TI　TMS320C2000　I2C 模块参考指南

1　I2C 总线总体特征

I2C 总线只包含两条数据线：SDA 数据线和 SCL 时钟线。在 I2C 总线传输过程中，只有主机设备牢牢控制着 SCL 时钟信号。SDA 信号线则可以由主机或从机控制。

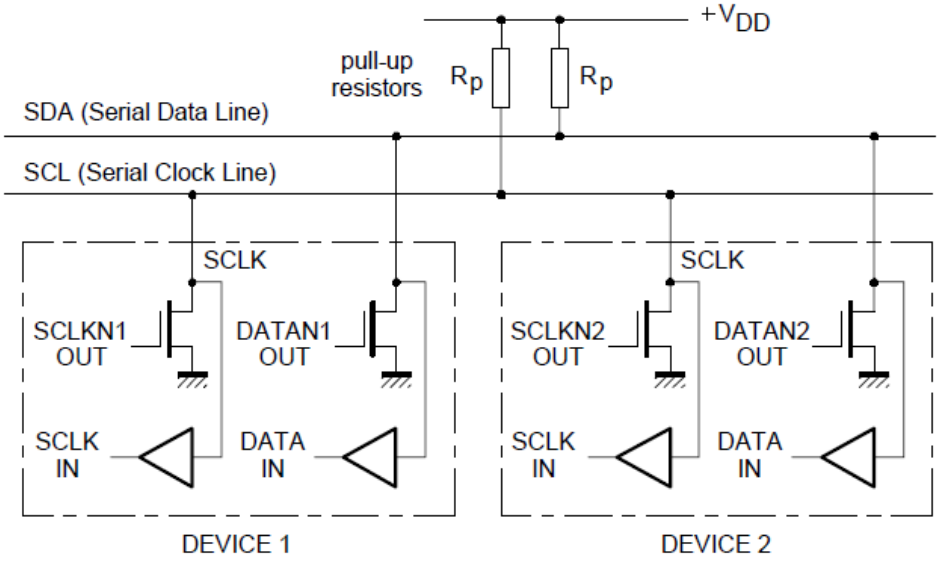

图 1　I2C 总线连接图

SDA 和 SCL 都是双向线路，通过一个上拉电阻连接到正的电源电压（如上图），总线特征包括：

a) 当场效应管接通时总线相当于接地，处于低电平状态；
b) 当场效应管断开时上拉电阻 Rp 将总线上拉至高电平；
c) 只要有设备将总线拉至低电平时，不管其它设备状态怎样，总线都将一直保持低电平；
d) 当总线空闲时时钟线和数据线路都处于高电平状态。

2　I2C 模块的时钟

I2C 模块作为一个外设模块受到外设时钟控制寄存器 PCLKCR0 管理，在配置 I2C 模块前应该确认 I2C 模块的时钟被打开（SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;　// I2C）。

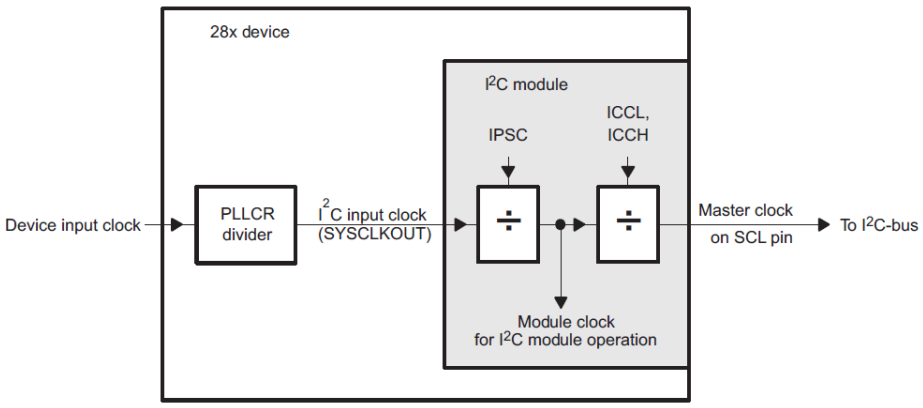

图 2　I2C 模块的时钟控制模块

如上图所示，系统时钟输出（SYSCLKOUT）直接输入到I2C模块，通过I2C分频寄存器（I2CPSC）控制的分频器进行分频，生成的I2C模块时钟，模块时钟必须在7M~12M之间，这个时钟才是模块直接使用的时钟信号。模块时钟可由下公式计算最终频率：

$$\text{模块时钟频率} = \frac{\mathbf{SYSCLKOUT}}{\mathbf{I2CPSC + 1}}$$

I2C 模块的分频器配置必须在 I2C 模块处于复位状态下进行，即 I2caRegs.I2CMDR.bit.IRS==0 时；分频器在 I2C 模块退出复位状态时开始工作；当 I2C 处于工作状态时操作 I2CPSC 寄存器是无效的。

当 I2C 模块在 I2C 总线上被配置为主设备时，主设备一直控制 SCL 时钟总线，I2CCLKH 和 I2CCLKL 寄存器分别用于控制 SCL 时钟总线的高电平和低电平持续时间，如下图所示。

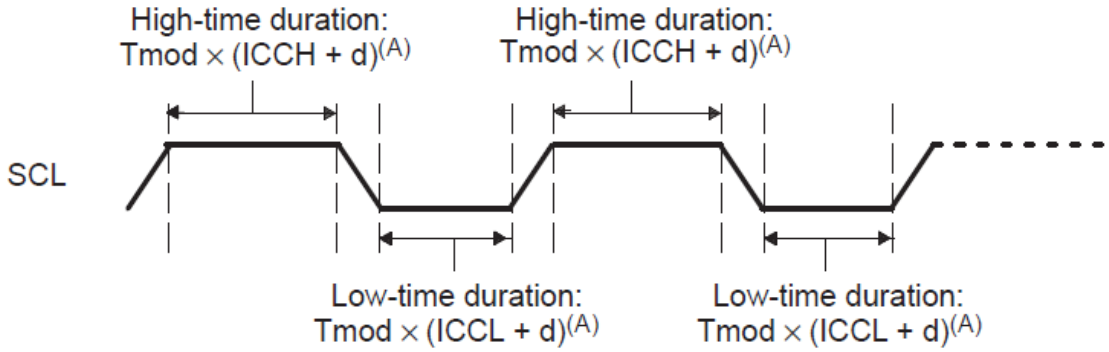

图 3　SCL 时钟高低电平时间控制逻辑

SCL 时钟总线的高电平和低电平持续时间可以下式表示：

$$T_{\mathrm{H}} = \mathrm{T}_{mod} \times (I2CCLKH + d) = \mathrm{T}_{\mathrm{sys}} \times (I2CCLKH + d) \times (I2CPSC + 1)$$

$$T_{\mathrm{L}} = \mathrm{T}_{mod} \times (I2CCLKL + d) = \mathrm{T}_{\mathrm{sys}} \times (I2CCLKL + d) \times (I2CPSC + 1)$$

其中T_{mod}表示 I2C 模块时钟的周期值，$\mathrm{T}_{\mathrm{sys}}$表示系统时钟的周期，d 表示分频延时值，可由下表决定。

表 1　I2C 分频延时值判定表

IPSC	d
0	7
1	6
Greater than 1	5

综上所述，对 I2C 模块的时钟配置流程：

a) 先配置外设时钟控制寄存器（PCLKCR0）使能 I2C 模块的输入时钟（SYSCLKOUT）；
b) 配置 I2CPSC 寄存器以生成 I2C 模块时钟，操作必须在 I2C 模块复位期间；
c) 配置 I2CCLKH 和 I2CCLKL 寄存器以控制 SCL 时钟总线的高低电平持续时间；
d) 在使能 I2C 模块后配置生效。

3　I2C 总线数据格式

一次典型的 I2C 总线传输的包含：开始条件、地址位、数据流向控制位、应答位、数据位和停止条件，如下图所示：

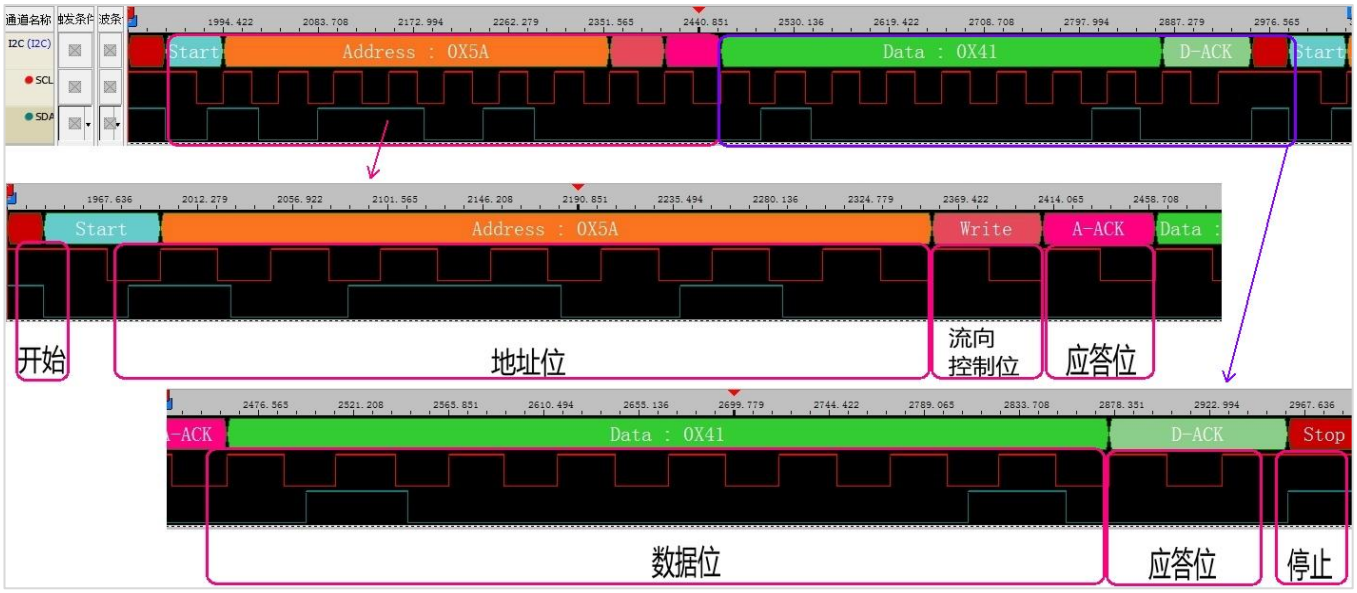

图 4　典型 I2C 传输

I2C 总线上的信号必须遵循以下规定：

a) SDA 线上的数据位、数据流向控制位、地址位、应答位必须在时钟的**高电平周期保持稳定**，数据线的电平改变只能在**SCL时钟总线的低电平时**进行；
b) 只有需要产生开始和停止条件时，SDA数据总线才能够在SCL时钟信号高电平期间发生电平改变；
c) 不管是地址还是数据都是先发送字节的最高位（MSB），即从高位到低位依次发送。

3.1 开始和停止条件

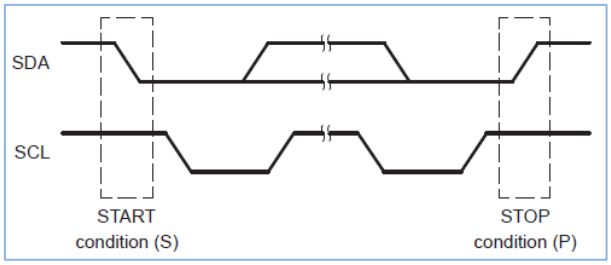

图 5　I2C 总线的开始和停止条件

I2C 总线的开始条件和停止条件如上图所示，表达的意思是：

a) 在 SCL 时钟总线处于高电平时 SDA 数据总线产生一个**下降沿**表示发生一个开始条件；
b) 在 SCL 时钟总线处于高电平时 SDA 数据总线产生一个**上升沿**表示发生一个停止条件。

I2C 总线的起始和停止条件由主机产生，总线在产生起始条件后被认为处于忙的状态，只有在产生停止条件的某段时间后总线被认为再次进入空闲状态。

特别地：如果总线传输过程中，主机想要更换寻址的从机或者改变数据传输方向并且不想释放总线的控制权时，可以再次在总线产生一个开始信号，叫做重复起始条件。因此在总线上可以有多个开始信号，但一次传输只有一次停止条件。

在 TMS320F28027 中（也适用于其它 DSP&MCU 的 I2C 模块）中，开始条件和停止条件的产生方法：

a) 当总线处于空闲状态时，如果 I2caRegs.I2CMDR.bit.MST（主机模式控制位）和 I2caRegs.I2CMDR.bit.STT（开始条件控制位）都设置为 1 时，即在设置为主机模式后产生一个开始条件，这应该非常好理解，因为开始和停止条件只能是主机产生。
b) 当总线处于忙状态（I2caRegs.I2CSTR.bit.BB==1）时，如果 I2caRegs.I2CMDR.bit.MST（主机模式控制位）和 I2caRegs.I2CMDR.bit.STP（停止条件控制位）都为 1 时，如果满足一个附加条件即会产生一个停止条件。这个外加条件是：
    - 主机处于非重复模式时（I2caRegs.I2CMDR.bit.RM == 0），如果 I2caRegs.I2CCNT 计数器减至零并且满足上述产生停止条件的条件就会产生一个停止条件；
    - 主机处于重复模式时（I2caRegs.I2CMDR.bit.RM == 1），当满足上述产生停止条件的条件时将会在下一个应答位后产生一个停止条件。

3.2 I2C 总线地址格式

I2C 总线规范中定义了两种地址格式：七位地址格式和十位地址格式。

I2C 模块中的 I2caRegs.I2CMDR.bit.XA 控制着地址格式，当 XA 为 1 时地址处于 10 位格式模式，当 XA 为 0 时处于七位地址模式。同时还需要确认自由格式控制位已被关闭（I2caRegs.I2CMDR.bit.FDF = 0），即七位地址和十位地址模式都不适用于自由模式。

不管是七位地址还是十位地址模式，都有相当的数据传送和控制方式，包括重复起始条件、停止条件、应答等，唯一不同的只是地址位数而已！

I2caRegs.I2CSAR 寄存器用于存储从机的地址。当满足开始发送条件的条件时，在发送完开始条件后，I2C 模块将 I2caRegs .I2CSAR 中的地址复制到移位寄存器并按位发送出去。

3.2.1　七位地址格式

当总线发起开始条件后，紧跟着的就是被寻址的从机地址（I2caRegs.I2CSAR），先发送高位（MSB）即 I2CSAR 的第 6 位，再依次发送其它低位，发送格式如下图所示。

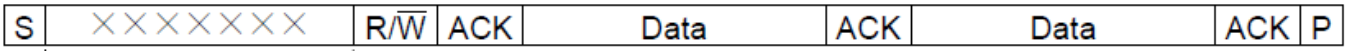

图 6　七位地址模式

在总线发送完 7 位地址后跟随的是 R/W 位，此位决定了数据流的方向，本人叫它数据流方向控制位（或许不够规范）：

a) R/W=1 时，主设备读数据（接受）到从地址设备，此时主机转换为接收器，处于主收模式；
b) R/W=0 时，主设备写数据（发送）到从地址设备，此时主机为发送器，处于主发模式。

数据流方向控制位 R/W 的值由 I2caRegs.I2CMDR.bit.TRX 位控制，其值与 R/W 值对应。所以当 MST=TRX=1 时主机处于主收模式，当 MST=1 且 TRX=0 时处于主发模式。

下图显示了一次实际的发送过程，从机的地址为 0x5A，R/W 位为 1 处于主发模式。

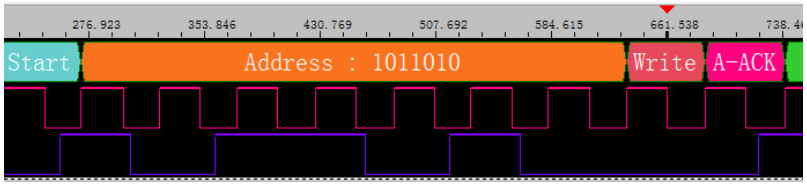

图 7　一次七位地址模式传输的时序图

3.2.2　十位地址模式

十位地址模式与七位地址模式相似，不同的是地址是以两字节发送出去的，发送格式如下图的示。

图 8 十位地址模式

当选用十位地址模式后，第一字节传输的是 11110xx，其中 xx 为十位地址的高两位，接着发送 R/W 位，第二个字节发送的是剩余的八位地址。

当 I2caRegs.I2CMDR.bit.XA=1 时 I2C 模块处于十位地址模式，其它操作与七位地址模式完全相同。

图 9 一次十位地址模式传输的时序图

上图所示为一次十位地址模式传感的时序图，从机的地址为 0x5A，大家可以与 7 位地址模式做对比。

3.3 应答位（ACK)

在 I2C 总线传输过程中，主机每发送一个字节，从机都应该产生一个额外的应答（ACK）或非应答（NACK）信号。

应答（ACK）信号的产生方式：当主机完成一个字节数据后在 SCL 低电平期间释放 SDA 数据总线，如果没有从机的作用，SDA 将会由 Rp 上拉至高电平。此时从机如果要发出应答信号就应该在 SCL 时钟线低电平期间将 SDA 数据总线拉低，主机检测到这个低电平即表示从机做出了应答。图 7 的时序图已经展示了应答信号，就不再截图了~

3.3.1 非应答位（NACK）

如果从机不想做出应答或总线出错，当主机释放 SDA 数据总线后没有从机 SDA 拉低，即表示产生一个非应答（NACK）信号。

非应答（NACK）信号主要用处有：

a) 工作在主收模式工作时，主机接收一定字节的数据后可向从机发送一个非应答（NACK）信号告诉从机不需要再发送新的数据；
b) 工作在主发模式工作时，有一个非应答（NACK）中断可用于主机对 NACK 位进行处理。

图 8 为一实例：寻址地址为 0x55 的从机但这个从机并不存在，那么主机发送第一个地址字节后将会释放 SDA 总线，但又没有从机拉低 SDA 数据总线，因而产生了一个非应答（NACK）响应。从时序图上可以看到，当主机检测到非应答（NACK）响应时将 SCL 时钟总线拉至低电平，以将阻塞其它设备的操作，所以此时需要进入 NACK 中断进行处理。

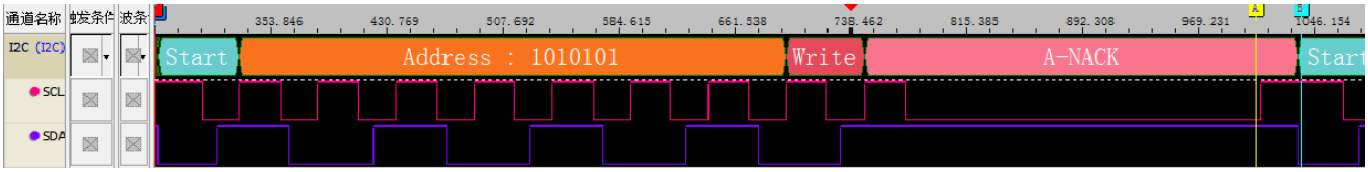

图 10 当产生 NACK 响应时的时序图

在中断处理程序中可以通过 I2caRegs.I2CMDR.bit.IRS 位清零从而复位 I2CSTR 状态寄存器，实现 I2C 模块的复位。

```
interrupt void I2CINT1A_ISR(void)
{
     Uint16 status;
     status = I2caRegs.I2CISRC.all;
     if(status == 0x02)  //Detect NACK condition
     {
          I2caRegs.I2CMDR.bit.IRS = 0;
     }
     else if(status == 0x04) // Reseive data ready
     {
          buf = I2caRegs.I2CDRR;
     }
     PieCtrlRegs.PIEACK.bit.ACK8 = 1;
}
```

3.3.2 数据位

I2C 模块支持 1-8 位数据位格式，通过 I2caRegs.I2CMDR.bit.BC 位可以控制，见表 1：

表 2　I2caRegs.I2CMDR.bit.BC 与数据位对应表

I2caRegs.I2CMDR.bit.BC 值	数据位数
3’b000	8 位数据
3’b001	7 位数据
3’b010	6 位数据
3’b011	5 位数据
3’b100	4 位数据
3’b101	3 位数据
3’b110	2 位数据
3’b111	1 位数据

在传输数据字节时，我们需要将要传输的数据写入 I2caRegs.I2CDXR 中，然后 CPU 将数据自动移至发送移位寄存器(I2CXSR)中按位移出。数据位在发送完地址位后开始传输。

如果 I2C 模块处于 FIFO 模式，则数据先写入 I2CDXR 中，系统自动并自动转存到 FIFO 队列中，发送移位寄存器将从 FIFO 中依次取数并发送出去。

4　I2C 模块的中断请求

I2C模块可以产生在七种基本的中断请求，其中两个告诉CPU什么时候写发送数据、什么时候读取接受数据。如果你想队列处理发送和接收数据，你也可以使用FIFO队列中断。

4.1 基本 I2C 中断请求

I2C模块能产生表2中描述的中断请求。如图9所示，一个仲裁器对中断请求进行仲裁，决定最高优先级的中断请求并向CPU发出中断请求，因此每次只能产生一个中断请求，但是被仲裁掉的中断请求并不会丢失，会保留在中断请求队列中。

每个I2C中断请求都在I2C状态寄存器（I2CSTR）中有一个相应的标志位，同时在I2C中断使能寄存器（I2CIER）中也有相应的中断请求使能位。每当发生中断请求时，I2C中断源寄存器（I2CISRC）会生成一个中断源特征码，用户通过I2CISRC可以确定确切的中断源，详见I2CISRC寄存器描述。

当其中一个指定I2C中断请求发生时，I2C状态寄存器（I2CSTR）中相应的标志位置1，如果I2C中断使能寄存器（I2CIER）中相应的使能位为0，中断请求将会被阻止；如果I2C中断使能寄存器（I2CIER）中相应的使能位为1，请求作为I2C中断请求发送给CPU。

I2C的基本中断请求被链接到一个可屏蔽的PIE中断中（I2CINT1A），I2C中断服务程序通过读取I2C中断源寄存器(I2CISRC)以判断中断源，可以使用IF语句调用适当的分支处理程序。

当CPU 读取中断源寄存器后，I2CISRC寄存器将会被清除，同时还会发生下列事件：

a) I2C中断状态寄存器（I2CSTR）相应的中断源被清除，除ARDY，RRD 以及XRDY 位外。在相应的位写1，该位被清除；
b) 仲裁器将会再次处理保留在中断请求队列中的中断请求，仲裁器将会将中断请求队列中具有最高优先级的中断请求发送至CPU产生中断，同时向I2C中断源寄存器（I2CSRC）中写入该中断

请求的特征码。

表 3　　基本I2C中断请求的描述

I2C中断请求	中断描述
XRDY_INT	发送寄存器准备好中断：由于先前的数据已从数据发送寄存器I2CDXR转移到发送移位寄存器I2CXSR中，数据发送寄存器I2CDXR准备好接收新的数据。
RRDY_INT	接收寄存器准备好中断：做为接收器时，接收到的新数据已经从接收移位寄存器中转存至数据接收寄存器I2CDRR中，通过RRDY_INT通知CPU数据准备好被读取。
ARDY_INT	寄存器访问准备好中断：由于先前编程的地址、数据以及命令值已经被使用，I2C模块寄存器准备好被访问。
NACK_INT	非应答条件中断：I2C模块被配置为主发送机时，当从接收机没有发送应答信号时主机产生一个非应答信号中断。
AL_INT	仲裁丢失中断：I2C模块失去和其他主发送机的仲裁竞争。
SCD_INT	检测到停止条件中断：在I2C总线上检测到停止条件
AAS_INT	被寻址中断：在I2C总线上的其他主设备把I2C配置为从设备

注意：

a) 当I2C模块处于FIFO模式时，I2C基本中断请求中的发送数据寄存器准备好（XRDY_INT）和接收数据寄存器准备好（RRDY_INT）不应该被使用，而应该使用FIFO中断；

b) 所有的I2C基本中断请求都可以使用轮询I2CSTR中的相应状态位的方法进行操作。

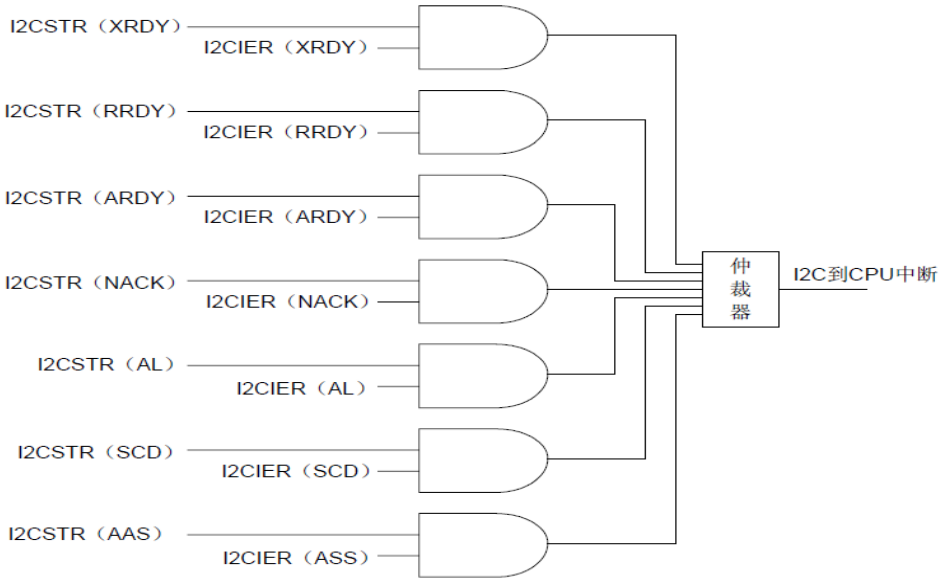

图 11　I2C 中断请求功能示意图

4.2 I2C FIFO 模式中断

除七个基本的I2C中断外，在FIFO模式中发送和接收FIFO队列各有一个FIFO模式中断。

发送FIFO可以被配置成发送一定数量的字节后产生一个FIFO发送中断请求，字节数最大可以达到4个（根据不同的MCU&DSP有所不同，如F28027等含有的是4级深度的FIFO、F2812等含有的是16级深度的FIFO）；

接收FIFO同样可以配置成接收一定数量的字节后产生一个FIFO接收中断请求，字节数最大可以达到4个。

这两个FIFO中断源集成在一个可屏蔽的PIE中断（I2CINT2A）中，中断服务程序可以通过读取FIFO中断状态寄存器来确定中断源，详见I2C发送FIFO寄存器和I2C接收FIFO寄存器的描述。

5 禁用和复位 I2C 模块

可以通过以下三种方式来复位/禁止I2C模块：

a) 禁用:通过PCLKCRO禁用I2C模块的时钟输入，整个I2C模块处于冻结状态，从而禁用I2C模块。当I2C长时间不工作时，建议禁用其时钟输入以达到省电的目的；
b) 复位：对I2C模式寄存器（I2CMDR）的I2C复位位（IRS）写0，所有的状态位被复位至默认值，在I2C复位位变为1之前，I2C模块将会一直保持为禁止状态；
c) 复位：CPU复位管脚XRS脚置0，初始化MCU&DSP复位，整个DSP一直保持复位直到XRS脚置拉高。当XRS脚为高电平时，所有的I2C 模块寄存器被复位到默认值。

当你配置或重新配置I2C模块时，I2C模块必须处于复位状态（IRS位为0），强制IRS位为0还可以被用来节约电源和用于清除错误情况。

6 I2C 模块寄存器

I2C 模块使用的寄存器列表如表 3 所示，除了发送和接收移位寄存器外所有的寄存器都直接与 CPU 连接，用户可以直接访问。

表 4 I2C 模块寄存器表

序号	寄存器名	地址	寄存器描述
1	I2COAR	0x7900	I2C 模块自身地址寄存器
2	I2CIER	0x7901	I2C 模块中断全能寄存器
3	I2CSTR	0x7902	I2C 模块状态寄存器
4	I2CCLKL	0x7903	I2C 模块低电平时钟分频值寄存器（控制低电平持续时间）
5	I2CCLKH	0x7904	I2C 模块高电平时钟分频值寄存器（控制高电平持续时间）
6	I2CCNT	0x7905	I2C 模块数据计数器寄存器（用于对输出字节进行计数）
7	I2CDRR	0x7906	I2C 模块数据接收寄存器（I2C 从总线上接收到一个字节数据后，数据将从 I2CRSR 中转移至 I2CDRR，供 CPU 读取）
8	I2CSAR	0x7907	I2C 模块从地址寄存器
9	I2CDXR	0x7908	I2C 模块数据发送寄存器(数据写入 I2CDXR 后将从转移至 I2CXSR，并逐次移到 SDA 总线上）
10	I2CMDR	0x7909	I2C 模块模式寄存器

11	I2CISRC	0x790A	I2C 模块中断源寄存器（用于查询确切的中断源）
12	I2CEMDR	0x790B	I2C 模块扩展模式寄存器
13	I2CPSC	0x790C	I2C 时钟预分频寄存器（对 SYSCLKOUT 分频输出到 I2C 模块作为 I2C 模块的工作时钟）
14	I2CFFTX	0x7920	I2C 模块 FIFO 模式下的发送寄存器
15	I2CFFRX	0x7920	I2C 模块 FIFO 模式下的接收寄存器
16	I2CRSR		I2C 模块接收/发送数据移位寄存器（不可人为操作，所以不必关心其内容）
17	I2CXSR		

注意：I2C模块的时钟来源由时钟控制寄存器PCLKCR0控制，在使用I2C模块之前需要确认PCLKCR0中I2C模块的时钟控制位已经置1。

详细寄存器定义请参考TI官方手册**SPRUFZ9** — **TMS320x2802x, 2803x Piccolo Inter-Integrated Circuit (I2C) Reference Guide** describesthe features and operation of the inter-integrated circuit (I2C) module.

7 典型应用过程

下列范例只是演示 I2C 模块的典型应用过程，并不是完整的程序，请根据自身的情况修改。

1. 全能 I2C 时钟

```
void InitPeripheralClocks(void)
{
   EALLOW;
     //省略……
   SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;      //使能 I2C时钟
     //省略……
   EDIS;
}
```

2. 初始化 GPIO

```
void InitI2CGpio(void)
{
   EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO28 = 0;    // Enable pull-up for GPIO28 (SDAA)
    GpioCtrlRegs.GPAPUD.bit.GPIO29 = 0;    // Enable pull-up for GPIO29 (SCLA)
//使能上拉电阻
    GpioCtrlRegs.GPAQSEL2.bit.GPIO28 = 3;  // Asynch input GPIO28 (SDAA)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO29 = 3;  // Asynch input GPIO29 (SCLA)
//选择GPIO为I2C功能管脚
    GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 2;   // Configure GPIO28 for SDAA operation
    GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 2;   // Configure GPIO29 for SCLA operation
    EDIS;
}
```

3. 初始化 I2C 模块

主要完成：配置模块地址、保持 I2C 复位、设置分频寄存器、设置高低电平分频寄存器、配置中断使能寄存器、退出 I2C 复位状态。

```
void InitI2Ca(void)
{
    InitI2CGpio(); //调用初始化GPIO程序
   // Initialize I2C
   I2caRegs.I2COAR = 0x005a;  //模块自身地址，用于被其它主机寻址
   I2caRegs.I2CMDR.bit.IRS = 0;//保持I2C为复位状态

   // I2CCLK = SYSCLK/(I2CPSC+1)
   I2caRegs.I2CPSC.all = 5;        // 预分频计数器，用于生成I2C使用的时钟
                                  // need 7-12 Mhz on module clk
   I2caRegs.I2CCLKL = 16;           //I2C时钟总线的低电平分频计数器，不能为0
   I2caRegs.I2CCLKH = 22;           // I2C时钟总线的高电平分频计数器，不能为0
   I2caRegs.I2CIER.bit.RRDY = 1;       // 使能RRDY中断
   I2caRegs.I2CIER.bit.NACK = 0;       //禁用NACK中断
   I2caRegs.I2CIER.bit.XRDY = 0;       //禁用XRDY中断
                                       //用户应该在此处禁用或使能中断
   I2caRegs.I2CMDR.all = 0x0020;    //退出复位模式

}
```

4. 配置 PIE

完成的工作：使能 **I2CINT1A_ISR** 中断、使能 ENPIE 位、使能 IER 相应位、开全局中断。

```
    InitPieCtrl();
    InitPieVectTable();
    PieCtrlRegs.PIEIER8.bit.INTx1 = 1;
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
    IER |= M_INT8;
    EINT;
```

5. 配置中断处理程序

主要完成的工作是读取 I2C 中断源寄存器，确定中断源后进行相应处理。

```
 interrupt void I2CINT1A_ISR(void)
 {
    Uint16 status;
    status = I2caRegs.I2CISRC.all;
    if(status == 0x05)  // Transmit data ready
    {
        //发送数据寄存器准备好中断的处理程序
    }
    else if(status == 0x02)  //Detect NACK condition
    {
        //检测到NACK中断的处理程序
    }
    else if(status == 0x04)    // Reseive data ready
```

```
        {
            buf = I2caRegs.I2CDRR;
            //接收数据准备好中断的处理程序
        }
        //其它中断源中断处理程序……….
        PieCtrlRegs.PIEACK.bit.ACK8 = 1;
    }
```

6. 配置发送数据程序

主要完成：检测 I2C 状态确认为空闲状态、配置从机地址、配置 I2C 工作模式。

```
int I2CWriteData(char data){
    if(I2caRegs.I2CSTR.bit.BB == 1)
        return 1;
    if(I2caRegs.I2CMDR.bit.STP == 1)
        return 2;
    I2caRegs.I2CSAR = 0X5A;
    //  I2caRegs.I2CCNT = times;
    I2caRegs.I2CDXR = data;         //如果是主发模式需要将数据写入I2CDXR中
                                    //如果是主收模式刚不需要，在中断中接收数据
    /*
    I2caRegs.I2CMDR.bit.IRS = 1;   //使能I2C模块
    I2caRegs.I2CMDR.bit.RM = 1;    //重复模式
    I2caRegs.I2CMDR.bit.FREE = 1;  //仿真状态下自由运行
    I2caRegs.I2CMDR.bit.MST = 1;   //设置为主设备
    I2caRegs.I2CMDR.bit.TRX = 1;   //设置为发送设备，如果TRX=0,则为接收设备
    I2caRegs.I2CMDR.bit.XA = 1;    //地址模式，1时为10位地址，0时为7位地址
    I2caRegs.I2CMDR.bit.STT = 1;   //置1后，满足停止条件时发送开始条件，详见3.1节内容
    I2caRegs.I2CMDR.bit.STP = 1;   //置1后，满足停止条件时发送停止条件，详见3.1节内容
    */
    I2caRegs.I2CMDR.all = 0x67A0;
    return 0;
}
```