SCDATA 线，由于它是双向的，所以 5V 逻辑电平(VIH/VIL)与它不相匹配。这就是必须用开漏缓冲器的原因。外部的上拉电阻需要连接到 VCC（根据智能卡的种类选择 5V 或 3.3V）。

13.3 协议

ISO 标准为异步协议规定了位时间单位，它按一种称为 ETU 的时间单位来计量，ETU 与智能卡输入时钟频率有关。一个位时间为一个 ETU 的长度。在外部应将 UART 发送器输出端和接收器输入端相连。为了从 STR71x 发送数据到智能卡，UART 应该设置为智能卡模式。

图 69 ISO 7816-3 异步协议



注意：STR71xx 能够通过硬件检测到从智能卡接收到的数据字节中的奇偶校验错误。但是，从读卡机接收到的数据字节中检测出的校验错误必须由软件进行处理。

13.4 智能卡时钟发生器

智能卡时钟发生器为所连接的智能卡提供一个时钟信号。智能卡用这个时钟提取出用于智能卡和另一个 UART 之间串行 I/O 的波特率时钟。如果卡中有 CPU，它同时也为卡中的 CPU 提供时钟信号。智能卡接口的操作要求，当智能卡 CPU 在执行代码时，提供给卡的时钟速率可以调整，以便可以改变波特率或者提高智能卡的性能。ISO7816-3 标准详细说明了管理这些时钟频率的协商和改变的协议。由于这个时钟用作智能卡的 CPU 时钟，所以其时钟速率的更新必须与智能卡的时钟 Clk 输入保持同步，即时钟高、低电平脉冲宽度不能短于原始的和新的编程值。时钟发生器的时钟源是 PCLK1。有两个寄存器分别控制时钟的周期和运行。

注：时钟发生器和 UART 的波特率是相互独立的。

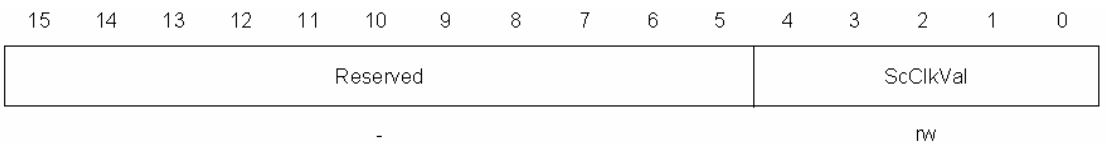
13.5 寄存器描述

智能卡可以通过寄存器来编程控制，这些寄存器映射到 STR71x 的地址空间。在存储器地址的章节中给出了智能卡寄存器的基地址。注：所有寄存器的复位值都为 0。

13.5.1 智能卡时钟预分频值（SC_CLKVAL）

地址偏移量：40h

复位值：00h



SC_CLKVAL 寄存器确定了智能卡时钟频率。寄存器设定的值乘 2 给出对输入时钟频率 的分频因子。

比特 15:5＝保留，必须保持在复位值(0)

比特 4:0＝SC_CLKVAL[4:0]

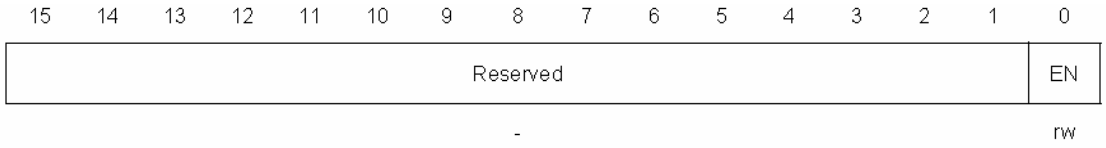
这些位确定了源时钟分频值。这个值乘 2 给出时钟分频因子：

ScClKVal 4:0	分频
00000	保留，不要编程为此值
00001	源时钟频率被 2 分频
00010	源时钟频率被 4 分频

13.5.2 智能卡时钟控制寄存器(SC_CLKCON)

地址偏移量:44h

复位值:00h



SC_CLKCON 寄存器控制时钟来源，确定智能卡时钟输出是否被允许。当智能卡时钟控制寄存器的使能位被设置成 0 时，可编程分频器和输出端复位。

位 15:1＝保留位，它必须保持复位值（0）。

位 0＝使能(EN)

智能卡时钟发生器使能位。

0　停止时钟，将输出置为低电平并复位分频器

1　使能时钟发生器

13.6 寄存器映射

表 47.智能卡接口寄存器映射

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
40	SC_ClkVal	Reserved											ScClkVal				
44	SC_ClkCon	Reserved															EN

基地址见表 2。

14 USB 从设备接口（USB）

14.1 简介

USB 外设实现了 APB 总线与全速 USB2.0 总线之间的接口。
支持 USB 的挂起/恢复功能，这一功能允许通过停止设备时钟来降低功耗。

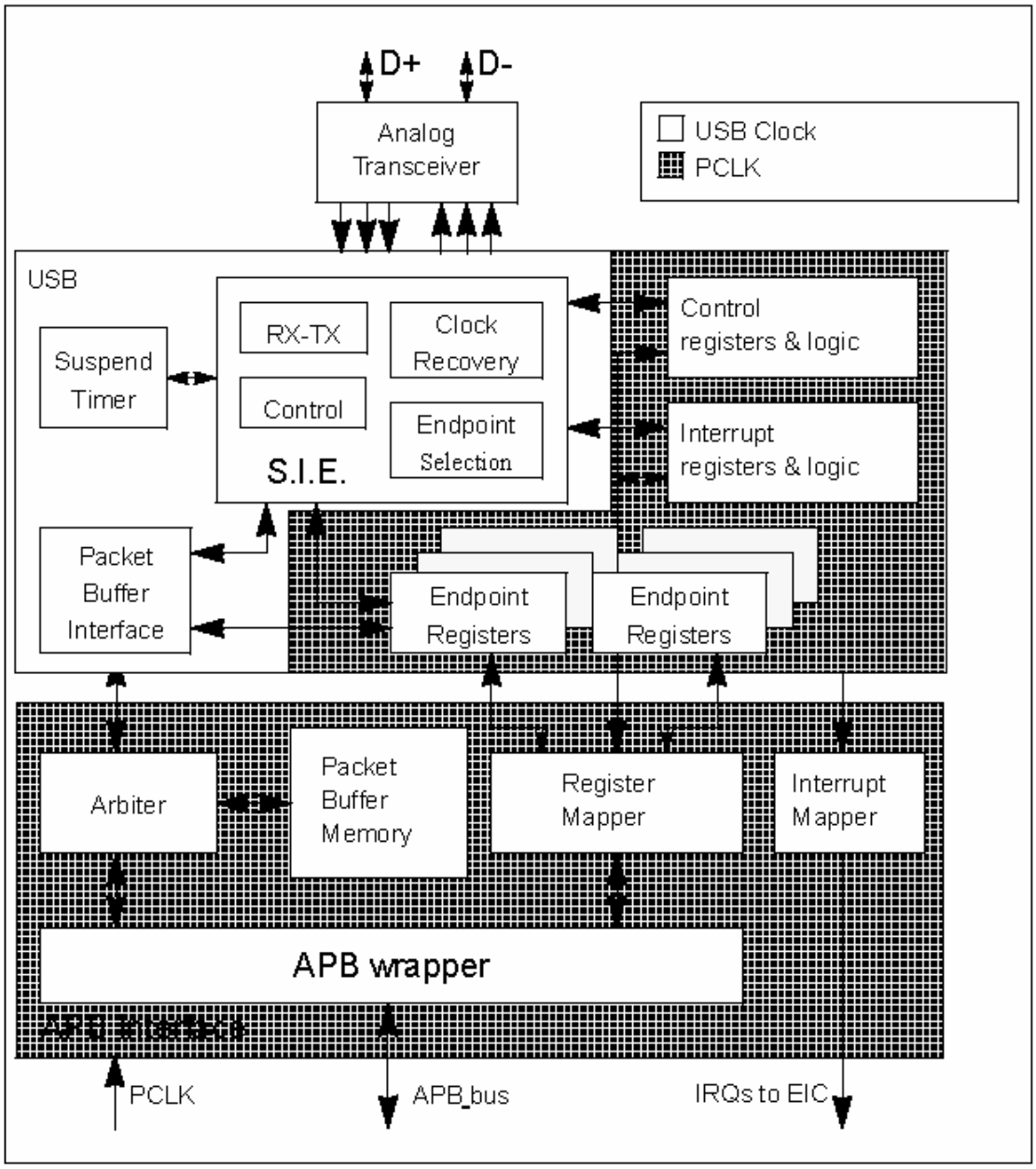
14.2 主要特色

- 最多 16 个双向端点；
- 循环冗余检查（CRC）产生/校验，反向不归零（NRZI）编码/解码以及位填充；
- 支持等时传输；
- 支持双缓冲的批量端点；
- 支持 USB 的挂起/恢复操作；
- 两个中断请求 IRQ 源（高 IRQ 优先级用于等时和双缓冲批量端点，低 IRQ 优先级用于端点）；
- 帧锁定方式的时钟脉冲产生。

14.3 结构框图

图 70 显示了 USB 外设的框图。

图 70.　USB 外设框图



14.4 功能描述

USB 外设提够了一个 PC 主机与由微控制器实现的功能之间的符合 USB 规范的连接。在 PC 主机与系统存储器之间的数据传送通过一个专用的 512 字节包缓冲存储器来进行，这个缓存可被 USB 外设直接访问。这个缓存的大小必须根据使用的端点数目以及最大的数据包大小来决定。USB 外设与 USB 主机相连接，按照 USB 标准的需要探测令牌包、处理数据发送/接收、处理握手信息包。事务格式形成由硬件来执行，包括 CRC 产生和校验。

每个端点与一个缓冲描述块相关联，描述块指示与该端点相关联的存储器在什么位置、这个区域有多大以及多少个字节必须被传送。当一个有效功能/端点对的令牌被 USB 外设识别出来，相关的数据传输（如果需要并且端点已经设置好了）就发生了。由 USB 外设缓冲的数据被加载到在一个内部的 16 位寄存器上，执行对专用缓冲的存储器访问。当所有的数据被传输完成时，如果需要的话，根据传输的方向可以产生或等待 USB 上的握手包。

在处理事务结束时，产生一个与特定端点相关的中断。通过读取状态寄存器并且（或者）使用不同的中断响应程序，微控制器可以确定：

- 哪个端点需要服务；
- 如果有错误产生的话（位填充、格式、CRC、协议、丢失 ACK、过载或欠载等等），是哪种类型的处理事务发生了。

USB 外设产生两种中断：一个 IRQ 收集高优先级的端点中断（等时或双缓冲的批量端点）；另一个 IRQ 收集所有其他的中断源（对于详细的中断源映射请查阅 IRQ 的中断矢量表）。

这一外设对于等时传输和高吞吐量批量传输有特殊的支持，实现了双缓冲。利用双缓冲使得 USB 外设在微控制器使用一个缓冲器时，总是有另一个缓存器可用。

如果需要的话，这一部件可以通过写入控制寄存器将其置于低功耗模式（SUSPEND 模式）。这时所有的静态功率消耗都被消除了，USB 时钟能够被减慢或停止。在低功耗模式下，检测到 USB 输入端的活动将设备异步地唤醒。一个特殊的中断源可以直接连接到一个唤醒线，使得系统能立即重新启动正常时钟的产生，并且（或者）支持对时钟的直接启动/停止。

14.4.1 USB 子模块的描述

USB 外设用实现与 USB 接口有关的所有功能，包括以下子模块：

- 串行接口引擎（SIE）：这一子模块的功能包括：同步模式识别，位填充，CRC 产生及校验，PID 的检验和产生，以及握手的确认。这就必须要与 USB 收发器有接口，使用由本地数据存储器的包缓存接口提供的虚拟缓冲器。这一单元也能根据 USB 外设事件产生出信号，例如：帧起始（SOF），USB 复位，数据错误等，也对与端点相关的事件产生信号，例如像传送结束或者是正确接收到一个包。这些信号就被用于产生中断。
- 挂起定时器：这个模块对于需要帧启动同步信号的外部装置产生帧锁定时钟脉冲，并且当有 3mS 没有传送流量时，它探测全局挂起信号（来自主机）。
- 包缓冲区接口：发送和接收时，这一模块都以一种灵活的方式来管理本地存储器，实现一组缓冲器。它能根据来自 SIE 的请求来选择合适的缓冲器，并根据端点寄存器指针在存储器地址上定位它们。每次字交互完之后它会增加地址，直到这一数据包的结尾，并随时跟踪交换字节的数目以防止超过缓冲器最大容量。
- 端点相关寄存器：每个端点有一个关联的寄存器，存储端点的类型以及它目前的状态。对于单向/单缓冲器端点，一个单独的寄存器可被用于实现两个不同的端点（IN 和 OUT）。STR71x 处理器中 USB 的 IP 核包括 16 个端点寄存器，允许多达 16 个双缓冲端点或最多 32 个单向单缓冲端点以任何形式的组合。
- 控制寄存器：这些寄存器存储整个 USB 外设的状态信息，并用来强迫某些 USB 事件的发生，例如恢复和掉电。
- 中断寄存器：这些寄存器存储了中断掩码和事件的记录。这些能被用于查明中断原因、中断状态或者清除中断等待状态。

USB 外设与 APB 总线通过 APB 接口相连接，包括以下子模块：

- 包存储器：这是物理上存放包缓冲区的本地存储器。它可以被负责建立数据结构的包缓冲区接口使用，并且能被应用软件直接访问。这个包存储器的大小是 512 字节，结构是 256 字×16 位。
- 仲裁器：这个模块接收来自 APB 总线和 USB 接口两方面的存储器请求。它解决请求冲突的办法是，优先考虑 APB 访问，同时总是预留一半的存储器带宽用于完成所有 USB 传输。这种分时双工方案实现了一个虚拟的双端口 RAM，允许在 USB 传输的同时对存储器进行访问。这个方案还可以实现任何长度的多字 APB 传送。
- 寄存器映射器：这个模块将 USB 外设的各种各样的字节宽度和位宽度的寄存器收集在 可由 APB 寻址的 16 位宽的字结构中。
- 中断映射器：这个模块用于选择 USB 事件产生中断的方式，并将这些事件映射给 EIC 的中断线。
- APB 封装器：它给存储器和寄存器提供了一个与 APB 相连的接口，同时也将整个 USB 外设映

射到 APB 地址空间里。

14.4.2 编程考虑

下面的部分描述了 USB 外设与应用程序之间预期的互动，以方便应用软件的开发。

14.4.3 通用的 USB 设备编程

这个部分描述了为了符合 USB 要求，应用软件需要完成的主要任务。与大多数一般 USB 事件相关的动作都被考虑进来，有一些段落专门叙述双缓冲端点和等时传输这两个特殊情形。除了系统复位之外，动作都是由 USB 外设发起，由下面描述的 USB 事件之一来驱动的。

14.4.4 系统和上电复位

在系统和上电复位之时，应用软件应该执行的第一个操作就是给 USB 外设提供所有所需要的时钟信号，然后撤销它的复位信号以便能够访问其寄存器。整个的初始化过程从这里此开始描述。

作为第一步，应用软件需要通过设备时钟管理逻辑提供的相关控制位来激活寄存器宏单元的时钟，撤消与宏单元相关的复位信号。

之后，必须利用 CNTR 寄存器中的 PDWN 位去接通与 USB 收发器相关的设备的模拟部分的电源，这需要一个特殊的处理。这一位的用途是将供给端口收发器的参考电压源开启。因为这个电路有一个限定的开启时间，这一开启时间之内 USB 收发器的行为是不确定的。将 CNTR 寄存器中的 PDWN 位置高之后，需要等待这段开启时间，然后就可以撤除 USB 部分的复位状态（将 CNTR 寄存器中的 FRES 位清零），将 ISTR 寄存器清零，以移除任何虚假的等待中断，在此之后，才能允许执行其他的宏单元操作。

作为最后一步，需要用设备时钟管理逻辑提供的相关控制位来激活 USB 规定的 48MHz 时钟。

系统复位时，微控制器必须初始化所有需要的寄存器和包缓冲器的描述表，以使得 USB 外设能够正确地产生中断信号和进行数据传输。所有的不与任何端点关联的寄存器，都必须根据应用软件的需要被初始化（选择已经允许的中断，选择包缓冲器的地址，等等）。然后任务进程继续进入到对 USB 复位的动作（见后面的段落）。

14.4.4.1 USB 复位 （REST 中断）

这一事件发生时，USB 外设被置于的状态，与被系统复位且经过前面描述的初始化之后的状态相同：USB_DADDR 寄存器被复位，所有端点寄存器中通信被禁止（USB 外设不会响应任何包）。作为对 USB 复位事件的响应，USB 功能必须被启动，此时 USB 地址为 0，只实现默认的控制端点（端点地址也是 0）。启动是通过设置 USB_DADDR 寄存器的使能功能（EF）位，再初始化 EP0R 寄存器以及与其相关的包缓冲器来完成。在 USB 枚举过程中，主机为这个设备分配一个惟一的地址，该地址必须被写入 USB_DADDR 寄存器的 ADD[6：0]位。主机还配置任何需要的其他端点。

当一个 RESET 中断信号被接收到时，应用软件负责在从引起中断的复位序列结束开始算起的 10mS 内再次使能 USB 功能 0 的缺省端点。

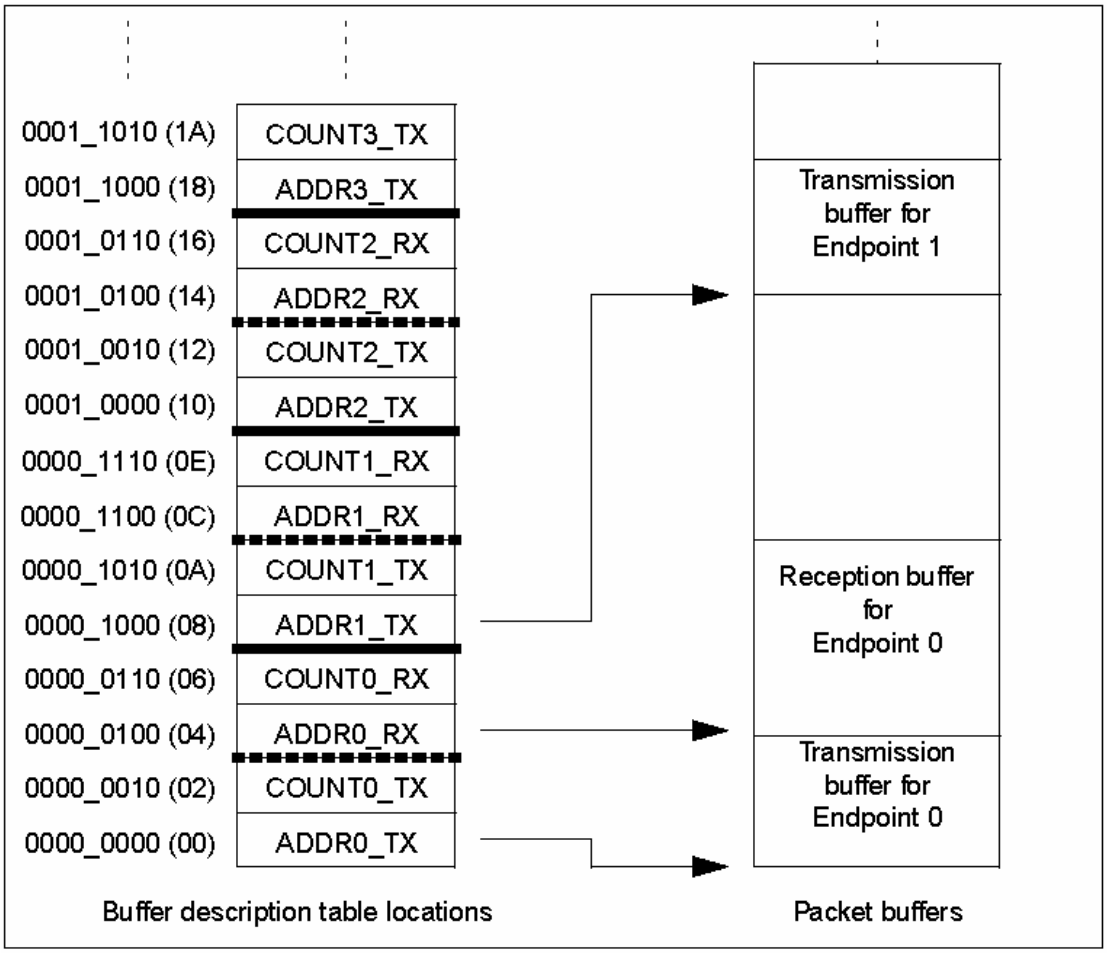
14.4.4.2 包缓冲器的结构和用法

每一个双向端点都可能从主机接收或向主机发送数据。接收到的数据被存储到一个为那个端点保留的专用存储缓冲区之中，而另外一个存储缓冲区存储着要被这个端点发送的数据。访问这个存储器要由包缓冲器接口模块来完成，它传递访问请求等待确认。由于包缓冲器的内存也需要被微控

制器访问，因此有一个仲裁逻辑来处理访问冲突，它将 APB 周期的一半用于微控制器的访问，另一半用于 USB 外设的访问。用这种方法，两者都可执行，就像包存储器是一个双端口的 RAM 一样，就算微控制器正在执行背对背的访问也不用考虑任何的冲突。USB 外设的逻辑使用专用的时钟，其频率按照 USB 的标准被固定为 48MHz，该时钟频率可以不同于与 APB 总线的接口的时钟频率。APB 设置的时钟频率高于或低于 USB 外设的时钟频率都是可能的。但是，由于 USB 数据速度以及包存储器接口的需要，APB 时钟频率必须高于 8MHz，以避免出现数据过载或欠载问题。

每一个端点都与两个包缓冲区相连（通常是一个用于发送而另一个用于接收）。缓冲区可以被置于包存储器内部的任何位置，因为它们的位置和大小已经在一个缓冲器描述表中指定了，这个描述表也位于包存储器内部，其地址由 USB_BTABLE 寄存器指定。每一个表项都与一个端点寄存器相关连，由四个 16 位字组成，这样表格的开始地址必须总是对齐到一个 8 字节的边界（USB_BTABLE 寄存器的最低 3 位总是“000”）。缓冲器描述表的各个表项在“缓冲器描述符表”这一节中叙述。如果一个端点是单向的，而且它既不是等时传输又不是双缓冲的批量传输，这时就只需要一个包缓冲区（与支持的传送方向相关的一个）。与不支持的传送方向或不用的端点相关的表位置可由用户使用。等时和双缓冲的批量端点在包缓冲区中有特殊的处理（分别参考“等时传输”和“双缓冲端点”部分）。缓冲器描述表表项和包缓冲器区域的关系见图 71。

图 71. 包缓冲器区域与缓冲器描述表表项



每一个包缓冲器在接收或发送时被用于从底部开始使用。USB 外设永远不会改变与分配的存储缓冲器相邻的存储器区域的内容；如果一个大于分配的缓冲器长度的数据包被接收到时（缓冲器过载情况），数据最多将只被拷贝到存储器的最后一个可用位置。

14.4.4.3 端点初始化

初始化一个端点的第一步是向 ADDRn_TX/ ADDRn_RX 寄存器写入一个适当的值，使得 USB 外设知道要被传输的数据已经准备好了，以及要接收的数据可以被缓冲了。USB_EPnR 寄存器中的 EP_TYPE 位必须根据端点类型来设定，最终用 EP_KIND 位来启动任何特定需要的功能。对于发送，端点必须用 USB_EPnR 寄存器中的 STAT_TX 位来使能，COUNTn_TX 必须被初始化。对于接受，STAT_RX 位必须被置位以允许接收功能，并且要在 COUNTn_RX 的 BL_SIZE 和 NUM_BLOCK 域中写入已分配的缓冲区大小。除了等时和双缓冲的批量端点外，单向的端点仅需要初始化与它们所支持方向相关的位和寄存器。一旦发送和/或接收被使能后，寄存器 USB_EpnR 以及 ADDRn_TX/ADDRn_RX、COUNTn_TX/ COUNTn_RX 位置（分别地）都不能够被应用软件所更改，因为硬件运行过程中可以改变它们的值。当 CTR 中断信号事件通报数据传输操作完成时，它们能够被再次访问来重新启动一个新的操作。

14.4.4.4 IN 包（数据发送）

当接收到 IN 令牌信息包时，如果接收到的地址与一个已经配置的有效端点相匹配，USB 外设就访问 ADDRn_TX 和 COUNTn_TX 位置的内容，这些位置在与被寻址端点相关的缓冲器描述符表项中。这些位置的内容存储在内部的 16 位寄存器 ADDR 和 COUNT 中（不能被软件访问）。包存储器被再次访问，读取要被处发送的第一个字（参见“包存储器的结构和用法”这一节），并且根据 USB_EpnR 寄存器的 DTOG_TX 位开始送出一个 DATA0 或 DATA1 的 PID。当 PID 完成时，从缓冲存储器中读取的字中的第一个字节被载入输出移位寄存器中，准备被发送到 USB 总线上。最后一个数据字节发送之后，发送计算出的 CRC。如果被寻址的端点无效，根据 USB_EPnR 寄存器中的 STAT_TX 位，一个 NAK 或者是 STALL 握手包将代替数据包被发送。

ADDR 内部寄存器被用来指向当前缓冲存储器位置，而 COUNT 被用来计数要被发送的剩余字节数目。每一个从数据包缓冲存储器读出的字在 USB 总线上发送时都是从最低字节开始。发送缓冲存储器从 ADDRn0_TX 指定的地址开始读取，共读取 COUNTn_TX/2 个字。如果一个要发送的包由奇数个字节组成，最后一个字只用到较低的半个字。

当接收到来自主机的 ACK 回应时，寄存器 USB_EPnR 将按照如下方式更新：DTOG_TX 位被取反，通过设置 STAT_TX=10 （NAK）使端点无效，且将 CTR_TX 位置位。应用软件必须首先辨认请求控制器关注的端点，通过检查寄存器 USB_ISTR 的 EP_ID 和 DIR 位来确认。对 CTR_TX 事件的服务从中断位的清零开始，然后应用软件再次用准备要发送的数据填满一个缓冲器，用下一传输中将被传送字节的数目来更新 COUNTn_TX 表位置，最后设置 STAT_TX 为“11”（VALID）来重新启动发送。当 STAT_TX 位等于“10”（NAK）时，任何寻址到该端点的 IN 请求都被用 NAK 回应，标志着一个流控制状态：USB 主机将重试该事务直到成功为止。发送过程的操作序列必须严格按照上述顺序进行，这样才能避免紧跟在触发 CTR 中断的 IN 事务之后、寻址到同一端点的第二次 IN 处理的通知被丢失。

14.4.4.5 OUT 和 SETUP 包（数据接收）

USB 外设处理这两个令牌采用差不多相同的方法；处理 SETUP 包时的不同点在后面关于控制传输的内容中详细介绍。当接收到一个 OUT/SETUP 包标识符时，如果地址跟一个有效端点匹配，USB 外设就访问与被寻址端点相关的缓冲器描述表项内部的 ADDRn_RX 和 COUNTn_RX 位置的内容。ADDRn_RX 的内容被直接存储到它的内部寄存器 ADDR 中。而 COUNT 此时被清除，从 COUNTn_RX 内容之中读取的 BL_SIZE 和 NUM_BLOCK 位区域的值用来初始化 BUF_COUNT，它是一个内部 16 位计数器，用来检查缓冲器过载状态（所有的这些内部寄存器都不能用软件访问）。USB 外设随后接收到的数据字节被打包成字（接收到的第一个字节被存储为低字节），然后被传送到从内部 ADDR 寄存器里包含的地址开始的包缓冲区。每次字节传输时，BUF_COUNT 递减，COUNT 递增。当探测到 DATA 包的结尾时，测试接收到的 CRC 的正确性，仅当接收过程没有错误

时，才将一个 ACK 握手信息包送回发送主机。万一出现 CRC 错误或其他种类的错误（位填充违规，帧错误，等等），至少在错误检出点之前，数据字节仍然会被复制到包存储缓冲器之中，但是 ACK 包不会被送出，且 USB_ISTR 寄存器中的 ERR 位被置位。然而，在这一情形下通常不需要软件的动作：USB 外设从接收错误中恢复，并且仍然准备好下一次传输的到来。如果被寻址端点无效，一个 NAK 或 STALL 握手包将代替 ACK 被发送，这要取决于 USB_EpnR 寄存器中的 STAT_RX 位，并且不会有数据写入接收存储缓冲器中。

接收存储缓冲器被写入的位置从 ADDRn_RX 中包含的地址开始，写入的字节数对应于收到数据包的长度，还包括 CRC（也就是数据有效负载长度+2），或者一直写到由 BL_SIZE 和 NUM_BLOCK 定义的、预先分配的存储器的最后一个位置，两者任一个满足时写入即停止。通过这种方式，USB 外设写入从不会被超过被分配的接收存储缓冲区域的边界。如果数据包有效负载（应用使用的实际字节数）的长度超过分配的缓冲器大小，USB 外设将检测出一个缓冲器过载状态。这种情况下，将发送一个 STALL 握手包而不是通常的 ACK，以此向主机通报这一问题，而且不产生中断信号，认为传输失败。

当传输正确地完成时，通过发送 ACK 握手包，内部的 COUNT 寄存器被复制回缓冲器描述表项内的 COUNTn_RX 位置中，而 BL_SIZE 和 NUM_BLOCK 域不被影响，通常它们不需要被重写。USB_EPnR 寄存器按照如下方式被更新：DTOG_RX 位被取反，通过设置 STAT_RX =“10”（NAK）使端点无效，同时 CTR_RX 位被置位。如果传输由于错误或缓冲器过载而失败，任何前面列举的动作都不会发生。应用软件必须首先辨认请求微控制器关注的端点，这是通过检查寄存器 USB_ISTR 中的位 EP_ID 和 DIR 来进行的。对 CTR_RX 事件的服务，首先要确定事务的类型（根据 USB_EPnR 寄存器中的 SETUP 位）；应用软件必须将中断标志位清零，并且从与被处理端点关联的缓冲器描述表项内部的 COUNTn_RX 位置读取接收字节的数目。处理完接收到的数据之后，应用软件应该将 USB_EPnR 中的 STAT_RX 位设置为“11”（有效），允许下次传输事务。当 STAT_RX 位等于“10”（NAK），任何寻址为该端点的 OUT 请求都被用 NAK 回应，标志出一种流控制状态：USB 主机将重试传输直至成功为止。接收过程的操作序列必须严格按照上述顺序执行，这样才能避免紧跟在触发 CTR 中断的 OUT 事务之后、寻址到同一端点的第二次 OUT 处理的通知被丢失。

14.4.4.6 控制传输

控制传输的组成包括一个 SETUP 传输，跟随着零个或多个数据阶段（均为同一方向），再跟一个状态段（一个零字节的反方向传输）。SETUP 传输仅由控制端点处理，与 OUT 传输（数据接收）非常相象，除了被寻址端点寄存器的 DTOG_TX 和 DTOG_RX 位的值分别设置为 1 和 0，用来启动控制传输，以及 STAT_TX 和 STAT_RX 都设置为“10”（NAK），以便让软件根据 SETUP 的内容决定随后的传输必须是 IN 或 OUT。控制端点必须在每一个 CTR_RX 事件时都检查 USB_EPnR 寄存器中的 SETUP 位，以便从 SETUP 传输中区别出正常的 OUT 传输来。USB 设备可以通过解释在 SETUP 阶段中传输的数据来决定数据阶段的数量和方向，发生错误的要求对该传输事务发出 STALL。为了实现这一目的，最后一个字段前的所有字段在其不使用的方向上都应该被设置为 STALL，这样如果主机过早反转了传输方向的话，它将收到一个 STALL 状态段。当启动最后一个数据段时，反方向应该被设置为 NAK，这样如果主机立即反转了传输方向（以执行状态段）的话，它将一直等待控制操作的完成。如果控制操作成功完成，软件将 NAK 改为 VALID，否则将改为 STALL。与此同时，如果状态阶段将是一个 OUT，就应该置位 STATUS_OUT（寄存器 USB_EpnR 中的 EP_KIND）位，这样如果状态阶段事务不是以零字节数据传输时就会报告一个错误。当状态事务被服务时，应用软件将 STATUS_OUT 位清零，将 STAT_RX 设置为 VALID（以便能接受新的命令），并设置 STAT_TX 为 NAK（以延迟可能紧随在下一次 SETUP 后的状态阶段）。

由于 USB 规范规定 SETUP 包不能被 ACK 以外的握手响应，最终将放弃先前发布的命令而开始新的命令，所以 USB 逻辑不允许控制端点使用 NAK 或 STALL 包来回应从主机接收到的 SETUP 令牌。

当 STAT_RX 位被设定为“01”（STALL）或“10”（NAK）而接收到 SETUP 令牌时，USB 接受数据，完成要求的数据传输，并发送回一个 ACK 握手。如果那个端点有一个之前发布的 CTR_RX 请求还没被应用软件确认（也就是 CTR_RX 位仍然是由前一个已完成的接收来设定的），USB 会丢弃这个 SETUP 传输事务，并且无论它的状态怎样都不会用任何握手包去应答，这样就模拟成一个接收错误，强制主机再发送一遍 SETUP 令牌。这样做是为了避免丢失紧跟在触发了 CTR 中断的事务之后、寻址到同一端点的 SETUP 事务的通知。

14.4.5 双缓冲端点

USB 标准定义的所有不同的端点类型代表着不同的流量模型，描述了不同种类数据传输操作的典型要求。当大量的数据要在 PC 主机与 USB 功能设备之间传输时，批量端点类型是最合适的模式。这是因为主机对批量传输的调度的方式是填满一个帧内的所有可用带宽，因此只要 USB 功能准备好处理被寻址的批量传输，就可实现最高的实际传输速率。当下一次传输到来时，如果 USB 功能仍然被之前的传输所占用，它将用 NAK 握手信号去应答，PC 主机将重复发布同样的传输一直到 USB 功能设备做好处理的准备，这样就由于带宽被重新发送所占据，降低了实际的传输速率。为此，一个称为“双缓冲的”专用功能可以用在批量端点中。

当“双缓冲”功能被激活时，USB 数据包奇偶切换（data toggle）的顺序就被用来选择，哪个缓冲区被 USB 外设用来执行要求的数据传输。将同时使用“发送”和“接收”包存储区域来管理每一次成功传输的缓冲区切换，以便当 USB 外设填充一个缓冲器时，总会有另一个完整的缓冲器可以由应用软件使用。例如：在一个指向一个“接收”双缓冲批量端点的 OUT 传输过程中，当一个缓冲器正在被来自 USB 主机的新数据填充时，另一个可被微控制器软件使用（对于“发送”双缓冲批量端点以及 IN 传输也是同样的道理）。

因为切换缓冲管理需要使用所有的 4 个缓冲器描述表位置来放置地址指针和所分配的存储缓冲器的长度，用来实现双缓冲批量端点的 USB_EPnR 寄存器被强制作为单方向端点寄存器。因此只有一对 STAT 位必须被设定为不同于“00”（无效）的值：如果启动双缓冲批量端点的接收功能，使用 STAT_RX；如果启动双缓冲批量端点的发送功能，用 STAT_TX。如果需要同时启动双缓冲批量端点的发送和接收功能，则必须使用两个 USB_EPnR 寄存器。

为了充分利用双缓冲功能以达到可能的最高传输速率，前面内容中描述的端点流控制结构必须进行修改，目的是只有在 USB 外设与应用软件之间发生缓冲器冲突的时候才将端点状态转换为 NAK，而不是在每次成功传输之后就这样做。当前正在被 USB 外设使用着的存储缓冲器由与端点方向相关的 DTOG 位定义：“接收”双缓冲批量端点为 DTOG_RX（寄存器 USB_EPnR 的第 14 位），“发送”双缓冲批量端点为 DTOG_TX（寄存器 USB_EPnR 的第 6 位）。为了实现新的流控制方案，USB 外设应该知道哪个包缓冲区目前正在被应用软件所使用，以便知悉任何的冲突。由于在寄存器 USB_EPnR 中有两个 DTOG 位而只有一个被 USB 外设用于数据和缓冲器的排序（由于双缓冲功能所要求的单方向限制），另外一个可被应用软件用来显示目前哪个缓冲器正在被使用。这个新的缓冲器标志叫做 SW_BUF。在下面的表格中说明了对于 “发送”和“接收”双缓冲批量端点，寄存器 USB_EPnR 的位与 DTOG/SW_BUF 定义之间的对应关系。

表 48 双缓冲缓冲器标志定义

Buffer flag	‘Transmission’ endpoint	‘Reception’ endpoint
DTOG	DTOG_TX (USB_EPnR bit 6)	DTOG_RX (USB_EPnR bit 14)
SW_BUF	USB_EPnR bit 14	USB_EPnR bit 6

正在被 USB 外设使用着的存储缓冲器由缓冲器标志 DTOG 定义，而正在被应用软件使用着的缓冲器由缓冲标志 SW_BUF 来标识。缓冲标志的值与所使用的包缓冲区之间的关联在这两种情况中

都一样，列举如下表：

表 49　双缓冲存储缓冲器用法

DTOG or SW_BUF bit value	Packet buffer used by **USB Peripheral** (DTOG) or application software (SW_BUF)
0	ADDRn_TX / COUNTn_TX buffer description table locations.
1	ADDRn_RX / COUNTn_RX buffer description table locations.

批量端点的双缓冲功能的激活是通过：

- 将寄存器 USB_EpnR 中的 EP_TYPE 位区域写为“00”，将端点定义为批量，且
- 在同一寄存器中，设定 EP_KIND 位为“1”（DBL_BUF）。

应用软件负责根据所用的第一个缓冲器对 DTOG 和 SW_BUF 位进行初始化；这样做必须考虑到这两位特殊的只能被切换的性质。在设置 DBL_BUF 之后的第一次传输的结尾触发了特殊的双缓冲批量端点流控制，这种控制被寻址这一端点的所有其他传输所使用，直到 DBL_BUF 设置不再保持。在每次传输的结尾，依据设定的方向，被寻址端点寄存器 USB_EPnR 的 CTR_RX 或 CTR_TX 位被置位。同时，受影响的寄存器 USB_EpnR 的 DTOG 位被硬件取反，使得 USB 外设缓冲器的切换完全独立于软件。与普通的传输和在 DBL_BUF 设定之后的第一次传输不同，两个 STAT 位不受传输终止的影响，它的值一直保持在“11”(Valid)。然而，当接收到新传输的令牌包时，如果检测到 USB 外设与应用软件之间的缓冲冲突（这种状态的识别是 DTOG 和 SW_BUF 的值相同），实际的端点状态将被屏蔽为“10”(NAK) 。应用软件对 CTR 事件通告的响应是清除中断标志并且开始对完成的传输进行所需要的处理。当应用对包缓冲区的使用结束时，软件切换 SW_BUF 位，即在这一位写入“1”，来通知 USB 外设这一缓冲器可以使用了。这样，被 NAK 应答的传输的数量只取决于应用软件对被传输数据的处理时间：如果处理时间比在 USB 总线上完成一次传输的时间短，就不会出现由于流控制引起的重新传送，实际的传输速率将只受到 PC 主机的限制。

应用软件总是能够覆盖掉对于双缓冲的批量端点实现的特殊流控制，只要在相关的 USB_EPnR 寄存器的两个 STAT 位写入一个明确的不同于“11”（Valid）的状态就可以做到。在这种情况下，USB 外设将总是使用被编程的端点状态，而不管缓冲器的使用状况。

14.4.6 等时传输

USB 标准支持一种需要固定和精确的数据产生/消耗频率的全速外设，将这种类型的传输定义为“等时”。这种数据流的典型例子为：音频采样，压缩的视频流，以及一般的任何一种对于供给频率的精确性都有着严格的要求的采样数据。当一个端点在枚举阶段被定义为“等时端点”时，主机将在每帧分配所需要的带宽，并在每一帧都根据端点的方向精确地发送一个 IN 或 OUT 信息包。为了限制对带宽的需求，等时传输中失败的传输是不会重新发送的；这将使得等时传输没有握手阶段，数据包之后也不需要发送或等待 ACK 包。由于同样的原因，等时传输不支持数据奇偶切换（toggle sequencing），并总是使用包标识符 DATA0 来启动所有数据包。

端点的等时传输行为是通过设定 USB_EPnR 寄存器的 EP_TYPE 位为“10”来选择的；由于没有握手阶段，STAT_RX/STAT_TX 位的唯一合法值只能是“00”（Disabled）和“11”（Valid），其他的值将会产生不符合 USB 标准的结果。等时端点实现双缓冲使得应用软件开发更容易，此时“发送”和“接收”包存储区域都被用来管理每一次成功传输的缓冲交换，以便当 USB 外设填充一个缓冲器的时候，总有另一个完整的缓冲器可以被应用软件所使用。

当前被 USB 外设使用着的存储缓冲区由与端点方向有关的 DTOG 位来定义（在相关的寄存器 USB_EPnR 中，DTOG_RX 用于“接收”等时端点，DTOG_TX 用于“发送”等时端点），如表 50 所示。

表 50 等时存储缓冲区的用法

DTOG bit value	DMA buffer used by **USB Peripheral**	DMA buffer used by application software
0	ADDRn_TX / COUNTn_TX buffer description table locations.	ADDRn_RX / COUNTn_RX buffer description table locations.
1	ADDRn_RX / COUNTn_RX buffer description table locations.	ADDRn_TX / COUNTn_TX buffer description table locations.

与双缓冲的批量端点的情况一样，过去执行等时传输的寄存器 USB_EPnR 被强制作为单方向端点来使用。如果需要等时端点能够同时接受和发送，那么就必须使用两个 USB_EPnR 寄存器。

应用软件负责根据第一个要被使用的缓冲器来初始化 DTOG 位；这一动作必需要考虑到这两位特殊的只能被切换的特性。在每次处理的末尾，要根据允许的方向来置位被寻址端点的 USB_EPnR 寄存器的 CTR_RX 或 CTR_TX 位。同时，USB_EPnR 寄存器中受到影响的 DTOG 位由硬件切换，使得缓冲器的切换完全不依靠于软件。STAT 位不受传输完成的影响。由于没有握手阶段等时传输不能进行流控制，端点一直保持“11”（Valid）。在等时 OUT 传输中发生的 CRC 错误或者缓冲过载情况不论在什么情况下都被认为是正确的传输，并且他们总是触发一个 CTR_RX 事件。然而，CRC 错误无论如何都会置位 USB_ISTR 寄存器中的 ERR 位，通知软件注意可能的数据损坏。

14.4.7 挂起/ 恢复事件

USB 标准定义了一种特殊的外设状态，叫做挂起（SUSPEND），此时 USB 总线上吸收的平均电流必须不超过 500μA。这一要求对于总线供电设备是至关重要的，但是自供电设备不需要服从这个严格的功率消耗限制。在挂起模式下，PC 主机在 USB 总线上发送挂起通知的方式是通过在 USB 总线上不发送任何位流超过 3 毫秒：由于 SOF 包在正常操作中必须每毫秒发送一次，USB 外围设备如果检测到缺少连续的三个 SOF 信息包则认为是主 PC 机的一个挂起请求，并且设定 USB_ISTR 寄存器中的 SUSP 位为“1”，如果允许了中断话就会产生一个中断。设备一旦被挂起，也能够通过一个称为恢复（RESUME）的过程来还原其正常操作，这一过程可以从 PC 主机启动或者是直接从外设本身启动，但是总是由 PC 主机终止。无论怎样，挂起的 USB 设备必须能够检测出一个 RESET 的过程，并且把这事件作为正常的 USB 复位事件对其做出反应。

下面给出对典型的挂起步骤的简要描述，重点讨论负责响应 USB 外设 SUSP 通知的应用软件程序中与 USB 相关的方面：

1. 设定寄存器 USB_CNTR 中的 FSUSP 位为“1”。这个动作是为了在 USB 外设内部激活挂起模式。只要挂起模式被激活，对 SOF 接收的检查就马上被禁止，防止在 USB 挂起过程中出现任何新的 SUSP 中断。
2. 消除或减少 USB 外设以外的模块中的任何静态功率消耗。
3. 设定 USB_CNTR 寄存器中的 LP_MODE 位为“1”，以消除在模拟 USB 收发器中的静态功率消耗，但是保持他们检测恢复动作的能力。
4. 可选择地关掉外部振荡器和设备的 PLL 来停止任何设备内部的活动。

在设备处于 SUSPEND 模式时，如果有 USB 事件发生，必须激活 RESUME 过程来恢复正常的时钟以及重新获得正常的 USB 行为。当唤醒事件是 USB 复位时序时，特别要注意确保这一过程所花费的时间不超过 10 毫秒（详见“通用串行总线规范”）。在 USB 外设被挂起的状态下，一个恢复或者是复位时序的启动会将 USB_CNTR 寄存器中的 LP_MODE 位异步地清零。即使这一事件能够触发一个 WKUP 中断（如果被允许的话），中断响应程序的使用必须被谨慎地考虑，这是因为系统时

钟的重新启动需要很长的等待时间；为了在恢复正常的时钟之前有少一点的时间延迟，建议将恢复程序紧挨着放在挂起程序的后面，这样只要系统时钟重新启动，它的代码就会立即执行。为了防止静电释放或者任何其他种类的噪声把系统唤醒（挂起模式的退出是一个异步事件），对数据线状态的一个适合的模拟滤波器在挂起过程中被启动，滤波器的宽度大约为 70 纳秒。

下面是恢复程序应当完成的一系列动作：

1. 可选择地接通外部振荡器和设备的 PLL。
2. 将 USB_CNTR 寄存器中的 FSUSP 位清零。
3. 如果需要辨别恢复触发事件，寄存器 USB_FNR 中的 RXDP 和 RXDM 位可以根据表 51 被使用，表中还列举了在这些事件中软件的应做的动作。如果需要的话，恢复或复位时序的末尾可被检测到，为此要监视上述位的状态，检查它们何时达到“10”组合，该组合代表了空闲总线状态；此外，在复位时序的末尾，USB_CNTR 寄存器中的 RESET 位被置“1”，如果允许的话将发布一个中断，这个中断也应照常被处理。

表 51 恢复事件的检测

[RXDP,RXDM] Status	Wake-up event	Required resume software action
"00"	Root reset	None
"10"	None (noise on bus)	Go back in Suspend mode
"01"	Root resume	None
"11"	Not Allowed (noise on bus)	Go back in Suspend mode

作为对与 USB 协议没有直接联系的特殊事件的响应，设备也许需要退出挂起模式（例如鼠标移动会唤醒整个系统）。在这种情况下，可以启动恢复时序的方式为：设定 USB_CNTR 寄存器的 RESUME 位为“1”，在经过 1mS 到 15mS 的时间间隔后再将该位清为 0（这一时间间隔能够用 ESOF 中断来测量，当系统时钟按照正常频率运转时，该中断的周期为 1mS）。一旦 RESUME 位被清零，PC 主机将会完成恢复时序，其结束也可以用 USB_FNR 寄存器的 RXDP 和 RXDM 位来监视。

注意 RESUME 位无论如何都只能在 USB 外设已经处于挂起状态之后才能被使用，此时设定了寄存器 USB_CNTR 中的 FSUSP 位为 1。

14.5 寄存器描述

USB 外设寄存器可以划分成如下几组：

- 公用寄存器：中断和控制寄存器
- 端点寄存器：端点配置和状态
- 缓冲器描述表：包存储器的位置，用于定位数据缓冲器。

所有寄存器的地址都表示成相对于 USB 外设寄存器基地址 0xC000 8800 的偏移量，但缓冲器描述表的位置除外，它是起始于由寄存器 USB_BTABLE 指定的地址。由于受到 APB 桥的字寻址能力的共同限制，所有的寄存器尽管都是 16 位宽的，但其地址也都被对齐到 32 位的字边界。同样的地址对齐方式也被用于访问包缓冲存储器区域 PMA，其地址从 0xC000 8000 开始。下面的缩写词将在本节中用到：

read/write(rw)　　软件可以读和写这些位。
read-only(r)　　软件只能读这些位
write-only(w)　　软件只能写这些位

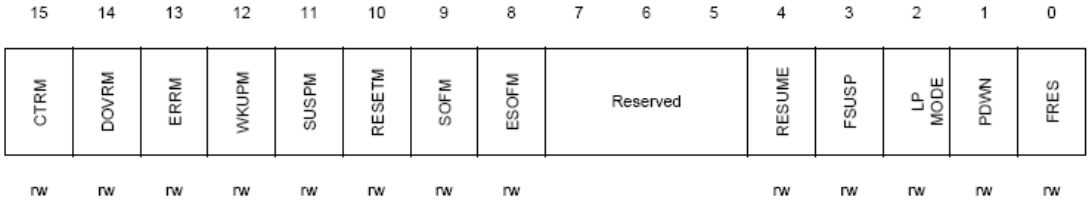
read-clear(rc_w0)	软件只能读，或写’0’清除这一位，写’1’无效
toggle(t)	软件只能写’1’来对这一位取反。写’0’无效
PMA	包存储区域

14.5.1 公共寄存器

这些寄存器影响 USB 外围设备定义的操作命令的基本行为，中断操作，设备地址以及与当前最新的 PC 机相连接的接口。

14.5.1.1 USB 控制寄存器（USB_CNTR）

地址偏移：40h
复位值：0000 0000 0000 0011（0003h）



位 15 CTRM: 正确的传输中断掩码
0：正确传输中断（CTR）被禁止
1：CTR 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 14 DOVRM：DMA 上溢/下溢中断掩码
0：DOVR 中断被禁止
1：DOVR 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 13 ERRM：错误中断码
0：ERR 中断被禁止
1：ERR 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 12 WKUPM：唤醒中断掩码
0：WKUP 中断被禁止
1：WKUPM 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 11 SUSPM: 挂起模式中断掩码
0：挂起模式请求（SUSP）中断被禁止
1：SUSP 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 10 RESETM：USB 复位中断掩码
0：RESET 中断被禁止
1：RESET 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 9 SOFM：帧起始(SOF)中断掩码
0：SOF 中断被禁止

1：SOF 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 8　ESOFM：预期帧起始(ESOF)中断掩码

0：ESOFM 中断被禁止

1：ESOFM 中断允许，当 USB_ISTR 寄存器中相应位被置 1 时，产生中断请求。

位 7-5　保留位

这些位是保留位，他们永远被读为‘0’，必须写入‘0’。

位 4　RESUME：恢复请求

微控制器可以对此进行置位来送出一个恢复信号给主机。根据 USB 规范，恢复信号的激励应保持不少于 1ms，不长于 15ms。在此之后，PC 主机就准备好驱动恢复时序直到其结束。

位 3　FSUSP：强制挂起

当接收到 SUSP 中断时，软件必须将这一位置位。当 3ms 内 USB 外围设备没有接收到位流后，发布 SUSP 中断。

0：不起作用

1：进入挂起模式。在模拟收发器中的时钟和静态功率消耗不受影响。如果对挂起时功率消耗有要求（总线供电设备），应用软件在 FSUSP 之后应该对 LP_MODE 位进行置位，如下面所解释的。

位 2　LP_MODE：低功耗模式

当挂起模式限制要求（除了必须供电给外部上拉电阻器外）消除所有的静态功率消耗时，应使用这个模式。当应用软件已经准备好停止所有的系统时钟，或者降低它们的频率来满足 USB 挂起情况下的功率消耗要求时，应当进入这种模式。在挂起模式下 USB 的活动（WKUP 事件）会异步地清除这一位（它也可以被软件清除）。

0：没有低功耗模式

1：进入低功率模式。

位 1　PDWN：电源关闭

如果出于某种原因需要彻底关闭 USB 外设，这一位用来关断所有 USB 相关的模拟部件的电源。当这一位被置位时，USB 外设与收发器分离且不能被使用。　　0：退出电源关闭

1：进入电源关闭模式。

位 0　FRES：强迫 USB 复位

0：清除 USB 复位

1：强迫 USB 外设复位，与 USB 总线上的一个复位信号作用相同。USB 外围设备保持复位状态直到软件清除此位。如果允许，将会产生“USB 复位”中断。

14.5.1.2 USB 中断寄存器（USB_ISTR）

地址偏移：44h

复位值：0000 0000 0000 0000（0000h）

这个寄存器包含所有中断源的状态，允许应用软件用来确定哪些事件引起了一个中断请求。

这个寄存器的高位部分包含独立的位，每一位代表一个特定事件。当相关的事件发生时，这些位被硬件置位；如果 USB_CNTR 寄存器中的相应位被置位了，就会产生一个中断。中断程序将检查每一位，完成所有必要的动作，最后会清除已经服务过的位。如果它们中任何一位没被清除，中断将仍被认为是等待处理状态的，中断线也会再次变高。如果有几位被同时置位，将只有一个中断产生。

对端点传输事务完成的处理有几种方式来减少中断响应延迟。当一个端点成功完成一个传输时，CTR 位将会立即被硬件置位；同时，如果 USB_CNTR 寄存器的相应位被置位则会产生一个中断请求。一个端点专用中断的触发条件将独立于 USB_CNTR 寄存器的 CTRM 位。两种中断条件将会一直保持到软件清除了 USB_EPnR 寄存器的相应的等待位（CTR 位实际为只读位）。USB 外设有两个中断请求线：

- 优先级较高的 USB IRQ：端点中断请求在等待，这些端点有优先级高的任务（等时和双缓冲批量传输）且它们不能被屏蔽。
- 优先级较低的 USB IRQ：所有其它的中断条件，这些可以是与低优先级传输事务相关的不可屏蔽中断请求，或者是所有其他由 USB_ISTR 高字节通告的可屏蔽事件。

对于和端点相关的中断，软件可以用 DIR 和 EP_ID 只读位来辨别哪一个端点产生了最近的中断请求，并调用相应的中断服务程序。

用户可以选择同时等待的 USB_ISTR 事件的相对优先级，方法是指定中断服务程序中软件检查 USB_ISTR 中位的顺序。之后只把对应于处理完事件的位清除。在服务程序结束时，另外一个中断将请求处理剩余的事件。

为避免一些位被虚假清除，推荐用一个加载指令来清除它们，指令中不需要改变的位都写“1”，所有要被清除的位都写“0”（这些位只能被软件清除）。读—修改—写周期应该被避免，因为在读操作与写操作之间，另一位可能会被硬件置位，在微处理还没来得及处理它之前，下一次的写操作可能会把它清除掉。

下面详细描述每一位：

位 15　CTR：正确传输

这个位被硬件置位，来指示一个端点成功的完成了一个传输；用 DIR 和 EP_ID 位软件可以确定哪个端点请求了中断。这一位是只读的。

位 14　DOVR：DMA 上溢/下溢

如果微处理器没能够及时响应 USB 存储器的请求，这一位置位。USB 外设按下面方法处理这一事件：在接收过程中，不发送 ACK 握手包；发送过程中，在发送位流中强迫发送一个位填充错误；在这两种情况下，主机都会重新进行传输事务。在正常操作中，DOVR 中断不应再发生。因为失败的事务会被主机重试，所以在对此中断处理时应用软件有机会来加速对设备的操作，准备好下一次的重传。但这在等时任务时是不会发生的（等时传输不会被重试），因此导致在这种情况下的数据丢失。这一位是读/写位，但只可写 ‘0’，写‘1’时不起作用。

位 13　ERR：错误

每当下面错误之一发生时，这一位都会被置位：

NANS：没有应答。主机响应的时间限已经超过。
CRC：循环冗余校验错误。接受到的令牌或数据中的 CRC 有错误。
BST：位填充错误。位填充错误被检测出，在 PID、数据和/或 CRC 中的任何位置。
FVIO：成帧格式违规。一个非标准帧被接受（EOP 不在正确的位置上，错误的令牌顺序等）。
USB 软件通常可以忽略错误，因为 USB 外围设备和 PC 主机以完全透明的方式管理出错重发。这个中断可以用在软件开发阶段，或用来监控 USB 总线上传输的质量，或用于向用户提示可能的问题（比如连接器松动，环境噪声太重，USB 电缆中断线等）。这一位是读/写位，但只可写 ‘0’，写‘1’时不起作用。

位 12 WKUP：唤醒
在挂起模式下，检测到唤醒了 USB 外围设备的活动时，这一位会被硬件置为‘1’。这一事件异步地清除了 CTLR 寄存器中的 LP_MODE 位，并激活 USB_WAKEUP 信号线，该信号可以被用来通报设备中其他模块（如唤醒单元）恢复过程的开始。这一位是读/写位，但只可写 ‘0’，写‘1’时不起作用。

位 11 SUSP: 挂起模式请求
当有 3ms 在总线上没有流量时，这一位被硬件置位，标志着一个来自 USB 总线的挂起模式请求。当 USB 复位后，挂起条件的检查功能立即被启动，当挂起模式被激活（FSUSP=1）时检测功能被硬件关闭，直到恢复时序结束时再被打开。这一位这一位是读/写位，但只可写 ‘0’，写‘1’时不起作用。

位 10 RESET：USB 复位请求
当 USB 外围设备在它的输入端探测到一个有效的 USB 复位信号时这一位被置位。USB 外设在响应复位时，复位它的内部协议状态机，如果在 USB_CNTR 寄存器中 RESETM 位被置位，则产生一个中断。接收和发送被关闭，直到 RESET 位被清除。所有配置寄存器不被复位：微控制器必须显示地清除这些寄存器（这是为了保证 RESET 中断可以被安全的发送，紧跟着复位的任何任务可以被完成）。USB 功能地址和端点寄存器可以被 USB 复位事件清除。
这一位这一位是读/写位，但只可写 ‘0’，写‘1’时不起作用。

位 9 SOF：帧起始
这一位通知一个新的 USB 帧的开始，当一个 SOF 包通过数据总线到达时，这一位被置位。中断服务程序可以监控 SOF 事件使其对主机有 1ms 的同步事件，并安全地读取 USB_FNR 寄存器中的数据，该寄存器在收到 SOF 包时被更新（这对等时应用软件是有用的）。这一位这一位是读/写位，但只可写 ‘0’，写‘1’时不起作用。

位 8 ESOF：预期帧起始
当一个 SOF 包被预期但没收到时这一位被硬件置位。主机每毫秒发送一个 SOF 包，但是如果集线器没有正确地收到它，挂起定时器发布这个中断。如果有三个连续的 ESOF 中断发生（即丢失了三个 SOF 包），且在三个中断之间没有任何总线活动发生，则产生一个 SUSP 中断。即使在挂起定时器还没有被锁定时发生了 SOF 包丢失时，这一位也会被置位。这一位这一位是读/写位，但只可写 ‘0’，写‘1’时不起作用。

位 7-5 保留位
这些位是保留位，他们总是被读为‘0’，必须写入‘0’。

位 4　DIR：传输事务的方向

根据成功传输的方向，这一位被硬件写入，成功的传输会产生中断请求。

如果 DIR 位=0，与产生中断端点相关的 USB_EPnR 寄存器中的 CTR_TX 位被置位。产生中断的事务是 IN 类型（数据由 USB 外围设备发送到主机）。

如果 DIR 位=1，与产生中断端点相关的 USB_EPnR 寄存器中的 CTR_RX 位或 CTR_TX/CTR_RX 两位都被置位。产生中断的事务是 OUT 类型（USB 外设接收到来自主机的数据），或者有两个传输事务在等待处理。

这个信息可以被应用软件用来访问与触发中断的事务相关的 USB_EPnR 位，因为它指示出了等待中断的传输事务的方向。这一位是只读的。

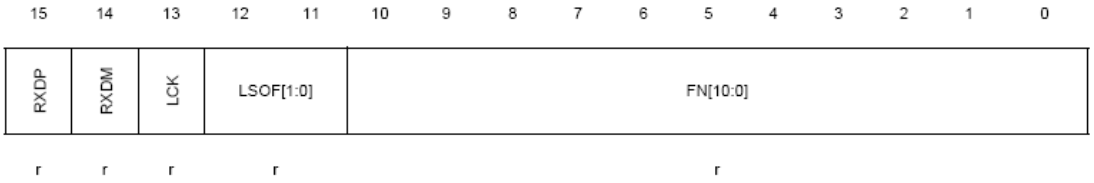
位 3-0　EP_ID[3:0] 端点标识符

这几位由硬件根据产生中断请求的端点编号来写入。如果有多个端点任务等待中断，硬件写入有最高优先级的端点的端点标识符，优先级按照下面的方式定义：先将端点分为两组：按优先级顺序，首先考虑等时和双缓冲器批量端点，然后再检查其它的端点。如果从同一组中有多个端点请求中断，USB_ISTR 寄存器中的 EP_ID 位的赋值将按照最低编号端点优先原则进行，EP0R 优先级最高，接着是 EP1R，等等。应用软件可以能够根据这个优先级顺序，为每个端点指派一个寄存器，以便用合适的顺序排列并发的端点请求。这些位是只读的。

14.5.1.3 USB 帧序号寄存器（USB_FNR）

地址偏移：48h

复位值：0000 0xxx xxxx xxxx（0xxxh）



位 15　RXDP，接收 D+线状态

这一位可以用来观查接收到的上游端口的数据线的 D+线状态。它可以在结束挂起程序中被用来帮助确定唤醒事件。

位 14　RXDM，接收 D-线状态

这一位可以用来观查接收到的上游端口的数据线的 D- 线状态。它它可以在结束挂起程序中被用来帮助确定唤醒事件。

位 13　LCK：锁定

在 USB 复位条件结束后或一个 USB 恢复序列结束后，当至少有两个连续的 SOF 包被接收时，这一位被硬件置位。一旦锁定，帧定时器将保持在这个状态直到 USB 复位或 USB 挂起事件发生。

位 12-11　LOSF[1:0]：丢失 SOF

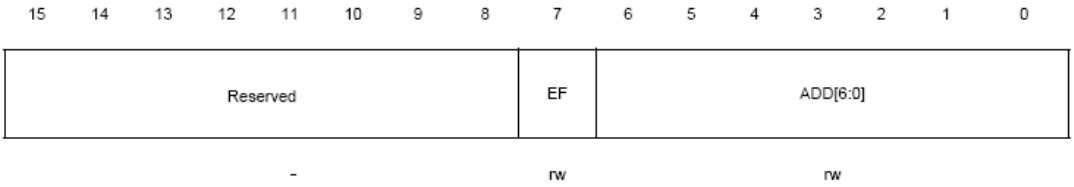
当 ESOF 中断发生时这两位被硬件写入，用来计数连续 SOF 包丢失的个数。在接收到一个 SOF 包时这两位将被清除。

位 10-0　FN[10:0]：帧序号

这个位域包括了包含在最后接收到的 SOF 包中的 11 位帧序号。帧序号在主机每发送一帧时递增，这对等时传输是有用的。当 SOF 中断产生时，这些位被更新。

14.5.1.4 USB 设备地址（USB_DADDR）

地址偏移：4Ch
复位值：0000 0000 0000 0000（0000h）



当从 USB 总线接收到 USB 复位，或通过 USB_CNTR 寄存器中 FRES 位强迫复位时，这个寄存器将会被复位。

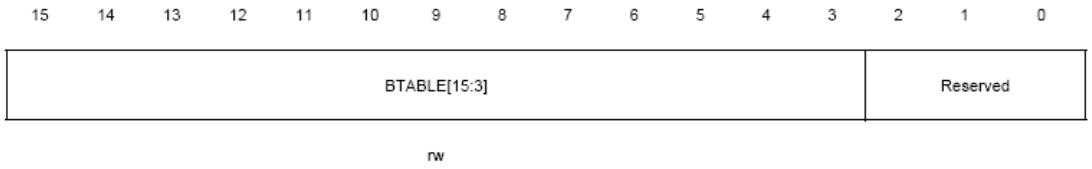
位 15-8　保留位
这些位是保留位，他们总被读为‘0’，必须写入‘0’。

位 7　EF：功能允许
这一位被软件来置位来使能 USB 设备。该设备的地址存放在 ADD[6:0]位。如果这一位是‘0’则没有传输事务被处理，而与 USB_EPnR 寄存器中的设置无关。

位 6-0　ADD[6:0]：　设备地址
这些位包含在枚举过程中被主机分配的 USB 功能地址。本地址域和在相关的 USB_EPnR 寄存器中的端点地址（EA）域必须与一个 USB 令牌子中的信息相匹配，才能处理对要求端点的传输事务。

14.5.1.5 缓冲器表地址（USB_BTABLE）

地址偏移：50h
复位值：0000 0000 0000 0000（0000h）



位 15-3　BTABLE[15:3]：缓冲器表
这些位包含位于专用包存储器内部的缓冲器分配表的开始地址。这个表描述了每个端点缓冲器位置和尺寸，其位置必须是对齐到 8 字节边界（最低 3 位永远为‘0’）。在寻址到这个设备的每个传输事物开始时，USB 外围设备读取这个表中与被寻址端点相关的表项，来获得该端点的缓冲器开始位置和大小（参见“包缓冲器的结构和使用”）。

位 2-0　保留位

这些位是保留位，他们总被读为‘0’，必须写入‘0’。

14.5.2 端点相关的寄存器

STR71x USB 外围设备支持最多 16 个双向端点。每个 USB 设备必须支持一个控制端点，且这个控制端点的地址（EA 位）必须被置为 0。如果多个端点被使能且有相同的端点编号值，USB 外围设备的表现不确定。对于每一个端点来说，一个 USB_EPnR 寄存器可以用来存储端点相关的信息。

14.5.2.1 USB 端点 n 寄存器（USB_EPnR）

地址偏移：00h—3Ch
复位值：0000 0000 0000 0000b（0000h）

当从 USB 总线收到 USB 复位或通过 CTLR 寄存器的 FRES 位强迫 USB 复位时，这些寄存器都会被复位；但 CTR_RX 和 CTR_TX 位除外，这两位保持不变，以避免丢失紧跟在一个 USB 复位事件后的正确的信息包通告。每个端点有它自己的 USB_EPnR 寄存器，其中 n 是端点标识符。

对这些寄存器的读—修改—写周期应该避免，这是因为在读和写操作之间，某些位可能被硬件置位，这样的话下一个写操作会修改了这些变化位，使微处理器没有时间可以探测到这些改变。出于这个目的，所有被这个问题影响的位都有‘不变’值，在不要求对他们进行修改时，必须采用这些‘不变’值。建议采用一个加载指令来修改这些寄存器，该指令中所有只能被硬件修改的位，都被写为其‘不变’值。

位 15　CTR_RX：接收的正确传输

当一个 OUT/SETUP 传输在这一端点成功的完成时，这一位被硬件置位；软件只能清除这一位。如果 USB_CNTR 寄存器中的 CTRM 位因此被置位，一个普通的中断条件产生，并将会伴随着与端点有关的中断条件产生，而这些端点中断都是激活的。所发生的传输事务的类型，OUT 或 SETUP，可以用以下所描述的 SETUP 位决定。

以 NAK（不应答）或 STALL（中止）握手结束的任务不设置这一位，因为没有数据被实际传输，在协议错误或数据同步切换错误时也是同样情况。

这一位是读/写位，但只能写入‘0’，写‘1’时不起作用。

位 14　DTOG_RX：用于接收传输的数据切换

如果端点不是等时的，这一位包含下一个要接收的数据包预期的数据切换（0=DATA0，1=DATA1）位的值。在一个接收到的数据包有一个匹配的数据 PID 值，因而 ACK 握手包被送到 USB 主机后，硬件切换这一位。如果端点被定义为控制端点，在接收到寻址到这个端点的 SETUP PID 时，这一位被硬件清除。

如果端点使用了双缓冲器特性，这一位也被用来支持包缓冲区交换（参见“双缓冲器端点”）。

如果端点是等时的，这一位只用来支持包缓冲器交换，因为对于这类端点不使用数据切换，

只传输 DATA0 包（参见“等时传输”）。当数据包接收刚一结束结束，硬件就切换这一位，因为等时传输中不使用握手信息。

这一位也可以被软件切换来初始化它的值（当端点不是一个控制端点时必须要初始化），或用于强制特殊数据切换或特定包缓冲器的使用。当应用软件写入‘0’时，DTOG_RX 的值保持不变；当写入‘1’时，使得这位的值切换（取反）。这一位可以读/写，但只有在写 1 时可以切换。

位 13-12 STAT_RX[1:0]：状态位，用来接收传输

这两位包含关于端点状态的信息，列于表 52，“接收状态编码”。这两位可以被软件切换来初始化它们的值。当应用软件写‘0’时，值保持不变；当写‘1’时，位值切换。当寻址到该端点的一个 OUT 或 SETUP（只是控制）传输任务正确传输（CTR_RX=1）时，硬件把 STAT_RX 位设置为 NAK；使得软件在确认一个新的事务之前有时间来处理接收到的数据。

双缓冲的批量端点实现一个特殊的传输事务流控制，它根据缓冲器的可用状态来进行控制（参考章节“双缓冲端点”）。

如果端点定义为等时的，其状态只能是“VALID”或“DISABLE”，因此当一个传输成功后，硬件不能更改端点的状态。如果软件对等时端点设置 STAT_RX 位为‘STALL’或‘NAK’，USB 外围设备的行为是不确定的。这些位可以读/写，但只有在写 1 时可以切换。

位 11 SETUP：设置事务完成

这一位只可读，当最后一个完成的传输事务是 SETUP 时，它被硬件置位。这一位只为控制端点改变它的值。在成功接收（CTR_RX 事件）的情况下，必须检查这一位来确定发生的事务的类型。为了保护中断服务程序不会受到下一个来到的令牌引起 SUTUP 位改变的影响，当 CTR_RX 位为‘1’时，这一位保持冻结；当 CTR_RX 位为‘0’时，它的状态可改变。这一位是只读的。

位 10-9 EP_TYPE[1:0]：端点类型

这两位设置这个端点的行为，如表 53“端点类型编码”所描述。端点 0 必须总是一个控制器端点，且每一个 USB 功能至少必须有一个地址为‘0’的控制端点，如果需要的话，可能还有其它的控制端点。只有控制端点才处理 SETUP 传输事务，这些传输事务被其它类型的端点所忽略。SETUP 传输不能用 NAK 或 STALL 来应答。如果控制端点被定义为 NAK 时，在接收时，当接收到一个 SETUP 传输时，USB 外围设备将不会响应，模拟一个接收错误。如果控制端点被定义为 STALL 时，无论如何 SETUP 包将会被接收，传送数据并产生 CTR 中断。即使端点是控制端点，OUT 传输的接收也会被被用正常的方式处理。批量端点和中断端点的行为非常相似，区别只在于使用 EP_KIND 配置位时可以得到的特殊功能不同。等时端点使用的解释见“等时传输”。

位 8 EP_KIND：端点性质

这一位的意义由 EP_TYPE 位设定的端点类型来决定。表 54 总结了不同的含义。

DBL_BUF：这一位被软件置位，用来使能对此批量端点的双缓冲特征。双缓冲的批量端点的使用见“双缓冲端点”。

STATUS_OUT：这一位被软件置位用来指出所预计为状态输出传输：在这种情况下，所有的包含非 0 个数据字节的 OUT 传输事务被响应为‘STALL’而不是‘ACK’。这一位可以被用来提高应用软件对于在控制传输中出现的协议错误的抵抗能力，其使用仅限于控制端点。当 STATUS_OUT 被清除时，OUT 传输事务可以按需要有任何字节数。

位 7　CTR_TX：对发送的正确传输

当端点成功完成了一个 IN 传输时，这一位被硬件置位；软件只能清除这一位。如果 USB_CNTR 寄存器中的 CTRM 位被置位，一个普通的中断条件产生，同时伴随着与端点相关的中断条件，它通常是被激活的。

注意：以 NAK 或 STALL 握手结束的传输事务不对此位进行置位，因为实际中没有数据传输，就像是协议错误和数据切换不匹配一样。

这一位可以读/写，但只可以写入‘0’。

位 6　DTOG_TX：数据切换，用于发送传输

如果端点不是等时的，这一位包含下一个即将发送的数据包的需要的数据切换位的值（0＝DATA0,1=DATA1）。在发送一个包之后，当从 USB 主机接收到 ACK 握手时，硬件切换这一位。如果端点被定义为控制端点，在接收到寻址到这个端点的 SETUP PID 时，硬件把这一位置为‘1’。

如果端点使用双缓冲功能，这一位被用来支持包缓冲器交换（参见 “双缓冲端点”）。

如果端点是等时的，这一位用来支持包缓冲器交换，因为对这类端点没有数据转切换，只有 DATA0 包被发送（参考“等时传输”）。由于等时传输没有使用握手，数据包发送结束后，硬件立即切换这一位。

这一位也可以被软件切换，用来初始化它的值（当端点不是一个控制端点时必须这样做），或用来强制一种特殊的数据切换或包缓冲器的使用。当应用软件写入‘0’时，DTOG_TX 的值保持不变；当写入‘1’时，这位的值切换。这一位可以读/写，但只有在写 1 时可以切换。

位 5:4　STAT_TX [1:0]：状态位，用于发送传输

这两位包含关于端点状态的信息，在表 55 中列出。这两位可以被软件切换用来初始化它们的值。当应用软件写入‘0’时，值保持不变；当写入‘1’时，位的值切换。当与这个端点相关的一个 IN 或 SETUP（仅为控制传输）事务对应的正确的传输发生（CTR_TX＝1）时，硬件把 STAT_TX 位置为 NAK。然后等待软件准备下一组要发送的数据。

双缓冲批量端点实现特殊传输事务流控制，这个控制基于缓冲器的可用状态（参考“双缓冲器端点”）。

如果端点定义为等时的，它的状态只能是“VALID”或“DISABLED”。因此，在成功传输后，硬件不能改变端点的状态。对于等时端点，如果软件把 STAT_TX 位设为‘STALL’或‘NAK’，USB 外围设备的行为是不确定的。这两位是可以读/写的，但是它们只能通过写入‘1’被切换。

位 3-0　EA[3:0] 端点地址

软件必须在这个域写入用来标识指向这个端点的传输事务的 4 位地址。在使能对应的端点之前，必须先赋值。

表 52 接收状态编码

STAT_RX[1:0]	Meaning
00	**DISABLED:** all reception requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all reception requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all reception requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for reception.

表 53．端点类型编码

EP_TYPE[1:0]	Meaning
00	BULK
01	CONTROL
10	ISO

EP_TYPE[1:0]	Meaning
11	INTERRUPT

表 54　端点性质的含义

EP_TYPE[1:0]		EP_KIND Meaning
00	BULK	DBL_BUF
01	CONTROL	STATUS_OUT
10	ISO	Not used
11	INTERRUPT	Not used

表 55．传输状态编码

STAT_TX[1:0]	Meaning
00	**DISABLED:** all transmission requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all transmission requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all transmission requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for transmission.

14.5.3 缓冲器描述符表

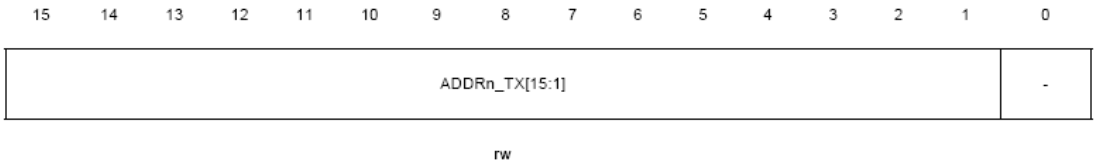
尽管这个表被放在包缓冲存储器内，它的表项可以被认为是附加的寄存器，用来配置用以交换 USB 和 STR71x 之间数据的包缓冲器的位置和大小。由于公共 APB 桥在字寻址方面的限制，所有的包存储器位置都被 APB 用 32 位对齐的地址来访问，而不是 USB 外围设备用来访问 USB_BTABLE 寄存器和缓冲区描述表位置的实际存储器位置地址。在下文中将说明两个位置地址：一个在访问包

存储器时，被应用软件来使用；另一个是和 USB 外围设备访问相关的局部地址。软件在访问包存储器时，为了获得正确的 STR71x 的存储器地址，实际存储器位置的地址必须乘以 2。第一个包存储器位置在 0Xc000 8000。

与 USB_EPnR 寄存器相关的缓冲器描述表项在下面叙述。包缓冲器和缓冲器描述表使用的详细描述见“包缓冲器的结构和使用”。

14.5.3.1 发送缓冲器地址 n（USB_ADDRn_TX）

地址偏移：[USB_BTABLE]*2+n*16
USB 局部地址：[USB_BTABLE]＋n*8

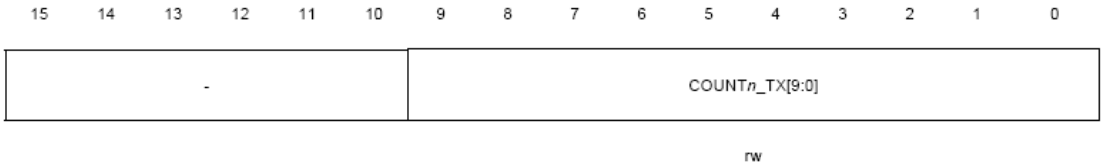


位 15-1　ADDRn_TX[15:1]：发送缓冲器地址
这些位指出包缓冲器的起始地址，该缓冲器包含与 USB_EPnR 寄存器关联的端点在下一个寻址到该端点的 IN 令牌来到时要发送的数据。

位 0　这一位必须被写为‘0’，因为包存储器是 16 位字宽的，且所有的包缓冲器必须是字对齐的。

14.5.3.2 发送字节计数 n（USB_COUNTn_TX）

地址偏移：[USB_BTABLE]*2+n*16+4
USB 局部地址：[USB_BTABLE]＋n*8+2

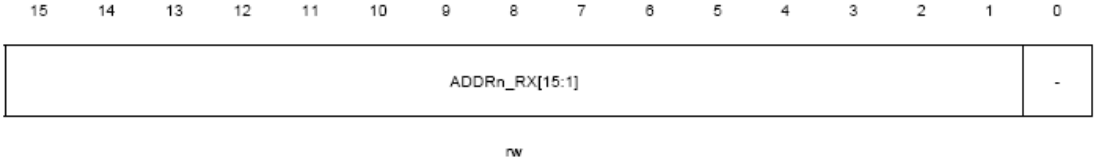


位 15-10　这些位不使用，因为包的大小被 USB 规范限制在 1023 字节内。它们的值不被 USB 外设考虑。

位 9-0　COUNTn_TX[9:0] 发送字节计数
这些位包含与 USB_EPnR 寄存器关联的端点在下一个寻址到该端点的 IN 令牌来到时要发送的字节数。

14.5.3.3 接收缓冲器地址 n（USB_ADDRn_RX）

地址偏移：[USB_BTABLE]*2+n*16+8
USB 局部地址：[USB_BTABLE]＋n*8+4



位 15-1　ADDRn_RX[15:1]：接收缓冲器地址

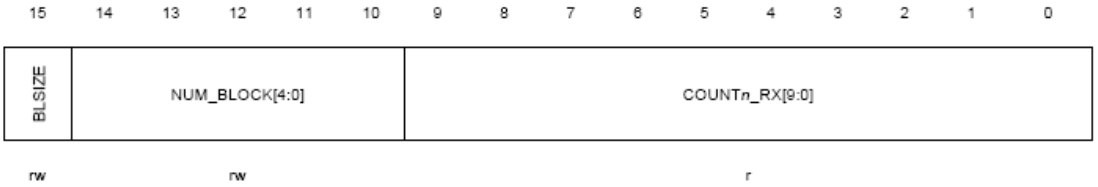
这些位指向的起始地址，这些包缓冲器将存放与 USB_EPnR 寄存器关联的端点在下一个寻址到该端点的 OUT/SETUP 令牌到来时接收到的数据。

位 0　　这一位必须被写为‘0’，因为包存储器是 16 位字宽的，所有的包缓冲器必须是字对齐的。

14.5.3.4 接收字节计数 n（USB_COUNTn_RX）

地址偏移：[USB_BTABLE]*2+n*16+12

USB 局部地址：[USB_BTABLE]＋n*8+6



这个表位置用来储存两个不同的值，在包接收中这两个值都是需要的。高位中包含括已分配缓冲器大小的定义，以便进行缓冲器的溢出探测，而低位部分在接收的结束时被 USB 外围设备回写，用来给出实际接收到的字节数。由于在可用位上的限制，缓冲器大小用分配的存储器块的数目来代表，而块的大小可以选择，用来确定一种在细粒度/小缓冲区与粗粒度/大缓冲区之间的权衡。被分配缓冲器的大小是端点描述符的一部分，通常在枚举过程中依据它的 maxPacketSize 参数值来定义（见“通用串行总线规范”）。

位 15　　BL_SIZE：块大小

这一位选择用来定义分配缓冲器大小的存储器块的大小。

- 如果 BL_SIZE＝0，存储器块是 2 字节大，这是在一个字宽的存储器中允许的最小块。用这种块的大小，分配缓冲器大小的范围从 2－62 字节。
- 如果 BL_SIZE＝1，存储器块是 32 字节大，以此能实现在 USB 规格中允许达到的最大包长度。用这种块的大小，分配缓冲器大小的范围从 32－512 字节，这是 USB 规范中所允许的最大的包尺寸。

位 14:10　　MUN_BLOCK[4:0]：块的数目

这些位定义了分配给包缓冲器的存储器块的数目。实际分配的存储器大小与 BL_SIZE 值有关，如表 56 中所列。

位 9:0　　COUNTn_RX[9:0]：这些位存放与 USB_EPnR 寄存器相关的端点在最后一个寻址到该端点的 OUT/SETUP 传输事务中接收到的字节数。

表 56 确定被分配的缓冲存储器

Value of NUM_BLOCK[4:0]	Memory allocated when BL_SIZE=0	Memory allocated when BL_SIZE=1
0 (‘00000’)	Not allowed	32 bytes
1 (‘00001’)	2 bytes	64 bytes
2 (‘00010’)	4 bytes	96 bytes
3 (‘00011’)	6 bytes	128 bytes
...	...	...
15 (‘01111’)	30 bytes	512 bytes
16 (‘10000’)	32 bytes	not used
17 (‘10001’)	34 bytes	
18 (‘10010’)	36 bytes	
...	...	...
30 (‘11110’)	60 bytes	
31 (‘11111’)	62 bytes	not used

14.6 寄存器映射

为了能找到每个寄存器的正确的地址偏移量，表 57 显示出了所有 USB 外设寄存器的地址映射。

表 57. USB 外设寄存器页映射

Offset	Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USB_EP0R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x04	USB_EP1R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x08	USB_EP2R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x0C	USB_EP3R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x10	USB_EP4R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x14	USB_EP5R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x18	USB_EP6R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x1C	USB_EP7R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x20	USB_EP8R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x24	USB_EP9R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			

表 57. USB 外设寄存器页映射

Offset	Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	USB_EP10R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x2C	USB_EP11R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x30	USB_EP12R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x34	USB_EP13R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x38	USB_EP14R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x3C	USB_EP15R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x40	USB_CNTR	CTRM	DOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved			RESUME	FSUSP	LP	PDWN	FRES
0x44	USB_ISTR	CTR	DOVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	Reserved			DIR	EP_ID[3:0]			
0x48	USB_FNR	RXDP	RXDM	LCK	LSOF [1:0]		FN[10:0]										
0x4C	USB_DADDR	Reserved								EF	ADD[6:0]						
0x50	USB_BTABLE	BTABLE[15:3]													Reserved		

基地址请见表 2 “APB1 存储器映射”。

15 高级数据链路控制器（HDLC）

15.1 主要特性

- 自动标记检测及插入
- 自动零位删除及插入
- 帧校验序列(FCS)自动产生和检测（CRCCCITT）
- 中止检测及发送
- 空闲检测及发送
- 32 位可屏蔽私有地址段识别
- 4 个单字节可屏蔽群地址段识别
- 检测每帧收到的字节数整数
- 多至 16 位可编程前同步序列
- 多至 16 位可编程后同步序列
- 两个独立的输入时钟线（一个在接收，一个在发送）
- NRZ，NRZI，FM0 和曼彻斯特模式都可用在 RX/TX
- 用于时钟恢复的数字锁相环（DPLL）
- 输入 / 输出帧的可补性
- 自动回环模式
- 全双工操作模式
- 2×128 字节 RAM 缓冲区的用于发送和接收

15.2 HDLC 功能描述

15.2.1 HDLC 帧格式

所有的用户信息和协议消息都以帧的形式发送。下图显示 HDLC 的帧格式.

最多 16 位	01111110	32 位	N 字节	16 位	01111110	最多 16 位
前同步信号	开始标志	地址段	数据字节	FCS段	结束标志	后同步信号

HDLC 从最低位开始接收 / 发送一个字节的串行位.

以下是对帧内每个段的介绍：

- 前同步信号

开始标志之前可出现 0 到 16 位。

- 后同步信号

结束标志之后可出现 0 到 16 位。

- 标志

标志是独特的二进制数模式（01111110）。它规定了帧的边界和每个帧段的位置的参考点。单一的标志能被用作一个帧的结束标志和下个帧的开始标志.

- 地址段

跟在开始标志后的段定义为地址段。它的长度大于 32 位。用几个可编程值对地址段进行校验，可确定是否接收这个帧.

- 信息段

信息段在 FCS 段之前。它包含任意字节序列.

- 帧校验序列段（FCS）

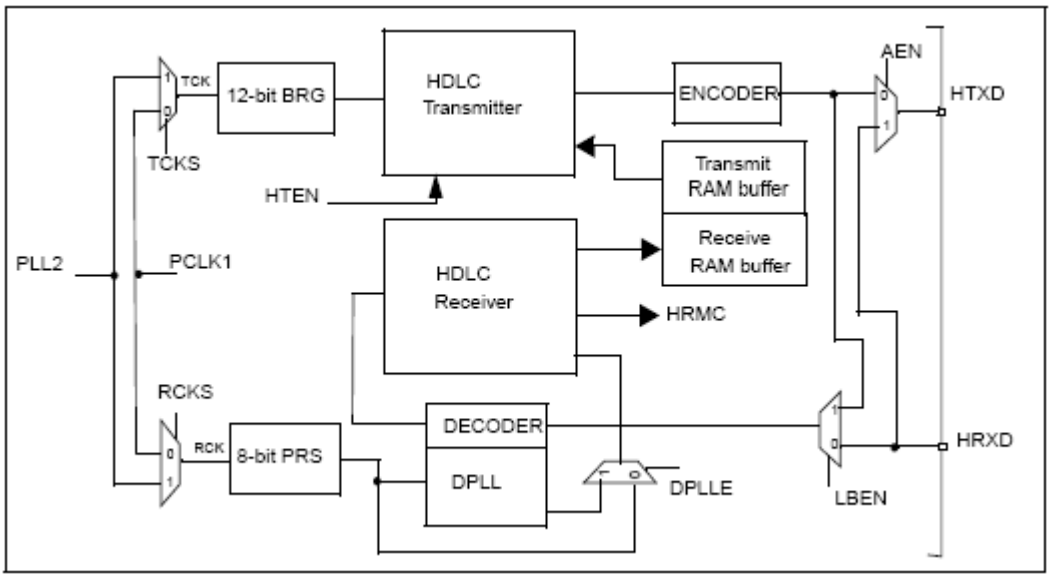
在结束标志之前的 16 位是 FCS 段。FCS 是一种错误检测编码，用帧中（除去标志）余下的位计算出来。使用的编码是循环冗余校验 CCITT（CRCCCITT，X16+X12+X5+1）。

发送和接收的多项式寄存器都可独立的被初始化为”1”或”0”。

15.2.2 基本结构

基本的 HDLC 包含两部分：

一个接收部分（HDLC 接收器）和一个是发送部分（HDLC 发送器）。



注释：

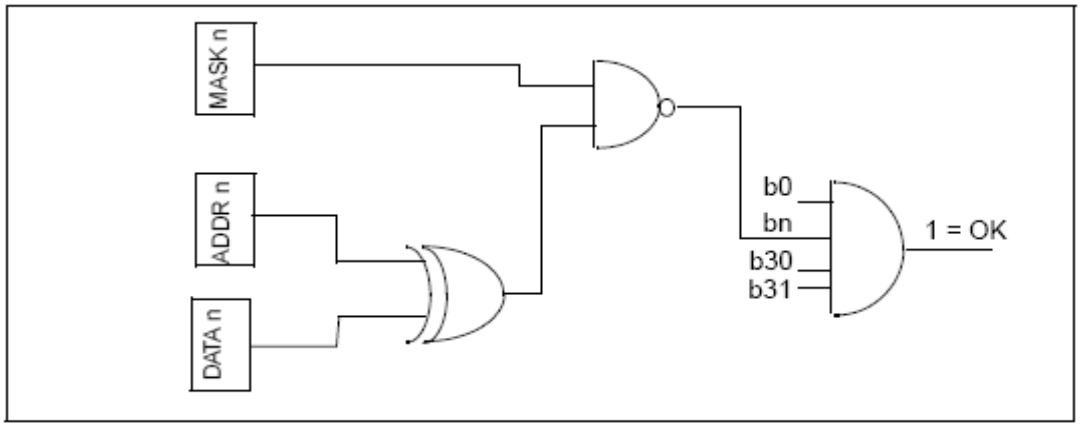
1. HTEN（HDLC 发送使能）信号被内部连接到定时器 Timer2 的输出比较 B（T2.OCMP_B），它允许通过定时器 2 触发发送的开始，允许对这个事件的精确定时。
2. HRMC(HDLC 接收消息完成)信号被内部连接到定时器 2 输入捕捉 B。

15.2.2.1 接收器

接收器部分完成以下基本操作：

- **标志检测**：一个 0 后接连续 6 个 1 和另一个 0，被识别为一个标志。
- **零删除**：在 HDLC 帧里，5 个连续的 1 后面的一个 0 被删除掉。
- **CRC 校验**：CRC 段是根据生成多项式 CRCCCITT 来的。结果显示在帧状态字节寄存器 HDLC_FSBR 的循环冗余校验位（bit1=CRC）。
- **中止检查**：一帧里有 7 个或更多连续的 1 被解释为是终止条件。终止条件的检测结果显示在帧状态字节寄存器 HDLC_FSBR 的接收中止位（bit0=RAB）。
- **空闲检查**：15 或更多连续的 1 被解释为是空闲条件。空闲条件的检测结果显示在 HDLC 状态寄存器 HDLC_PSR 的 RLS 位.
- **地址段识别**:为接收一帧可识别出一个 32 可屏蔽位的私有地址,和 4 组一个可屏蔽字节的地址,其中一个可用作广播地址。如果地址段至少匹配了其中一个地址寄存器的值，这个帧就被接收。匹配是根据对应的掩码值，即，如果掩码寄存器的位 n 是 " 0 " ，接收地址段的位 n 与地址寄存器的位 n 的比较不起作用。图 73 显示了地址过滤机制。如果秘有地址掩码寄存器（HDLC_PAMH 和 HDLC_PAML）被清零，任何输入帧都可被接收。群地址识别和秘有地址必须通过在接收控制寄存器的对应位的设置来打开。被比较的群地址是在开始标志后的第一个字节。

图 73 地址段识别



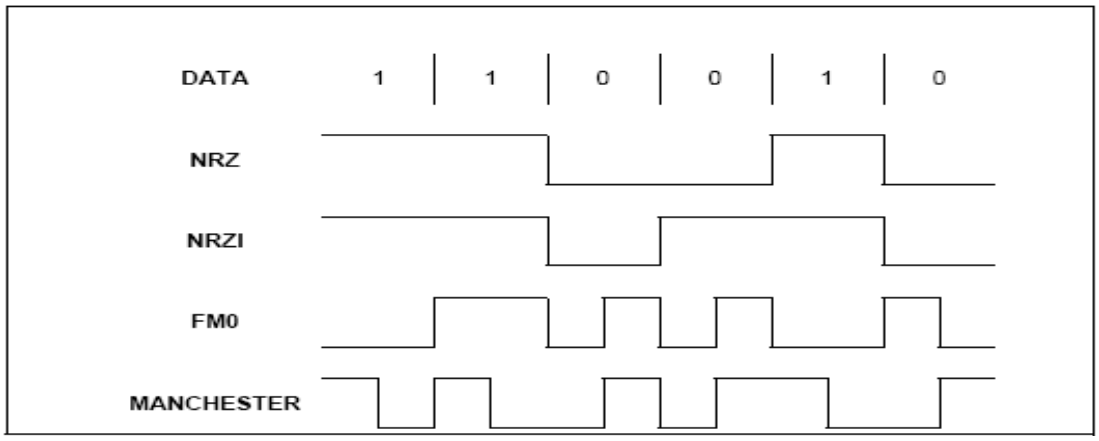
15.2.2.2 发送器

发送器部分执行以下基本操作：

- **前同步产生**：在前同步寄存器中编程的 0 到 16 位值，可在开始标志前发送。前同步位数由发送控制寄存器定义。
  该功能必须通过将发送控制寄存器的 PREE 位置位来使能
- **后同步产生**：在后同步寄存器中编程的 0 到 16 位值，可在结束标志后发送。后同步位数由发送控制寄存器定义。
  该功能必须通过将发送控制寄存器的 POSE 位置位来使能。
- **标志产生**：标志在每个帧的开始和结尾产生。
- **零插入**：在 HDLC 帧五个连续的 1 后面差入一个零。
- **CRC 产生**：发送器帧的 FCS 段根据发生器多项式 CRCCCITT（X16+X12+X5+1）来产生。
- **中止序列产生**：当 HDLC 帧被中止（编程 HDLC 命令寄存器的 TEN 位为 " 0 " ）或当发射过载产生（TDU=1）时，就产生一个中止序列。
- **帧间时间填充**：根据 TCTL 寄存器的帧间时间填充位，空闲状态或标志可以在帧间时间内发送。

15.2.2.3 数据编码/解码及时钟

- **数据编码/解码**：
  根据经典的 NRZ 编码，HDLC 控制器能处理 NZRI，FM0 和 MANCHESTER 编码。（见图 74）。
  图 74．数据编码举例



发送器和接收器编码通过接收控制寄存器的 RCOD 位和发送器控制寄存器的 TCOD 位独立选

择。

- **时钟：**

  在 HDLC 发送器和接收器模块有三种可用的时钟

  - PLL2 输出
  - PCLK1（APB1 时钟）
  - DPLL（数字锁相环）

- **DPLL（数字锁相环）**

  一个内部锁相环允许从正在接收的帧恢复时钟信息。来自预分频器的 DPLL 输入时钟必须比接收器数据速率快 16 倍。

  如果锁相环被关闭（使用在 HDLC_RCTL 寄存器中的 DPLLE 位），HDLC 使用 PLL2 输入时钟，只能工作在 NRZ 和 NRZI 模式。

  当 DPLL 使能时，其内部计数器从 0 到 15 计数（将理想位宽分为 16 个子单元）并生成一个输出时钟。

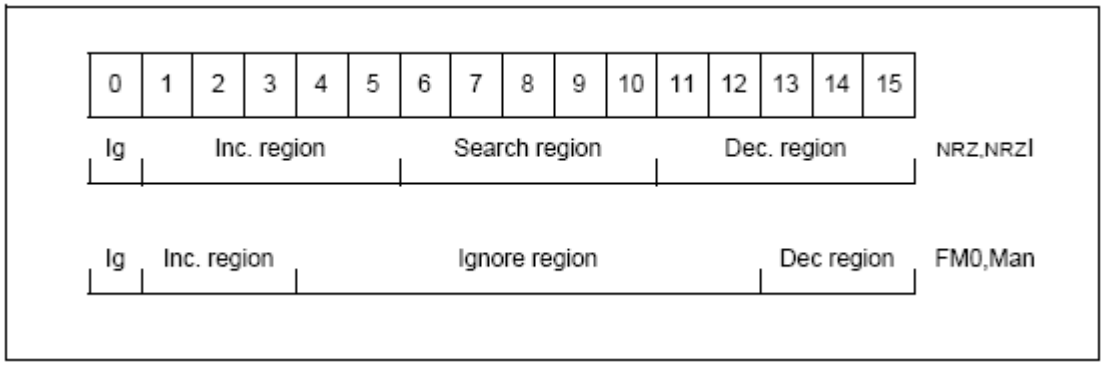
（a） NRZ，NRZI 模式下的 DPLL

在 DPLL 的复位状态之后，如果被使能，则 DPLL 进入搜索模式。在第一个数据沿，DPLL 进入同步模式并从 0 开始计数.

在 NRZ 和 NRZI 模式下，DPLL 把位单元划分为以下四个区（见图 7 5 ）：

- 忽略区（计数 0）.
- 增量区（从计数 1 到计数 5）.
- 递减区（从计数 11 到计数 15）.
- 搜索区（从计数 6 到计数 10）.

图 75 DPLL 工作区



（b） FM0,MANCHESTER 模式下的 DPLL

在 FM0 和 MANCHESTER 模式下，DPLL 把位单元划分为以下 3 个区（见图 75）

- 忽略区（计数 0，和从计数 4 到计数 12）.
- 增量区（从计数 1 到计数 3）.
- 递减区（从计数 13 到计数 15）.

当 DPLL 处于同步模式时，一个数据输入信号跳变沿伴随下列行为之一，取决于它发生在哪个区：

- DPLL 忽略该沿（在忽略区）
- DPLL 加倍下次计数 0 （在增量区）
- DPLL 跳过下次计数 0 （在递减区）

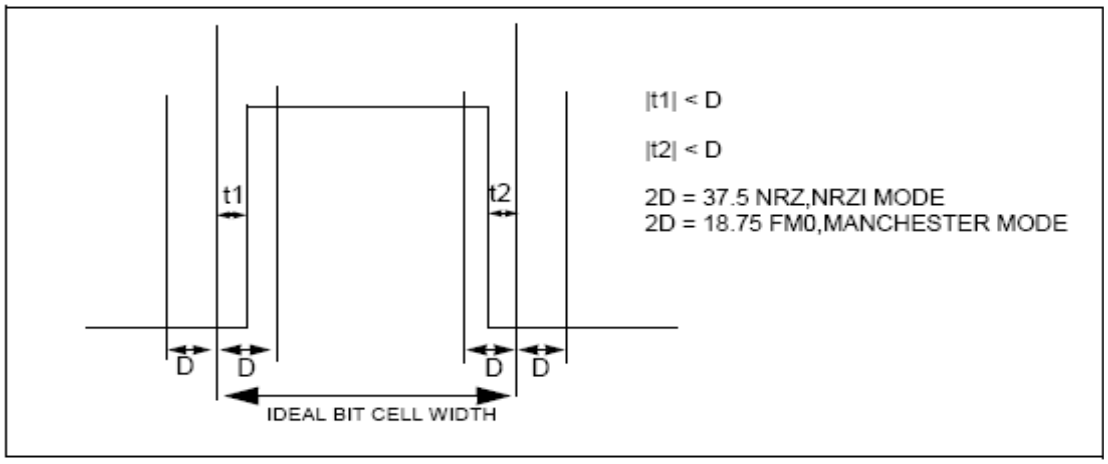
如果以下条件发生 DPLL 停止内部计数：

- FM0 模式：1 个位单元的结束边界边沿丢失。
- MANCHESTER 模式：1 位的中间边沿缺少（在此模式下，中间沿定为位单元的分界）。

DPLL 将在下个数据沿从计数 0 重新开始。在一个复位条件后 DPLL 进入搜索模式.

在第一个边沿，DPLL（如果激活了）进入常态模式，从计数 0 开始。

图76 DPLL相位抖动



一旦 DPLL 同步在引入数据中上，它能在输入脉冲脉宽变化（相位抖动）时保持同步，只要变化小于：

在in NRZ,NRZI模式下 37.5%

在FM0,MANCHESTER模式下 18.75%

15.2.2.4 波特率发生器和预分频器

HDLC 在发送中有 12 位波特率发生器，和接收时 8 位预分频器。（见图 72）。

波特率和预分频时间常数在 HDLC_BRR 和 HDLC_PRS 寄存器中编程，给出的分频因子分别是从 1 到 4096 和从 1 到 256。

发送中的最大波特率是 16MHz。

接收中的最大波特率是f_{PCLK1}/16。

预分频器或波特率发生器的输出时钟频率 Fckout 是：

Fckout = Fckin / 分频因子

其中分频因子等于HDLC_BRG+1或 HDLC_PRS+1值，Fckin是预分频器或波特率发生器的输入时钟频率。

15.2.2.3 循环冗余校验

HDLC帧校验序列由循环冗余校验程序按CCITT多项式 X16+X12+X5+1计算。

根据CRC初始设定选择位（HDLC_TCTL寄存器的TCRCI位 和HDLC_RCTL寄存器的RCRCI位），内部CRC发生器和校验器可以被独立地预先设置为全"1"或全"0".

以下 C 语言程序说明 CRCCCITT 模式是如何产生的：

CRCCCITT 计算例子

```
/*-------------------------------------------------------*/
/*-------------------Frame Format-------------------------*/
/*__FLAG|DATIN[0],[1],...,[NDATA]|FF[0],...,FF[15]|FLAG___*/
/*-------------------------------------------------------*/
/*--------------CRC Initialization-----------------------*/
for ( i=0; i<16; i++ )
{ FF[i]=0 } /* CRC initialization to“0s” */
```

```
/* FF[i]=1 to initialize to “1s” */
/*-------------------CRC Generation-----------------------*/
for ( i=0; i<NDATA; i++ )
{ R=XOR ( FF[0], DATIN[i] );
FF[0] = FF[1]; /* FF[0] is 1st bit transmitted *
FF[1] = FF[2];
FF[2] = FF[3];
FF[3] = XOR ( R, NOT( FF[4] ) );
FF[4] = FF[5];
FF[5] = FF[6];
FF[6] = FF[7];
FF[7] = FF[8];
FF[8] = FF[9];
FF[9] = FF[10];
FF[10]= XOR ( R, NOT( FF[11] ));
FF[11]= FF[12];
FF[12]= FF[13];
FF[13]= FF[14];
FF[14]= FF[15];
FF[15]= R;
}
```

For instance, with the data pattern:

DAT

[7][0] = d1 (hex)

DAT[15][8]= d2 (hex)

the CRC output pattern (with initialization to “1s”) is:

1100000100110010 (FF[0]=1).

15.2.2.6 中断工作模式

- 发送

只要 HDLC_PCR 中的 TEN 位为低，只有空闲和标志帧间信息按照 HDLC_TCTLR 的 ITF 位值被发送。

要启动一个帧传输，必须进行以下写操作：

- 数据写入发送缓冲器（所有的包，或直到缓冲器满）见图 77，
- 要发送的字节数写到 HDLC_TFBC 寄存器
- TEN 位置"1"（由软件，或当 TEN=1 由硬件）。

HDLC_TFBC 一被设定，HDLC 就启动后同步信号发送，并且从缓存器中读取数据直到缓存器半空或已发送的字节数与 HDLC_TFBC 寄存器的值相匹配。在第一种情况下将产生一个 TBE（发送缓冲为空）中断，而在第二种情况下则计算 CRC，产生结束标志和后同步信号。

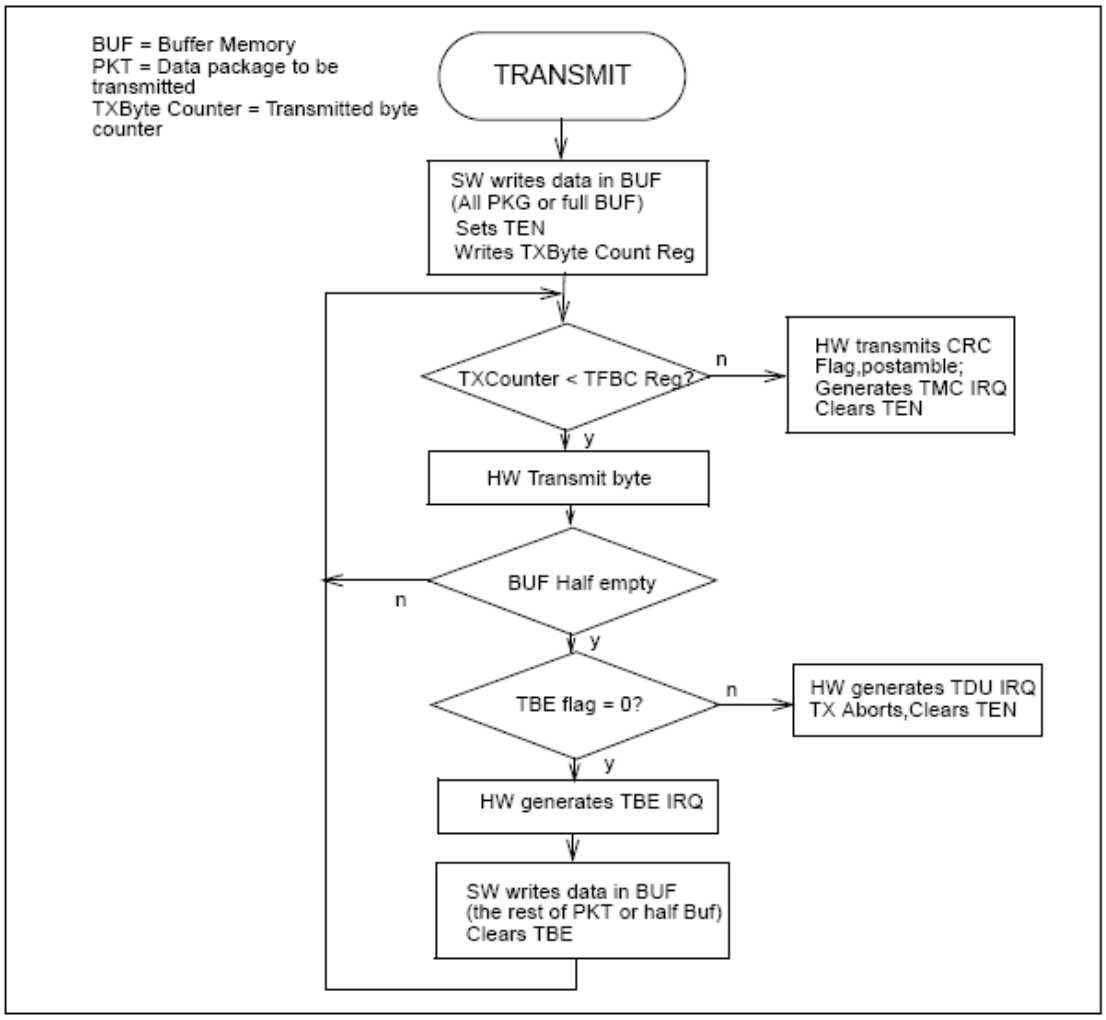
当后同步信号（如果有）的最后位，或标志，或中止序列的最后位已经被发送了，TEN 就被软件置位，产生 TMC（发送报文完成）中断。

如果 TEN 位被复位并且帧发送没完成，将产生一个中断序列。

如果发生一个 TBE 中断，且上一次中断还没完成（位 TEN 仍置"1"），一个错误条件，即 TDU（发送数据欠载 ）中断会产生，并且该帧被丢弃。

图 77 显示发送流程。

图 77 发送程序流程



- 接收

当 HDLC_PCR 寄存器的 REN(接收使能)为低时，不能接收任何帧，从不产生 RBF 中断。当 REN 写为置"1"，HDLC 可以接收帧。

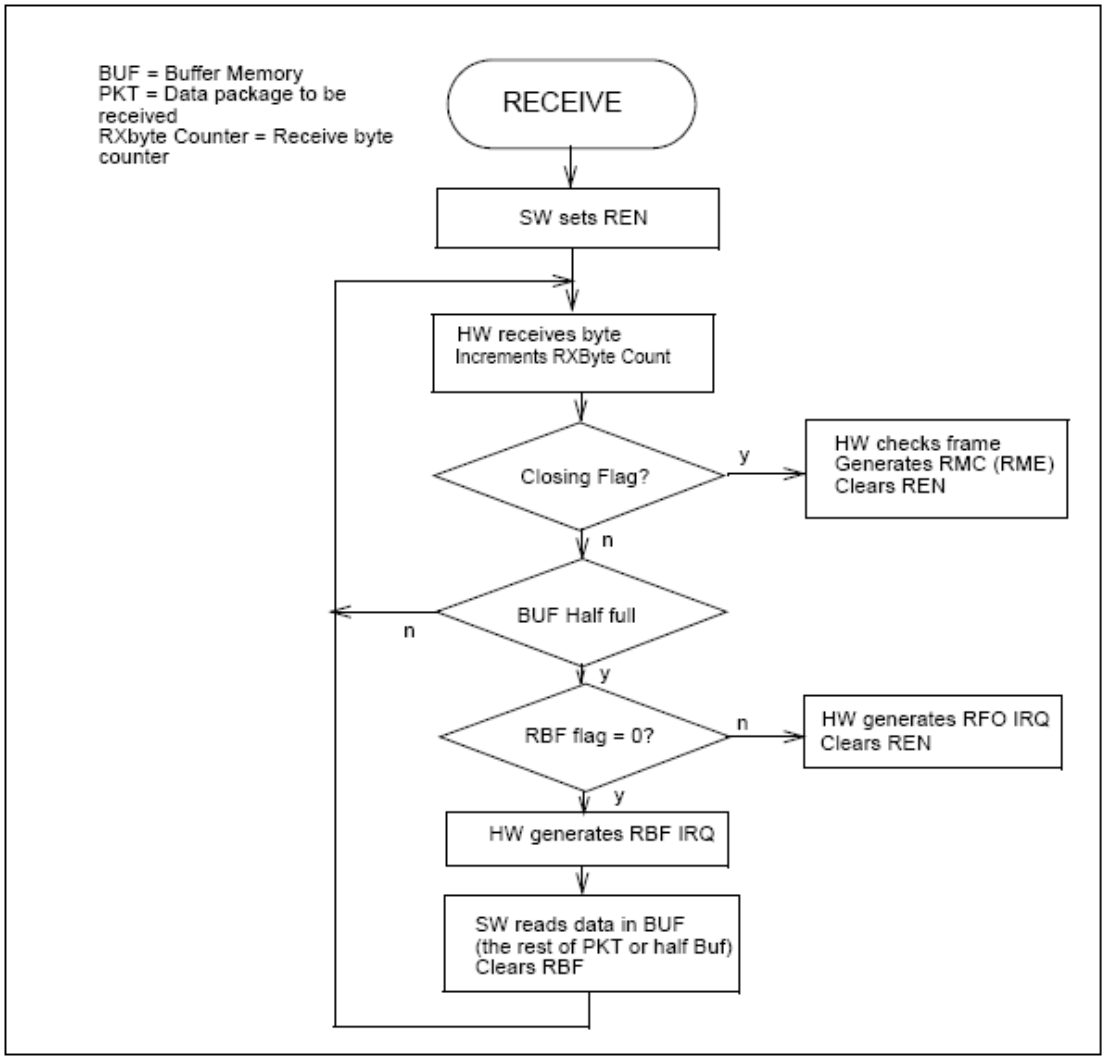
在开始标志后HDLC开始校验地址，如果在公共和群地址中至少一个是有效的，或所有的地址检验关闭，它就开始将数据（包含地址）写入接收缓冲器。每次在缓冲器半满时，会产生一个RBF（接收缓存器满）中断。

当最后一个数据写入接收缓冲寄存器，且后同步序列（如果存在）结束了，根据帧是否成功接收（通过了所有校验），就会产生一个RMC（接收报文完成）或RME(接收报文错误)，中断。此外，HDLC_RFBC (接收帧字节计数)寄存器的值给出接收字节总数（FCS字节未被写入缓冲器），HDLC_FSBR（帧状态字节）寄存器给出关于接收帧正确性的信息。

当一个RBF中断发生，并且上一中断仍未完成（位RBF仍置1），则产生错误条件。在这种情况下产生RFO（接收帧过溢）。

当发生RMC或RME时，硬件复位REN位，且如果RMCE位置位，则输出触发信号HRMC被激活。图78显示了接收流程。

图78 接收程序流程



15.3 寄存器描述

HDLC外围寄存器通常是16位读 / 写寄存器。对保留位的读取值为0。

15.3.1 私有地址高位寄存器（HDLC_PARH）

地址偏移：0000h
复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAB3								PAB2							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:8=**PAB3[7:0]:** 私有地址字节3
在开始标志后的第四地址字节。
位7:0=**PAB2[7:0]:** 私有地址字节2
在开始标志后的第三地址字节。

15.3.2 私有地址低位寄存器（HDLC_PARL）

地址偏移：0004h
复位值：0000h

位15:8=**PAB1[7:0]:** 私有地址字节1

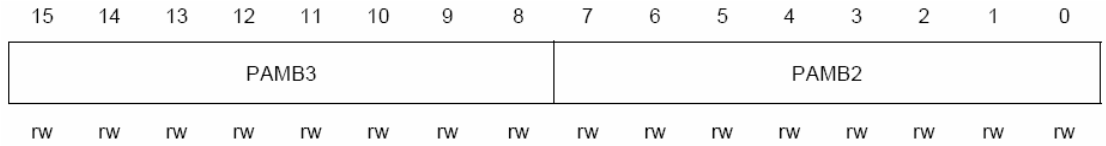
在开始标志后的第二地址字节。

位7:0=**PAB0[7:0]:** 私有地址字节0

在开始标志后的第一地址字节。

15.3.3 私有地址掩码高位寄存器（HDLC_PAMH）

地址偏移：0008h
复位值：0000h



位15:8=**PAMB3[7:0]:** 私有地址掩码字节3

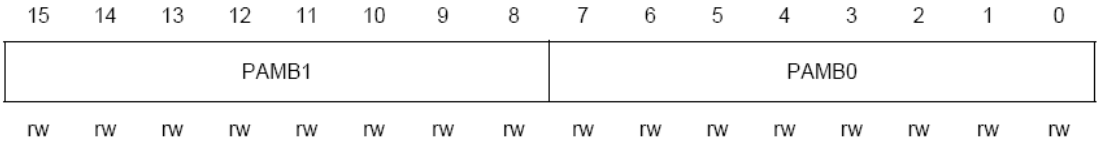
如果PAMB3x位是0， PAB3x与地址段中的相应位的比较被忽略。否则不被忽略.

位7:0=**PAMB2[7:0]:** 私有地址掩码字节2

如果PAMB2x位是0， PAB2x与地址段中的相应位的比较被忽略。否则不被忽略.

15.3.4 私有地址掩码低位寄存器（HDLC_PAML）

地址偏移：000Ch
复位值：0000h



位15:8=**PAMB1[7:0]:** 私有地址掩码字节1

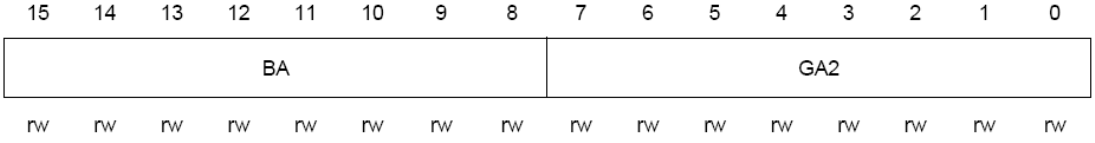
如果PAMB1x位是0， PAB1x与地址段中的相应位的比较被忽略。否则不被忽略.

位7:0=**PAMB2[7:0]:** 私有地址掩码字节0

如果PAMB0x位是0， PAB0x与地址段中相应位的比较被忽略。否则不被忽略.

15.3.5 群地址寄存器 1(HDLC_GA1)

地址偏移：0010h
复位值：0000h



位15:8=**BA[7:0]**：广播地址

如果此功能被使能，开始标记后的第一字节被识别为广播地址。

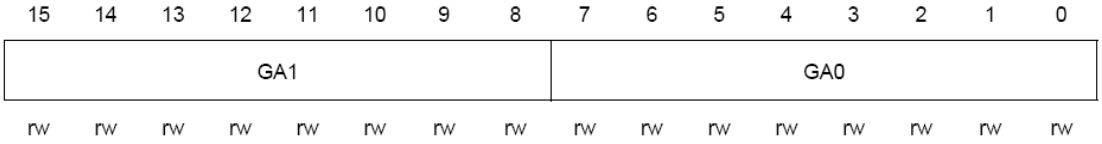
位7:0**=GA2[7:0]:** 群地址2

如果此功能被使能，开始标记后的第一字节被识别为群地址。

15.3.6 群地址寄存器 0（HDLC_GA0）

地址偏移：0014h

复位值：0000h



位15:8**=GA1[7:0]:** 群地址1

如果此功能被使能，开始标记后的第一字节被识别为群地址。

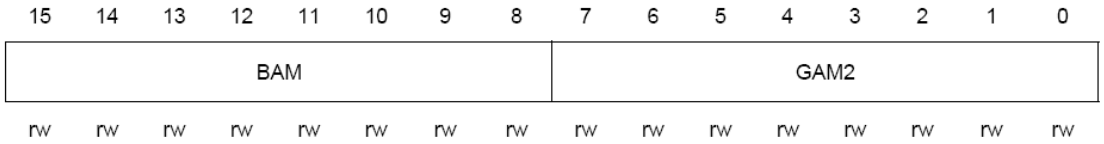
位7:0**=GA0[7:0]:** 群地址0

如果此功能被使能，开始标记后的第一字节被识别为群地址。

15.3.7 群地址掩码寄存器 1（HDLC_GAM1）

地址偏移：0018h

复位值：0000h



位15:8**=BAM[7:0]**：广播地址掩码

如果BAMx位是0， BAx与地址段中相应位的比较被忽略。否则不被忽略。

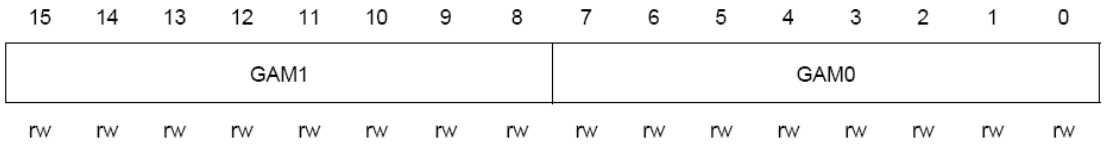
位7:0**=GAM2[7:0]:** 群地址掩码2

如果GAM2x位是0， GA2x与地址段中相应位的比较被忽略。否则不被忽略。

15.3.8 群地址掩码寄存器 0（HDLC_GAM0）

地址偏移：001Ch

复位值：0000h



位7:0**=GAM1[7:0]:** 群地址掩码1

如果GAM1x位是0， GA1x与地址段中相应位的比较被忽略。否则不被忽略。

位7:0**=GAM0[7:0]:** 群地址掩码0

如果GAM0x位是0， GA0x与地址段中相应位的比较被忽略。否则不被忽略。

15.3.9 前同步序列寄存器（HDLC_PRES）

地址偏移：0020h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRESEQ															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 15：0＝PRESEQ[15:0]：前同步序列

在开始标记位之前，从最低位开始，最多 16 位可被发送。

15.3.10 后同步序列寄存器（HDLC_POSS）

地址偏移：0024h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
POSS															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 15：0＝POSS[15：0]：后同步序列

在结束标记位之后，从最低位开始，最多 16 位可被发送。

15.3.11 发送控制寄存器（HDLC_TCTL）

地址偏移：0028h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HTEN	SOC	ITF	TCRCI	TCOD		PREE	POSE	NPREB				NPOSB			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 15＝HTEN：硬件发送时能

当为 1，TEN 位可被硬件和软件设置。

当为 0，TEN 位仅可被软件设置。

位 14＝SOC：串行输出取补

当为 1，串行输出帧被逐位取补。

当为 0，串行输出帧被原样发送。

位 13=ITF：帧间时间填充

空闲状态或者标记可在两个连续的帧之间发送。

当 ITF=1 时，标记被发送。

当 ITF=0 时，空闲状态被发送。

位 12=TCRCI：发送 CRC 初始化

值为 0 时，CRC 用'0'初始化。

值为 1 时，CRC 用'1'初始化。

位 11：10＝TCOD[1:0]：发送数据编码

发送数据通过下表被编码：

表 58. 发送数据编码

TCOD1	TCOD0	MODE
0	0	NRZ
0	1	NRZI
1	0	FM0
1	1	MANCH

位 9＝PREE：前同步序列使能
当为 0 时，不发送前同步序列。
当为 1 时，发送前同步序列。

位 8＝POSE：后同步序列使能
当为 0 时，不发送后同步序列。
当为 1 时，发送后同步序列。

位 7:4 = NPREB[3:0]：前同步序列位数
如果功能被使能（PREE=1），在开始标志之前，（NPREB[3:0]+1）位被传输。

位 3:0 = NPOSB[3:0]：后前同步序列位数
如果功能被使能（POSE=1），在结束标志之后，（NPOSB[3:0]+1）位被传输。

15.3.12 接收控制寄存器（HDLC_RCTL）

地址偏移：002Ch
复位值: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	RMCE	DPLLE	PAE	BAE	GA2E	GA1E	GA0E	AEN	LBEN	SIC	RCRCI	RCOD	
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 15:13 = 保留位，一直读为 0。

位 12＝RMCE：接收报文完成使能
当为 1，允许产生触发输出信号 HRMC；
当为 0，不允许产生触发输出信号 HRMC。

位 11=DPLLE：DPLL 使能
值为 1 时 DPLLE 是使能的
值为 0 时 DPLLE 关闭。

位 10＝PAE：私有地址使能
值为 1 时私有地址被识别
值为 0 时私有地址被忽略。

位 9＝BAE：广播地址使能

值为 1 时广播地址被识别

值为 0 时广播地址被忽略

位 8＝GA2E：群地址 2 使能

值为 1 时群地址 2 被识别

值为 0 时群地址 2 被忽略

位 7＝GA1E：群地址 1 使能

值为 1 时群地址 1 被识别

值为 0 时群地址 1 被忽略

位 6＝GA0E：群地址 0 使能

值为 1 时群地址 0 被识别

值为 0 时群地址 0 被忽略

位 5＝AEN：自动回应使能

当 AEN=1 时，发送器处于自动回应模式：发送器与 TX 数据输出 HTXD 断开。HTXD 由引脚 HRXD 的 RX 信号直接驱动。接收器与 HRXD 保持连接。

位 4=LBEN：回环使能

当 LBEN＝1 时，TX 输出 HTXD 置“1”，RX 输入 HRXD 断开，发送器的输出回送至接收器输入。

位 3=SIC：串行输入取补

当 SIC＝1 时，串行输入帧逐位取补。

当 SIC＝0 时，串行输入帧按原样传送。

位 2＝RCRCI：接收 CRC 初始化

值为 0 时 CRC 用 0 初始化

值为 1 时 CRC 用 1 初始化。

位 1:0 ＝ RCOD[1:0]：接收数据译码

接收数据根据下表译码：

表 59 接收数据译码

RCOD1	RCOD0	MODE
0	0	NRZ
0	1	NRZI
1	0	FM0
1	1	MANCH

15.3.13 波特率寄存器（HDLC_BRR)

地址偏移：0030h

复位值：0000h

位 15:13 = 保留位，一直读为 0。

位 12＝TCKS：发送时钟选择

0：PCLK1 时钟被选择作为发送时钟源。

1：PLL2 输出被选择作为发送时钟源。

当选择了一个新的发送时钟源，外围设备将会被复位来避免虚假脉冲产生，所以任何先前存储的数据都会丢失。

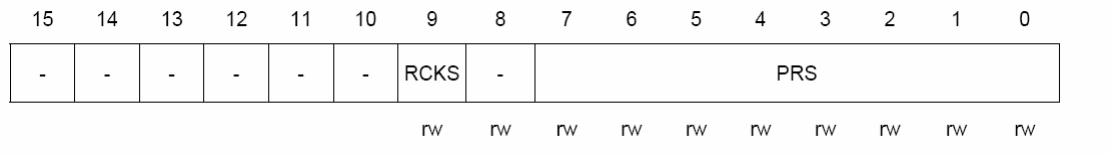
位 11:0 ＝ BRG[11:0]：发送波特率

当使能 NRZ 或者 NRZI 能编码时，发送时钟 TCK 被 BRG[11:0]+1 倍分频；当采用 FM0 或者曼彻斯特编码时，发送时钟 TCK 被 2*(BRG[11:0]+1) 倍分频。

15.3.14 预分频寄存器（HDLC_PRSR）

地址偏移：0034h

复位值：0000h



位 15:10＝保留位，一直读为 0

位 9＝RCKS：接收时钟选择

0：PCLK1 时钟被选择作为接收时钟源。

1：PLL2 输出被选择作为接收时钟源

当选择了一个新的接收时钟源时，外围设备将被复位，以避免虚假脉冲发生，所以任何先前存储的数据将会丢失。

位 8＝保留位。这个位可读/写，但是必须被写入零。

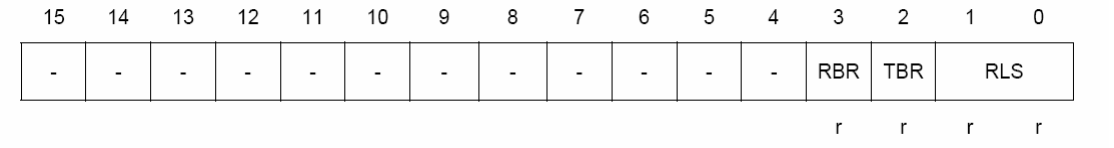
位 7:0＝PRS[7:0]：接收预分频数

接收时钟会 RCK 被 PRS[7:0]+1 倍分频。

15.3.15 外设状态寄存器（HDLC_PSR）

地址偏移：0038h

复位值：0000h



位 15:4＝保留，始终读为 0

位 3＝RBR：接收半缓冲器就绪

值为 0 时，接收缓冲器低半区已满

值为 1 时，接收缓冲器高半区已满。

位 2＝TBR：发送半缓冲器就绪

值为 0 时，发送缓冲器低区空

值为 1 时，发送缓冲器高半区空

位 1:0＝RLS[1:0]：接收线状态

接收线状态根据下表编码：

表 60 接收线状态

RLS1	RLS0	STATUS
0	0	Noise or abort is being received
0	1	Idle is being received
1	0	Interframe flags are being received
1	1	Frame is being received

此寄存器不断被更新。

15.3.16 帧状态字节寄存器（HDLC-FSBR）

地址偏置：003Ch

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	RBC	RDO	CRC	RAB	-	-	-	-
								r	r	r	r				

位 15:8=保留位，始终读为 0

位 7=RBC：接收字节计数

为 1 时，接收字节的长度不是 8 字节的整数倍

位 6=RDO：接收数据溢出

为 1 时，由于发布了中断请求，但前一中断还没有被确认，从而造成至少一字节已经丢失。

位 5 = CRC：FCS 校验

为 1 时，接收到的 FCS 字节错误

位 4=RAB：接收中止

为 1 时，接收的帧被放弃

位 3:0 = 保留位，始终读为 0

当 RMC（接收报文完成）或 RME（接收报文错误）中断发生时，此寄存器被更新。状态字节为 00h 指示一个正确接收的帧。

15.3.17 发送帧字节计数寄存器（HDLC-TFBCR）

地址偏置：0040h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TFBC															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit15:0=TFBC[15:0]：发送传输帧字节计数

在开始发送前，要发送的字节数必须由软件装载到此寄存器。每发送一字节，TFBC 的值都要更新，显示剩余要发送的字节数。

15.3.18 接收帧字节计数寄存器（HDLC_RFBCR）

地址偏移：0044h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RFBC															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位 15:0＝RFBC[15:0]：接收帧字节计数

RFBC 值给出了以接收的数据数目。

15.3.19 外设命令寄存器（HDLC_PCR）

地址偏移：0048h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	TEN	REN
														rw	rw

位 15:2＝保留位，一直读为 0。

位 1＝TEN：发送使能

当被写为 1 时，发送器被使能，如果允许，会产生中断请求。在发送结束后，该位被硬件复位。

当被写为 0 时，发送器处在复位状态，中断请求不再发生。如果在当前帧的结束标记之前发生，会产生一个异常中止序列。

如果功能（HTEN=1）被使能，那么这个位就会被触发信号 HTEN 设置。

位 0＝REN：接收使能位

当它被写为 1 时，接收器被使能；如果允许，中断请求会发生。在接收结束时，这个位被硬件复位。

当它被写为 0 时，接收器处在复位状态，中断请求不再发生。

15.3.20 中断状态寄存器（HDLC_ISR）

地址偏移：004Ch

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	TMC	TBE	TDU	RBF	RFO	RMC	RME
									rc	rc	rc	rc	rc	rc	rc

位 15：7＝保留位，一直读为 0。

位 6＝TMC：发送报文完成中断等待位

当结束标志的最后一位或者后同步序列（如果存在）已经被发送了，这个等待位被设置，如果被允许（TMCM=1）则产生一个中断请求；PCR 中的 TEN 为被硬件置位。

位 5＝TBE：发送缓冲器空中断等待位

当发送缓冲区为半空，这个等待位被置位；若允许（TBEM=1）则产生一个中断。

位 4＝TDU：发送数据欠载中断等待位

当一个欠载条件发生时，这个等待位被置位；若允许（TDUM=1）则产生一个中断。

位 3＝RBF：接收缓冲器满中断等待位

当接收缓冲器半满时，这个等待位被置位；若允许（RFBM=1）则产生一个中断。

位 2＝RFO：接收帧溢出中断等待位

当一个溢出条件发生时，这个等待位被置位；若允许（RFOM=1）则产生一个中断

位 1＝RMC：接收报文完成中断等待位

当检测到帧结束且 CRC 校验正确时，这个等待位被置位；若允许（RMCM=1）则产生一个中断，若允许（RMCE=1）则输出触发信号 HRMC 被激活。

位 0＝RME：接收报文错误中断等待位

当检测到帧结束且CRC校验失败，或检测到一个中止时，这个等待位被置位；若允许（RMEM=1）则产生一个中断，若允许（RMCE=1）则输出触发信号 HRMC 被激活。

这些位被硬件置位，软件仅可写入零。

15.3.21 中断屏蔽寄存器（HDLC_IMR）

地址偏移：0050h

复位值：0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	TMCM	TBEM	TDUM	RBFM	RFOM	RMCM	RMEM
									rw	rw	rw	rw	rw	rw	rw

位 15：7＝保留位，一直被读为 0。

位 6＝TMCM：传输报文完成中断屏蔽位

当为 0,TMC 中断不能发生。

当为 1,TMC 中断能够发生。

位 5＝TBEM：发送缓冲器为空中断屏蔽位

当为 0，TBE 中断不能发生。

当为 1，TBE 中断能够发生。

位 4＝TDUM：发送数据欠载中断屏蔽位

当为 0，TDU 中断不能发生。

当为 1，TDU 中断能够发生。

位 3＝RBFM：接收缓冲器满中断屏蔽位

当为 0，RBF 中断不能发生。

当为 1，RBF 中断能够发生。

位 2＝RFOM：接收帧溢出中断屏蔽位

当为 0，RFO 中断不能发生。

当为 1，RFO 中断能够发生。

位 1＝RMCM：接收报文完成中断屏蔽位

当为 0，RMC 中断不能发生。

当为 1，RMC 中断能够发生。

位 0＝RMEM：接收报文错误中断屏蔽位

当为 0，RME 中断不能发生。

当为 1，RME 中断能够发生。

15.4 HDLC 寄存器映射

表 61 总结了 HDLC 寄存器映射

表 61 HDLC 寄存器映射

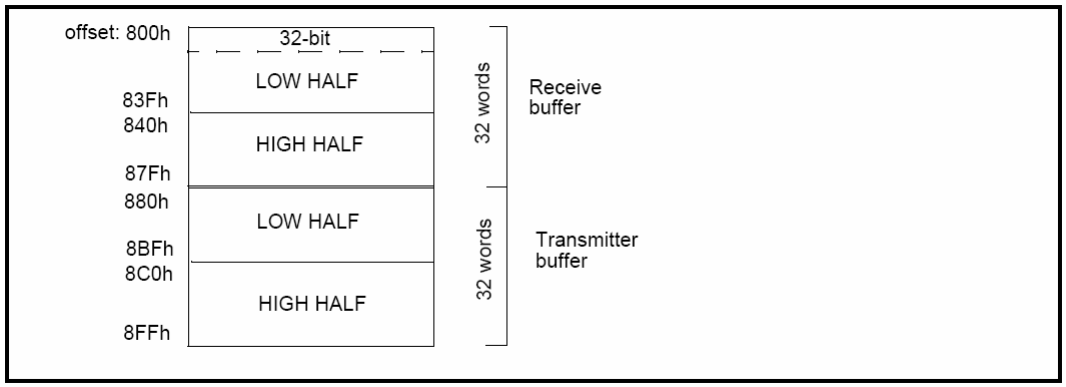
Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	HDLC_PARH	PAB3								PAB2							
04	HDLC_PARL	PAB1								PAB0							
08	HDLC_PAMH	PAMB3								PAMB2							
0C	HDLC_PAML	PAMB1								PAMB0							
10	HDLC_GA1	BA								GA2							
14	HDLC_GA0	GA1								GA0							
18	HDLC_GAM1	BAM								GAM2							
1C	HDLC_GAM0	GAM1								GAM0							
20	HDLC_PRES	PRESEQ															
24	HDLC_POSS	POSS															
28	HDLC_TCTL	HTEN	SOC	ITF	TCR-CI	TCOD		PREE	POSE	NPREB				NPOSB			
2C	HDLC_RCTL	Reserved			RCME	DPLLE	PAE	BAE	GA2E	GA1E	GA0E	AEN	LBEN	SIC	RCR-CI	RCOD	
30	HDLC_BRR	Reserved			TCKS	BRG											
34	HDLC_PRSR	Reserved						RCKS	-	PSR							
38	HDLC_PSR	Reserved												RBR	TBR	RLS	
3C	HDLC_FSBR	Reserved												RBC	RDO	CRC	RAB
40	HDLC_TFBCR	TFBC															
44	HDLC_RFBCR	RFBC															
48	HDLC_PCR															TEN	REN
4C	HDLC_ISR	Reserved									TMC	TBE	TDU	RBF	RFO	RMC	RME
50	HDLC_IMR	Reserved									TMCM	TBEM	TDUM	RBFM	RFOM	RMCM	RMEM

基地址请看表 1“存储器块的映射”。

15.4.1 随机存储器缓冲器映射

两个存储器缓冲器，一个用于数据接收，另一个用于数据发送，它们的地址映射分别为：地址偏移从 800h 到 87Fh，和地址偏移从 880h 到 8FFh。这两个缓冲器都是 128 字节大小，组织成 32 个 32 位字。

图 79 HDLC 发送和接收存储缓冲器映射



接收数据必须从地址偏移 800h 到 83Fh 或者是从 840h 到 87Fh 读取，根据哪一半是满的来选择（可以检查 PSR 中的 RBR 位）。要发送的数据必须被装载到发送缓冲器，地址偏移从 880h 到 8BFh，或者从 8C0h 到 8FFh，根据哪一半是空的来选择（可以检查 PSR 中的 TBR 位）。

第一个被接收到的字节是在地址偏移 800h 处的缓冲器位置的最低字节。

第一个要发送的字节是位于地址偏移 880h 处的缓冲器位置的最低字节。

16 A/D 转换器（ADC）

16.1 介绍

ADC 用于测量信号强度和其它变化缓慢的信号，提供了四条输入通道，每条的转换速率达到 1kHz。

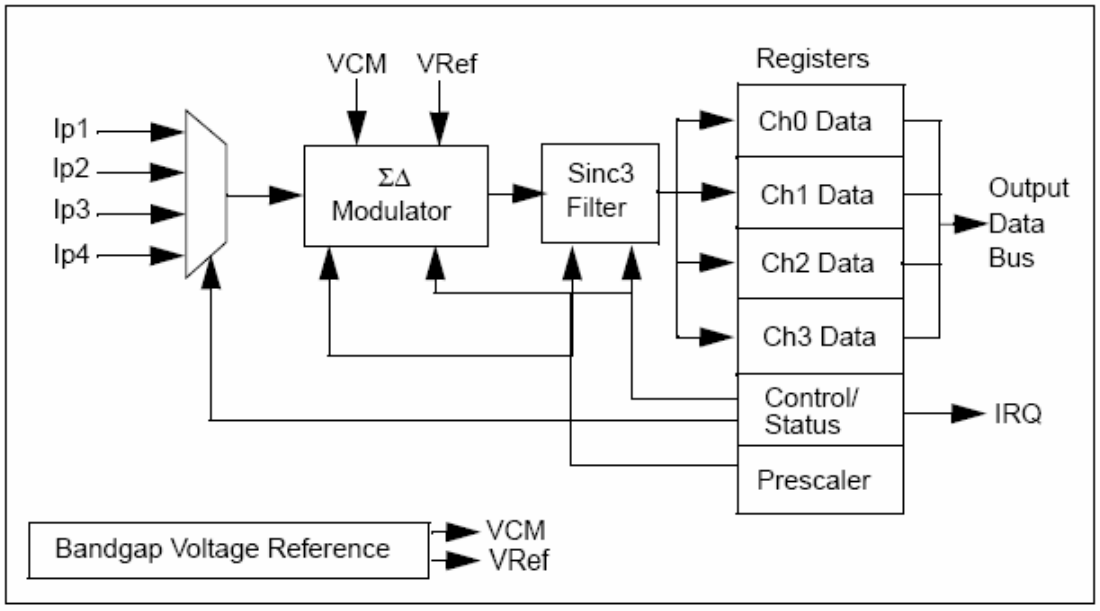
16.2 主要特性

- 12 位转换精度
- 0-2.5V 动态范围
- 4 输入通道
- 多通道轮寻或单通道模式
- 可屏蔽中断
- 可编程预分频器

16.3 功能描述

ADC 由一个带有 511-tap $Sinc^3$ 数字滤波器的 4 通道单比特 Σ-Δ 调制器和一个带隙基准电压源组成。转换器的结构图如图 80 所示。寄存器 PCU_BOOTCR 里的 ADC_EN 为转换器的模拟部分的使能位。

图 80 ADC 结构框图



16.3.1 ADC 的标准（轮询）工作模式

在标准工作模式下，转换器对每个输入通道进行 512 个过采样时钟周期的采样。在第一个时钟周期，Σ–Δ 调制器复位且数字滤波器清零。调制器剩余的 1 位采样值序列被送入 $Sinc^3$ 滤波器滤波，在第 511 个时钟周期以后提供一个 16 位的输出采样到有关数据寄存器。在这段时间内，$Sinc^3$ 滤波

器被填充并稳定下来。信道选择被切换到下一个输入信道，复位信号再次在第一个周期内进行，滤波器再次在 511 个周期中被填充以产生一个采样值。这一过程将对每一个通道不断轮流重复进行。

16.3.2 单通道模式

当对单通道采样时，该单一通道被选择作为模拟信号将输入到 Σ–Δ 调制器。转换器的功能将和上面一样，转换器每 512 个周期产生一个有效的采样信号，而后被复位。然而，为了保持转换器有同样的输出频率，每四个采样信号中将只选取一个，即每 2048 个时钟周期产生一个有效的采样。

注意：为了提高 ADC 对模拟信号的转换速度，可以采用多通道轮询模式，把一个输入信号连接到 ADC 的四个输入通道。（参考 AN1798: STR71x ADC Conversion Speed-Up）。

16.3.3 时钟时序

Σ–Δ 调制器工作频率(f_{Mod},过采样速率)必须低于 2.1MHz 的 ADC 频率。转换器由 PCLK2 提供时钟信号。用双时钟同步作为数据交叉时钟的分界来避免亚稳态情况。基于下列公式，要求用户正确地为预分频器编程以产生基于 PCLK2 的正确的过采样频率。

f_s = 输入信号的采样频率

f_{mod} = Σ–Δ 调制器的采样率

$f_S = f_{Mod} / \{512*4\}$

$f_{Mod} = f_{PCLK2} / \{预分频因子\}$

例如，如果f_{PCLK2}是16MHz，希望的输入信号的采样频率f_s是500Hz，Σ–Δ调制器必须工作在1.024MHz，预分频器因子必须被设置成0x8(在ADC_CPR寄存器里)使预分频因子为16。

注：如果预分频器被设置产生一个高于给定值的采样频率，转换性能和精度就不能保证了。

16.3.4 增益误差和偏移误差

一个片上带隙基准电压提供一个 1.22V 的参考电压，用来产生两个供调制器使用的电压——V_{CM} 和 V_{Ref}。V_{CM} 被设计成 1.25V，是转换器的输入电压范围的中点；Vref 是 1.85V，为反馈参考电压。由于带隙基准电压未经调整，V_{CM} 和 V_{Ref} 的绝对值的不准确度可达±5%，这将导致转换器的增益误差和偏移误差，如有必要此误差可以用软件来校准。

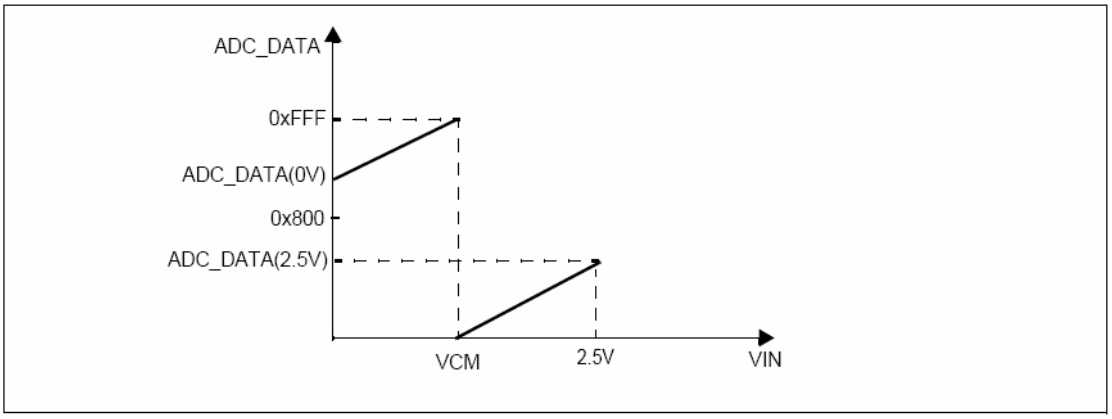
为对转换器进行校准，需要输入支持的最小和最大输入值——0V 和 2.5V。0V 输入时的数字输出就是偏移，转换器的增益由下式给出：G = (2.5V 的输出 － 0V 的输出) / 2.5。偏移和增益校正因子可以存为数值，并应用到所有四路输入转换的输出。

16.3.5 ADC 输出编码

Σ–Δ 转换器每 512 个过采样时钟对每一模拟输入产生一个数字采样。从 $Sinc^3$ 数字滤波器输出的数字采样存储在四个 ADC_DATA[n]寄存器中，存为 16 位字，其中仅前 12 位是有意义的。存储的转换值是带符号的 2 的补码，正比于差值（V_{IN}-V_{CM}）。 若 V_{IN}=V_{CM}，其理想值为 0。

下图给出了 ADC 输出（12 位编码）与输入电压值关系：

图 81 ADC 输出



ADC_DATA(0V)和 ADC_DATA(2.5V)分别是对 0V 和 2.5V 的转换结果。这两个值必须要确定，以便用以下公式计算 ADC 增益。

$$G = \left[0XFFF - ADC\_DATA(0V) + ADC\_DATA(2.5V)\right] / 2.5$$

注： 模拟输入电压不能超过 Σ–Δ 调制器中心电压的 2 倍(2* V_{CM})，否则不能保证转换器性能。同样 V_{CM} 的精确度误差为+/-5%，要求必须对转换器进行校准。

16.3.6 低功耗特性

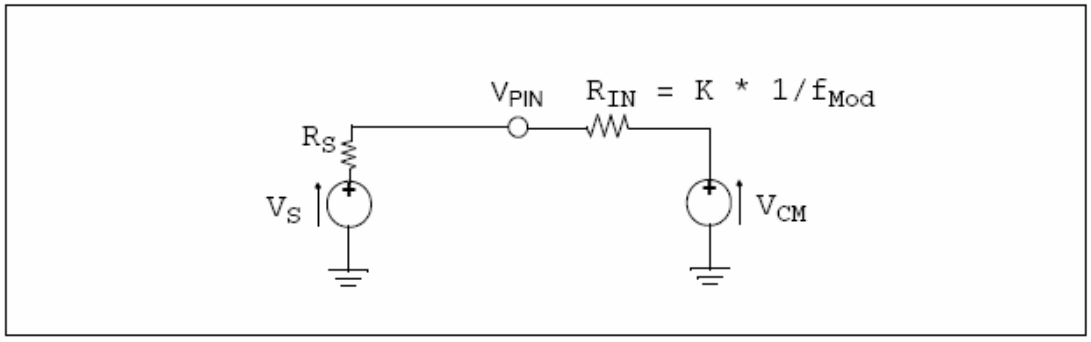
当 PCU_BOOTCR 寄存器中的“ADC_EN”被置位时，ADC 模块的模拟电路开启。默认情况下，在复位过程中该位被清零，AV_{DD}/AV_{SS} 引脚的功耗达到最小。如果先前被使用了，推荐在进入低功耗模式前用软件关闭 ADC。

注： 使用“ADC_EN”开关来禁止 ADC，只关闭了 Σ–Δ 转换器的模拟部分。如果不使用 ADC，也可以利用 APB2_CKDIS 寄存器的“位 7”来关闭其数字部分。这就关闭了 ADC 的时钟。

16.3.7 ADC 的输入等效电路

输入电容存储了取自被测量输入信号的能量，由于它以 f_{Mod} 频率进行开关动作，输入等效电路如下图所示：

图 82 ADC 输入等效电路



图中，V_S 是被测量的电压，R_S 是被测电压源的输出电阻，V_{PIN} 是实际被转换的电压，R_{IN} 是 ADC 的输入等效电阻。R_{IN} 与调制器的过采样时钟成反比例。常数 K 由转换器的工作模式决定，其值为：

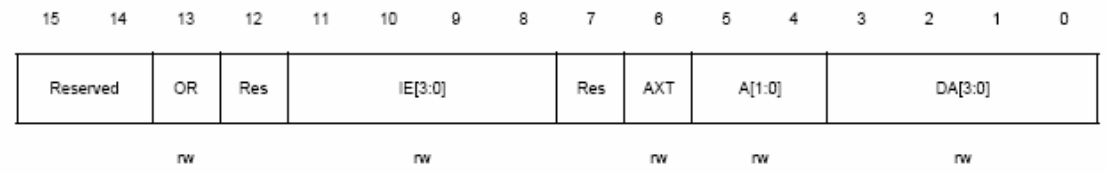
- 750[kΩ][MHz] +/-20% ——单输入转换模式
- 3000[kΩ][MHz] +/-20% ——轮转转换模式.。

16.4 寄存器描述

16.4.1 ADC 控制/状态寄存器（ADS_CSR）

地址偏移：20h

复位值：0000h



此寄存器用于控制 ADC 的工作模式，设置中断允许，并包含数据可用标志、以及发生数据读取前被覆盖错误的标志。

位 [15:14] 保留位，必须保持在复位状态(0)。

位 13 OR：过载

此位可读可写，用于告知应用软件某一通道数据未读之前被覆写的情况。

0：正常运行。无过载情况发生。

1：过载情况发生。当过载被检测到时，此位被硬件置位。必须通过软件对此位清零。

位 12 保留位，必须保持在复位状态(0)。

位 [11:8] IE[3:0]：中断允许

这些位允许单独设定每个 ADC 通道的中断请求的使能或禁止。IE[n]对应 ADC 的第 n 个通道。

位 7 保留位，必须保持为复位状态(0)。

位 6 AXT：外部寻址允许信号。

此位用于设定单通道操作，配置 ADC 重复转换由本寄存器的 A[1:0]确定的通道。

0：轮转寻址使能

1：单通道寻址使能

位 [5:4] A[1:0]：通道地址

当外部寻址有效时，此位用于选择外部采样通道。

位 [3:0] DA[3:0]：数据有效位

这些位可读些，用来确定在哪个通道的数据寄存器中，有新采样值可读。 DA[n]对应 ADC 的通道 n。

0：对应通道无有效取样值

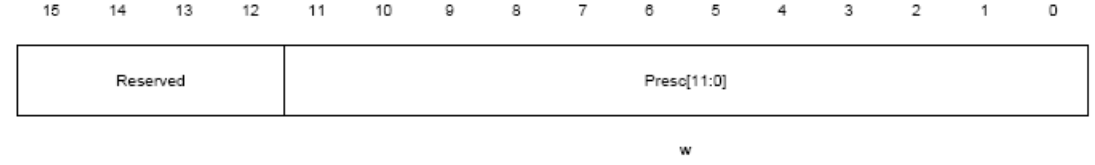
1：对应通道有新的可用采样值

一旦有了新的采样值，此位就被硬件置位；必须通过写“0”由软件来清除，向此位写“1”无效。这些位也用来作为对应通道的中断标志。

16.4.2 ADC 时钟预分频寄存器

偏移地址：30h

复位值：05h



位[15:12] 保留位，必须保持复位值(0)。

位[11:0] PRESC[11:0]：预分频量

由本寄存器给定的这 12 位二进制数据用于确定 ADC 输入时钟分频倍数，用来产生用于∑-△调制器的过采样时钟，实际分频因子是 PRESC 寄存器值的两倍，如表 62 所示。

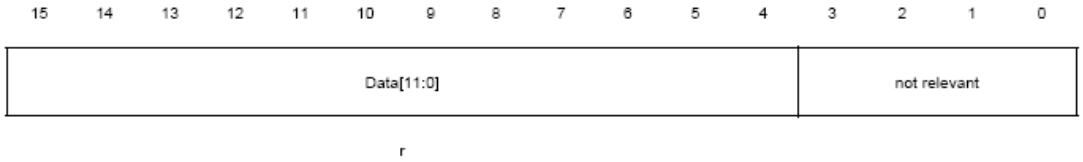
放在这个寄存器的值，必须使因此产生的过采样时钟频率，不高于接入 ADC 的 2.5Mhz 的 PCLK2 时钟。这些位只能由软件写入，任何读操作均返回 0。

表 62 ADC 预分频设置

Setting	Prescaling Factor
0x0, 0x1	Not available
0x2	4
0x3	6
0x4	8
0x5	10
...	...
0xFFE	8188
0xFFF	8190

ADC 数据寄存器 n，n＝0…3(ADC_DATA[n])

地址偏移：0x00h(通道 0)
地址偏移：0x08h(通道 1)
地址偏移：0x10h(通道 2)
地址偏移：0x18h(通道 3)
复位值：0000h



存在 4 个数据寄存器，每一个用于一个模拟输入通道。每个寄存器的最高 12 位存放转换的结果，而最低 4 位应被忽略。在多通道轮转模式下，数据寄存器按通道编号顺序填充。在单通道模式下，只有选定的通道被刷新。

位[15:4]　DATA[11:0]：数据采样
此只读寄存器含有对应通道的最新采样值。

16.5 ADC 寄存器映射

表 63 ADC 寄存器映射

00	ADC_DATA0	DATA0[11:0]										N/U		
08	ADC_DATA1	DATA1[11:0]										N/U		
10	ADC_DATA2	DATA1[11:0]										N/U		
18	ADC_DATA3	DATA3[11:0]										N/U		
20	ADC_CSR	-	-	OR	-	IE[3:0]		-	AXT	A[1:0]		DA3	DA2	DA1	DA0
30	ADC_CPR	-							PRE[6:0]						

基址部分见表 3 “APB2 存储器映射”。

17 APB 桥寄存器

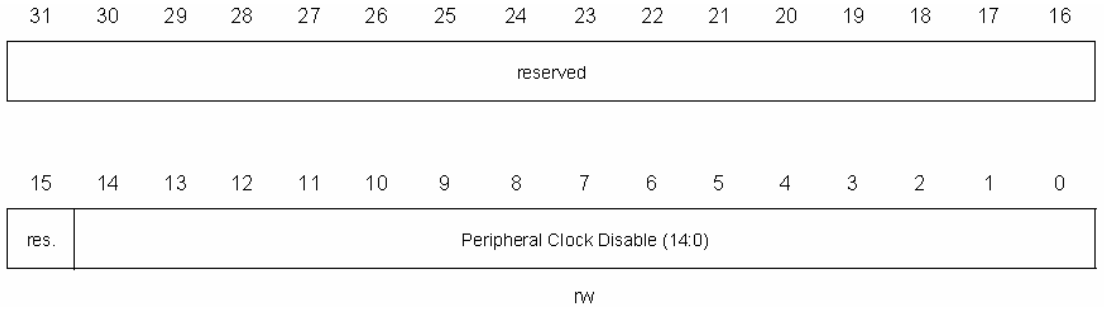
片上的外围设备通过两个 APB 桥来寻址：APB1 和 APB2。

每个桥有两个 32 位的寄存器。在软件的控制下,每个外围设备可以用 SWRES 寄存器来单独复位。输入给每个外围设备的 PCLK 信号(除了看门狗以外)也可以用 CKDIS 寄存器来独立地来使能/禁用。CKDIS 寄存器也用来使能/关闭 CKOUT 引脚上的信号，这个引脚上的时钟输出就是 PCLK2，其频率通过 PRCCU 可编程(APBDIV 寄存器)。

注意：APB 桥寄存器必须用 32 位对齐的操作来存取(即，字节/半字的访问周期是不允许的)。

17.1 APB 时钟禁止寄存器(APBn_CKDIS)

地址偏移:0×10h
偏移量: 0×0000 0000h



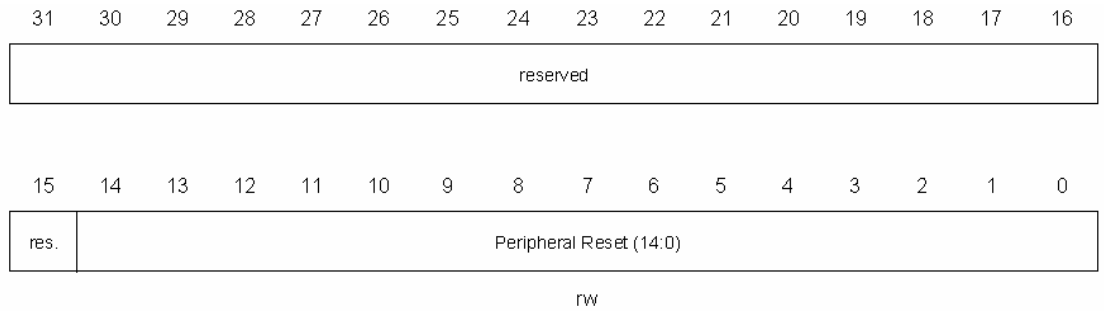
位 31:15 = 保留位，必须保持在复位值(0)。
位 14:0 = 外围时钟禁止位(14:0)

这些位的每一位控制一个外设的时钟，位的顺序与外设在桥存储器映射（见表 2 和表 3）中的顺序相同，位 0 在位置 1 的外设(APB1 的 I2CO 或者 APB2 的 XTI)，等等。

0：外设时钟允许
1：外设时钟禁止

17.2 APB 软件复位寄存器(APBn_SWRES)

偏移地址:0×14h
复位值: 0×0000 0000h



位 31:15=保留位，必须保持在复位值(0)。
位 14:0 =外设复位(14:0)

这些位的每一位控制一个外设复位信号的激活，位顺序与外设在桥存储器映射中的顺序相同(见表 2 和表 3)。位 0 控制在位置 1 的外设(APB1 的 I2CO 或者 APB2 的 XTI)，等等。　　0：外围设备通过系统范围的复位信号来复位

1：外围设备由复位位单独控制，并与系统复位分开.

17.3 APB 寄存器映射

表 64 APBn 寄存器映射

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	**Reserved**	reserved																															
04	**Reserved**	reserved																															
08	**Reserved**	reserved																															
0C	**Reserved**	reserved																															
10	**APBn_CKDIS**	reserved																	Peripheral clock disable[14:0]														
14	**APBn_SWRES**	reserved																	Peripheral reset [14:0]														

基地址见表 2 和表 3。

18 JTAG 接口

18.1 概述

STR71x 是围绕 ARM7TDMI 内核制造的，该内核的调试接口是基于 JTAG 的。 ARM7TDMI 包含了用于高级调试功能的硬件扩展。调试扩展允许内核运行可以被停止于给定的取指操作（断点），或数据访问（观察点），或异步的调试请求。当运行中断的时候 ARM7TDMI 就被称为处于*调试*状态。在这个瞬间，核的内部状态和系统的外部状态可以被检查。一旦检查完成，内核和系统可以被还原，程序运恢复。

ARM7TDMI 可以由一个外部接口上信号的请求强迫进入调试状态，也可以由一个叫做在电路仿真单元(ICE)的内部功能单元强迫进入调试状态。一旦处于调试状态，内核就把自身与存储器系统隔离，然后内核可以被检查，而所有其他的系统活动继续如常。

ARM7TDMI 的内部状态可以通过一个 JTAG 风格的串行接口来检测，该接口允许指令串行进入内核的流水线，而不需使用数据总线。这样，当处于调试状态时，一个多字存储（STM）指令可被插入到指令流水线，这样可以收集 ARM7TDMI 寄存器的内容。这些数据能被串行地移出，而不影响系统的其他部分。

18.2 调试系统

ARM7TDMI 仅仅是复杂调试系统的一个部件。调试系统通常由三部分构成：调试主机，协议转换器和 ARM7TDMI 核。

18.2.1 调试主机

调试主机通常是一台的运行软件调试器的计算机。调试主机允许用户发布高级命令，比如“在 XX 位置设置断点”或者“检测存储器地址 0X0～0X100 的内容”。

18.2.2 协议转换器

调试主机将通过一个接口（如 RS232）与 ARM7TDMI 的开发系统连接。通过该连接发送的消息必须转换成内核的接口信号。这个功能由协议转换器实现。

18.2.3 ARM7TDMI

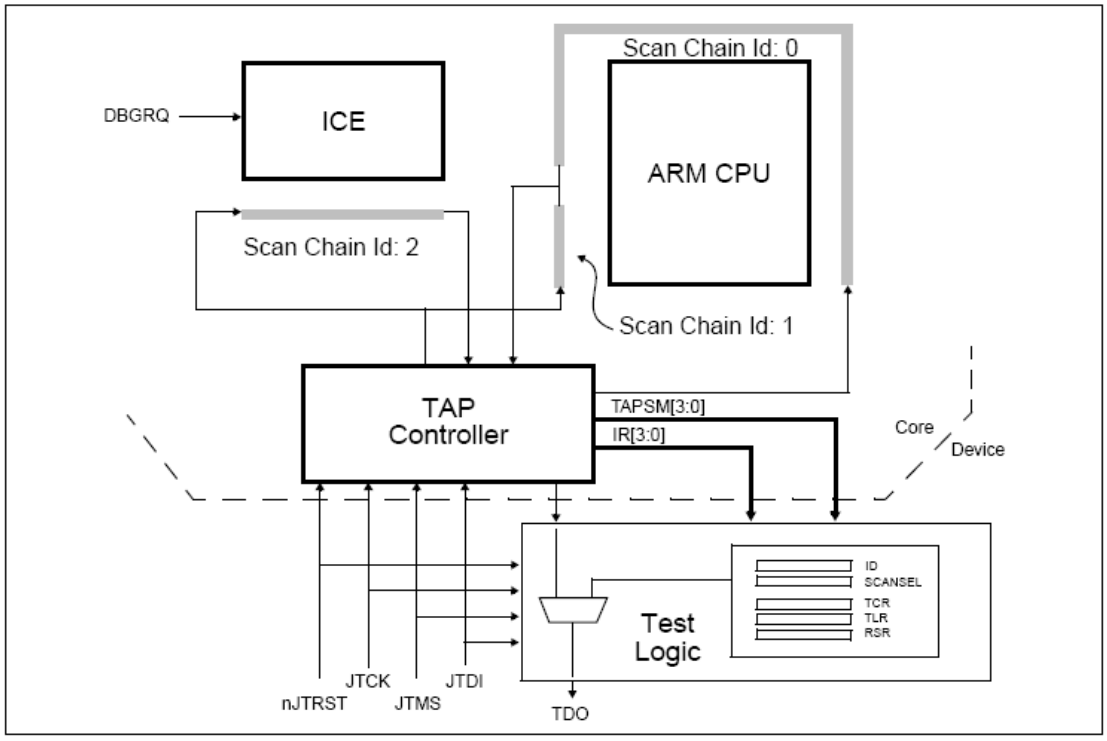
ARM7TDMI 是系统的最底层，它的调试扩展功能允许用户中止内核对程序的执行，检查其内部状态和存储器系统的状态，然后恢复程序的执行。

18.3 ARM7TDMI 调试界面

ARM7TDMI 的用于高级调试的硬件扩展的详细结构如图 83 所示。其中主要的模块是：

- CPU 核，可硬件支持调试。
- 在电路仿真单元（ICE），它通过一组寄存器和比较器产生调试异常（断点或者观察点）。
- 测试访问端口（TAP）控制器。
- 扫描链 0，1 和 2，它们围绕着 ARM CPU 和 ICE。
- 器件测试逻辑。

表 83 ARM7TDMI 扫描链分布



18.3.1 物理接口信号

根据 IEEE 1149.1 标准(JTAG)，TAP 控制器的物理接口基于 5 个信号。除了他们的用法之外，此种标准同时对于复位状态和上拉电阻的使用提供了一套说明。这些信号是：

- **nJTRST:** 测试复位。低电平有效的复位信号，用于复位 TAP 控制器有限状态机。这个引脚在上电时必须保持低电平，以便让控制器初始化（复位）。当退出初始化的时候，引脚必须被拉高。当不使用 JTAG 接口时，可以通过把 nJTRST 引脚接地来保持其复位状态。
- **JTDI**：测试数据输入。需通过外部电阻上拉。
- **JTMS**：测试模式选择。需通过外部电阻上拉。当 nJTRST 从 0 变成 1 的过程中必须是高电平。

- **JTCK**：测试时钟。用来驱动 TAP 控制器有限状态机。当 JTCK 保持低电平时，TAP 状态机无限期地保持其状态。也可以选择保持 TCK 引脚高电平来得到类似的行为。标准没有强制要求加上拉或下拉电阻。为避免静态功率消耗，建议不使用悬浮输入。
- **JTDO**：测试数据输出。这是边界扫描逻辑的输出。当不使用时是高阻输出。

根据这些标准，所有的接口信号都应该有一个如下的上拉或下拉电阻：

nJTRST: 上拉

JTDI: 上拉

JTMS: 上拉

JTCK: 下拉或上拉

JTDO: 悬浮或上拉/下拉（无论如何都不会有静态功耗）

在 STR71x 中，上述电阻都没有在内部实现。因此，需要在应用电路板上实现。在这种情况下，若不使用 JTAG 接口，所有的接口引脚都可以被永久接地。这将不会引起任何附加的功率消耗，因为没有内部上拉电阻。下面介绍的附加信号，虽不是 IEEE 1149.1 标准中的信号，也引到芯片外，使得外部用户逻辑可以请求调试事件：

DBGRQ: 处于高电平时，系统请求 ARM7TDMI 无条件地进入调试状态。当仿真功能关闭时，这个引脚必须保持低电平。

19 修订历史

表 65. 修订历史

日期	修订	修改描述
2004.3.17	1	第一个版本
2004.4.2	2	更新表 8 把文章中的 BOOTCONF 寄存器改成了 BOOTCR 改进了 18.1 章
2004.4.8	2.1	改正 STR712F 引脚输出。Pin42/43 互换。
2004.4.15	2.2	改正 PDF 中的表格链接。把 P1.11 和 P1.12 中的 USB 替换结构移除
2004.7.19	3	改正对 STDBY, V18, VSS18, V18BKP VSSBKP 的描述 更新 BSPI 的最大波特率 更改保留的系统内存模式 更改 APB 寄存器 移除多余的 I/O 图表 更新闪存编程保护部分 更新 PRCCU(48 页 3.7 章) 更新备用模式(44 页 3.6.5 章) 加入电压校准器(37 页 3.3 章) 加入 LVD (38 页 3.3 章) 更新功率(33 页 3.1 章) 更新 12C 频率公式(10.5 章) 更新 ADC(347 页 16 章) 加入计数器特殊结构(133 页 8.3 章) 把 FLASH POR 改为 LVD RES(3.7.5 章) 更新 JTAG(359 页 18 章) 移除 Emulation 章节

2004.10.9	4	改正闪存部分 B1F0/F1 地址(16 页表 1) 更新 BCON1 和 2 的复位值(31 页表 2.3.6) 改正 22 页表 1 TPFP64 TESTpin (17 改为 16) 加入 TQPFP64 pin 7 BOOTEN 和 pin 17 加入定义表格(52 页 3.7.5 章) 添加备注 UART(258 页 12.3.5 章) 改进 UART 备注(263 页 12.3.5 章) 加入 FrameError 位(263 页 12.4.3 章) 将 UART RxBufFul Flag 改成 RxBufNotEmpty(12 章)
2004.12.21	5	移除闪存编程章节在 STR7Flash 编程手册中加入参考书目 把 PU/PD 改为 Reset state(22 页表 1，器件引脚描述) 添加备注 5(47 页 3.6.5.2 章) 在 RCCU_CC 中加入 EN_HALT 位(52 页 3.7.5 章) 更新中断(81 页 5.1 章) 更新 USB(275 页 14 章)和 CAN(155 页 9 章) 把 PLL1 最大乘法器从 28 改成 24(48 页 3.7 章) 更新对 PLL 自由运定模式的描述(50 页 3.7.3 章) 在 59 页 3.7.5.4 章添加备注 更改寄存器名字改为 RCCU 和 PCU(48 页 3.7 章) 把“rw_w0”改为“rc_w1”改为“read/clear weite 0”改为“read/clear write 1”
2005.3.9	6	移除结构描述和引脚输出(更新数据表中的数据) 在“HDL 结构描述”(306 页)加入最大波特率 改正 145 页输入脉冲宽度调制公式
2005.11.24	7	移除系统内存模式更新 LVD 描述(第 3 章) 在 3.6.1 章 53 页加入 CKSTOP_EN 位 更新寄存器名字和地址信息(第 15 章) 更新 145 页 8.4.9 章 更新 USB 缓冲器地址(14 章 266 页) 改进 341 页对“ADC Data Regiter n,n=0…3(ADC_DATA[n])”的描述