

I²C设备开发用户手册

采用I²C总线技术可以使系统的硬件设计大大简化、系统的体积减小、可靠性提高；同时，系统的更改和扩充极为容易。

I²C标准协议之中允许有双I²C主机，但大多数应用并没有使用。为了简单起见，我们认为一个I²C总线之中只有一个I²C主机的方式来使用I²C接口。以下函数是基于这样的应用而编写的。

I²C通讯由用户主机即用户单片机系统发起，分为开始、写字节或读字节及停止等几种状态，使用普通的IO口即可以有效地模拟出I²C总线的时序，获得较好的通讯性能。我们设计了4个函数，分别对应I²C的开始、写字节、读字节及停止等4个状态，这4个函数已经过多种不同类型的单片机系统应用，驱动了无数种不同类型的I²C设备，可靠性、稳定性及兼容性都较高，用户可以简单地将其移植到自己的单片机系统之中。

- **void MI2C-Bus-Start();**

I²C总线开始，初始化总线及产生I²C的开始信号。

- **BOOL MI2C-Bus-Write (BYTE gLocal-1);**

I²C总线写字节。将传入的8位参数输出到I²C设备，而且接收ACK信号，调用时读取返回值即可以了解I²C设备应答的是ACK或者NAK了。一般如果返回NAK，即表示对I²C设备写入失败，应该重试或作相应的提示。

- **BYTE MI2C-Bus-Read (BOOL FLocal-NAK);**

I²C读字节从I²C设备读回一个字节，而且输出ACK/NAK应答信号。参数FLocal-NAK只有最后一个字节的读取才需要设置为1，即NAK读取；其余字节都为0。I²C字节读函数需要特别注意这个应答位，因为用错应答位将不能传输有效的停止位，导致整个传输失败。

- **void MI2C-Bus-Stop();**

I²C总线停止。产生相应的停止信号及恢复I²C总线，使I²C总线进入空闲状态。

在I²C写入数据的函数之中有返回值，这个值为读取从I²C设备返回的ACK/NAK的值。在大多数的应用之中可以忽略这个返回值。事实上在我们的应用之中，从来没有理会过这个返回值。对此，在有人机对话的产品之中一般不会有问题的。如果写入到I²C设备出错，一般使用者都会再动作一次，这样就人为地重发了一次，确保写入相应的数据到I²C设备。但是，如果是机器自动控制的或者对写入数据要求较严格的，一般调用时判断返回值。如果返回NAK的则重发一次，多次重发后还是返回NAK的做错误提示。

写I²C设备的寄存器的方法:

```
MI2C-Bus-Start();  
MI2C-Bus-Write (I2C设备写地址);  
MI2C-Bus-Write (寄存器索引值);           // 可能是多个字节  
MI2C-Bus-Write (寄存器内容);             // 可能是多个字节  
MI2C-Bus-Stop();
```

在I²C总线开始后，先写一个字节的I²C设备地址，这个地址一般可以在I²C设备的数据手册之中可以找到;



深圳市龙珠科技有限公司

Hard & Soft Technology Co., LTD.

<http://www.HSAV.com>

地址: 深圳市西乡龙吟二路 199 号 2 楼

技术支持: support@HSAV.com

hsavd107.pdf

电话/传真: 0755-27951479 27950879

业务联系: sales@HSAV.com

2009 年 12 月 24 日



紧跟着为一个字节或多个字节的寄存器索引；最后为一个或多个字节的数值。在OTG15E之中，寄存器索引值为1个字节，数值可以从一个字节到多个字节不等，具体可参阅《OTG15X系列I²C软件用户手册》。最后要调用停止，这样就可以写入数据到I²C设备了。

从I²C设备读取数据一般分为两种：

一种为寄存器型读取，例如24C01、OTG15X、DA32UD等，这种设备的特点是先做一个假写的动作再读取数据；另一种为通讯型读取，例如IP72C等，这种设备的特点是用户主机无法直接指定需要读取数据的顺序，I²C设备会将数据以队列的形式准备好，在用户主机读完一组后再传输下一组直到所有的数据被读完。

在I²C系统之中，因为很多时候I²C设备内的状态是随时变化的，所以一般需要另外增加一个INT脚，从I²C设备输出到用户主机，用户主机可以在INT变低后读取数据，这样可以减少用户主机查询的时间。这样的系统会额外需要用户主机多一个输入脚，而且一般都是在主循环之中不停检测这个INT输入脚，当发现INT变低后开始调用读取函数进行读取。

寄存器型读取例如OTG15X的读取方法：

```
if (!HAL_I2C_INT()) { // 检测到I2C从机产生中断, INT为低
    MI2C-Bus-Start();
    MI2C-Bus-Write(I2C设备写地址索引值);
    MI2C-Bus-Write(中断寄存器); // 可能是多个字节
    MI2C-Bus-Stop();

    MI2C-Bus-Start();
    MI2C-Bus-Write(I2C设备读地址);
    gLocal_1 = MI2C-Bus-Read(0); // 读取数据，可能是多个字节
    gLocal_1 = MI2C-Bus-Read(1); // 最后一次读取数据输入参数必须为1
    MI2C-Bus-Stop();
}
```

当用户主机检测到INT为低时，先读取中断寄存器以了解产生中断的原因；然后根据中断的标志查找对应的中断位的寄存器索引值；最后通过索引值再读取相应的寄存器数据。寄存器读取方式通常是需读取的数据是相对确定的，而且可以通过中断寄存器来指示正在变化的寄存器。这样先写入寄存器索引值后读取的方式，在一些手册中叫做假写；这时并没有任何数据写入I²C设备，只是提醒I²C设备准备相应的数据供下一步读取。

通讯型读取例如IP72X的读取方法：

```
if (!HAL_I2C_INT()) { // 检测到I2C从机产生中断, INT为低
    MI2C-Bus-Start();
    MI2C-Bus-Write(I2C设备读地址);
    gLocal_1 = MI2C-Bus-Read(0); // 读取寄存器索引值，可能是多个字节
    switch(gLocal_1) { // 从读取到的索引值作相应的数据读取
        case 0x01:
            gLocal_1 = MI2C-Bus-Read(0); // 读取数据，可能是多个字节
            break;
        case: // 其他需要处理的case
            break;
    }
```



```
}  
MI2C-Bus_Read(1);           // 最后一次读取数据输入参数必须为1  
MI2C-Bus_Stop();  
}
```

当用户检测到INT为低时，调用I²C总线开始，写入I²C设备的读地址，之后即可先读取寄存器索引值，再根据索引值来读取对应的数据。这样不需要再使用写地址作假写以让I²C设备根据这个假写的索引值准备数据，因而在读取时的效率较高。

在读取I²C设备数据时，开发时需要注意的就是传入到函数的参数，这个参数一般读取都为0，即应答ACK，表示接下来还有读的动作；但是，最后一个字节读取时必须为1，即应答NAK，表示这个字节为读的最后字节，跟随的将会是I²C总线停止信号。

可以下载I2C-bus.c及d文件直接编译使用。在我们的SDK软件包之中可以找到这2个文件，其中的# if 0之中为示范的定义方式，用户可以直接在自己的头文件中定义使用。

用好这4个函数即可以方便地开发出控制I²C设备的产品。

定义三个宏的注意事项：

```
#define HAL_I2C_SCL(b)           {P10=b;}           //定义SCL  
#define HAL_I2C_SDA(b)           {P11=b;}           //定义SDA输出  
#define HAL_I2C_IN_SDA()          (P11)              //定义SDA输入
```

如果为标准8051的I/O口类型，因为8051的I/O口其实与I²C是相同的，即谁也没有输出高电平，高电平是依靠上拉电阻实现的，这样直接按照电平来定义即可；另外一种带输出控制的真双向口I/O口，例如PIC系列单片机，这时当需要输出低电平时将输出寄存器设置为输出，同时将输出电平置为低，当需要高电平时，仅需要将输出寄存器变为输入即可，这样就可以与标准的I²C电平相兼容，如下面的宏所示。

```
#define HAL_I2C_SCL (b)           if (b) {OUTA0(0);} else {OUTA0(1); OPA0(0);} }  
#define HAL_I2C_SDA (b)           if (b) {OUTA1(0);} else {OUTA1(1); OPA0(0);} }  
#define HAL_I2C_IN_SDA()          (PA)
```

以上宏假设调用OUTA0(0)时，将PA0设置为输入；调用OUTA0(1)时，将PA0设置为输出。调用OPA0(0)时，PA0输出低电平，实际上需要SCL输出高时将PA0变为输入；在SCL输出低时将PA0变为输出，而且输出低电平。以上处理可以获得与I²C电气规格相同的效果

I²C的函数之中调用了MI2C_100K_DELAY及MI2C_400K_DELAY两个延时函数，这两个必须在用户程式之中定义。其中100K为低速的，400K为高速的。一般的I²C设备分为高速及低速设备，主要的分别是字节之间处理的速度有差异。因为完成一个字节后可能需要时间准备下一个字节，所以需要较长的延时，但字节之间的位延时可以短一些，这样可以有效提高传输的速度。当然，如果对传输的速度没有要求，这两个函数可以定义成相同的。

请注意：延时时间可以比I²C规格长，即变慢，但不能变快。所以在刚调试系统时是先使用较大的值获得较长的延时时间，这样免除因为延时不够造成的控制失灵的情况出现。

在系统正常工作及稳定之后逐渐减少延时的值以获得更好的性能。



```

void MI2C_100K_DELAY(); // 按照100Kbps的标准延时, 以确保状态稳定
    BYTE gLocal_1;
    gLocal_1 = 10;
    while (--gLocal_1 != 0); // 这里只是复制给一个局部变量自减以达到延时的目的
    return;
}

void MI2C_400K_DELAY() { // 按照400Kbps的标准延时, 可以有效地提高通讯的速度
    return; // 一般的单片机这里直接返回就可以了。如果太快则需要按上面的方法延时
}

```

请注意I²C设备地址的表示方法, 一般数据手册有两种写法: 一种是标准的地址, 使用时需要向左移一位; 另外一种已经是左移了一位, 这时将设备地址分为写地址及读地址, 直接写入即可。OTG15X及IP72X都是采用后者, 例如IP72X的地址为0x72, 则在使用到读数据时地址为0x73, 不需再移位。

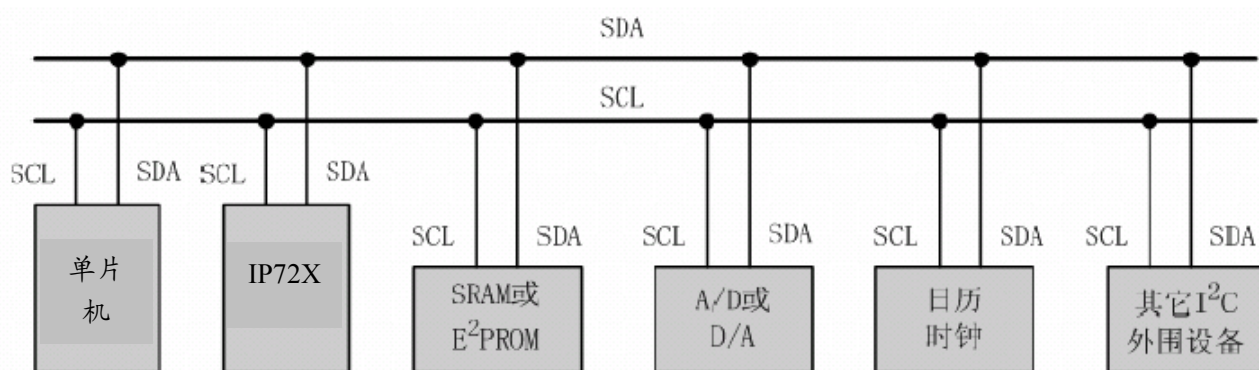
总结上述如下:

1. 定义MI2C_100k_DELAY及400K延时函数;
2. 定义HAL_I2C_SCL及HAL_I2C_SDA等3个宏;
3. 使用开始、读字节、写字节及停止4个函数作I²C的读写。这样你的的系统之中即拥有了2个I²C设备控制的能力。如果需要深入了解I²C, 请参阅下一节具体说明。

◆ I²C总线硬件的详细说明

I²C总线一般应用于单片机系统中对多个设备进行控制, 最大特点是所有的I²C设备并联在一起, 通讯速度为100Kbps或400Kbps。

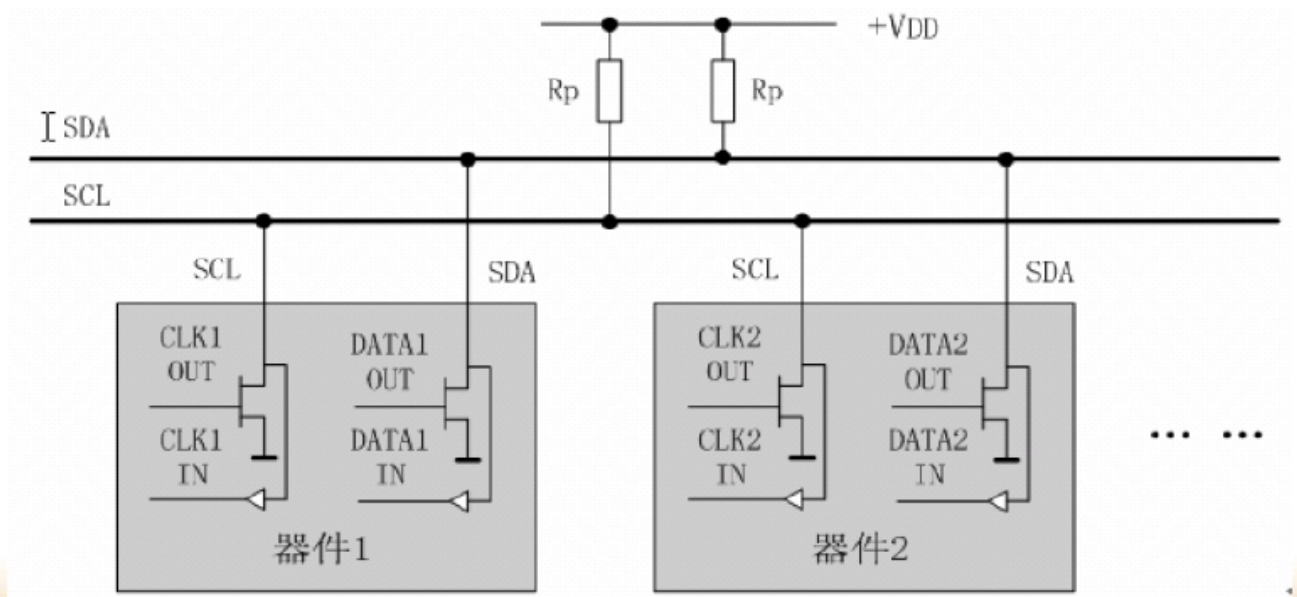
I²C总线只有两根双向信号线。一根是数据线SDA, 另一根是时钟线SCL。



I²C总线通过上拉电阻接正电源。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。

每个接到I²C总线上的器件都是唯一的地址。主机与其他器件间的数据传送可以由主机发送数据到其他器件, 这时主机即为发送器。由总线上接收数据的器件则为接收器。

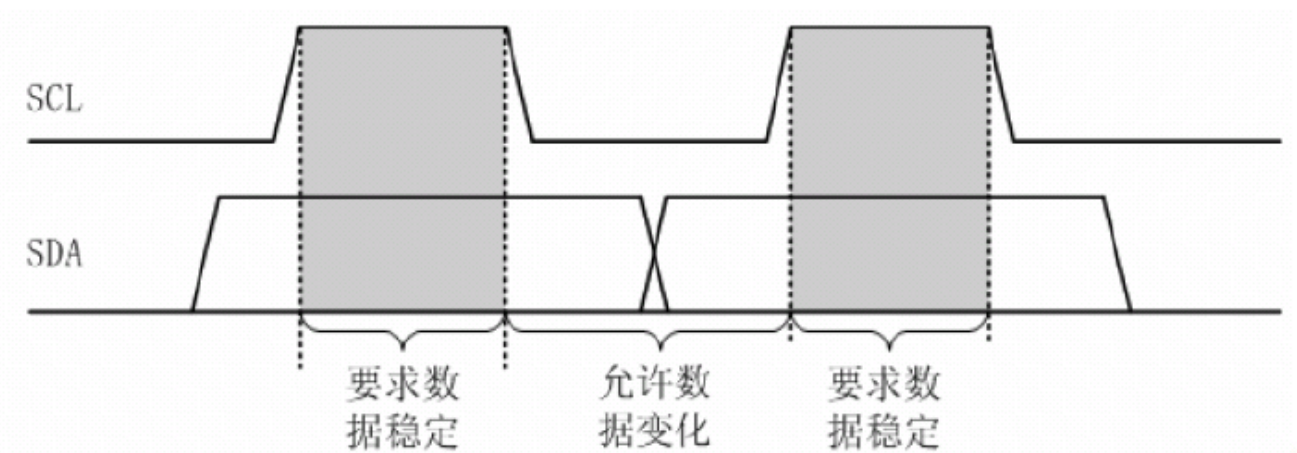
在多主机系统中, 可能是同时有几个主机企图启动总线传送数据。为了避免混乱, I²C总线要通过总线仲裁, 以决定由哪一台主机控制总线。



◆ I²C总线的数据传送

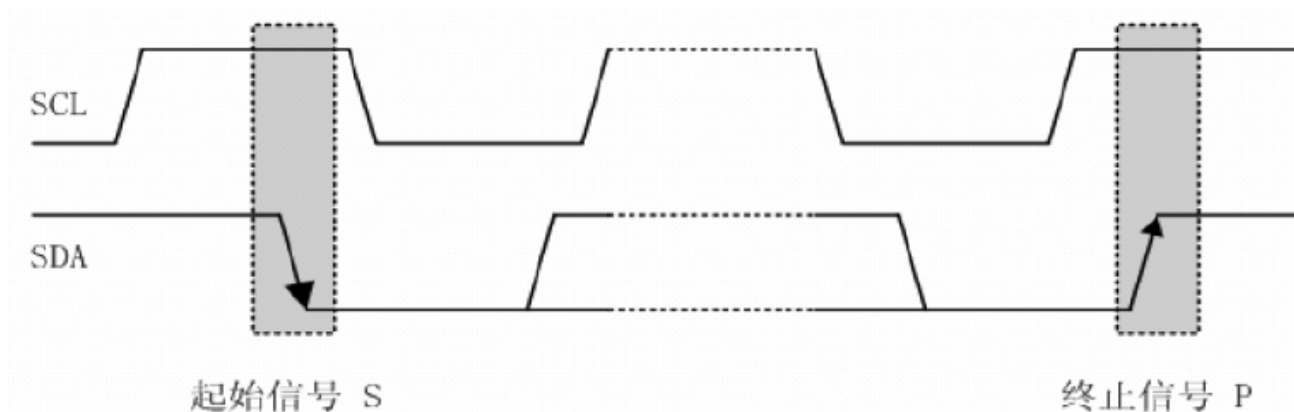
一，数据位的有效性规定

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。



二，起始和终止信号

SCL线为高电平期间，SDA线由高电平向低电平的变化表示起始信号；SCL线为高电平期间，SDA线由低电平向高电平的变化表示终止信号。



起始和终止信号都是由主机发出的，在起始信号产生后，总线就处于被占用的状态；在终止信号产生后，总线就处于空闲状态。

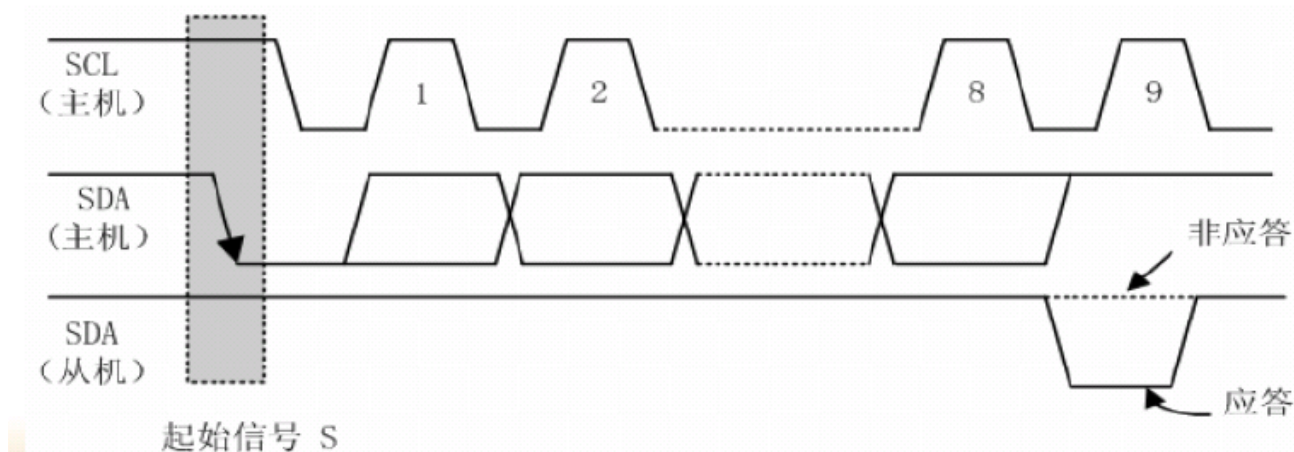
连接到I²C总线上的器件，若具有I²C总线的硬件接口，则很容易检测到起始和终止信号。对于不具备I²C总线硬件接口的有些单片机来说，为了检测起始和终止信号，必须保证在每个时钟周期内对数据线SDA采样两次。

接收器件收到一个完整的数据字节后，有可能需要完成一些其他工作，如处理内部中断服务等，可能无法立刻接收下一个字节，这时接收器件可以将SCL线拉成低电平，从而使主机处于等待状态。直到接收器件准备好接收下一个字节时，再释放SCL线使之为高电平，从而使数据传送可以继续进行。

三、数据传送格式

(1) 字节传送与应答

每一个字节必须保证8位长度。数据传送时，先传送最高位（MSB），每一个被传送的字节后面都必须跟随一位应答位（即一帧共有9位）。



由于某种原因从机不对主机寻址信号应答时（如从机正在进行实时性的处理工作而无法接收总线上的数据），它必须将数据线置于高电平，而由主机产生一个终止信号以结束总线的数据传送。

如果从机对主机进行了应答，但在数据传送一段时间后无法继续接收更多的数据时，从机可以通过对无法接收的第一个数据字节的“非应答”通知主机，主机则应发出终止信号以结束数据的继续传送。

当主机接收数据时，它收到最后一个数据字节后，必须向从机发出一个结束传送的信号。这个信号是由对从机的“非应答”来实现的。然后，从机释放SDA线，以允许主机产生终止信号。



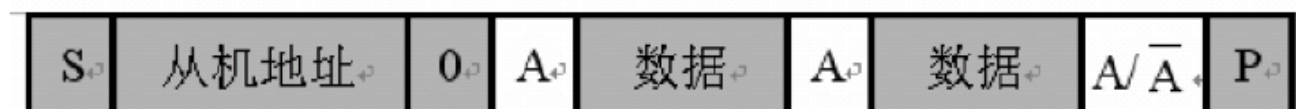
(2) 数据帧格式

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/），用“0”表示主机发送数据（）， “1”表示主机接收数据（R）。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。

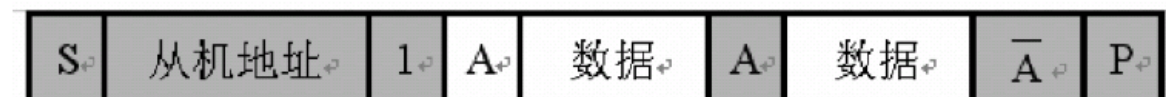
在总线的一次数据传送过程中，可以有以下几种组合方式：

A，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：

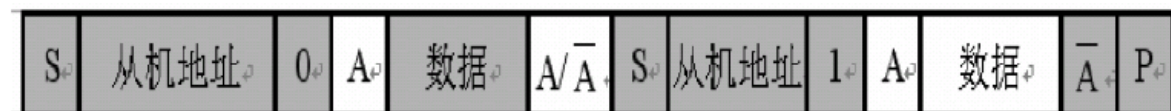


注：有阴影部分表示数据由主机向从机传送，无阴影部分则表示数据由从机向主机传送。A表示应答， \bar{A} 表示非应答（高电平）。S表示起始信号，P表示终止信号。

B，主机在第一个字节后，立即由从机读数据。



C，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/写方向正好相反。



四，总线的寻址

I²C总线协议有明确的规定：采用7位的寻址字节（寻址字节是起始信号后的第一个字节）。

(1) 寻址字节的位定义



D7～D1位组成从机的地址。D0位是数据传送方向位，为“0”时表示主机向从机写数据，为“1”时表示主机由从机读数据。

主机发送地址时，总线上的每个从机都将这7位地址码与自己的地址进行比较，如果相同，则认为自己正被主机寻址，根据R/位将自己确定为发送器或接收器。

从机的地址由固定部分和可编程部分组成。在一个系统中可能希望接入多个相同的从机，从机地址中可编程部分决定了可接入总线该类器件的最大数目。如一个从机的7位寻址位有4位是固定位，3位是可编程



位，这时仅能寻址8个同样的器件，即可以有8个同样的器件接入到该I²C总线系统中。

(2) 寻址字节中的特殊地址

固定地址编号0000和1111已被保留为特殊用途。

地 址 位						R/W	意 义
0	0	0	0	0	0	0	通用呼叫地址
0	0	0	0	0	0	1	起始字节
0	0	0	0	0	1	X	CBUS 地址
0	0	0	0	0	1	0	为不同总线的保留地址
0	0	0	0	0	1	1	保留
0	0	0	0	1	X	X	
1	1	1	1	1	X	X	
1	1	1	1	0	X	X	十位从机地址

I²C总线特殊地址表

起始信号后的第一字节的8位为“0000 0000”时，称为通用呼叫地址。通用呼叫地址的用意在第二字节中加以说明。格式为：

第一字节（通用呼叫地址）									第二字节							LSB	
0	0	0	0	0	0	0	0	A	X	X	X	X	X	X	X	B	A

第二字节为06H时，所有能响应通用呼叫地址的从机器件复位，并由硬件装入从机地址的可编程部分。能响应命令的从机器件复位时不拉低SDA和SCL线，以免堵塞总线。

第二字节为04H时，所有能响应通用呼叫地址并通过硬件来定义其可编程地址的从机器件将锁定地址中的可编程位，但不进行复位。

如果第二字节的方向位B为“1”，则这两个字节命令称为硬件通用呼叫命令。

在这第二字节的高7位说明自己的地址。接在总线上的智能器件，如单片机或其他微处理器能识别这个地址，并与之传送数据。硬件主器件作为从机使用时，也用这个地址作为从机地址。格式为：

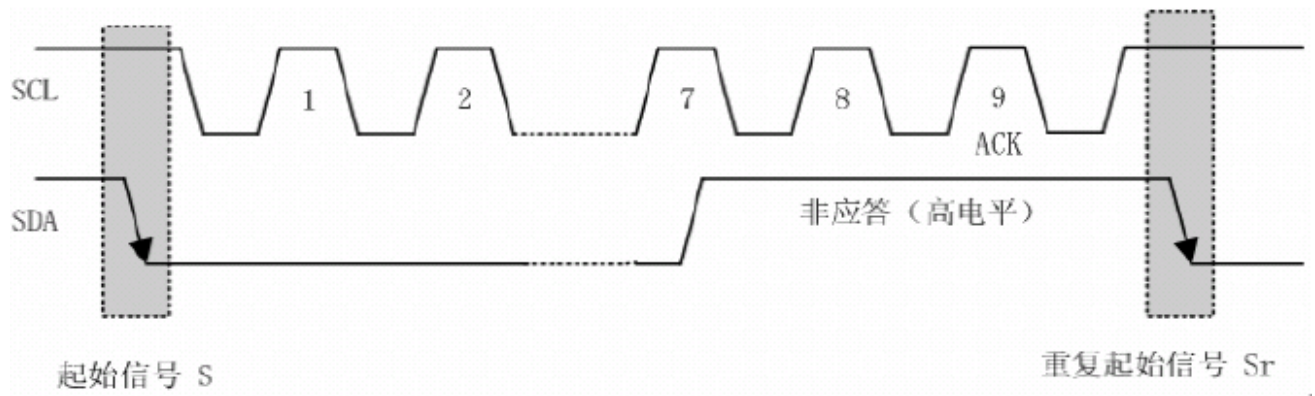
S	0000 0000	A	主机地址	1	A	数据	A	数据	A	P
---	-----------	---	------	---	---	----	---	----	---	---

在系统中另一种选择可能是系统复位时硬件主器件工作在从机接收器方式，这时由系统中的主机先告诉硬件主器件数据应送往的从机器件地址，当硬件主器件要发送数据时就可以直接向指定从机器件发送数据了。

(3) 起始字节

起始字节是提供给没有I²C总线接口的单片机查询I²C总线时使用的特殊字节。

不具备I²C总线接口的单片机，则必须通过软件不断地检测总线，以便及时地响应总线的请求。单片机的速度与硬件接口器件的速度就出现了较大的差别，为此，I²C总线上的数据传送要由一个较长的起始过程加以引导。



引导过程由起始信号、起始字节、应答位、重复起始信号 (Sr) 组成。

请求访问总线的主机发出起始信号后，发送起始字节 (0000 0001)，另一单片机可以用一个比较低的速率采样SDA线，直到检测到起始字节中的7个“0”中的一个为止。在检测到SDA线上的高电平后，单片机就可以用较高的采样速率，以便寻找作为同步信号使用的第二个起始信号Sr。

在起始信号后的应答时钟脉冲仅仅是为了和总线所使用的格式一致，并不要求器件在这个脉冲期间作应答。