# IIC通信协议1（字节）

（1）概述

I2C(Inter-Integrated Circuit BUS) 集成电路总线，该总线由NXP（原PHILIPS）公司设计，多用于主控制器和从器件间的主从通信，在小数据量场合使用，传输距离短，任意时刻只能有一个主机等特性。

经常ＩＩＣ和ＳＰＩ接口被认为指定是一种硬件设备，但其实这样的说法是不尽准确的，严格的说**他们都是人们所定义的软硬结合体**，分为**物理层**（四线结构）和**协议层**（主机，从机，时钟极性，时钟相位）。

ＩＩＣ，ＳＰＩ的区别不仅在与物理层，ＩＩＣ比ＳＰＩ有着一套更为复杂的协议层定义。下面来分别说明一下ＩＩＣ的物理层和协议层。

（２）ＩＩＣ的物理层

a．只要求**两条总线线路**，一条是串行数据线ＳＤＡ，一条是串行时钟线ＳＣＬ。（IIC是半双工，而不是全双工）。

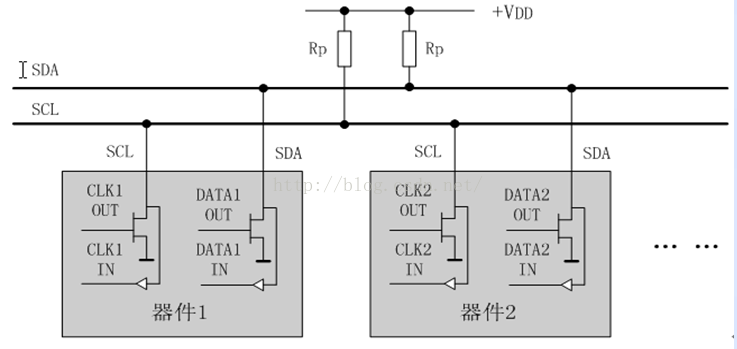
b.每个连接到总线的器件都可以通过**唯一的地址**和其它器件通信，主机/从机角色和地址可配置，主机可以作为主机发送器和主机接收器。

c.IIC是真正的**多主机总线**，（而这个SPI在每次通信前都需要把主机定死，而IIC可以在通讯过程中，改变主机），如果两个或更多的主机同时请求总线，可以通过冲突检测和仲裁防止总线数据被破坏。

d.传输速率在标准模式下可以达到100kb/s,快速模式下可以达到400kb/s。

e.连接到总线的IC数量只是受到总线的最大负载电容400pf限制。

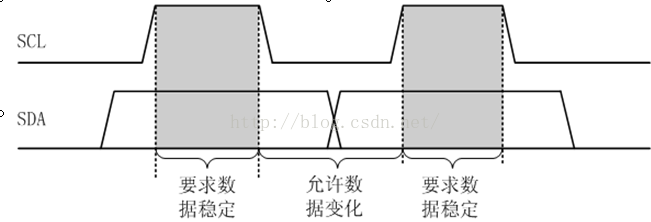
一个典型的IIC接口如下图所示



（3）IIC的协议层

IIC的协议层才是掌握IIC的关键。现在简单概括如下：

a.数据的有效性：在**时钟**的**高电平**周期内，**SDA**线上的数据必须保持**稳定**，数据线仅可以在**时钟SCL为低电平时改变**。如图所示

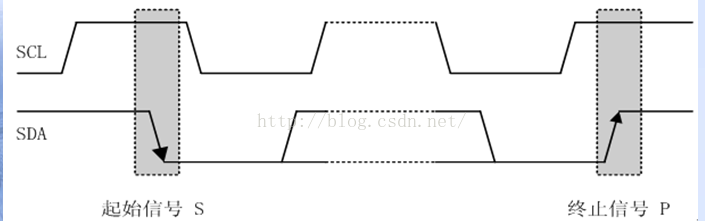


b.起始和结束条件

起始条件：当**SCL为高电平**的时候，**SDA**线上由高到低的跳变**（下降沿）**被定义为起始条件。

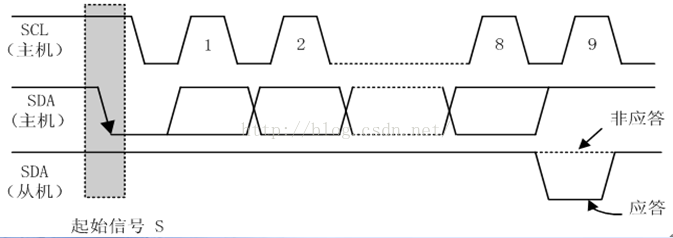
结束条件：当**SCL为高电平**的时候，**SDA**线上由低到高的跳变**（上升沿）**被定义为停止条件。

要注意起始和终止信号都是由主机发出的，连接到I2C总线上的器件，若具有I2C总线的硬件接口，则很容易检测到起始和终止信号。总线在起始条件之后，视为忙状态，在停止条件之后被视为空闲状态，对起始条件和结束条件的描述如下图所示。



c.应答

**每当主机向从机发送完一个字节的数据（8位）**，主机总是需要等待从机给出**一个应答信号**，以确认从机是否成功接收到了数据，**从机应答主机所需要的时钟仍是主机提供的**，应答出现在每一次主机完成8个数据位传输后紧跟着的时钟周期，**低电平0表示应答，1表示非应答**，如图所示。



d.数据帧格式

I2C总线上传送的**数据信号是广义的**，**既包括地址信号，又包括真正的数据信号**。

在起始信号后必须传送一个**从机的地址（7位）**，第8位是数据的**传送方向位（R/T）**，用“0”表示主机发送数据（T），“1”表示主机接收数据（R）。{这里小编在驱动MPU6050模块的时候，就犯过这样的错误，它写的MPU6050从机地址是0x68,因为发送从机地址的时候，要加一位读写方向位，因为刚开始应该是向这个MPU6050里写从机里某个寄存器的地址，所以应该是7位地址   0x68(1101000)+二进制位0=11010000）也就是0xD0,表示要向该IIC设备里写东西，然后再紧接着写入IIC设备里的寄存器地址，而我直接写入了0x68,导致出错}，每次数据传送总是由主机产生的终止信号结束。但是，若主机希望继续占用总线进行新的数据传送，则可以不产生终止信号，马上再次发出起始信号对另一从机进行寻址。

在总线的一次数据传输过程中，可以有以下几种组合方式：

[1] 主机向从机发送数据，**数据传送方向在整个传送过程中不变**：

https://img-blog.csdn.net/20150908151520584?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

注：有阴影部分表示数据由主机向从机传送，无阴影部分则表示 数据由从机向主机传送。

A 表示应答(低电平)， A 非表示非应答（高电平）。 S 表示起始信号， P 表示终止信号。

[2]主机在第一个字节后，**立即从从机读数据**:

https://img-blog.csdn.net/20150908151718332?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

[3]在传送过程中，当需要**改变传送方向**时，起始信号和从机地址都被重复产生一次，但两次**读/写方向位正好反相**：



一般情况下，[3]是比较常见的，比如MPU6050模块，

发送起始信号

等待从机应答

写一个从机地址+0(表示写)，

等待从机应答

发送一个字节的MPU6050加速度存储寄存器地址，

等待从机应答

再发送一次起始信号

等待从机应答

写一个从机地址+1（表示读）

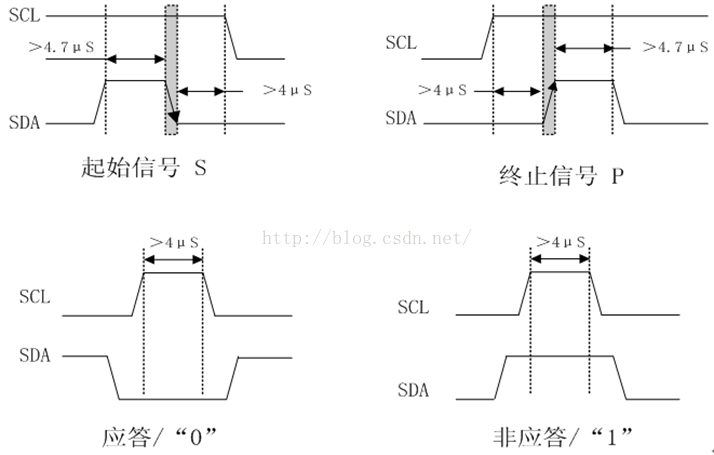
等待从机应答

读取MPU6050传感器数据

主机非应答

e.IIC信号的模拟

主机可以采用不带I2C总线接口的单片机，如80C51、AT89C2051等单片机，利用软件实现I2C总线的数据传送，即**软件与硬件结合的信号模拟**。**即使是含有IIC硬件的单片机（如stm32 103系列）也有一定的缺陷，所以一般也会模拟IIC的时序**。现将具体时间截图如下：



具体的程序代码如下：

//产生起始信号

void I2C\_Start(void)

{

I2C\_SDA\_OUT();//配置一下引脚,引脚设置为输出

I2C\_SDA\_H;//把数据线拉高

I2C\_SCL\_H;//把时钟线拉高

delay\_us(5);//延时5微秒,要求大于4.7微秒

I2C\_SDA\_L; //拉低，产生下降沿

delay\_us(6);//这个过程大于4微秒

I2C\_SCL\_L;//最后一定要把这个时钟线拉低，因为只有时钟线拉低的时候才允许数据变化。

}

//产生停止信号

void I2C\_Stop(void)

{

I2C\_SDA\_OUT();

I2C\_SCL\_L;

I2C\_SDA\_L;

I2C\_SCL\_H;

delay\_us(6);

I2C\_SDA\_H;

delay\_us(6);

}

//主机产生应答信号ACK

void I2C\_Ack(void)

{

I2C\_SCL\_L;

I2C\_SDA\_OUT();

I2C\_SDA\_L;

delay\_us(2);

I2C\_SCL\_H;

delay\_us(5);

I2C\_SCL\_L;

}

//主机不产生应答信号NACK

void I2C\_NAck(void)

{

I2C\_SCL\_L;

I2C\_SDA\_OUT();

I2C\_SDA\_H;

delay\_us(2);

I2C\_SCL\_H;

delay\_us(5);

I2C\_SCL\_L;

}

//等待从机应答信号，用户只负责主机应答信号的产生，不控制从机应答信号。

//返回值：1 接收应答失败，0 接收应答成功

u8 I2C\_Wait\_Ack(void)

{

u8 tempTime=0;

I2C\_SDA\_IN(); //配置为上拉输入。

I2C\_SDA\_H; //主机释放数据总线，等待从机产生应答信号

delay\_us(1);

I2C\_SCL\_H;

delay\_us(1);

//等待从机对数据总线的操作。低电平代表应答

while(GPIO\_ReadInputDataBit(GPIO\_I2C,I2C\_SDA))

{

tempTime++;

//这个属于软件延时，不一定准确。

if(tempTime>250) //如果时间超时，没有应答就停止。

{

I2C\_Stop();

return 1;  //没有响应的话返回1.

}

}

I2C\_SCL\_L;

return 0; //如果有响应的话就返回0.

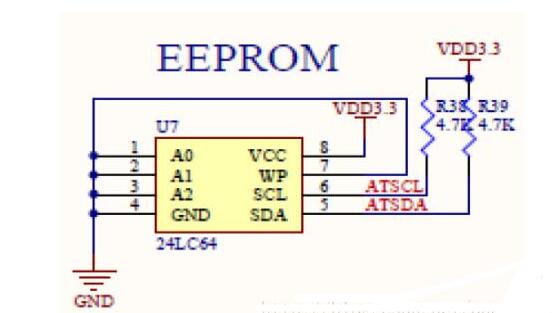
}

# IIC通信协议2（连串）

IIC协议是二线制，信号线包含SDA和SCL，且信号线是双向的，开路结构，需要通过上拉电阻到VCC，具体的电阻值影响的是信号反应速度和驱动能力。

首先，IIC通信与UART，还有SPI统称为串行接口通信，不过它们之间还是有区别的，如UART的负电平逻辑，还有UART通信不需要时钟，只需要特定的波特率即可，SPI与IIC都可以有一个主机，多个从机的情况，不过IIC适用于短距离传输，如片间通信，摄像头的配置等场景。

要搞定IIC首先来看IIC的硬件接口：

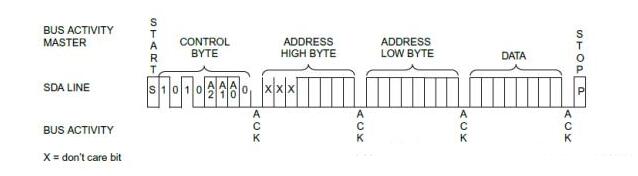


如图所示，我们知道IIC一个主机可以悬挂多个从机，所以**地址线A2，A1，A0** 可以实行片选的功能，那么WP这个引脚的功能就是当**WP悬空或者接地的时候，表示这时的EEPROM既可以读，也可以写，当WP接电源时，则只可以读而不能写**。

**SCL与SDL这两个引脚，必须上拉**，否则驱动能力不够，无法进行正常的IIC通信。

OK，硬件接口已经介绍清楚了，那么我们现在开始来看协议了。

首先IIC分为**字节读写和页面读写**，首先来看字节读写的协议：



如上图所示，如果我们要向EEPROM中写入一个字节的数据，得有如下几个步骤：

1.开始信号——在SCLK的高电平器件，拉低SDA的信号（由1 变为0）。

2.控制字节——即器件地址，就是你操作那一块EEPROM。

3.ACK信号——由从机发出，主机为接收，所以在此阶段，sda\_link必须置为0，即为读取这个应答信号，所以在SCLK的高点平期间。

4.字节地址——即某一块EEPROM里面的哪一个地址。

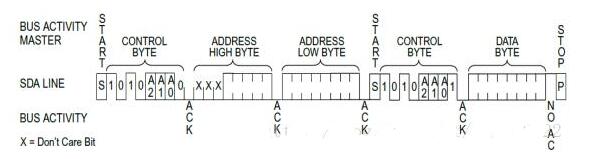
5.ACK信号——与上述相同。

6.数据信号——即你往某个地址里面写入的8位数据。

7.ACK信号——上述相同。

8.结束信号——在SCLK的高电平期间，拉高SDA信号，表示通信结束。

再来看读的时序：



由上图可看出读时序的前面处理方式与写相同，不同的时在第三个ACK信号来了之后，如果是读，那么会又有一个起始信号，紧接着读器件地址，然后应答，再然后读数据，再然后在SCLK的低电平期间发送一个NO ACK信号，要记住这个信号由主机发出，然后紧接着一个结束信号。

由上述读写时序我们可知，通信的起始均在SCLK的高电平期间发生跳变，这就据定了我们其他信号跳变均在SCLK的下降沿，SCLK高电平期间数据稳定，适用于读（即低电平改变数据，高电平采集数据）。

具体过程如下：

首先板子上电来个初始化需要来个延时，具体多少用计数器自己搞定。

# IIC通信协议3（进阶）

I2C（Inter-integrated Circuit集成电路总线）总线支持设备之间的短距离通信，用于处理器和一些外围设备之间的接口，它只需要两根信号线来完成信息交换。I²C的一个特殊优势是[微控制器](https://en.wikipedia.org/wiki/Microcontroller)只需两个[通用I / O](https://en.wikipedia.org/wiki/General-purpose_I/O)引脚和软件即可控制器件芯片网络。I2C最早是飞利浦在1982年开发设计并用于自己的芯片上，一开始只允许100kHz、7-bit标准地址。1992年，I2C的第一个公共规范发行，增加了400kHz的快速模式以及10-bit扩展地址。

在I2C的基础上，1995年Intel提出了“System Management Bus” (SMBus)，用于低速设备通信，SMBus 把时钟频率限制在10kHz~100kHz，但I2C可以支持0kHz~5MHz的设备：

**普通模式**（100kHz即100kbps）、

**快速模式(Fm）**（400kHz）、

**快速模式+(Fs+)**（1MHz）、

**高速模式(Hs）**（3.4MHz）、

**超高速模式(UFm)**（5MHz）。

注：基于IIC是Master与Slave模式，所以两者间的通信要保持时钟频率的一致。IIC是半双工。

2、IIC应用

I²C适用于外围设备，其简单性和低制造成本比速度更重要。I²C总线的常见应用包括：

通过小型ROM配置表描述可连接设备，以实现“ [即插即用](https://en.wikipedia.org/wiki/Plug_and_play) ”操作，例如：

[双列直插式内存模块](https://en.wikipedia.org/wiki/Dual_in-line_memory_module)（DIMM）上的[串行存在检测](https://en.wikipedia.org/wiki/Serial_Presence_Detect)（SPD）EEPROM

通过[VGA](https://en.wikipedia.org/wiki/VGA" \t "_blank)，[DVI](https://en.wikipedia.org/wiki/DVI" \t "_blank)和[HDMI](https://en.wikipedia.org/wiki/HDMI" \t "_blank)连接器为显示器提供[扩展显示识别数据](https://en.wikipedia.org/wiki/Extended_Display_Identification_Data" \t "_blank)（EDID）。

通过[SMBus](https://en.wikipedia.org/wiki/SMBus)对PC系统进行系统管理;

SMBus引脚分配在[常规PCI](https://en.wikipedia.org/wiki/Conventional_PCI" \t "_blank)和[PCI Express](https://en.wikipedia.org/wiki/PCI_Express" \t "_blank)连接器中。

访问保持用户设置[的实时时钟](https://en.wikipedia.org/wiki/Real-time_clock)和[NVRAM](https://en.wikipedia.org/wiki/NVRAM)芯片。

访问低速[DAC](https://en.wikipedia.org/wiki/Digital-to-analog_converter" \t "_blank)和[ADC](https://en.wikipedia.org/wiki/Analog-to-digital_converter" \t "_blank)。

更改显示器中的对比度，色调和色彩平衡设置（通过[显示数据通道](https://en.wikipedia.org/wiki/Display_Data_Channel)）。

改变智能扬声器的音量。

控制小型（例如[功能手机](https://en.wikipedia.org/wiki/Feature_phone" \t "_blank)）[OLED](https://en.wikipedia.org/wiki/Organic_light-emitting_diode)或[LCD](https://en.wikipedia.org/wiki/Liquid_crystal_display" \t "_blank)显示器。

读取硬件监视器和诊断传感器，例如风扇的速度。

打开和关闭系统组件的电源

3、IIC协议

I2C协议把传输的消息分为两种类型的帧：

**地址帧** —— 用于master指明消息发往哪个slave；

**数据帧(单个或者连续）** —— 由master发往slave的数据（或由slave发往master），每一帧是8-bit的数据。

通常我们所说的IIC读写是相对于Master来说的。

SCL变为低电平后，数据置于SDA线上，并在SCL线变为高电平后进行采样。时钟边沿和数据读/写之间的时间由总线上的器件定义，并且在芯片与芯片之间会有所不同。

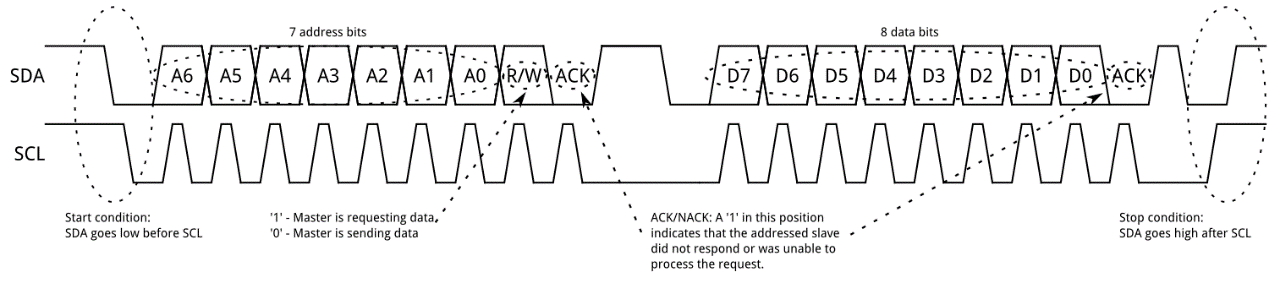
下图描述的是一个IIC完整时序图，从左往右依次看，大致总结为两类：

**IIC写寄存器的标准流程**：

1. Master发起START
2. Master发送I2C addr（7bit）和w操作0（1bit），等待ACK
3. Slave发送ACK
4. Master发送reg addr（8bit），等待ACK
5. Slave发送ACK
6. Master发送data（8bit），即要写入寄存器中的数据，等待ACK
7. Slave发送ACK第6步和第7步可以重复多次，即顺序写多个寄存器
8. Master发起STOP

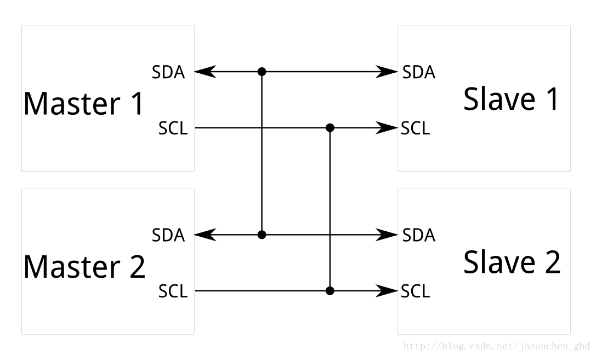
**IIC读寄存器流程：**

1. Master发送I2Caddr（7bit）和 W操作1（1bit），等待ACK
2. Slave发送ACK
3. Master发送reg addr（8bit），等待ACK
4. Slave发送ACK
5. Master发起START
6. Master发送I2C addr（7bit）和 R操作1（1bit），等待ACK
7. Slave发送ACK
8. Slave发送data（8bit），即寄存器里的值
9. Master发送ACK
10. 第8步和第9步可以重复多次，即顺序读多个寄存器



3.1 开始条件（Start Condition）

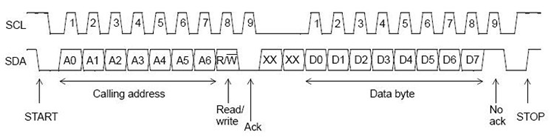
要启动地址帧，Master将SCL保持为高电平并将SDA拉低。也就是说，开始条件表现为：**当SCL为高电平时，SDA线上的高电平到低电平的跳变即定义了START条件**。这使得所有Slave都注意到传输即将开始。如果两个Master希望一次获得总线的所有权，则无论哪个设备将SDA拉低，第一个将SDA拉低的将获得对总线的控制权。Master可以发出重复启动，启动新的通信序列而不放弃对其他Master的控制; 我们稍后再谈。



3.2 地址帧(Address Frame)

在任何新的通信序列中，地址帧始终是第一个。对于7位地址，地址首先输出最高有效位（MSB），然后是R / W位，指示这是读（1）还是写（0）操作。

帧的第9位是NACK / ACK位。所有帧（数据或地址）都是这种情况。一旦发送帧的前8位，接收设备就可以控制SDA。如果接收设备在第9个时钟脉冲之前没有将SDA线拉低，则可以推断出接收设备要么没有接收到数据，要么不知道如何解析消息。在这种情况下，则由master来决定如何处理（stop或repeated start condition），也就是代码中等待超时需要做什么处理。



3.3 数据帧(Data Frame)

在发送地址帧之后，可以开始传输数据。Master将以规则的间隔继续生成时钟脉冲，数据将由Master或Slave置于SDA上，具体取决于R / W位是否指示读或写操作。数据帧的数量是任意的，并且大多数从器件将自动递增内部寄存器，这意味着后续读取或写入将来自下一个寄存器。

3.4 停止条件(Stop Condition)

一旦发送了所有数据帧，主设备将生成停止条件。停止条件由SCL上0-> 1转换后 SDA上的0-> 1（低到高）转换定义，SCL保持高电平。在正常的数据写操作时，SDA上的值应该不会当SCL为高电平改变，以避免错误的停止条件。图如3.1小节所示。

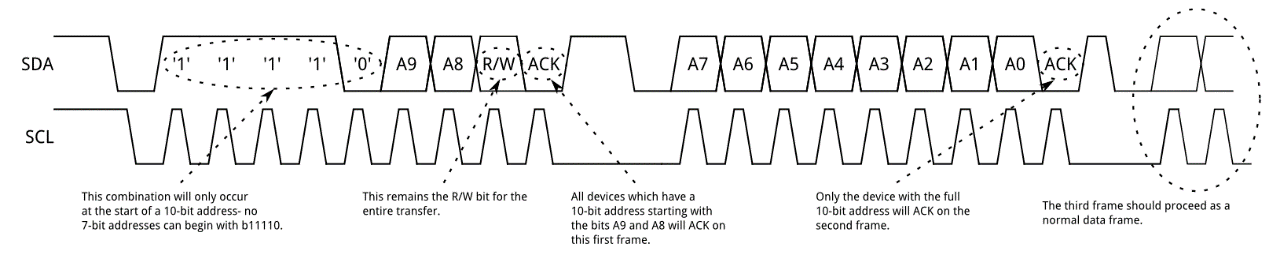
4、IIC协议的高级特性

4.1 10-bit地址

在10-bit地址的I2C系统中，需要两个帧来传输slave的地址。第一个帧的前5个bit固定为b11110，后接slave地址的高2位，第8位仍然是R/W位，接着是一个ACK位，由于系统中可能有多个10-bit slave设备地址的高2bit相同，因此这个ACK可能由多有slave设备设置。第二个帧紧接着第一帧发送，包含slave地址的低8位（7:0），接着该地址的slave回复一个ACK（或NACK）。

注意：10-bit地址的设备和7-bit地址的设备在一个系统中是可以并存的，因为7-bit地址的高5位不可能是b11110。实际上对于7-bit的从设备地址，合法范围为b0001XXX-b1110XXX，’X’表示任意值，因此该类型地址最多有112个（其他为保留地址[1]）。

两个地址帧传输完成后，就开始数据帧的传输了，这和7-bit地址中的数据帧传输过程相同。

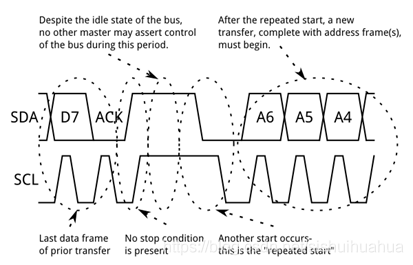


4.2 重复开始条件（repeated start condition）

有时master需要在一次通信中进行多次消息交换（例如与不同的slave传输消息，或切换读写操作），并且期间不希望被其他master干扰，这时可以使用“重复开始条件”

在一次通信中，master可以产生多次start condition，来完成多次消息交换，最后再产生一个stop condition结束整个通信过程。由于期间没有stop condition，因此master一直占用总线，其他master无法切入。

为了产生一个重复的开始条件，SDA在SCL低电平时拉高，然后SCL拉高。接着master就可以产生一个开始条件继续新的消息传输（按照正常的7-bit/10-bit地址传输时序）。重复开始条件的传输时序如下图所示：



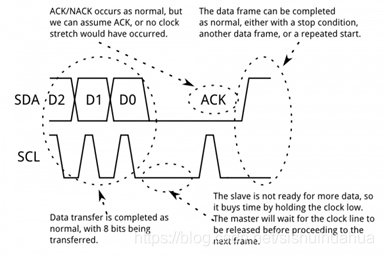
4.3 时钟拉伸

有时，低速slave可能由于上一个请求还没处理完，尚无法继续接收master的后续请求，即master的数据传输速率超过了slave的处理能力。这种情况下，slave可以进行时钟拉伸来要求master暂停传输数据。

通常时钟都是由master提供的，slave只是在SDA上放数据或读数据。而时钟拉伸则是slave在master释放SCL后，将SCL主动拉低并保持，此时要求master停止在SCL上产生脉冲以及在SDA上发送数据，直到slave释放SCL（SCL为高电平）。之后，master便可以继续正常的数据传输了。可见时钟拉伸实际上是利用了时钟同步的机制（见下文），只是时钟由slave产生。

如果系统中存在这种低速slave并且slave实现了clock stretching，则master必须实现为能够处理这种情况，实际上大部分slave设备中不包含SCL驱动器的，因此无法拉伸时钟。

所以更完整的I2C数据传输时序图为：



4.4 时钟同步和仲裁

如果两个master都想在同一条空闲总线上传输，此时必须能够使用某种机制来选择将总线控制权交给哪个master，这是通过时钟同步和仲裁来完成的，而被迫让出控制权的master则需要等待总线空闲后再继续传输。在单一master的系统上无需实现时钟同步和仲裁。

4.5时钟同步

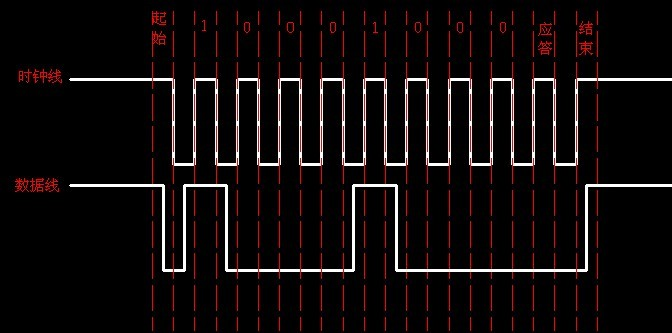
时钟同步是通过I2C接口和SCL之间的线“与”（wired-AND）来完成的，即如果有多个master同时产生时钟，那么只有所有master都发送高电平时，SCL上才表现为高电平，否则SCL都表现为低电平。

4.6总线仲裁

总线仲裁和时钟同步类似，当所有master在SDA上都写1时，SDA的数据才是1，只要有一个master写0，那此时SDA上的数据就是0。一个master每发送一个bit数据，在SCL处于高电平时，就检查看SDA的电平是否和发送的数据一致，如果不一致，这个master便知道自己输掉仲裁，然后停止向SDA写数据。也就是说，如果master一直检查到总线上数据和自己发送的数据一致，则继续传输，这样在仲裁过程中就保证了赢得仲裁的master不会丢失数据。

输掉仲裁的master在检测到自己输了之后也不再产生时钟脉冲，并且要在总线空闲时才能重新传输。

仲裁的过程可能要经过多个bit的发送和检查，实际上两个master如果发送的时序和数据完全一样，则两个master都能正常完成整个的数据传输。



# IIC通信协议4（GPIO端口模拟IIC总结）

# IIC通信协议5（内置10位地址IIC总结）