**1. I2C协议数据传输格式**

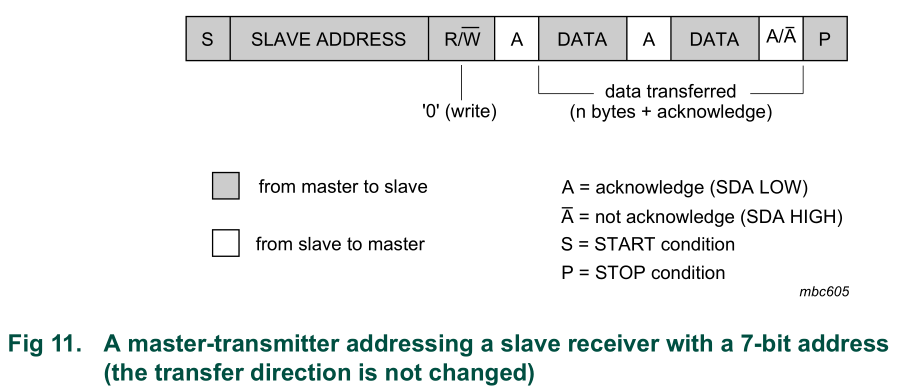


图 1 i2c协议数据传输格式

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 slave写：slave进行数据写传输的时候，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。此时slave作为接收数据的一方，完成每个byte接收后都要在下一个bit传输时拉低SDA线作为给master的一个ack。



图 2 i2c master写数据格式

I2C slave读：slave进行数据读传输的时候，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。此时slave既是接收数据的一方(接收slave设备地址、寄存器地址)，又是发送数据的一方(发送数据给master)，在完成slave设备地址、寄存器地址的接收后都要给master一个ack信号，处于发送数据的状态时，每发送完一个byte，都要接收来自master的应答信号，若为ack，则继续发送数据，否则回到起始状态。



图 3 i2c master读数据格式

**2. I2C slave接口列表**



图 4 i2c slave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 slave模块的时钟 |
| rst\_n | 1 | i | 异步 | 模块异步复位信号 |
| scl\_i | 1 | i | i2c\_clk | 滤波后的SCL输入 |
| sda\_i | 1 | i | i2c\_clk | 滤波后的SDA输入 |
| i2c\_slave\_addr | 7 | i | i2c\_clk | slave设备地址 |
| bus\_rdata | 8 | i | i2c\_clk | slave待发送的数据 |
| sda\_oe | 1 | o | i2c\_clk | slave SDA线使能，为“1”拉低SDA线 |
| bus\_sel | 1 | o | i2c\_clk | 一次有效的bus传输，“1”有效 |
| bus\_wr | 1 | o | i2c\_clk | bus读写标志，“1”写，“0”读 |
| bus\_wdata | 8 | o | i2c\_clk | slave接收到的数据，通过bus写入寄存器 |
| bus\_addr | 8 | o | i2c\_clk | bus寄存器地址 |

**3. I2C slave详细设计**

**3.1 总体设计**

1、状态机设计。I2C slave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 slave的状态分为以下7个。其中起始和停止信号都是由master产生的，slave模块只需要在检测到该信号产生后跳转到设计好的状态即可。其中检测到停止信号的产生状态机跳转到IDLE，检测到起始信号跳转到S\_ADDR。

0：IDLE 状态机的初态

1：S\_ADDR slave接收slave设备地址和读写位

2：R\_ADDR slave接收寄存器地址

3：W\_DATA0 slave接收master发送的数据

4：W\_DATA1 slave接收master发送的数据

5：RX\_DATA0 slave发送数据给master

6：RX\_DATA1 slave发送数据给master

2、i2c\_slave内设置一个bit\_cnt，用来对传输过程中的bit进行计数

3、i2c\_slave内设置tx\_buffer和rx\_buffer分别作发送和接收数据时的移位缓存寄存器。

4、SDA逻辑：i2c通信中时钟信号由master产生，本设计slave不具有强制拉低SCL的功能，故没有对SCL总线控制的逻辑。对SDA线的控制体现在以下几个方面：1、master发送数据时，slave每接收完一个byte需要拉低SDA表示一个ack。2、master接收数据时，slave通过控制SDA线传递数据。

5、ACK控制逻辑：在master作为接收数据的一方时，每接收完一个byte都要给slave一个应答信号，该信号是slave决定是否要结束数据发送的依据，通常规定的是，假设master从slave处读取N个byte，那么前N-1个byte数据master需要发送ack信号，最后一个byte master需要发送nack信号，这样slave收到nack信号后便不会再发送数据给master了。

**3.2 细节设计**

**3.2.1 状态机描述**

状态机描述中未出现的控制信号可以在XX小节中查看



if(!rst\_n)

state <= IDLE

else if (stop)

state <= #DLY IDLE

else if (start)

state <= #DLY S\_ADDR

else

state <= #DLY state\_next

**3.2.2 bit\_cnt的描述**

bit\_cnt用来对传输过程中的bit进行计数，并产生一些相关的控制信号。其控制逻辑为：复位时bit\_cnt等于0，start或stop脉冲到来时，bit\_cnt清“0”，scl\_neg脉冲到来时，如果cnt\_eq9为“1”则将bit\_cnt置为“1”，否则就将bit\_cnt自加1。

利用bit\_cnt分别产生cnt\_eq9、cnt\_eq8、cnt\_eq7电平信号，其分别表示bit\_cnt==9、bit\_cnt==8、bit\_cnt==7。

**3.2.3 tx\_buffer和rx\_buffer**

rx\_buffer是slave接收数据的移位缓存寄存器，包括缓存slave设备地址、寄存器地址、接收到的数据。其控制逻辑为:在start或stop脉冲到来时将rx\_buffer清“0”，在scl\_pos有效且cnt\_eq9为“0”时，将rx\_buffer左移一位，最低位补sda\_ok：

(rx\_buffer <= #DLY {rx\_buffer[6:0]，sda\_ok})，与其相关的操作有：rx\_buffer[6:0]会和i2c\_slave\_addr进行比较产生addr\_equal信号；在addr\_latch有效时会将rx\_buffer锁存到bus\_addr中；bus\_wdata与rx\_buffer直接相连。

tx\_buffer的控制逻辑为：复位时清零，rdata\_latch脉冲到来时将bus\_rdata锁存到tx\_buffer中，在rdata\_shift脉冲到来时，对tx\_buffer左移一位，最低位补1’b0。

rdata\_latch为bus\_sel & (~bus\_wr)打一拍得到。

**3.2.4 SDA逻辑**

slave通过寄存器信号sda\_oe控制SDA线，sda\_oe为“1”则拉低SDA，为“0”则释放SDA。其控制逻辑为：在复位时sda\_oe变“0”，其余状态根据当前状态进行译码。

IDLE: sda\_oe <= #DLY 1’b0

S\_ADDR: sda\_oe <= #DLY cnt\_eq9 ? sadd\_match : 1’b0

RADDR，W\_DATA0，W\_DATA1：sda\_oe <= #DLY cnt\_eq9 ? 1’b1 : 1’b0

R\_DATA0，RDATA1： sda\_oe <= #DLY cnt\_eq9 ? 1’b0 : ~tx\_buffer[7]

default： sda\_oe <= #DLY 1’b0

**3.2.5 ACK控制逻辑**

电平信号tx\_ack的功能为：i2c\_slave处于RDATA0或RDATA1状态时，状态跳转的判断信号。其逻辑设计为：复位时tx\_ack清“0”，在txack\_check脉冲到来时，将sda\_ok(滤波或同步后的SDA信号)取反后赋值给tx\_ack，在start或stop脉冲到来时将tx\_ack清“0”。

脉冲信号txack\_check = ack\_phase & at\_rdata；slave只有在RDATA0或RDATA1状态才需要接收来自master的应答信号。

脉冲信号ack\_phase = cnt\_eq9 & scl\_pos，一个byte传输完成后，在第9个bit进行ack逻辑判断。

**3.2.6 其他控制信号**

**3.2.6.1 bus相关信号**

**3.2.6.1.1 wr\_flag**

电平信号wr\_flag，“1”表示master写，“0”表示master读，在S\_ADDR状态控制状态机的跳转。其控制逻辑为：复位时wr\_flag清“0”，在rw\_check脉冲到来时，将sda\_ok(滤波或同步后的SDA信号)取反后赋值给wr\_flag，在start或stop脉冲到来时将wr\_flag清“0”。

脉冲信号rw\_check = scl\_pos & cnt\_eq8 & at\_saddr，该逻辑很容易理解，在S\_ADDR状态，第8bit是读写位，通过上升沿脉冲去抓取取反后的sda\_ok信号。

**3.2.6.1.2 bus\_wr**

脉冲信号bus\_wr将wr\_valid打一拍得到，bus\_wr是总线寄存器的读写标志。“1”表示将slave收到的数(bus\_wdata)据写入寄存器中，“0”表示将寄存器的数据(bus\_rdata)锁存到tx\_buffer中。脉冲信号wr\_valid = at\_wdata & ack\_phase，该逻辑很容易理解，ack\_phase产生时slave接收数据已经完成了，所以利用该信号造出wr\_valid脉冲。

**3.2.6.1.3 bus\_sel**

脉冲信号bus\_sel，将wr\_valid | rd\_valid 打一拍得到。其中脉冲信号

rd\_valid = (at\_saddr | at\_rdata) & ack\_phase & (~wr\_flag) & (~sda\_ok)，该信号约束较多，从以下几个方面进行理解，rd\_valid表示的是maste需要从slave读取数据，所以在发送数据前就必须通过总线将寄存器的数据锁存到tx\_buffer中。所以第一个约束逻辑就是(at\_saddr | at\_rdata)，因为必须在发送数据前将数据锁存到tx\_buffer中，所以对于第一个数据的发送就必须在S\_ADDR状态结束时将要发送的数据锁存，如果要发送第二个数据，那么就是在第一个R\_DATA状态将要发送的数据锁存；ack\_phase表示一个byte已经传输完毕，也就是S\_ADDR或R\_DATA状态完毕；(~wr\_flag）表示的是必须为master从slave

读数据才产生rd\_valid，从而避免在master向slave写数据的过程中在S\_ADDR状态时产生rd\_valid；最后一个约束条件是(~sda\_ok)，是为了避免master读取完最后一个数据后还产生rd\_valid，因为master读取完最后一个数据，在第9bit不会拉低SDA，从而避免产生了多余的rd\_valid。

**3.2.6.1.4 bus\_addr、bus\_wdata**

bus\_wdata直接与rx\_buffer，只有在bus\_sel与bus\_wr有效时才会将bus\_wdata数据写入。

bus\_addr是i2c与寄存器的地址接口，其控制逻辑为：复位时将bus\_addr清“0”，addr\_latch脉冲有效时将rx\_buffer锁存到bus\_addr中，addr\_incr脉冲有效时，bus\_addr自加1。

bus\_addr相关的控制信号有：

addr\_latch = at\_raddr & ack\_phase；

raddr\_incr = txack\_check；

waddr\_incr = at\_wdata & state\_end;

addr\_incr = waddr\_incr | raddr\_incr

**3.2.6.2 其他信号**

**3.2.6.2.1 SDA/SCL上升下降沿脉冲、start/stop**

sda\_ok和scl\_ok是滤波或同步后的信号，可以用来造出所需要的控制信号。先将sda\_ok信号打一拍得到sda\_ok\_r0，scl\_ok信号打一拍得到scl\_ok\_r0。利用打拍后的信号和

未打拍的信号造出以下若干信号：

sda\_pos = sda\_ok & ~sda\_ok\_r0

sda\_neg = ~sda\_ok & sda\_ok\_r0

scl\_pos = scl\_ok & ~scl\_ok\_r0

scl\_neg = ~scl\_ok & scl\_ok\_r0

start = sda\_neg & scl\_ok

stop = sda\_pos & scl\_ok

**4. I2C slave的最大速率**

整个设计中没有对SCL线拉低的操作，i2c slave的能够支持的最大速度主要受限于i2c模块的时钟，当然对输入的SDA和SCL进行同步与滤波也会使i2c的速率有所下降。