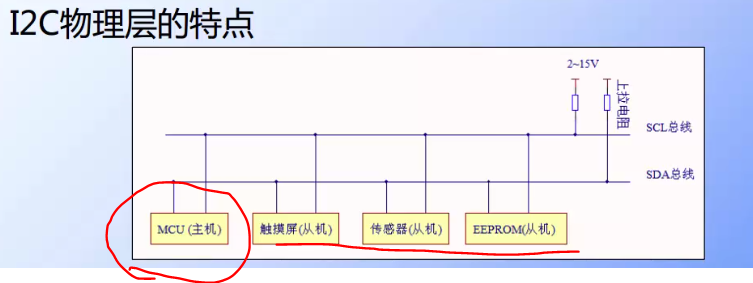
**一，I2C协议**

1. **I2C物理层和协议层**

**1.1物理层**



1. 一个主机，多个从机。SCL,SDA要有上拉电阻，同步通信(有时钟)
2. SDA：数据双方发送线(主到从，从到主)

SCL：串行时钟线

1. 根据寻址机制，选择不同的从机。即 每个从机都有**独立地址**

发送地址和发送数据没什么区别，都是通过SDA发送一串数。只不过协议层定义顺序不同(先发的数是地址后发的是数据)

1. SCL,SDA上拉电阻一般为4.7K。从机数量过多，根据协议改变。

上拉电阻实现**线与**特性：

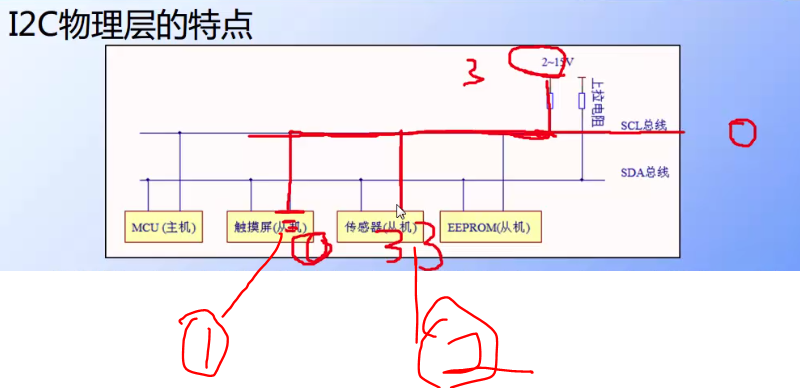
高电平 1 **高阻态**

低电平 0

高阻:输出一个非常大的电阻，相当于断开

IIC是MOS开漏输出一个高阻态相当于 从机与SDA,SCL 断开，则从机不会影响其他从机通信。

如果①不输出高阻，输出0的话，会**进行线与**将整个SCL置0.当②输出3.3v，这个电压会直接输入到①里，形成短路。



上拉电阻实现**线与**特性：防止从机冲突。

例如从机①输出0，会**进行线与**把整个SCL变为0，即牵制住SCL，当其他设备想通信时，SCL已经被线与成0，因此其他设备无法通信。因此,当所有从机是高阻态时SCL=1 SDA=1表示总线空闲可以通信，一旦哪个从机想通信就线与将SCL拉0，这样表示总线繁忙。

1. IIC的**高电平 就是输出 高阻态**

这就是SDA接上拉电阻的原因，如果从机①要输出逻辑1，当输出3.3v时，会有烧毁其他从机的风险(如上文)。不像上拉电源有上拉电阻限流，不会烧坏其他器件。

因此

从机①想**输出高电平**时，开漏输出直接输出高阻态(相当于断开)，反正SDA有上拉是高电平，因此主机读到SDA是。

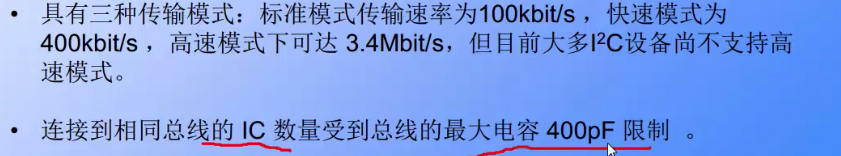
从机①想**输出低电平**时，输出GND,把整个SDA都拉低了，主机读到0。即**线与**

其实很多总线输出高电平都是高阻态，这样就不会影响其他从机



1. iic通信的速度取决于，从机的数据读取频率。所以要限定SCL的时钟频率。一班采取快速模式。

连接到总线的从机IC越多，总线的电容越大。当电容超过400pF就不行了。因此连接的IC从机有数量限制。



**1.2协议层**

IIC的协议定义了通讯的 **起始，停止，数据有效性，响应，仲裁，时钟同步，地址广播**。

**1.2.1基本读写过程**

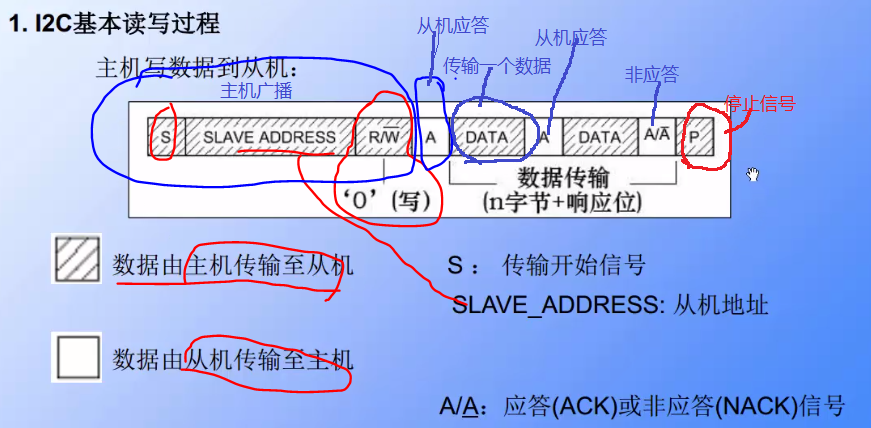
0是 主机发送到从机 **写**

1是 从机发送到主机 **读**

1. **写过程**

S/ P/ SLAVEIDress/ R/W /P /SCL信号线都只能由主机产生

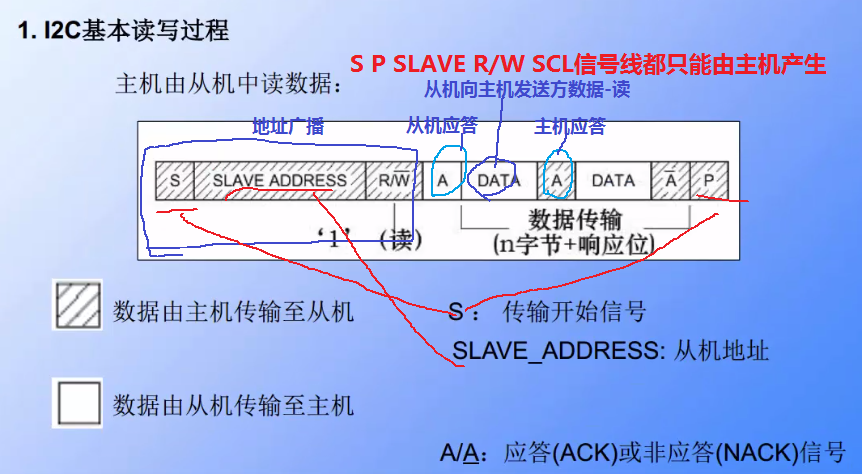
**主机开始S--写7位地址—写信号—从机应答—主机写数据—从机应答--主机写数据—从机非应答—主机结束P**

****

1. **读过程**

S/ P/ SLAVE/ R/W /P /SCL信号线都只能由主机产生

**主机开始S--写7位地址--读—从机应答—从机发送数据—主机应答--从机发送数据—主机非应答—主机结束P**



1. **实际IIC读(读从设备的寄存器值)**

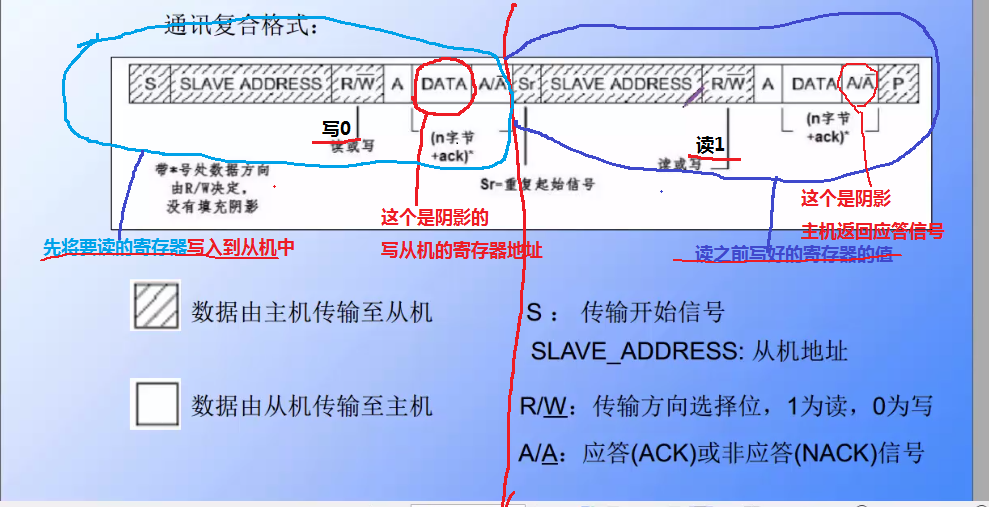
分为两大步：

**写从设备的寄存器**

主机开始S—从设备地址—写0—从设备应答—主机写从设备寄存器地址—从设备应答(注意没有停止P)

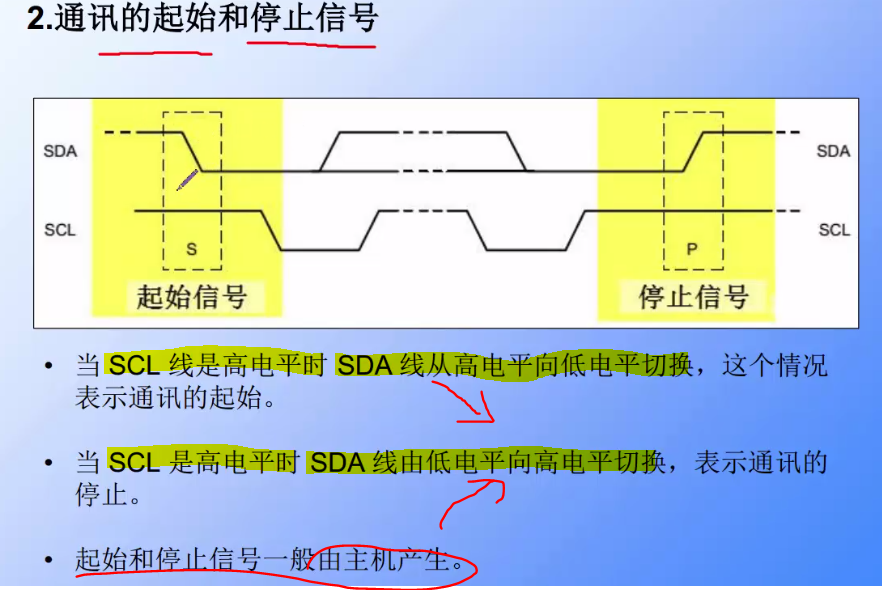
**读从设备的寄存器值**

主机重复起始Sr—从设备地址—读1—从机应答—从机寄存器返回数据(读)—主机应答—主机停止P

****

**1.2.2基本读写信号电平**

1. **起始/停止信号**

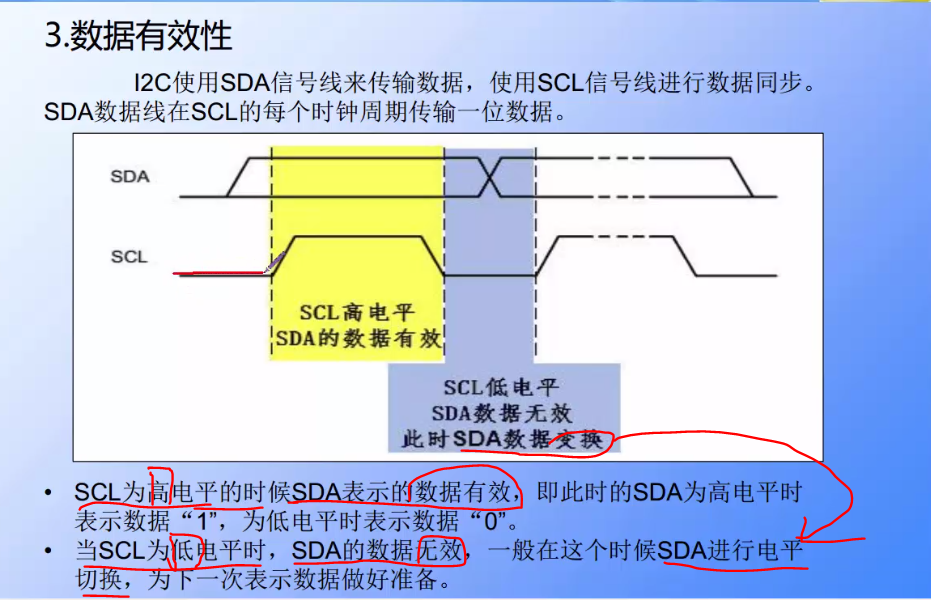


1. **数据有效性(将SCL与SDA同步)**

SCL高电平时，SDA的数据有效，从机/主机采集信号

SCL低电平时，SDA进行数据切换，此时数据无效，从机/主机不采集信号

采集8次，即8bit=1byte



**发送数据(数据先准备好)** SDA先准备好 再SCL1到0

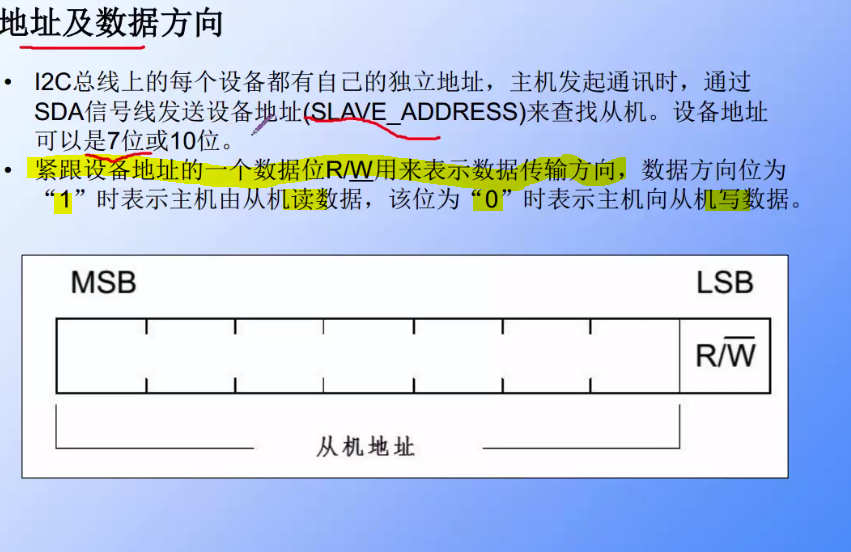
**读数据(时钟先准备好)** SCL先1到0 再读SDA的寄存器地址和数据方向

从机的7位设备地址+ 1

0x78 + 1 =8位设备的读地址=0Xf1

0x78 + 0 设备的写地址写地址=0Xf0

注意区分8位地址设备 和 7位地址设备分别指什么 (eeprom的8位地址是1010 0000 即A0)



MSB高位先行：发送一字节数据**从高位先开始**先发

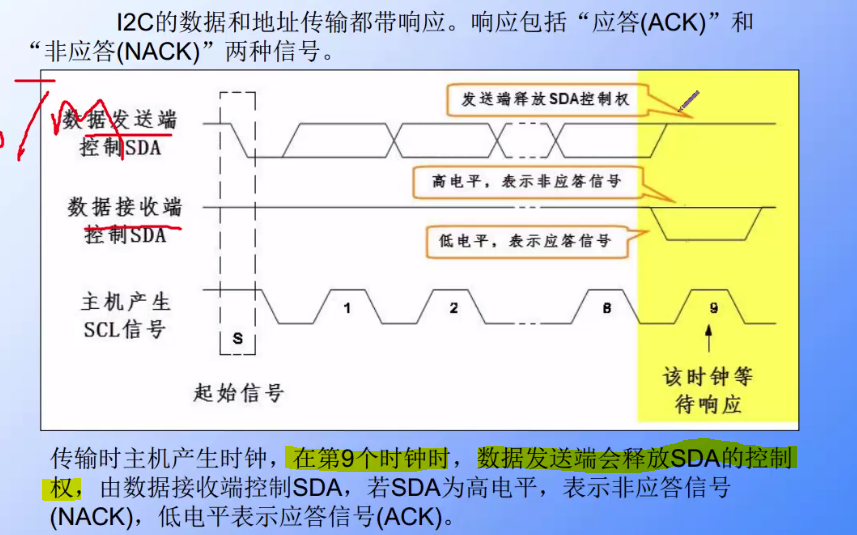
LSB地位先行：发送一字节数据从低位先开始先发

**IIC属于MSB**

1. **响应(ACK nACK)**

**释放控制权**：第9时钟**数据发送端(数据方向不同可以是从设备/主设备)**把SDA搞成高阻态(则SDA被上拉电阻拉成1)，这样SDA就空闲了。

如果**数据接收端**把SDA拉成0--ACK应答，如果**数据接收端**仍是1---nACK非应答



**软件模拟：**就主机在第九个时钟,控制SDA发送1。**数据SDA先准备好再控制时钟**

**1.3STM的IIC**

**软件模拟 硬件实现**

**1.3.1软件模拟**

通过两个GPIO模拟起始，停止，数据有效性，响应，仲裁等信号。软件模拟繁琐浪费CPU.

**1.3.2硬件实现**

直接配置好STM32的IIC,根据配置寄存器来配置IIC。则就直接连接从设备了，只用检测IIC外设状态和访问数据寄存器即可。大大减轻CPU负担。

此部分见STM32官方手册 **IIC工作过程 24.3.1**

**通讯数据发送过程24.3.3**

**寄存器24.6**

**二，EEPROM介绍**

手册在：7硬件资料---2芯片资料---AT24C02里

已经手册详细读，有疑问直接查询手册

**实际意义：**我们可以把一些数据放在EEPROM这样掉电不会丢失

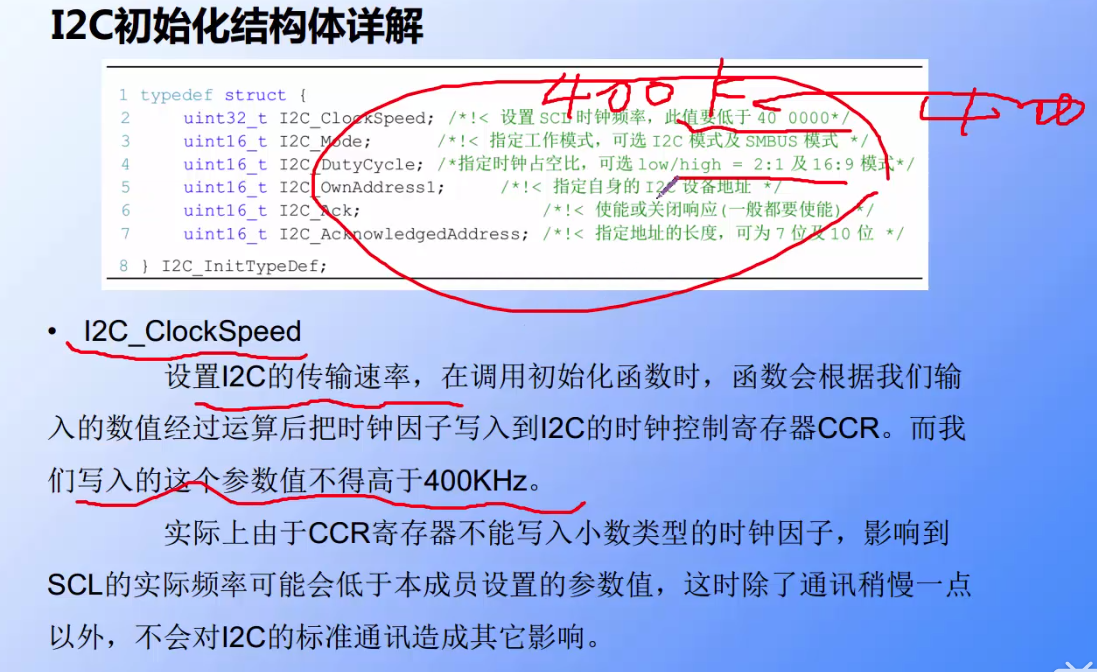
EEPROM页字节对齐表

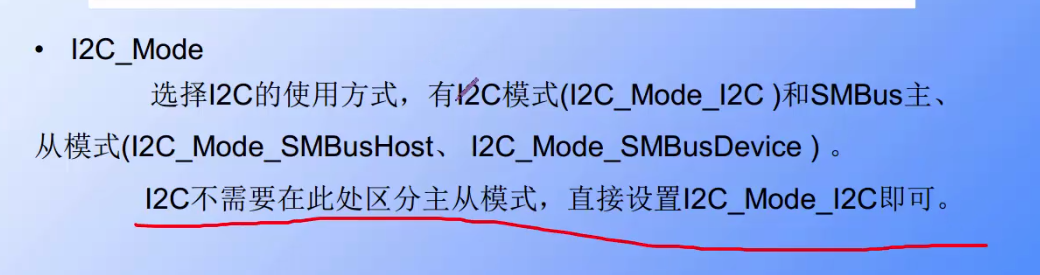


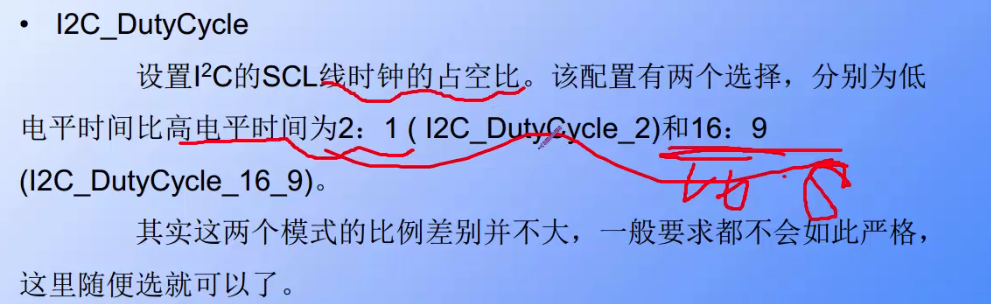
**三，编程问题**

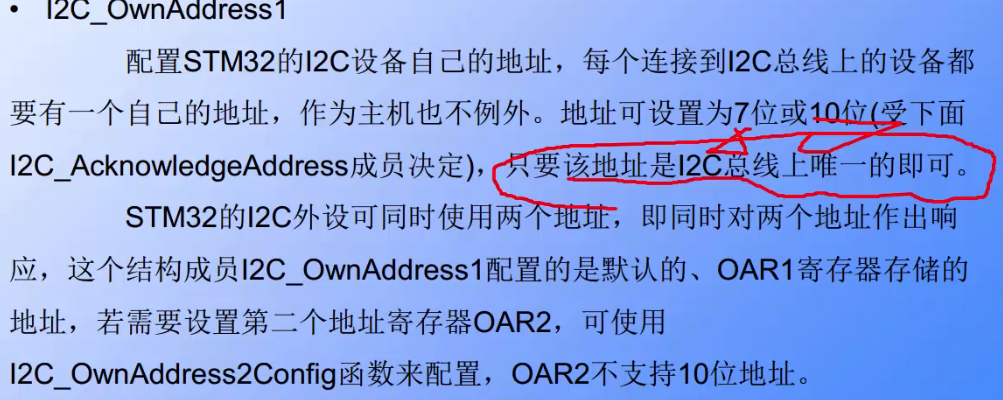
1. **硬件STM的IIC的库函数**

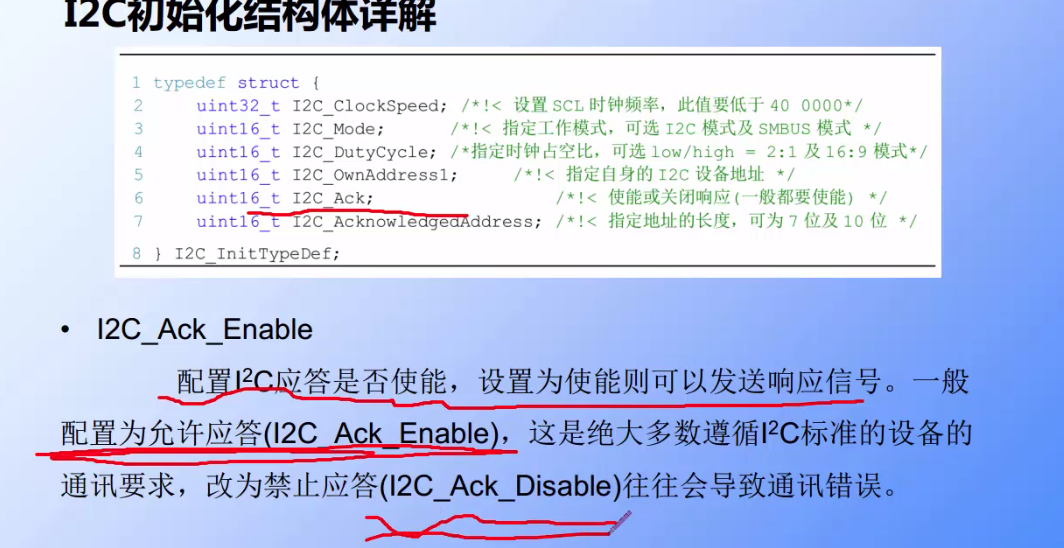
**实现效果：**向IIC写入数据，按下复位键通过串口通信把EEPROM数据读出来校验，看数据是否一致

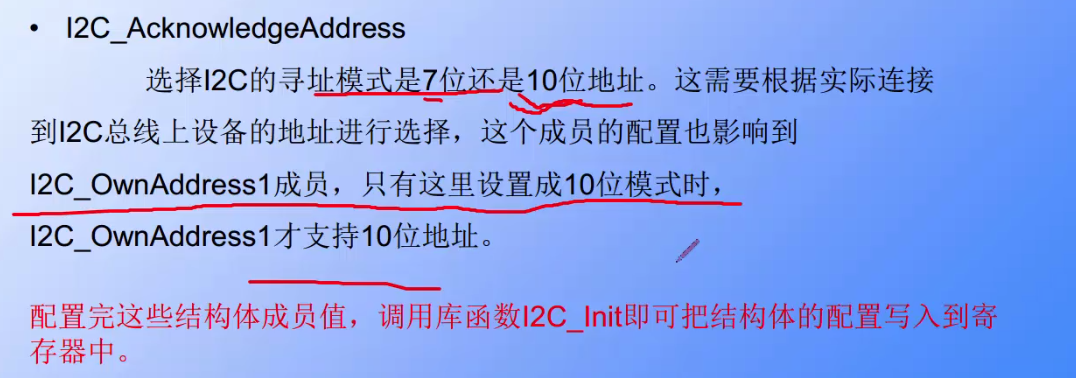












**1.1硬件编程的一些问题**

①为什么全速运行程序卡死，而调试模式的单步运行却不卡死？

时间问题：AT24C02手册第8页

*写完一个字节，如果EEPROM回应ACK=0代表这次写完了 才能执行下一次写操作*

即，写直接过程后，没有等待返回一个ACK信号，就立马继续开始读操作(读操作前几个仍是写)，然后这最开始的写地址操作就无法继续运行，就卡在这。

**解决方法**所以必须等待一段时间，等待写完最后一个字节再读。要么读写之间加延时(不建议，因为通信对时间要求非常严格)，要么加个**检测写结束函数**(推荐)

②连续写入，读出来的值不对？**数据要和地址对齐**

EEPROM连续写入多字节数，必须要和地址对齐。即写入地址addr%8=0，即写8个字节的话必须从0x00/0x08/0x16这些地址开始，不然数据会出错

1. 无优化版：简单运行硬件IIC

优化版：加了超时处理，打印错误原因，自动对齐，一次发送>8字节的函数

**2.软件模拟IIC**

用两根GPIO口模拟SDA SCL 产生起始信号，停止信号，ACK等信号。硬件IIC由于复用还有其他功能，会出现各种问题。

软件IIC没有各种检测位，只能简单的控制IO电平模拟IIC来读取高低电平来读取数据。

通过读取相应IO口的IDR寄存器来读取高电平次数

1. **发送数据(数据先准备好)**

**读数据(时钟先准备好)**

1. GPIO一般配置为“开漏输出GPIO\_Mode\_Out\_OD”

因为IIC接上拉电阻，输出高阻自动为1，如果输出1的话多个从机就会容易烧坏其他芯片。iic低电平拿事

(3) 输出和输入寄存器是分开的因此引脚就算配置成输出模式 如OD,AP

依然能读取数据，只需要读数据寄存器IDR即可.但是还是建议位带操作+宏定义把引脚配置成输入在读

1. 一共两个.c文件

第一个.c两个GPIO模拟出IIC的各种信号:起始，停止，应答等

第二个.c调用第一个.c写出一个完整的IIC读/写数据过程。

**疑问**

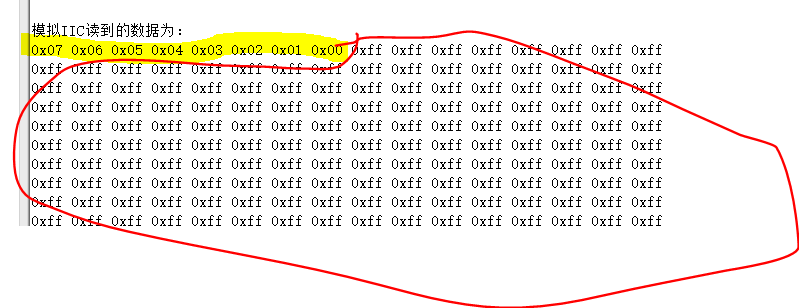
1. 循环明明是255 for(i=0; i<255:i++)理论上循环了255次 可为什么都出来256个数据

答：你读出来的最后一个数据肯定不对，把i定义成u16，循环256次才是真的对。IIC内部256个字节，想要读完必须循环256次

地址是256个：0~255 0x00~0Xff

数据存储是256个：循环256次

1. 数据突然出现这样：前8个1byte的数据正常后面的有问题，



问题在EEpromI2C.c-------ee\_WriteBytes()函数---//4.从设备有应答发送地址IIC\_SendByte(SaveAddr); 应该是IIC\_SendByte(addr); 写成SaveAddr**则关于addr的判断就无效了**。**那么IIC每次重新开始都从首地址**SaveAddr所以必然会出错，所以前8个数会有规律。

3.软件IIC加了SDA\_IN() SDA\_OUT()为什么会出错？

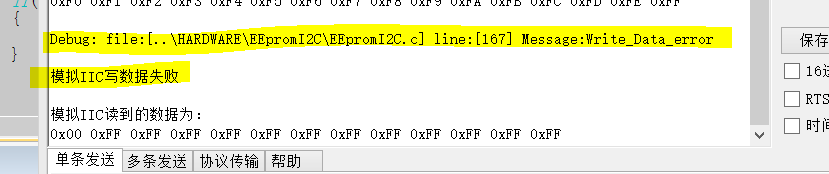
①初步通过debug的值#define SDA\_IN()的配置不起作用，CRL寄存器最高位一直为0

解答：左移是<< 我写成了< 肯定出错了导致寄存器配置不成功



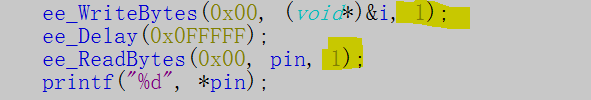
3.1更正后上位机读的数据乱码，为何？

单步运行显示写数据失败，明天再看



把 IIC\_ReadByte()，IIC\_WriteByte()，IIC\_WaitAck()的SDA\_IN SDA\_OUT注释了就不会出问题了。为什么呢？

IIC\_ReadByte()，IIC\_WriteByte()连续读/写数据，每次都得调用IIC\_ReadByte/IIC\_WriteByte 每调用一次就得配置一次SDA\_IN()，时间配合不上就卡死了。可是只调用写一个字节就不会出问题了，如下图：

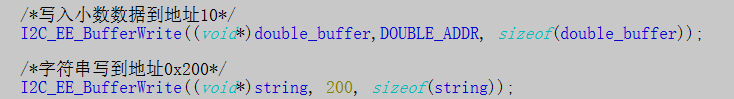


只写1次，因此只调用IIC\_ReadByte()1次，同理SDA\_IN()也配置一次，所以数据正常

**变些花样**

1. 在加上按键 按下按键KEY1写值,KEY0读值
2. EEPROM存储**小数**和**字符串**

写字符串和小数在表达上都一样 都是调用函数，如下：



区别在于，写整数一个int：4byte

double：8byte

char：1byte

这就牵扯到EEPROM的存储问题了，到底写几个数的问题。用sizeof(数组名)来计算写的IIC写的数的个数。这样就不会存在没写完或覆盖等问题。

比如写浮点数float 8byte，他就需要8个地址线才能写下。所以我们在调用函数时，必须给地址前加上(void\*)这样编译就不会出错。因为

* 1. 指针有两个属性：指向变量/对象的地址和长度，但是指针只存储地址,长度则取决于指针的类型；
  2. （void\*）xxx 表明xxx是个地址，但并不知其长度。具体的长度由函数决定。void类型指针可以指向任意类型的数据，即可用任意数据类型的指针对void指针赋值。
  3. 所以调用Write和Read函数时 void\* doble\_buffer **把float的8字节分割成1字节1字节，分别存储在EEEPROM。**读的时候也把这**8字节一字节一字节读出来**，然后再存储到double的数组里。

空指针传给u8指针的函数，就会被按照8bit来分割开

将读出来的数据按照响应的 %d %f %s 打印即可。要搞懂**浮点数在内存中是如何存储的**