1. 概述：

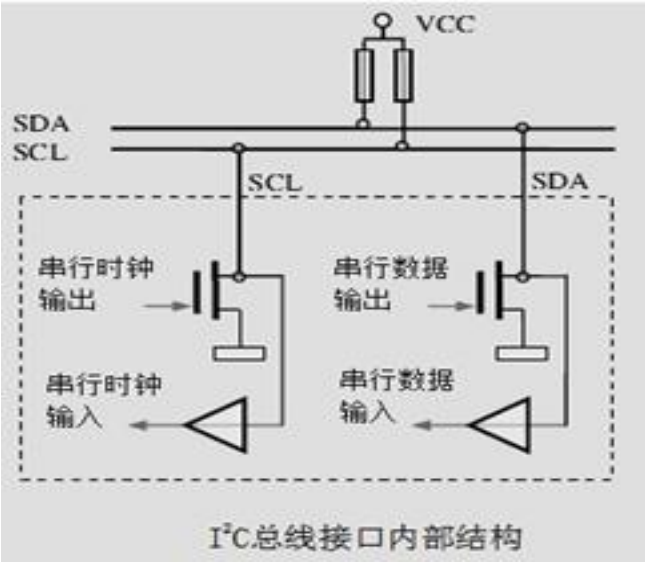
I²C 是 Inter-Integrated Circuit 的缩写，发音为"eye-squared cee" or "eye-two-cee" ，它是一种两线接口。

I²C 只是用两条双向的线，一条 Serial Data Line (SDA) ，另一条 Serial Clock (SCL)。

SCL：上升沿将数据输入到每个 EEPROM 器件中；下降沿驱动 EEPROM 器件输出数据。(边沿触发)

SDA：双向数据线，为 OD 门，与其它任意数量的 OD 与 OC 门成"线与"关系。

2. 输出级



每一个 I2C 总线器件内部的 SDA、SCL 引脚电路结构都是一样的，引脚的输出驱动与输入缓冲连在一起。其中输出为漏极开路的场效应管，输入缓冲为一只高输入阻抗的同相器，这种电路具有两个特点：

1)由于 SDA、SCL 为漏极开路结构(OD)，因此它们必须接有上拉电阻，阻值的大小常为 1k8, 4k7 and 10k ，但 1k8 时性能最好；当总线空闲时，两根线均为高电平。连到

总线上的任一器件输出的低电平，都将使总线的信号变低，即各器件的 SDA 及 SCL 都是线"与"关系。

2)引脚在输出信号的同时还将引脚上的电平进行检测，检测是否与刚才输出一致，为"时钟同步"和"总线仲裁"提供了硬件基础。

3. 主设备与从设备

系统中的所有外围器件都具有一个 7 位的"从器件专用地址码"，其中高 4 位为器件类型，由生产厂家制定，低 3 位为器件引脚定义地址，由使用者定义。主控器件通过地址码建立多机通信的机制，因此 I2C 总线省去了外围器件的片选线，这样无论总线上挂接多少个器件，其系统仍然为简约的二线结构。终端挂载在总线上，有主端和从端之分，主端必须是带有 CPU 的逻辑模块，在同一总线上同一时刻使能有一个主端，可以有多个从端，从端的数量受地址空间和总线的最大电容 400pF 的限制。

-
- 主端主要用来驱动 SCL line;
- 从设备对主设备产生响应;

二者都可以传输数据，但是从设备不能发起传输，且传输是受到主设备控制的。

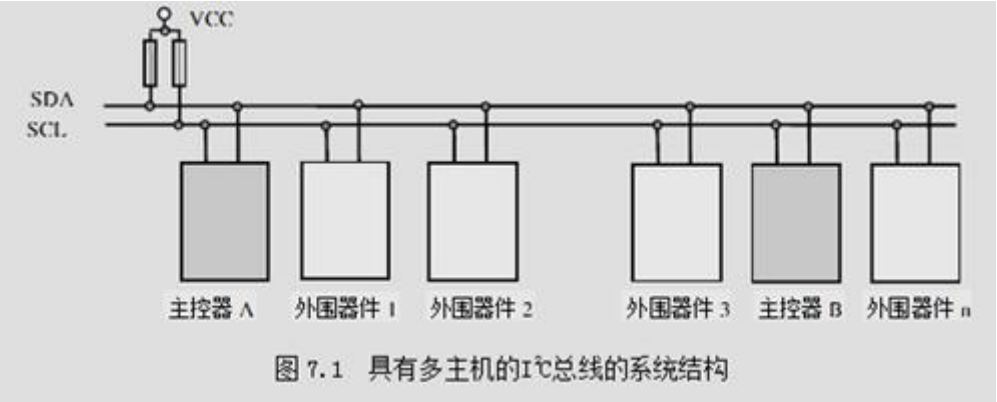


图 7.1 具有多主机的I²C总线的系统结构

4. 速率：

普通模式：100kHz；

快速模式：400kHz；

高速模式：3.4MHz；

没有任何必要使用高速 SCL，将 SCL 保持在 100k 或以下，然后忘了它吧。

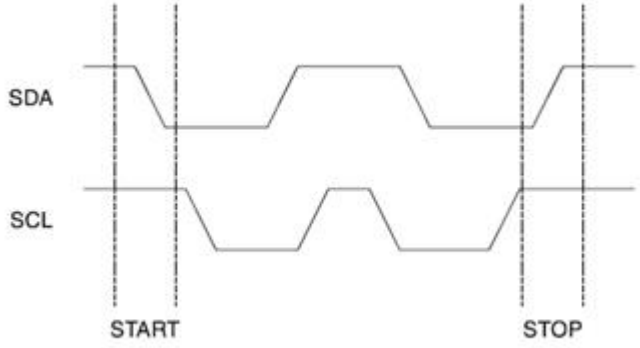
一、协议

1.空闲状态

I2C 总线总线的 SDA 和 SCL 两条信号线同时处于高电平时，规定为总线的空闲状态。此时各个器件的输出级场效应管均处在截止状态，即释放总线，由两条信号线各自的上拉电阻把电平拉高。

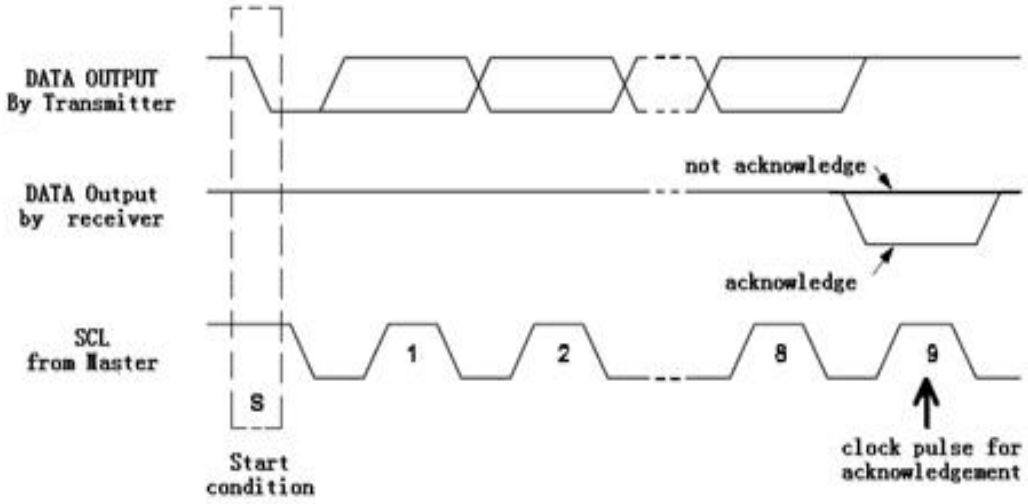
2.起始位与停止位的定义：

- 起始信号：当 SCL 为高期间，SDA 由高到低的跳变；启动信号是一种电平跳变时序信号，而不是一个电平信号。
- 停止信号：当 SCL 为高期间，SDA 由低到高的跳变；停止信号也是一种电平跳变时序信号，而不是一个电平信号。



3. ACK

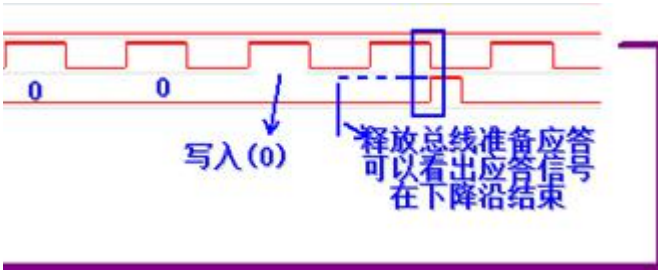
发送器每发送一个字节，就在时钟脉冲 9 期间释放数据线，由接收器反馈一个应答信号。应答信号为低电平时，规定为有效应答位（ACK 简称应答位），表示接收器已经成功地接收了该字节；应答信号为高电平时，规定为非应答位（NACK），一般表示接收器接收该字节没有成功。 对于反馈有效应答位 ACK 的要求是，接收器在第 9 个时钟脉冲之前的低电平期间将 SDA 线拉低，并且确保在该时钟的高电平期间为稳定的低电平。 如果接收器是主控器，则在它收到最后一个字节后，发送一个 NACK 信号，以通知被控发送器结束数据发送，并释放 SDA 线，以便主控接收器发送一个停止信号 P。



I²C 总线的响应

如下图逻辑分析仪的采样结果：释放总线后，如果没有应答信号，sda 应该一直持续为高电平，但是如图中蓝色虚线部分所示，它被拉低为低电平，证明收到了应答信号。

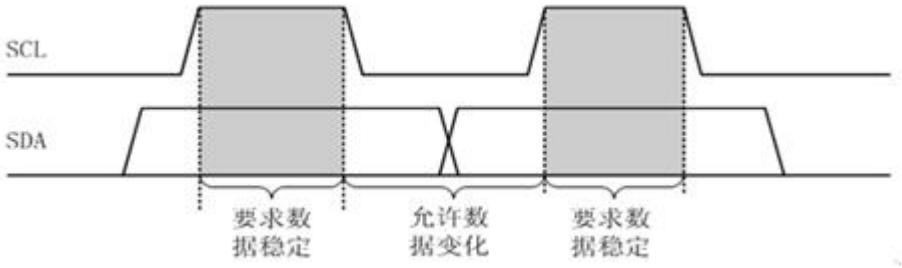
这里面给我们的两个信息是：1)接收器在 SCL 的上升沿到来之前的低电平期间拉低 SDA；2)应答信号一直保持到 SCL 的下降沿结束；正如前文红色标识所指出的那样。



4.数据的有效性：

I2C 总线进行数据传送时，时钟信号为高电平期间，数据线上的数据必须保持稳定，只有在时钟线上的信号为低电平期间，数据线上的高电平或低电平状态才允许变化。

我的理解：虽然只要求在高电平期间保持稳定，但是要有一个提前量，也就是数据在 SCL 的上升沿到来之前就需准备好，因为在前面 I2C 总线之(一)---概述一文中已经指出，数据是在 SCL 的上升沿打入到器件(EEPROM)中的。



5.数据的传送：

在 I2C 总线上传送的每一位数据都有一个时钟脉冲相对应（或同步控制），即在 SCL 串行时钟的配合下，在 SDA 上逐位地串行传送每一位数据。数据位的传输是边沿触发。

二、工作过程

总线上的所有通信都是由主控器引发的。在一次通信中，主控器与被控器总是在扮演着两种不同的角色。

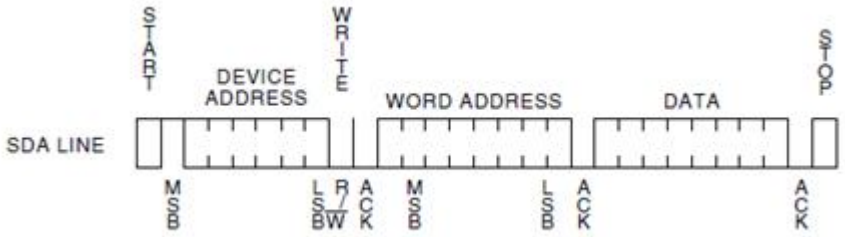
1. 主设备向从设备发送数据

主设备发送起始位，这会通知总线上的所有设备传输开始了，接下来主机发送设备地址，与这一地址匹配的 slave 将继续这一传输过程，而其它 slave 将会忽略接下来的传输并等待下一次传输的开始。主设备寻址到从设备后，发送它所要读取或写入的从设备的内部寄存器地址； 之后，发送数据。数据发送完毕后，发送停止位：

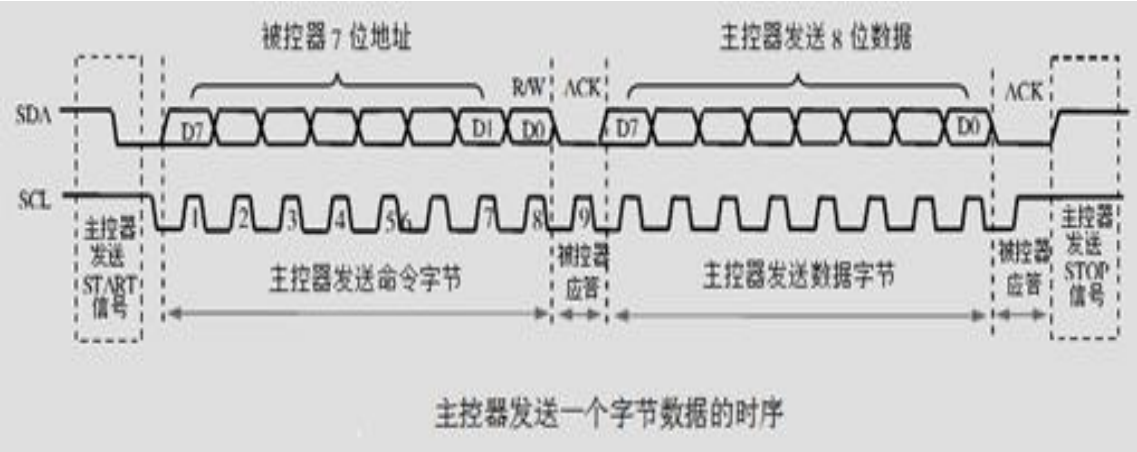
写入过程如下：

发送起始位

- 发送从设备的地址和读/写选择位；释放总线，等到 EEPROM 拉低总线进行应答；如果 EEPROM 接收成功，则进行应答；若没有握手成功或者发送的数据错误时 EEPROM 不产生应答，此时要求重发或者终止。
- 发送想要写入的内部寄存器地址；EEPROM 对其发出应答；
- 发送数据
- 发送停止位.
- EEPROM 收到停止信号后，进入到一个内部的写入周期，大概需要 10ms，此间任何操作都不会被 EEPROM 响应；（因此以这种方式的两次写入之间要插入一个延时，否则会导致失败，博主曾在这里小坑了一下）



详细：



主控器发送一个字节数据的时序

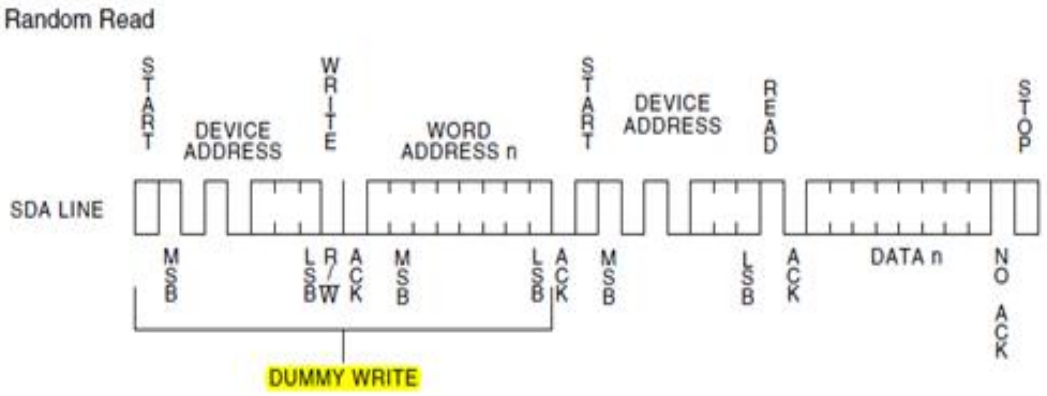
需要说明的是：①主控器通过发送地址码与对应的被控器建立了通信关系，而挂接在总线上的其它被控器虽然同时也收到了地址码，但因为与其自身的地址不相符合，因此提前退出与主控器的通信；

2. 主控器读取数据的过程：

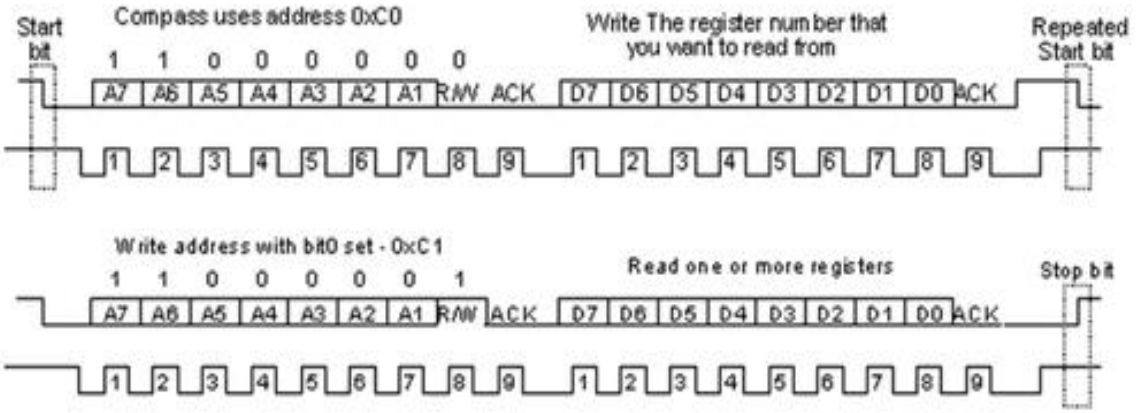
读的过程比较复杂，在从 slave 读出数据前，你必须先要告诉它哪个内部寄存器是你想要读取的，因此必须先对其进行写入(dummy write)：

- 发送起始位；
- 发送 slave 地址+write bit set；
- 发送内部寄存器地址；
- 重新发送起始位，即 restart；
- 重新发送 slave 地址+read bit set；
- 读取数据

  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。
- 发送停止位

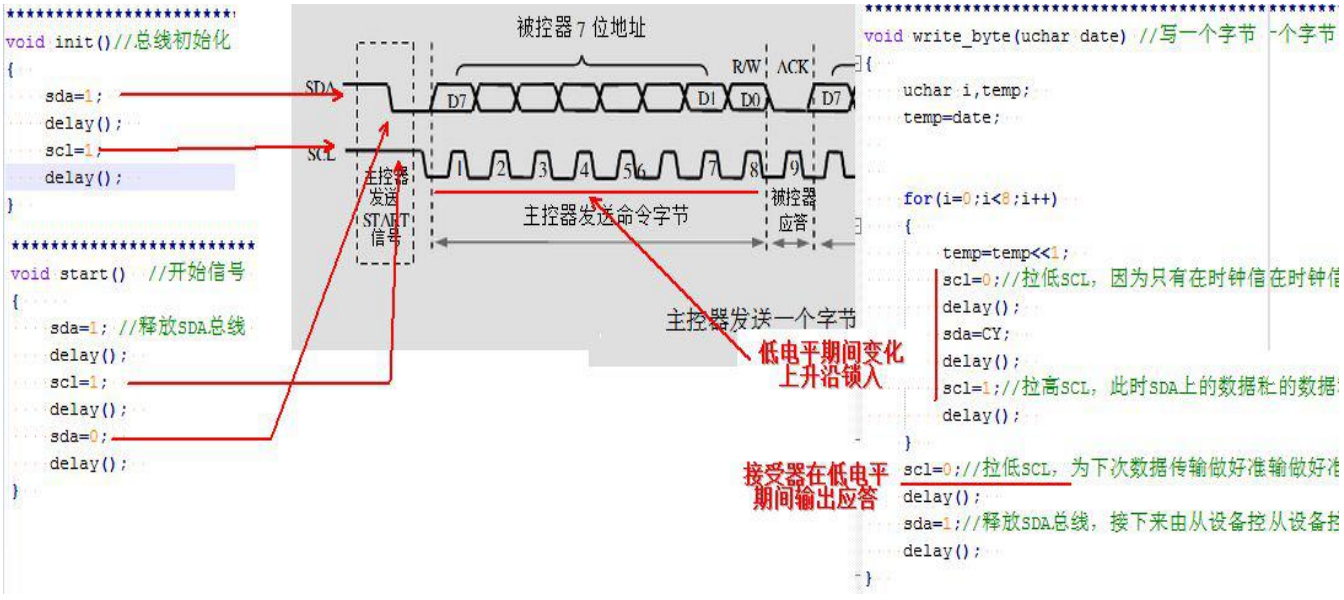


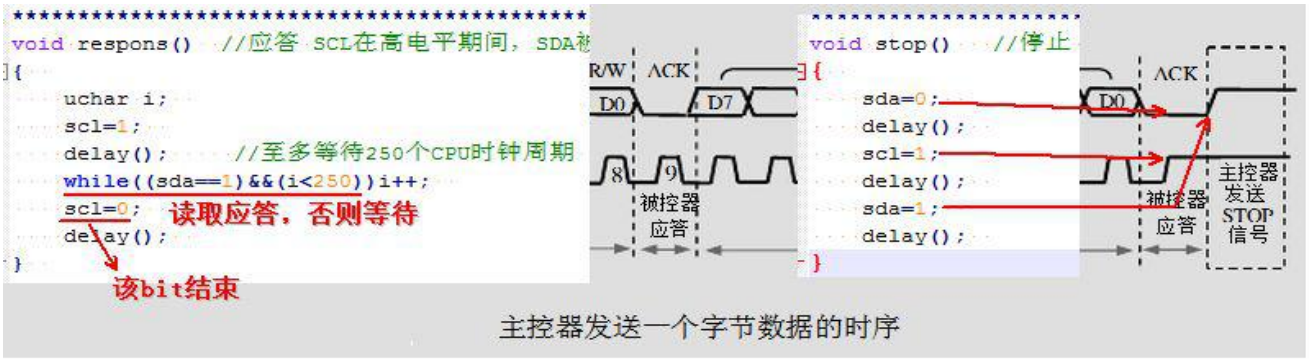
详细：



为了加深对 I2C 总线的理解，用 C 语言模拟 IIC 总线，边看源代码边读波形：

如下图所示的写操作的时序图：





主控器发送一个字节数据的时序

读时序的理解同理。对于时序不理解的朋友请参考"I2C 总线之(二)---时序"

完整的程序如下：

```
#include<reg51.h>
#define uchar unsigned char
#define uint unsigned int
#define write_ADD 0xa0
#define read_ADD 0xa1
uchar a;
sbit SDA=P2^0;
sbit SCL=P2^1;
void SomeNop();     //短延时
void init();    //初始化
void check_ACK(void);
void I2CStart(void);
void I2cStop(void);
void write_byte(uchar dat);//写字节
void delay(uint z);
uchar read_byte();     //读字节
void write(uchar addr,uchar dat);  //指定地址写
uchar read(uchar addr);       //指定地址读
bit flag;  //应答标志位
void main()
{
    init();
    write_add(5,0xaa); //向地址 5 写入 0xaa
    delay(10);      //延时,否则被坑呀！！！
     P1=read_add(5);      //读取地址 5 的值
     while(1);
}



//*****************************************************************************
void delay()//简单延时函数
{ ;; }
```

```
//*****************************************************************************
void start()  //开始信号 SCL 在高电平期间，SDA 一个下降沿则表示启动信号
{
    sda=1; //释放 SDA 总线
    delay();
    scl=1;
    delay();
    sda=0;
    delay();
}
//*****************************************************************************
void stop()   //停止 SCL 在高电平期间，SDA 一个上升沿则表示停止信号
{
    sda=0;
    delay();
    scl=1;
    delay();
    sda=1;
    delay();
}
//*****************************************************************************
void respons()  //应答 SCL 在高电平期间，SDA 被从设备拉为低电平表示应答
{
    uchar i;
    scl=1;
    delay();
    //至多等待 250 个 CPU 时钟周期
    while((sda==1)&&(i<250))i++;
    scl=0;
    delay();
}
//*****************************************************************************
void init()//总线初始化 将总线都拉高一释放总线  发送启动信号前，要先初始化总线。即总有检测到总线空闲才开始发送启动信号
{
    sda=1;
    delay();
    scl=1;
    delay();
}
//*****************************************************************************
void write_byte(uchar date) //写一个字节
{
    uchar i,temp;
    temp=date;


    for(i=0;i<8;i++)
    {
        temp=temp<<1;
```

```
        scl=0;//拉低 SCL，因为只有在时钟信号为低电平期间按数据线上的高低电平状态才允许变化；并在此时和上一个循环的 scl=1 一起形成一个上升沿
        delay();
        sda=CY;
        delay();
        scl=1;//拉高 SCL，此时 SDA 上的数据稳定
        delay();
    }
    scl=0;//拉低 SCL，为下次数据传输做好准备
    delay();
    sda=1;//释放 SDA 总线，接下来由从设备控制，比如从设备接收完数据后，在 SCL 为高时，拉低 SDA 作为应答信号
    delay();
}
//*****************************************************************************
uchar read_byte()//读一个字节
{
    uchar i,k;
    scl=0;
    delay();
    sda=1;
    delay();
    for(i=0;i<8;i++)
    {
        scl=1;//上升沿时，IIC 设备将数据放在 sda 线上，并在高电平期间数据已经稳定，可以接收啦
        delay();
        k=(k<<1)|sda;
        scl=0;//拉低 SCL，使发送端可以把数据放在 SDA 上
        delay();
    }
    return k;
}
//*****************************************************************************
void write_add(uchar address,uchar date)//任意地址写一个字节
{
    start();//启动
    write_byte(0xa0);//发送从设备地址
    respons();//等待从设备的响应
    write_byte(address);//发出芯片内地址
    respons();//等待从设备的响应
    write_byte(date);//发送数据
    respons();//等待从设备的响应
    stop();//停止
}
//*****************************************************************************
uchar read_add(uchar address)//读取一个字节
{
    uchar date;
    start();//启动
```

```
    write_byte(0xa0);//发送发送从设备地址 写操作
    respons();//等待从设备的响应
    write_byte(address);//发送芯片内地址
    respons();//等待从设备的响应
    start();//启动
    write_byte(0xa1);//发送发送从设备地址 读操作
    respons();//等待从设备的响应
    date=read_byte();//获取数据
    stop();//停止
    return date;//返回数据
}
```