I2C 通信原理

一、

I2C 的时序是比较复杂的，你如果能把 I2C 的时序弄清，那其他器件的时序都不成问题了。

我就按照我的理解来跟你讲吧。

直接用程序来说明吧。

```
NOP(), 一个机器周期时间的延迟,  12M 晶振时为 1 微秒
NOPS （）， 4 个 NOP() 。
sbit SDA P2^0;
sbit SCL P2^1;  定义数据线和时钟线接口
```

首先，I2C 有 2 个重要的线， SDA 数据线 SCL 时钟线， 当总线上没有进行信息传送时， SDA 和 SCL 都为高电平，我们称之为释放总线。

开始传送信息时，要有一个开始信号，

开始信号 ：定义为在 SCL 为高电平的时候， SDA 从高电平拉低。

```
start()
{
  SDA=1;
  NOP();        //同你图中 SDA/SCL 上升 / 下降所用时间 1US
  SCL=1;
  NOPS();       //建立开始信号（同你图中 TSU 起始信号建立时间一样 4US ）
  SDA=0;        //SDA 拉低
  NOPS();
  SCL=0;        //SCL 拉低， 钳住总线，准备发送或接收数据
  NOP();
}
```

结束信号：与开始信号相反，在 SCL 为高电平时， SDA 从低拉高

```
stop （）
{
  SDA=0;
  NOP();
  SCL=1;
  NOPS();    //建立信号时间
  SDA=1;      //拉高 结束。
  NOPS();
}
```

我晕，发现写了半天还有好多。。算了 帮人帮到底了

发送 /接收一个数据：数据的发送和接收都是在 SCL 为低电平的时候发生，因为 SCL 为高电平时已给了开始和结束信号。 发送数据时， 当数据准备读入时， 将 SCL 线暂时拉高 （SCL 为高时， SDA 无法改变状态），保持一段时间然后拉低（同你图中的 TDH ，数据输出保持时间），这时数据则发送完毕到 SDA 上。接收则与之大同小异。当 SDA 线上有数据过来时，先将 SCL 拉高，建立好时间，然后拉低，数据则被读入。（关于如何被发出以及如何

被读入则是芯片做的事，我们不用管，只需记住　　SCL 拉低，数据发出 /读入）

用程序来讲就是：

```
send(uchar c)                  //发送一个字节
{
   uint i;
   for(i=0;i<8;i++)             //该字节 8 位从高往低发送
      {
         if((c<<i)&0x80) SDA=1;
            else SDA=0;
           NOP();
         SCL=1;                 //建立信号时间
         NOPS();
         SCL=0;                 //发送完毕
      }
}
recieve()
{
    uchar r;uint i;
    r=0;SDA=1;
    for(i=0;i<8;i++)          //读取 8 位数据
     {
        NOP();
         SCL=0;
         NOPS();
         SCL=1;
         NOP();
         r<<=1;
         if(SDA==1) r+=1;
         NOP();
      }
       SCL=0;
       NOP();
       return (r);
}
```

程序可能有点难懂，不过没关系，使用　　I2C 时候，直接调用写好的程序，如我写的

start.stop.send receive　　等，这些程序应该有现成的。

发送一个字节后会有一个应答函数，应答函数定义为，当发送完一个字节也就是　　8 位的时候，如果这是 SDA 为低电平，则为有应答，反之则无。如果要发送多个字节，一个字节发完后需要一个应答函数才能继续发下面一个字节。程序也就大同小异。掌握好时序就行了。

I2C 用的最多的就是 E2PROM 了。

给你举个例子

如果我想往 E2PROM 的 0x50 单元写一个‘ a ’，则我这样写：

```
start();          //开始
send(0xa0);     //选择我要发送的器件。     0xa0 为 E2PROM  的器件地址
answer();         //应答
send(0x50);     //选择 E2PROM  中的 0X50 地址单元
answer();         //应答
send('a');       //发送数据‘  a ’
stop();           //停止
```

二

摒弃复杂的情况，这里只对 I2C 做简单的介绍。

一、 I2C 总线的一些特征：

? 只要求两条总线线路一条串行数据线 SDA 一条串行时钟线 SCL

? 每个连接到总线的器件都可以通过唯一的地址和一直存在的简单的主机从机关系软件设定地址主机可以作为主机发送器或主机接收器

? 它是一个真正的多主机总线如果两个或更多主机同时初始化数据传输可以通过冲突检测和仲裁防止数据被破坏

? 串行的 8 位双向数据传输位速率在标准模式下可达 100kbit/s 快速模式下可达 400kbit/s 高速模式下可达 3.4Mbit/s

? 片上的滤波器可以滤去总线数据线上的毛刺波保证数据完整

? 连接到相同总线的 IC 数量只受到总线的最大电容 400pF 限制

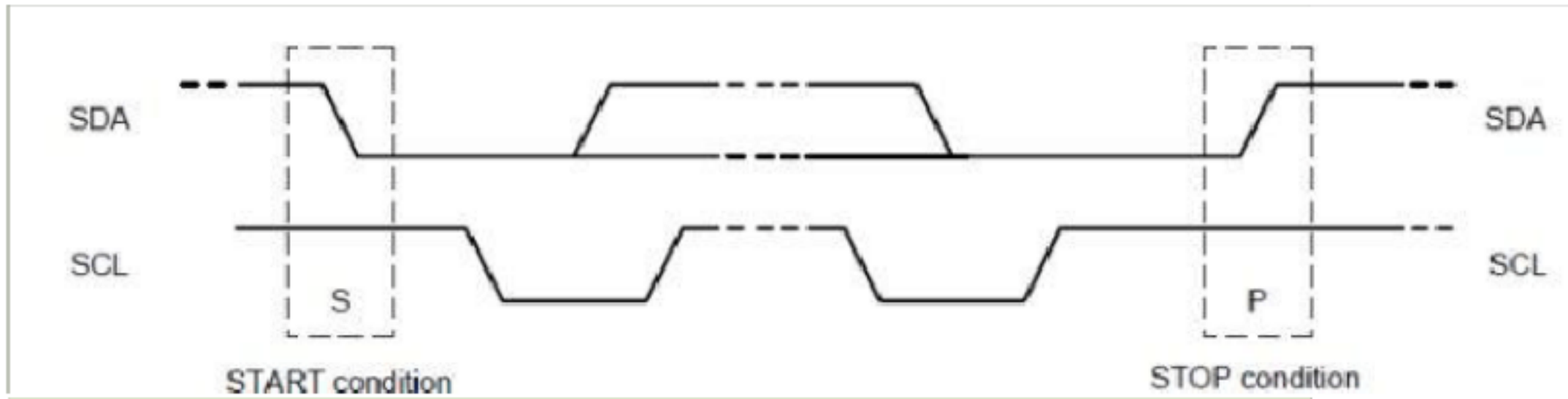
二、 I2C 总线在传送数据过程中共有三种类型信号：开始信号、结束信号和应答信号。

开始信号： SCL 为高电平时， SDA 由高电平向低电平跳变，开始传送数据。

结束信号： SCL 为高电平时， SDA 由低电平向高电平跳变，结束传送数据。

应答信号：接收数据的 IC 在接收到 8bit 数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据

起始和结束：




```
bool I2C_Start(void)
{
 SDA_H;
 SCL_H;
 I2C_delay();
 if(!SDA_read)return FALSE; // SDA 线为低电平则总线忙 ,退出
 SDA_L; // 拉低 SDA 线(当 SCL 为高电平时, SDA 由高电平向低电平跳变表示开始信号 )
 I2C_delay();
 if(SDA_read) return FALSE; // SDA 线为高电平则总线出错 ,退出
 SDA_L; // 数据为准备好时, 拉低 SCL 线
 I2C_delay();
 return TRUE;
}
```

发出开始信号之后，设备在数据未准备好时，拉低 SCL 线，这样主设备可知从设备未发送数据，从设备在数据准备好，可以发送的时候，停止拉低 SCL 线，这时候才开始真正的数据传输

```
void I2C_Stop(void)
{
 SCL_L;
 I2C_delay();
 SDA_L;
 I2C_delay();
 SCL_H; // SCL 为高电平时, SDA 由低电平向高电平跳变，结束传送数据
 I2C_delay();
 SDA_H;
 I2C_delay();
}
```

STOP 在单主环境下非必要，但在多主环境就非常必要，主控总线的设备发送 STOP 后，通知总线其他设备总线已经闲置 .

```
void I2C_Ack(void)
{
 SCL_L;
 I2C_delay();
 SDA_L;
 I2C_delay();
 SCL_H;
 I2C_delay();
 SCL_L;
 I2C_delay();
}
```

当主机作为接收设备时，主机对最后一个字节不应答，以向发送设备 (从设备)标识数据传送结束。 这是因为每次传输都应得到应答信号后再进行下一个字节传送。 如果此时接收机应答了，那它就接收的不是最后一个字节了。如果是最后一个字节，第 9 个时钟周期发送的是非应答信号 (此时发送的不是应答信号就是非应答信号)，最后发送停止信号。

1) 主发从收： 主 START-> 主发地址 -> 从 ACK->(主发数据 -> 从 ACK(循环))-> 主 STOP 或主 START 启动下一次传输

这一过程中，主控 SCL 线，从只在 ACK 时控 SDA 线，其他时刻主控 SDA 线。

2) 主收从发：主 START-> 从发地址 -> 主 ACK->(从发数据 -> 主 ACK(循环))-> 接受至最后一个字节时，主 NACK-> 主 STOP 或主 START 启动下一次传输

并非每传输 8 位数据之后，都会有 ACK 信号，有以下 3 中例外

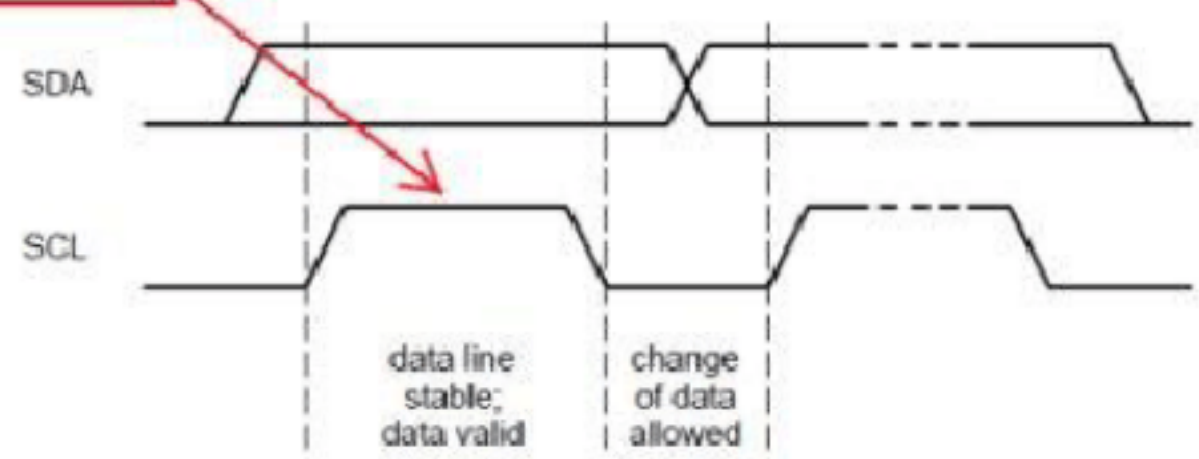
当从机不能响应从机地址时 (例如它正忙于其他事而无法响应 IIC 总线的操作，或者这个地址没有对应的从机)，在第 9 个 SCL 周期内 SDA 线没有拉低，即没有 ACK 信号。这时，主机发出一个 P 信号终止传输或者重新发出一个 S 信号开始新的传输。

如果从机接收器在传输过程中不能接收更多的数据时，它不会发出 ACK 信号。这样，主机就可以意识到这点， 从而发出一个 P 信号终止传输或者重新发出一个 S 信号开始新的传输。

主机接收器在接收到最后一个字节后，也不会发出 ACK 信号。于是，从机发送器释放 SDA 线，以允许主机发出 P 信号结束传输。

位传输：

SCL为高电平时，数据线上的数据必须保持稳定，否则表示开始或停止



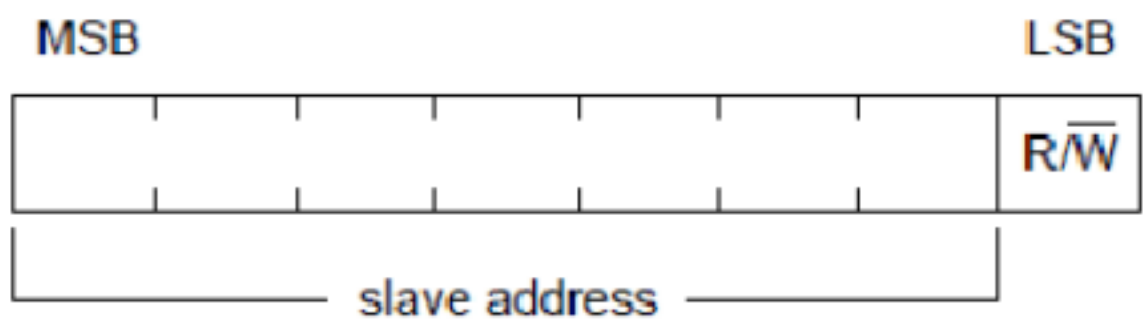
主机向从机发送一字节数据

```
void I2C_SendByte(u8 SendByte) //       数据从高位到低位
{
    u8 i=8;
    while(i--)
    {
        SCL_L;
        I2C_delay();
      if(SendByte&0x80)
        SDA_H;
      else
        SDA_L;
        SendByte<<=1;
        I2C_delay();
  SCL_H;
        I2C_delay();
    }
    SCL_L;
}
```

三、 7 位寻址

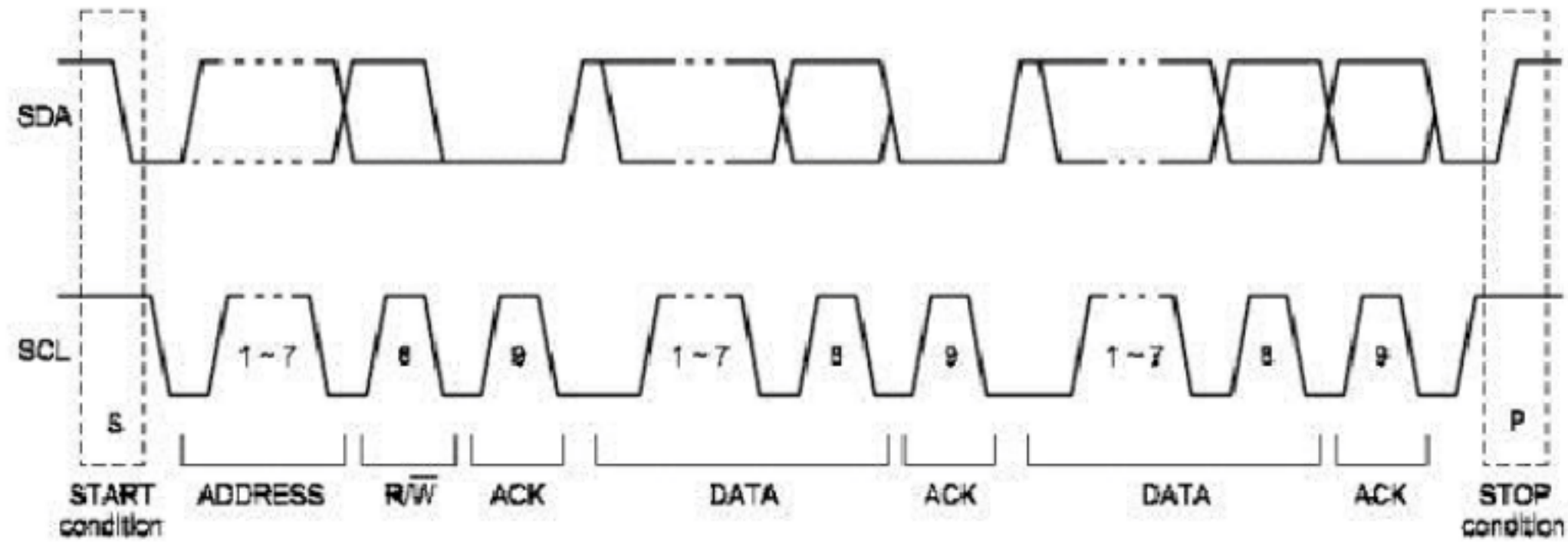
从机地址	R/$\overline{W}$ 位	描述
0000 000	0	广播呼叫地址
0000 000	1	起始字节 (1)
0000 001	X	CBUS 地址 (2)
0000 010	X	保留给不同的总线格式 (3)
0000 011	X	保留到将来使用
0000 1XX	X	Hs 模式主机码
1111 1XX	X	保留到将来使用
1111 0XX	X	10 位从机寻址

在起始条件 S 后发送了一个从机地址，这个地址共有 7 位，紧接着的第 8 位是数据方向位 R/W ，0 表示发送写、 1 表示请求数据读：



数据传输一般由主机产生的停止位 P 终止，但是如果主机仍希望在总线上通讯它可以产生重复起始条件 Sr 和寻址另一个从机，而不是首先产生一个停止条件。

完整的数据传输：



stm32 如何建立与 EEPROM 的通讯

1、 配置 I/O 端口，确定并配置 I2C 的模式，使能 GPIO 和 I2C 时钟。

2、 写：

检测 SDA 是否空闲；
->按 I2C 协议发出起始讯号；
->发出 7 位器件地址和写模式；
->要写入的存储区首地址；
->用页写入方式或字节写入方式写入数据；

2、 3、 读：
检测 SDA 是否空闲；
->按 I2C 协议发出起始讯号；
->发出 7 位器件地址和写模式（伪写）；
->发出要读取的存储区首地址；

->重发起始讯号；
->发出 7 位器件地址和读模式；
->接收数据；