**1. I2C协议数据传输格式**

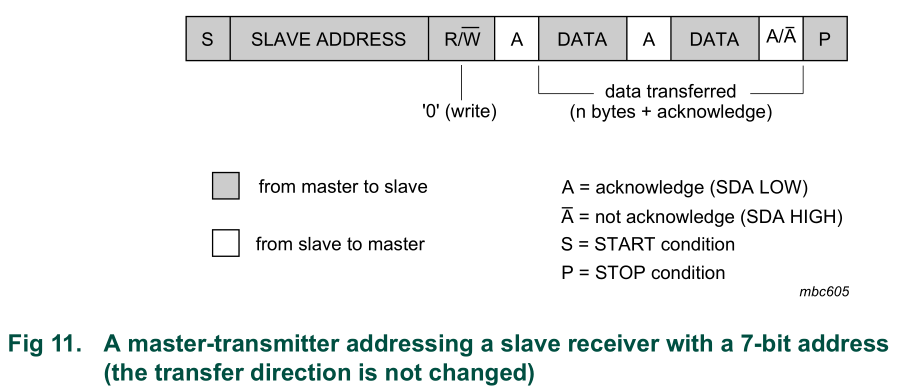


图 1 i2c协议数据传输格式

I2C数据传输以byte为单位进行，每个byte传输后都跟随一个应答位。图 1是最为常用的7位slave设备地址的I2C写数据传输格式，首先都发送起始位，然后发送7位slave设备地址，然后是读写位，“0”表示写，“1”表示读。然后由接收数据的一方给出应答位(SDA为“0”表示ack，SDA为“1”表示nack)， 传输完slave设备地址后就是传输数据，数据由若干个byte组成，同样在每个byte传输后由数据接收方给出应答位。最后是停止位。现代数字集成电路设计中，每个芯片通常对应了一个slave设备地址，同时该设备内的数据都是通过寄存器进行访问，所以就衍生出了寄存器地址的概念，基于这个理念，常用的7位slave地址的I2C master读写数据格式如下。

I2C master写：作为master进行数据写的时候，应答信号是slave设备回复给master的，可以得知整个过程数据的传输方向都是master->slave，所以数据的传输方向是没有改变的。在每个byte传输完成后master都需要去判断ack信号是否有效，如果发现slave无有效应答(SDA为“1”)则要产生一个nack\_flag电平，最终一次传输完成后产生一个done脉冲，通过done脉冲与nack\_flag电平信号判断本次传输是否有效。



图 2 i2c master写数据格式

I2C master读：作为master进行数据读的时候，应答信号既有slave设备回复给master的，也有master设备回复给slave的，所以这个数据的传输方向是双向的。可以发现i2c master读数据格式的前两个byte的传输与i2 master写的前两个byte一致，不同之处在于发送完寄存器地址后，需要重新产生一个restart信号，然后发送slave设备地址和“1”，然后开始接收从slave传送过来的数据。设master从slave处读取了N个byte数据，那么对于读取到的前N-1个byte数据，每个byte master都要将SDA拉低表示一次ack，但对于第N个byte，master不拉低SDA。



图 3 i2c master读数据格式

**2. I2C master接口列表**



图 4 i2c master接口信号框图

表 1 i2c master接口信号列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名 | 位宽 | 方向 | 时钟域 | 描述 |
| i2c\_clk | 1 | i | - | i2c master模块的时钟 |
| rst\_n | 1 | i | 异步 | 模块异步复位信号 |
| scl\_pos | 1 | i | i2c\_clk | SCL上升沿脉冲 |
| sda\_f | 1 | i | i2c\_clk | SDA滤波后的输入 |
| i2c\_master\_mode | 1 | i | i2c\_clk | master模式使能，“1”有效，若为“0”，则清“0”i2c\_nack\_flag信号，复位状态机为IDLE |
| i2c\_master\_start | 1 | i | i2c\_clk | 一次传输任务的开始，脉冲信号 |
| i2c\_read\_flag | 1 | i | i2c\_clk | master读写控制，电平信号 |
| i2c\_slave\_addr | 8 | i | i2c\_clk | slave设备地址 |
| i2c\_reg\_addr | 8 | i | i2c\_clk | slave设备寄存器地址 |
| i2c\_txdata | 8 | i | i2c\_clk | master发送的数据 |
| data\_length | 6 | i | i2c\_clk | master发送/接收数据的长度 |
| master\_sda\_oe | 1 | o | i2c\_clk | SDA线使能，为“1”拉低SDA线 |
| master\_scl\_oe | 1 | o | i2c\_clk | SCL线使能，为“1”拉低SCL线 |
| i2c\_master\_datao | 8 | o | i2c\_clk | master数据输出 |
| i2c\_master\_end | 1 | o | i2c\_clk | 一次传输任务的结束，脉冲信号 |
| i2c\_nack\_flag | 1 | o | i2c\_clk | master收到nack，电平信号 |

**3. I2C master详细设计**

**3.1 总体设计**

1、i2c\_master模块内的控制信号通过状态机和计数器产生，计数器有3个：scl\_cnt、bit\_cnt、trx\_data\_cnt。其中scl\_cnt是控制i2c master速率的计数器，bit\_cnt用来统计数据传输过程中的比特，trx\_data\_cnt主要用来统计发送/接收数据的长度从而产生TX\_DATA和RX\_DATA状态的结束信号。

2、I2C master的状态机根据I2C的读写协议进行设计，不管master是读数据还是写数据，首先都是先发送slave设备地址和寄存器地址，然后分别执行不同的操作，可以将I2C master的状态分为以下11个。

0：IDLE 状态机的初态

1：START master发送起始位

2：SADDR0 master发送slave设备地址和“0”

3：RADDR master发送寄存器地址

4：TX\_DATA master发送数据

5：RESTART master发送再起始位

6：SADDR1 master发送slave设备地址和“1”

7：RX\_DATA master接收数据

8：NACK master收到nack信号

9：STOP master发送停止位

10：DONE master完成一次任务传输

3、i2c\_master模块内置一个移位缓存器i2c\_shift\_buff，不同状态下对缓存数据输入不同，对缓存数据的操作也不同，通过计数器和状态机一同产生的控制信号控制i2c\_shift\_buff完成数据的缓存或移位。

4、SDA和SCL控制逻辑：i2c通信中时钟信号由master产生，master在发送数据时需要依据发送的数据控制SDA线，在接收数据的第9个bit需要给出ack或nack信号。

5、ACK控制逻辑：在master作为发送数据的一方时，每发送完一个byte都要对第9个bit(slave给的应答信号)进行逻辑判断决定是否产生i2c\_nack\_flag信号，而应答信号的有效与否也决定了状态机的控制流程。

**3.2 细节设计**

**3.2.1 scl\_cnt的描述**

scl\_cnt是控制i2c master速率的计数器，同时很多信号亦利用该计数器直接或间接产生控制信号。其RTL逻辑为：复位时scl\_cnt等于0，在IDLE、DONE状态和cnt\_cycl脉冲到来时scl\_cnt清0，其余情况则每个时钟周期scl\_cnt 自加1。直接利用scl\_cnt产生的控制信号有：

脉冲信号cnt\_cycl，在scl\_cnt==num\_cycl时产生，而num\_cycl是手动修改的值。num\_cycl是一个非常重要的参数，num\_cycl的值决定了i2c master传输1bit需要多少个时钟周期，也就是决定了i2c master的传输速率，譬如num\_cycl==11，那么一个SCL的持续周期为12个i2c\_clk，占空比50%。cnt\_cycl这个脉冲信号还间接参与了很多控制信号的产生

电平信号cnt\_half，其产生方式为cnt\_half = (scl\_cnt <= num\_half)。num\_half的值就是num\_cycl值的一半。

脉冲信号scl\_cntx，其产生方式为scl\_cntx = (scl\_cnt == 1)，scl\_cntx有效时更新master\_sda\_oe的值。

**3.2.2 bit\_cnt的描述**

bit\_cnt用来对一个byte传输过程中bit的数量统计。bit\_cnt的控制逻辑为：复位时bit\_cnt等于0，在NACK、STOP、DONE状态时将bit\_cnt清0，在RESTART状态时将bit\_cnt置“1”，在cnt\_cycl有效时更新bit\_cnt的值，如果bit\_cnt==9则将bit\_cnt置“1”，否则bit\_cnt自加1。

直接利用bit\_cnt产生的控制信号有：

电平信号bit\_cnt9，其产生方式为bit\_cnt==9时拉高。

**3.2.3 trx\_data\_cnt的描述**

trx\_data\_cnt用来统计i2c master模式下数据传输过程中Data的长度。trx\_data\_cnt的控制逻辑为：复位时trx\_data\_cnt等于0，在IDLE、DONE状态或trx\_data\_end脉冲有效时清“0”，在trxdata\_cnt\_add && state\_end有效时累加1，其余状态保持。

与trx\_data\_cnt相关的控制信号：

电平信号trxdata\_cnt\_add: trxdata\_cnt\_add= at\_txdata | at\_rxdata | at\_raddr

脉冲信号trx\_data\_end: trx\_data\_end = (trx\_data\_cnt==data\_length) & state\_end

**3.2.4 状态机的描述**

除上文所示，状态机描述中未出现的一些控制信号可以查看4.2.6小节



if(!rst\_n)

master\_cs <= IDLE

else

master\_cs <= #DLY master\_ns

**3.2.5 i2c\_shift\_buff的描述**

移位缓存器i2c\_shift\_buff的功能有如下几点：

在SADDR0状态到来前将{ i2c\_slave\_addr[7:1],1’b0}锁存到i2c\_shift\_buff中

在SADDR1状态到来前将{ i2c\_slave\_addr[7:1],1’b1}锁存到i2c\_shift\_buff中

在RADDR状态到来前将i2c\_reg\_addr锁存到i2c\_shift\_buff中

在TX\_DATA状态到来前将i2c\_txdata锁存到i2c\_shift\_buff中

在TX\_DATA状态或RX\_DATA状态通过移位的方式传输数据

i2c\_shift\_buff相关的控制信号6个，分别为enter\_txsaddr0、enter\_txsaddr1、enter\_txraddr、enter\_txdata、data\_rx\_shift、data\_tx\_shift。望文生意，从控制信号的命名可以明白其功能。这6个控制信号的优先级顺序为enter\_txsaddr0 > enter\_txsaddr1 > enter\_txraddr > enter\_txdata > data\_rx\_shift | data\_tx\_shift

这6个控制信号都是脉冲信号，其产生方式与功能如下：

enter\_txsaddr0 = cnt\_cycl & at\_start，该控制逻辑很容易理解，在进入START状态后scl\_cnt就开始计数，当scl\_cnt计数达到num\_cycl时就会产生cnt\_cycl信号。根据状态机描述可知下一个时钟周期进入SADDR0状态，将{ i2c\_slave\_addr[7:1],1’b0}锁存到i2c\_shift\_buff中。

enter\_txsaddr1 = cnt\_cycl & at\_restart，在master需要进行读数据操作时，需要将{ i2c\_slave\_addr[7:1],1’b1}锁存到i2c\_shift\_buff中。处于RESTART状态时，结束脉冲double\_end到来后状态机进入SADDR1。而enter\_txsaddr1与第一个cnt\_cycl脉冲对齐，完全满足。

enter\_txradd = state\_end & at\_saddr0，RADDR状态出现在SADDR0状态之后，所以在SADDR0状态结束时产生enter\_txradd脉冲。

enter\_txdata = state\_end & (at\_raddr | at\_txdata)，在RADDR状态结束或一个TX\_DATA状态结束但仍需要发送另一个数据时，会产生enter\_txdata脉冲。

data\_tx\_shift = at\_txbyte & cnt\_cycl & (~bit\_cnt9)，其中at\_txbyte是电平信号，

at\_txbyte = at\_saddr0 | at\_raddr | at\_txdata | at\_saddr1，data\_tx\_shift的逻辑可以这样理解：在SADDR0、、SADDR 、RADDR、TX\_DATA这几个状态都是master作为发送数据的一方，而每个bit的传输需要num\_cycl+1个时钟周期，第9个bit由master接收slave的应答，所以需要加上(~bit\_cnt9)逻辑。

data\_rx\_shift = at\_rxdata & scl\_pos & (~bit\_cnt9)，在RX\_DATA状态需要产生data\_rx\_shift信号，前8个bit在SCL的上升沿到来是进行移位，第9个bit是应答信号，不需要缓存，用另外的电路对其进行处理。

**3.2.6 控制信号的描述**

**3.2.6.1 state\_end**

脉冲信号state\_end，将cnt\_cycl & bit\_cnt9 打1拍得到state\_end信号，state\_end 信号可以理解为每个byte传输完成后产生的脉冲信号，当然用state\_end进行表述可能不太恰当，因为TX\_DATA和RX\_DATA可能需要传输多个byte，那么就会产生多个state\_end脉冲。

**3.2.6.2 start\_xfer**

脉冲信号start\_xfer，IDLE状态下，该脉冲有效时，master\_ns(状态机的下一状态)变为START，否则保持IDLE状态。若i2c master的时钟与系统控制时钟同步，则可以将i2c\_master\_start与start\_xfer直连，否则需要通过同步电路完成i2c\_master\_start与start\_xfer之间的转换。

**3.2.6.3 doubel\_end**

脉冲信号double\_end，RESTART状态下，该脉冲有效时，master\_ns变为SADDR1，否则保持RESTART状态。double\_end = double\_flag & cnt\_cycl

与double\_end相关的信号：

电平信号double\_flag，产生方式：

复位时置“0”，在非RESTART状态时清“0”，在RESTART状态下，cnt\_cycl脉冲有效时置“1”，其余状态保持。该信号的设计思路其实就是让RESTART的状态持续2倍num\_cycl+1个i2c时钟周期。

**3.2.7 ACK控制逻辑**

主要就是产生电平信号ack\_ok，该信号给状态机进行状态跳转的判断，用来表征i2c master作为发送数据一方时slave是否给与了有效的应答信号，高有效。其控制逻辑为：复位时清“0”，i2c\_master\_start脉冲到来时清“0”，ack\_check脉冲有效时更新ack\_ok的值(对sda\_f取反)，其余状态保持。

与ack\_ok相关的信号：

脉冲信号ack\_check，产生方式:

ack\_check = scl\_pos & bit\_cnt9 & (at\_saddr0 | at\_raddr | at\_txdata | at\_saddr1)

该逻辑也很容易理解，只有当master作为发送数据的一方时需要进行ack信号的检测，那么根据上述所规定协议，master在SADDR0、RADDR、TX\_DATA、SADDR1状态时是作为发送数据的一方，又ack信号的检测只需要在每个byte传输后进行，故将bit\_cnt9高电平作为与逻辑，在SCL上升沿脉冲到来时检测，所以用scl\_pos作与逻辑。

master处于SADDR0、RADDR、TX\_DATA、SADDR1状态时，每个byte传输完成后都会检测ack\_ok的电平状态，若ack\_ok拉低，则状态机跳转到NACK状态，由于NACK状态只持续一个是时钟周期，即产生的at\_nack是脉冲信号。

i2c\_nack\_flag是一个输出电平信号，其控制逻辑为，当i2c\_master\_mode为“0”或start\_xfer脉冲到来时清“0”，当at\_nack脉冲到来时置“1”，其余状态将信号锁存，如此一来，只要在i2c的数据传输过程中产生了at\_nack脉冲，i2c\_nack\_flag就会被拉高，直到i2c控制器的模式改变或下一次传输任务的到来，i2c\_nack\_flag才会被拉低，方便接口电路进行逻辑判断。

**3.2.8 SDA控制逻辑**

master对SDA线的控制主要有以下几个方面：

1. 在START和RESTART状态产生起始位
2. 在SADDR0、SADDR1、RADDR、TX\_DATA状态的非应答位依据需要发送的数据产生相应的SDA信号
3. 在RX\_DATA状态的应答位产生ack或nack。假设master从slave读取N个byte，那么对于前N-1个byte，master需要产生ack，对于第N个byte，master需要产生nack。

本设计中master\_sda\_oe寄存器信号直接决定了SDA的电平，master\_sda\_oe为“1”，拉低SDA线，master\_sda\_oe为“0”，释放SDA线。master\_sda\_oe的控制逻辑为：

复位时将master\_sda\_oe清“0”，IDLE状态时，将master\_sda\_oe清“0”，scl\_cntx脉冲有效时将sda\_oe\_pre的值更新到master\_sda\_oe中，其余状态master\_sda\_oe保持。

sda\_oe\_pre由组合逻辑产生，其逻辑设计为：

sda\_oe\_pre = rx\_set\_low | tx\_set\_low | at\_start | at\_stop | at\_restart & double\_flag

其中只要状态机一进入STOP或START，就在scl\_cntx有效时将sda\_oe\_pre=1的值更新到master\_sda\_oe中，将master\_sda\_oe拉低。在RESTART状态时，sda\_oe\_pre的状态由double\_flag决定，由于double\_flag在RESTART状态的第2个scl\_cnt的计数周期才会拉高，所以在RESTART状态，SDA线拉高的电平持续时间约为num\_cycl个时钟周期。

tx\_set\_low = at\_txbyte & (~bit\_cnt9) & (~i2c\_shift\_buff[7])，其中at\_txbyte状态包括SADDR0、SADDR1、RADDR、TX\_DATA。该逻辑也很容易理解，at\_txbyte拉高表示的是master需要发送数据的状态，在这些状态下。SDA的电平由发送的数据决定，master通过移位的方式将数据从最高有效位开始发送，所以需要和(~i2c\_shift\_buff[7])作与逻辑，而第9bit是接收来自slave的应答信号，故第9bit应该释放SDA线，所以需要和(~bit\_cnt9)作与逻辑。

rx\_set\_low = at\_rxdata & bit\_cnt9 & (~(trx\_data\_cnt==data\_length))，rx\_set\_low则是在master作为接收数据的一方(读数据)时的状态控制，master作为接收数据的一方时，SDA线上数据的传输由slave产生，master只负责将信号通过i2c\_shift\_buff存储，但在每个byte传输完成后都要给一个ack信号(最后一个byte除外，最后一个byte需要给nack信号)，所以rx\_set\_low需要用at\_rxdata、bit\_cnt9、（~(trx\_data\_cnt==data\_length)）作与逻辑，(~(trx\_data\_cnt==data\_length))确保了在最后一个byte完成传输后，rx\_set\_low的值为0，从而使得最后一个byte传输后产生nack信号。

**3.2.9 SCL控制逻辑**

本设计中SCL的控制逻辑由master\_scl\_oe决定，master\_scl\_oe为“1”，则拉低SCL线，master\_scl\_oe为“0”，则释放SCL线。master\_scl\_oe由scl\_set\_low打一拍得到。

scl\_set\_low完全由组合逻辑电路产生，其控制逻辑为：

scl\_set\_low = ( at\_txbyte | at\_rxdata | at\_stop | at\_nack | (at\_restart & (~double\_flag)) ) & cnt\_half

硬件i2c master的读写过程中时钟信号都由master产生，其中进入收发数据的状态(at\_txbyte、at\_rxdata)、停止(at\_stop)和非应答(at\_nack)状态对SCL的控制逻辑都是相似的，本设计中没有将SCL线拉低的操作，除了RESTART 状态，其余状态SCL的产生都是连续的，也就是说SCL的持续时间都是num\_cycl个时钟周期。而对于START状态，整个过程中SCL都是拉高的，进入SADDR0状态后，cnt\_half就拉高了，所以在过一个时钟周期SCL线就拉低，然后周期性的产生SCL信号；进入RESTART状态，对SCL的控制有稍许不一样，可以这样理解，整个RESTART状态如果不加逻辑控制，那么SCL线是一直拉高的，在cnt\_half为“1”且double\_flag为“0”时拉低SCL线，RESTART状态结束拉低SCL线，开始周期性产生SCL信号，这样就可以利用scl\_cnt产生i2c协议要求的重起始位。而在STOP状态，cnt\_half为“0”后，SCL过一个时钟周期就拉高了，等到cnt\_half重新拉高后，状态机已经进入DONE状态和IDLE状态了，自然也就不会在对SCL线进行操作。

**4. I2C master的最大速率**

整个设计中没有对SCL线拉低的操作，所以用多少个时钟周期去产生一个SCL时钟便成了衡量i2c master传输速率的标志。本设计中num\_cycl决定了一个SCL包含多少个时钟周期，对i2c master的速度启主导作用，除此之外，对PAD的输入信号是否有作同步和滤波也会影响i2c master的速度。