第10章　AT89S51单片机系统的串行扩展

单片机系统扩展除并行扩展外，串行扩展技术也得到广泛应用。

与并行扩展比，串口器件与单片机相连的I/O口线少（仅需1～4条），极大简化器件间连接，进而提高可靠性；串行接口器件体积小，占用电路板空间小，减少电路板空间和成本。除上述优点外，还有工作电压宽、抗干扰能力强、功耗低、数据不易丢失等特点。因此，目前串行扩展技术在单片机系统中已得到广泛应用。

常用串行扩展接口有I^2C（Inter Interface Circuit，内部接口电路）串行总线接口、单总线（1-Wire）接口以及SPI串行外设接口。

本章介绍上述几种串行接口总线的工作原理及特点，以及串行扩展的典型设计案例。

10.1 单总线串行扩展

单总线（也称1-Wire bus）由美国DALLAS公司推出的外围串行扩展总线。只有一条数据输入/输出线DQ，总线上所有器件都挂在DQ上，电源也通过这条信号线供给。

单总线系统中配置的各种器件，由DALLAS公司提供的专用芯片实现。每个芯片都有64位ROM，厂家对每一芯片都用激光烧写编码，其中存有16位十进制编码序列号，它是器件的地址编号，确保它挂在总线上后，可以唯一被确定。除了器件的地址编码外，芯片内还包含收发控制和电源存储电路，见图11-1。这些芯片耗电量都很小（空闲时几μW，工作时几mW），工作时从总线上馈送电能到大电容中就可以工作，故一般不需另加电源。

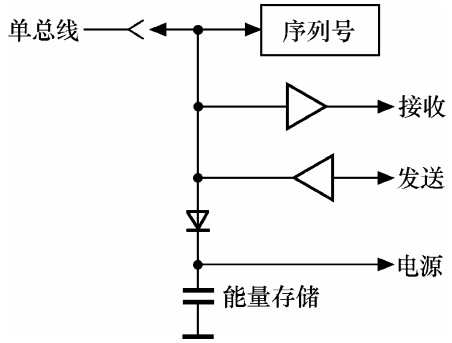

图10-1 单总线芯片内部结构示意图

10.1.1 单总线扩展的典型应用-DS18B20的温度测量系统

单总线应用典型案例是采用单总线温度传感器DS18B20的温度测量系统。

1. 单总线温度传感器DS18B20简介

DS18B20是美国DALLAS公司生产的数字温度传感器，体积小、低功耗、抗干扰能力强。可直接将温度转化成数字信号传送给单片机处理，因而可省去传统的信号放大、A/D转换等外围电路。

DS18B20测量温度范围-55～+128℃，在-10～+ 85℃范围内，测量精度可达±0.5℃，非常适合于恶劣环境的现场温度测量，也可用于各种狭小空间内设备的测温，如环境控制、过程监测过程监测、测温类消费电子产品以及多点温度测控系统。

图10-2为单片机与多个带有单总线接口的数字温度传感器DS18B20芯片的分布式温度监测系统，图11-2中多个DS18B20都挂在单片机的1根I/O口线（即DQ线）上。单片机对每个DS18B20通过总线DQ寻址。DQ为漏极开路，须加上拉电阻。

DS18B20的一种封装形式见图10-2。除DS18B20外，在该数字温度传感器系列中还有DS1820、DS18S20、DS1822等其他型号，工作原理与特性基本相同。

DS18B20每芯片都有唯一64位光刻ROM编码，它是DS18B20地址序列码，目的使每个DS18B20地址都不相同，这样可实现在一根总线上挂接多个DS18B20的目的。

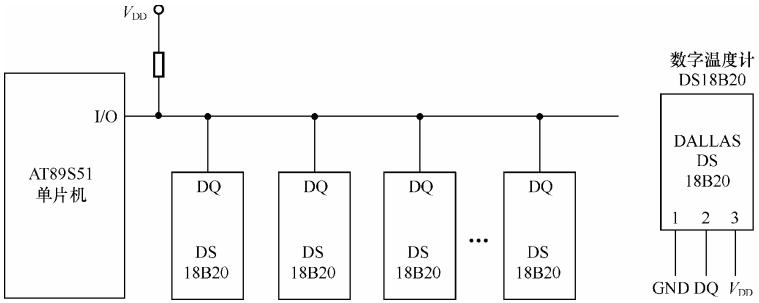

图10-2 单总线构成的分布式温度监测系统

片内有9个字节的高速暂存器RAM单元，9个字节具体分布如下：

温度低位	温度高位	TH	TL	配置	—	—	—	8 位 CRC
第1字节	第2字节							第9字节

第1字节和第2字节是在单片机发给DS18B20温度转换命令发布后，经转换所得的温度值，以两字节补码形式存放其中。一般情况下，用户多使用第1字节和第2字节。单片机通过单总线可读得该数据，读取时低位在前，高位在后。第3、4字节分别是由软件写入用户报警的上下限值TH和TL。第5个字节为配置寄存器，可对其更改DS18B20的测温分辨率，高速暂存器的第6、7、8字节未用，为全1。第9字节是前面所有8个字节的CRC码，用来保证正确通信。片内还有1个E2PROM为TH、TL以及配置寄存器的映像。

配置寄存器各位的定义如下：

TM	R1	R0	1	1	1	1	1

其中，TM位出厂时已被写入0，用户不能改变；低5位都为1；R1和R0用来设置分辨率。表10-1列出了R1、R0与分辨率和转换时间的关系。用户可通过修改R1、R0位的编码，获得合适的分辨率。

表10-1　R1、R0与分辨率和转换时间的关系

R1	R0	分辨率/位	最大转换时间/ms
	0	9	93.75
0	1	10	187.5
1	0	11	375
1	1	12	750

由表10-1，DS18B20转换时间与分辨率有关。当设定为9位时，转换时间为93.75ms；设定10位时，转换时间为187.5 ms；当设定11位时，转换时间为375ms；当设定为12位时，转换时间为750ms。

表10-2列出了DS18B20温度转换后所得到的16位转换结果的典型值。

表10-2　DS18B20温度数据

温度/℃	16 位 2 进制温度值																16 进制温度值
	符号位（5 位）					数据位（11 位）											
+125	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0	0x07d0
+25.0625	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	0x0191
-25.0625	1	1	1	1	1	1	1	0	0	1	1	0	1	1	1	1	0xfe6f
-55	1	1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0xfc90

下面介绍温度转换的计算方法。

当DS18B20采集的温度为+125℃时，输出为0x07d0，则：

实际温度=（0x07d0）/16=(0×16^3+7×16^2+13×16^1+0×16^0)/16=125℃

当DS18B20采集的温度为-55℃时，输出为0xfc90，由于是补码，则先将11位数据取反加1得0x0370，注意符号位不变，也不参加运算，则

实际温度=（0x0370）/16=(0×163+3×162+7×161+0×160)/16=55℃

注意，负号则需对采集的温度进行判断后，再予以显示。

2. DS18B20的工作时序

DS18B20对工作时序要求严格，延时时间需准确，否则容易出错。DS18B20的工作时序包括初始化时序、写时序和读时序。

（1）初始化时序，单片机将数据线DQ电平拉低480~960μs后释放，等待15~60μs，单总线器件即可输出一持续60~240μs的低电平，单片机收到此应答后即可进行操作。

（2）写时序，当单片机将数据线DQ电平从高拉到低时，产生写时序，有写“0”和写“1”两种时序。写时序开始后，DS18B20在15~60μs期间从数据线上采样。如果采样到低电平，则向DS18B20写的是“0”；如果采样到高电平，则向DS18B20写的是“1”。这两个独立的时序间至少需要拉高总线电平1μs的时间。

2. DS18B20的工作时序

工作时序要求严格，延时时间需准确，否则容易出错。

DS18B20的工作时序包括初始化时序、写时序和读时序。

（1）初始化时序，单片机将数据线电平拉低480~960μs后释放，等待15~60μs，单总线器件即可输出一持续60~240μs的低电平，单片机收到此应答后即可进行操作。

（2）写时序，当单片机将数据线电平从高拉到低时，产生写时序，有写“0”和写“1”两种时序。写时序开始后，DS18B20在15~60μs期间从数据线上采样。如果采样到低电平，则向DS18B20写的是“0”；如果采样到高电平，则向DS18B20写的是“1”。这两个独立时序间至少需拉高总线电平1μs时间。

（3）读时序，当单片机从DS18B20读取数据时，产生读时序。此时单片机将数据线电平从高拉到低使读时序被初始化。如果在此后15μs内，单片机在数据线上采样到低电平，则从DS18B20读的是“0”；如果在此后的15μs内，单片机在数据线上采样到高电平，则从DS18B20读的是“1”。

3. DS18B20的命令

DS18B20片内都有唯一的64位光刻ROM编码，出厂时已刻好。它是DS18B20的地址序列码，目的是使每个DS18B20的地址都不相同，这样就可实现在一根总线上挂接多个DS18B20的目的。64位光刻ROM的各位定义如下：

8 位产品类型标号	DS18B20 的 48 位自身序列号	8 位 CRC 码

DS18B20所有命令均为8位长，常用的命令代码见表10-3。

表10-3 DS18B20的部分命令

命令功能	命令代码
读 DS18B20 中 ROM 的编码（即 64 位地址）	33H
匹配 ROM，发出此命令之后，接着发出 64 位编码，访问与该编码对应的 DS18B20 并使其做出响应，为下一步对其进行读写做准备(总线上有多个 DS18B20 时使用)	55H
搜索 ROM(单片机识别所有的 DS18B20 的 64 位编码)	F0H
跳过读序列号的操作（总线上仅有 1 个 DS18B20 时使用）	CCH

下面介绍表10-3中命令的用法。当主机需要对多个单总线上的某一DS18B20进行操作时，首先应将主机逐个与DS18B20挂接，读出其序列号（33H）；然后再将所有的DS18B20挂接到总线上，单片机发出匹配ROM命令（55H），紧接着主机提供的64位序列号之后的操作就是针对该DS18B20的。

如果主机只对一个DS18B20进行操作，就不需要读取ROM编码以及匹配ROM编码，只要用跳过ROM（CCH）命令，就可按表10-4执行如下温度转换和读取命令。

表10-4　DS18B20的部分命令

命令功能	命令代码
启动温度转换	44H
读取暂存器中的温度数据	BEH
将温度上下限数据写入片内 RAM 的第 3、4 字节（TH、TL）	4EH
把片内 RAM 的第 3、4 字节的数据到复制暂存器 TH 与 TL 中	48H
将 E²PROM 第 3、4 字节的数据恢复到片内 RAM 中的第 3、4 字节	B8H
读供电方式，寄生供电时，DS18B20 发送 0；外部电源供电，DS18B20 发送 1	B4H
报警搜索，只有温度超过设定的上下限的芯片才做响应	ECH

10.1.2 设计案例：单总线DS18B20温度测量系统

【例10-1】利用DS18B20和LED数码管实现单总线温度测量系统，原理电路见图11-3。DS18B20测量范围是−55～128℃。本例只显示00～99。通过本例读者应掌握DS18B20特性及单片机I/O实现单总线协议的方法。

Proteus仿真时，用手动，即用鼠标单击DS18B20图标上的“↑”或“↓”来改变温度，注意手动调节温度同时，LED数码管会显示出与DS18B20窗口相同的2位温度数值。

电路中74LS47是BCD-7段译码器/驱动器，用于将单片机P0口输出欲显示的BCD码转化成相应的数字显示的段码，并直接驱动LED数码管显示。

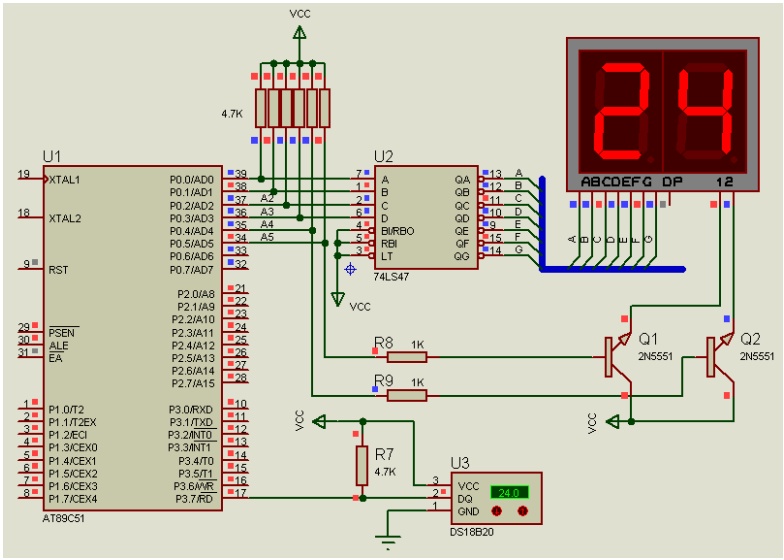

图10-3　单总线DS18B20温度测量与显示系统

参考程序如下：

```
#include "reg51.h"
#include "intrins.h"
#define uchar unsigned char
#define uint unsigned int
#define out P0
sbit smg1=out^4;
sbit smg2=out^5;
sbit DQ=P3^7;
void delay5(uchar);
void init_ds18b20(void);
uchar readbyte(void);
void writebyte(uchar);
uchar retemp(void);
```

```
void main(void)                          //主函数
{
     uchar i,temp;
     delay5(1000);
     while(1)
     {
        temp=retemp();
        for(i=0;i<10;i++)                        //连续扫描数码管10次
        {
                out=(temp/10)&0x0f;
                smg1=0;
                smg2=1;
                delay5(1000);                    //延时5ms
                out=(temp%10)&0x0f;
                smg1=1;
                smg2=0;
                delay5(1000);                    //延时5ms
        }
     }
}
```



```
void delay5(uchar n)                     //函数功能：延时5μs
{
        do
        {
                _nop_();
                _nop_();
                _nop_();
                n--;
        }
        while(n);
}

void init_ds18b20(void)          //函数功能：18B20初始化
{
        uchar x=0;
        DQ =0;
        delay5(120);
        DQ =1;
        delay5(16);
        delay5(80);
}
```



```
uchar readbyte(void)                     //函数功能：读取1字节数据
{
   uchar i=0;
   uchar date=0;
   for (i=8;i>0;i--)
   {
          DQ =0;
          delay5(1);
          DQ =1;                         //15μs内拉释放总线
          date>>=1;
          if(DQ)
          date|=0x80;
          delay5(11);
    }
   return(date);
}
```



```
void writebyte(uchar dat)                        //函数功能：写1字节
{
 uchar i=0;
 for(i=8;i>0;i--)
 {
     DQ =0;
     DQ =dat&0x01;                               //写"1" 在15μs内拉低
     delay5(12);                                 //写"0" 拉低60μs
     DQ = 1;
     dat>>=1;
     delay5(5);
     }
}
```

```
uchar retemp(void)                    //函数功能：读取温度
{
   uchar a,b,tt;
   uint t;
   init_ds18b20();
   writebyte(0xCC);
   writebyte(0x44);
   init_ds18b20();
   writebyte(0xCC);
   writebyte(0xBE);
   a=readbyte();
   b=readbyte();
   t=b;
   t<<=8;
   t=t|a;
   tt=t*0.0625;
   return(tt);
}
```


DS18B20体积小、适用电压范围宽，是世界上第一片支持“单总线”接口的温度传感器。

现场温度测量直接以“单总线”数字方式传输，大大提高系统抗干扰性。所以单总线系统特别适用于测控点多、分布面广、环境恶劣以及狭小空间内设备的测温以及现场温度测量，如环境控制、设备或过程控制、测温类消费电子产品等。

10.2 SPI总线串行扩展

SPI（Serial Peripheral Interface，串行外设接口）是Motorola公司推出的一种同步串行外设接口，允许单片机与多个厂家生产的带有标准SPI接口的外围设备直接连接，以串行方式交换信息。

10.2.1 SPI总线的扩展结构

SPI外围串行扩展结构见图10-4。使用4条线：串行时钟SCK，主器件输入/从器件输出数据线MISO，主器件输出/从器件输入数据线MOSI和从器件选择线 。

典型SPI系统是单主器件系统，从器件通常是外围接口器件，如存储器、I/O接口、A/D、D/A、键盘、日历/时钟和显示驱动等。

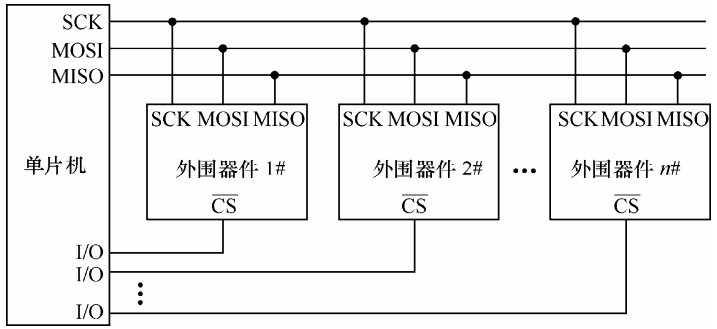

图10-4 SPI外围串行扩展结构图

单片机扩展多个外围器件时，SPI无法通过数据线译码选择，故外围器件都有片选端 。在扩展单个SPI器件时，外围器件的片选端CS*可接地或通过I/O口控制；在扩展多个SPI器件时，单片机应分别通过I/O口线来分时选通外围器件。

在SPI串行扩展系统中，如某一从器件只作输入（如键盘）或只作输出（如显示器）时，可省去一条数据输出（MISO）线或一条数据输入（MOSI）线，从而构成双线系统（ 接地）。

SPI系统中单片机对从器件选通需控制其CS*端，由于省去传输时的地址字节，数据传送软件十分简单。但在扩展器件较多时，需制较多的从器件CS*端，连线较多。

在SPI串行扩展系统中，主器件单片机在启动一次传送时，便产生8个时钟，传给接口芯片作为同步时钟，控制数据的输入和输出。数据的传送格式是高位（MSB）在前，低位（LSB）在后，见图11-5。数据线上输出数据的变化以及输入数据时的采样，都取决于SCK。但对不同的外围芯片，有的可能是SCK的上升沿起作用，有的可能是SCK的下降沿起作用。SPI有较高的数据传输速度，最高可达1.05Mbit/s。

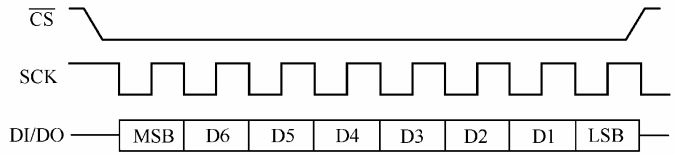

图11-5 SPI数据传送格式

目前世界各芯片公司为广大用户提供一系列具有SPI接口的单片机和外围接口芯片，例如Motorola公司存储器MC2814、显示驱动器MC14499和MC14489等各种芯片；美国TI公司的8位串行A/D转换器TLC549、12位串行A/D转换器TLC2543等。

SPI外围串行扩展系统的从器件要具有SPI接口。主器件是单片机。目前许多机型都带有SPI接口。由于AT89S51不带有SPI接口，可采用软件与I/O口结合来模拟SPI的接口时序。

10.2.2 设计案例：扩展带有SPI接口的8位串行A/D转换器TLC549

下面介绍SPI接口串行扩展应用案例，AT89S51扩展低价位、高性能的8位 A/D转换器TLC549。

TLC549是美国TI推出的一种低价位、高性能的8位 A/D转换器，它以8位开关电容逐次逼近方法实现 A/D转换，其转换速度小于 17μs，最大转换速率为 40kHz，内部系统时钟的典型值为4MHz，电源为 3~6V。它能方便地采用SPI串行接口方式与各种单片机连接，构成廉价的测控应用系统。

1. TLC549的引脚及功能

引脚见图11-6。

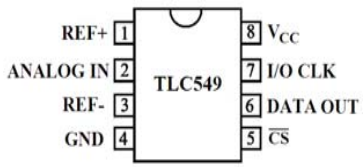

图11-6 TLC549的引脚

引脚功能如下：

- REF+：正基准电压输入 2.5V≤REF+≤Vcc+0.1V。
- REF-：负基准电压输入端，-0.1V≤REF-≤2.5V。且：（REF+）－（REF-）≥1V。
- V_{CC}：电源3V≤Vcc≤6V。
- GND：地。
- CS*：片选端。
- DATAOUT：转换结果数据串行输出端，与 TTL电平兼容，输出时高位在前。

- ANALOGIN：模拟信号输入端，0≤ANALOGIN≤Vcc，当ANALOGIN≥REF+电压时，转换结果为全“1”(0xff)，ANALOGIN≤REF- 电压时，转换结果为全“0”(0x00)。
- I/O CLOCK：外接输入/输出时钟输入端，同于同步芯片的输入输出操作，无需与芯片内部系统时钟同步。

2. TLC549 工作时序

工作时序见图11-7。从图可知：

（1）串行数据中高位A7先输出，最后输出低位A0；

（2）在每一次I/O COLCK高电平期间DATA OUT 线上数据产生有效输出，每出现一次I/O COLCK，DATA OUT线就输出1位数据。一个周期出现8次I/O COLCK 信号并对应8位数据输出；

（3）在CS*变为低电平后，最高有效位（A7）自动置于DATA OUT 总线 。其余7位（A6~A0）在前7 个I/O CLOCK 下降沿由时钟同步输出。B7～B0以同样方式跟在其后；

（4）tsu在片选信号CS*变低后，I/O COLCK 开始正跳变的最小时间间隔1.4μs；

（5）ten是从CS*变低到DATA OUT 线上输出数据最小时间（1.2μs）；

（6）只要I/O COLCK 变高就可读取DATA OUT 线上数据；

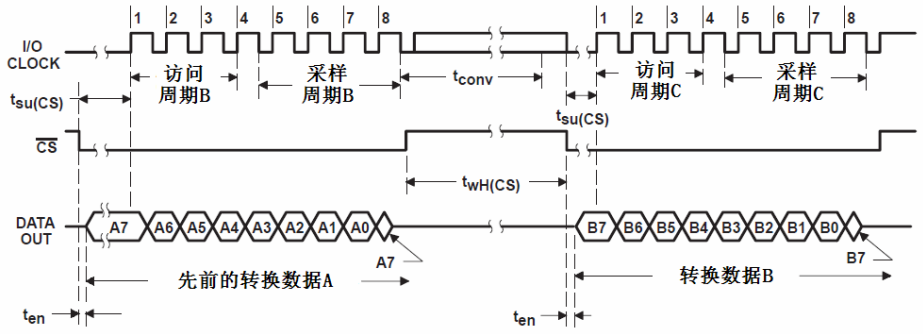

图11-7 TLC549工作时序

（7）只有在CS*端为低时，TLC549 才工作；

（8）TLC549的A/D转换电路没有启动控制端，只要读取前一次数据后马上就可以开始新的A/D 转换。转换完成后就进入保持状态。TLC549 每次转换所需时间是17μs， 它开始于 变为低电平后I/OCLOCK 的第8个下降沿，没有转换完成标志信号。

当CS*变为低电平后，TLC549芯片被选中， 同时前次转换结果的最高有效位MSB （A7）自 DATA OUT 端输出，接着要求从I/O CLOCK 端输入8个外部时钟信号，前7个 I/O CLOCK信号的作用，是配合 TLC549 输出前次转换结果的的A6~A0 位，并为本次转换做准备：

在第4个 I/O CLOCK 信号由高至低的跳变之后，片内采样/保持电路对输入模拟量采样开始，第8个 I/O CLOCK 信号的下降沿使片内采样/保持电路进入保持状态并启动 A/D开始转换。转换时间为 36 个时钟周期，最大为 17μs。

直到 A/D转换完成前的这段时间内，TLC549 的控制逻辑要求：或者 CS*保持高电平，或者 I/O CLOCK 时钟端保持36个系统时钟周期低电平。

由此可见，在TLC549的 I/O CLOCK 端输入8个外部时钟信号期间需要完成以下工作：读入前次A/D转换结果；对本次转换的输入模拟信号采样并保持；启动本次转换开始。

3. 单片机与TLC549 的接口设计

【例11-2】单片机控制串行8位A/D转换器TLC549进行A/D转换，原理电路与仿真结果见图11-8。由电位计RV1提供给TLC549模拟量输入，通过调节RV1上的“+”、“–”端，改变输入电压值。编写程序将模拟电压量转换成二进制数字量，本例用P0口输出控制8个发光二极管的亮与灭显示转换结果的二进制码，也可通过LED数码管将转换完毕的数字量以16进制数形式显示出来。

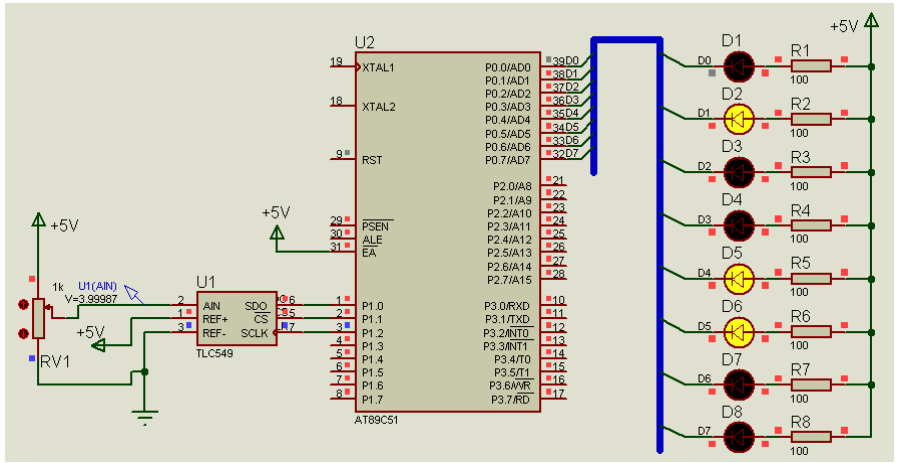

图11-8 单片机与TLC549接口的原理电路

参考程序如下：

```
#include<reg51.h>
#include<intrins.h>                    //包含_nop_()函数头文件
#define uchar unsigned char
#define uint unsigned int
#define  led  P0
sbit sdo=P1^0;     //定义P1.0与TLC549的SDO脚（即5脚DATA OUT）连接
sbit cs=P1^1;      //定义P1.1与TLC549的 脚连接
sbit sclk=P1^2;    //定义P1.2与TLC549的SCLK脚（即7脚I/O CLOCK）连接
```

```
void delayms(uint j)              //延时函数
{
   uchar i=250;
   for(;j>0;j--)
   {
while(--i);
        i=249;
        while(--i);
        i=250;
   }
}
```

```
void delay18us(void)                    延时约18μs函数
{
_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();
_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_(); nop_();_nop_();
}
uchar convert(void)
{
   uchar i,temp;
   cs=0;
   delay18us();
```

```
for(i=0;i<8;i++)
   {
        if(sdo==1)temp=temp|0x01;
        if(i<7)temp=temp<<1;
        sclk=1;
        _nop_(); _nop_(); _nop_();_nop_();
        sclk=0;
        _nop_(); _nop_();
   }
   cs=1;
   return(temp);
}
```

```
void main()
{
        uchar result;
        led=0;
        cs=1;
        sclk=0;
        sdo=1;
        while(1)
        {
                result=convert();
                led=result;              //转换结果从P0口输出驱动LED
                delayms(1000);
        }
}
```

由于TLC549 转换时间应大于17μs，本例采用延时操作方案，延时时间大约18μs，每次读取转换数据时间大于17μs即可。

10.3 I²C总线的串行扩展

I²C（Inter Interface Circuit）全称为芯片间总线，是目前使用广泛的芯片间串行扩展总线。目前世界上采用I2C总线有两个规范，荷兰飞利浦公司和日本索尼公司，现多采用飞利浦公司I2C总线技术规范，已成为电子行业认可的总线标准。采用I2C技术单片机以及外围器件种类很多，已广泛用于各类电子产品、家用电器及通信设备中。

10.3.1 I2C串行总线系统的基本结构

I2C串行总线只有两条信号线，一条是数据线SDA，另一条是时钟线SCL。SDA和SCL是双向的，I2C总线上各器件数据线都接到SDA线上，各器件时钟线均接到SCL线上。I2C总线系统基本结构见图10-6。

带有I2C接口单片机可直接与I2C总线接口的各种扩展器件（如存储器、I/O芯片、A/D、D/A、键盘、显示器、日历/时钟）连接。由于I2C总线采用纯软件寻址方法，无需片选线连接，大大简化总线数量。

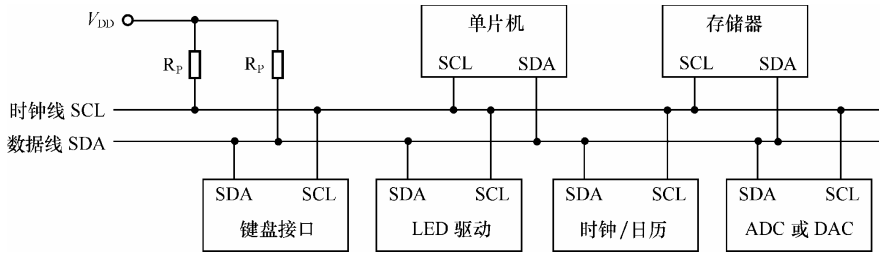

图10-6　I²C串行总线系统基本结构

I2C串行总线运行由主器件控制。主器件是指启动数据的发送（发出起始信号）、发出时钟信号、传送结束时发出终止信号的器件，通常由单片机来担当。从器件可以是存储器、LED或LCD驱动器、A/D或D/A转换器、时钟/日历器件等，从器件须带有I2C串行总线接口。

I2C总线空闲时，SDA和SCL两条线均为高。连接到总线上器件输出级必须是漏级或集电极开路，只要有一器件任意时刻输出低电平，都将使总线上信号变低，即各器件SDA及SCL都是“线与”关系。由于各器件输出为漏级开路，故须通过上拉电阻接正电源（见图10-6中两个电阻），以保证SDA和SCL在空闲时被上拉为高电平。SCL线上时钟信号对SDA线上各器件

间数据传输起同步控制作用。SDA线上数据起始、终止及数据的有效性均要根据SCL线上的时钟信号来判断。

在标准I2C普通模式下，数据传输速率为100kbit/s，高速模式下可达400kbit/s。总线上扩展器件数量不是由电流负载决定的，而是由电容负载确定。I2C总线每个器件接口都有一定等效电容，连接器件越多，电容值就越大，这会造成信号传输延迟。总线上允许器件数以器件电容量不超过400pF（通过驱动扩展可达4 000pF）为宜，据此可计算出总线长度及连接器件数量。每个I2C总线器件都有唯一地址，扩展器件时也要受器件地址数目限制。

I2C总线应用系统允许多主器件，由哪一个主器件来控制总线要通过总线仲裁来决定。如何进行总线仲裁，读者可查阅I2C总线仲裁协议。但是在实际应用中，经常遇到的是以单一单片机为主器件，其他外围接口器件为从器件的情况。

10.3.2 I2C总线的数据传送规定

1．数据位的有效性规定

I2C总线数据传送时，每一数据位传送都与时钟脉冲相对应。时钟脉冲为高电平期间，数据线上数据须保持稳定，在I2C总线上，只有在时钟线为低电平期间，数据线上电平状态才允许变化，见图10-7。

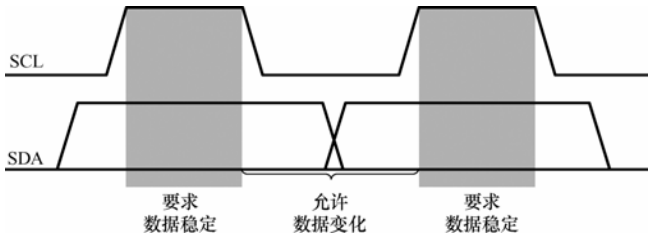

图10-7　数据位有效性规定

2．起始信号和终止信号

由I2C总线协议，总线上数据信号传送由起始信号（S）开始、由终止信号（P）结束。起始信号和终止信号都由主机发出，在起始信号产生后，总线就处于占用状态；在终止信号产生后，总线就处于空闲状态。下面结合图10-8介绍有关起始信号和终止信号的规定。

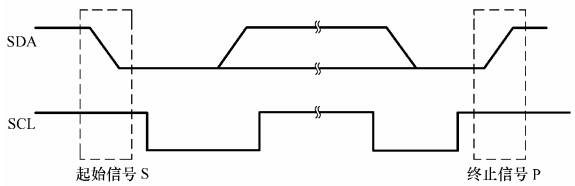

图10-8　起始信号和终止信号

（1）起始信号（S）。在SCL线为高期间，SDA线由高电平向低电平的变化表示起始信号，只有在起始信号以后，其他命令才有效。

（2）终止信号（P）。在SCL线为高期间，SDA线由低电平向高电平的变化表示终止信号。随着终止信号出现，所有外部操作都结束。

3．I2C总线上数据传送的应答

I2C总线数据传送时，传送字节数没有限制，但每字节须为8位长。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1位应答位（即1帧共有9位），见图10-9。

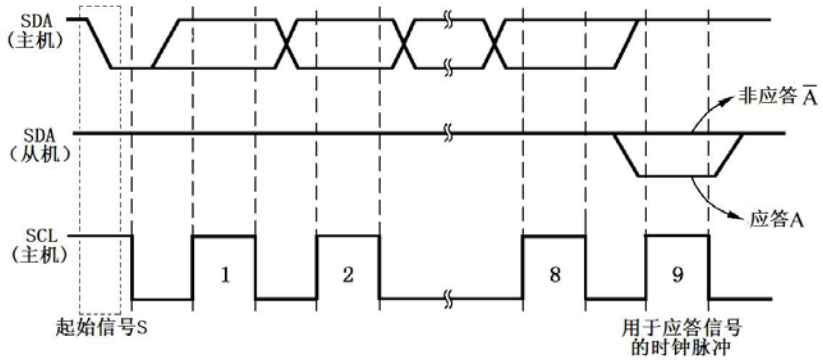

图10-9　I²C总线上的应答信号

I2C总线在传送每1字节数据后都须有应答信号A，应答信号在第9个时钟位上出现，与应答信号对应的时钟信号由主器件产生。这时发方须在这一时钟位上使SDA线处于高电平状态，以便收方在这一位上送出低电平应答信号A。

由于某种原因收方不对主器件寻址信号应答时，例如收方正在进行其他处理而无法接收总线上数据时，必须释放总线，将数据线置为高电平，而由主器件产生一个终止信号以结束总线的数据传送。

当主器件接收来自从机数据时，接收到最后一个数据字节后，须给从器件发送一个非应答信号（A*），使从机释放数据总线，以便主机发送一终止信号，从而结束数据传送。

4. I2C总线上的数据帧格式

I2C总线上传送数据信号即包括真正的数据信号，也包括地址信号。

I2C总线规定，在起始信号后必须传送一从器件地址（7位），第8位是数据传送的方向位（R/W*），用“0”表示主器件发送数据（W*），“1”表示主器件接收数据（R）。

每次数据传送总是由主器件产生的终止信号结束。但是，若主器件希望继续占用总线进行新的数据传送，则可不产生终止信号，马上再次发出起始信号对另一从器件进行寻址。因此，在总线一次数据传送过程中，可有以下几种组合方式：

（1）主器件向从器件发送n字节的数据，数据传送方向在整个传送过程中不变，数据传送格式：

S	从器件地址	0	A	字节 1	A	……	字节 (n-1)	A	字节 n	A/$\overline{A}$	P

其中：字节1~字节n为主机写入从器件的n字节的数据。格式中阴影部分表示主器件向从机发送数据，无阴影部分表示从器件向主器件发送，以下同。上述格式中的“从器件地址”为7位，紧接其后的“1”和“0”表示主器件的读/写方向，“1”-读，“0”-写。

（2）主器件读出来自从机的n字节。除第1个寻址字节由主机发出，n字节都由从器件发送，主器件接收，数据传送格式如下：

S	从机地址	1	A	字节 1	A	……	字节 (n-1)	A	字节 n	$\overline{A}$	P

其中：字节1~字节n为从器件被读出的n字节数据。主器件发送终止信号前应发送非应答信号，向从器件表明读操作要结束。

（3）主器件的读、写操作。在一次数据传送过程中，主器件先发送1字节数据，然后再接收1字节数据，此时起始信号和从器件地址都被重新产生一次，但两次读写的方向位正好相反。数据传送格式如下：

S	从器件地址	0	A	数据	A/$\overline{A}$	Sr	从器件地址 r	1	A	数据	$\overline{A}$	P

格式中的“Sr”表示重新产生的起始信号，“从器件地址r”表示重新产生的从器件地址。

由上可见，无论哪种方式，起始信号、终止信号和从器件地址均由主器件发送，数据字节传送方向则由主器件发出的寻址字节中的方向位规定，每个字节传送都须有应答位（A或A*）相随。

5．寻址字节

在上面介绍的数据帧格式中，均有7位从器件地址和紧跟其后的1位读/写方向位，即寻址字节。I2C总线的寻址采用软件寻址，主器件在发送完起始信号后，立即发送寻址字节来寻址被控的从器件，寻址字节格式如下：

寻址字节	器 件 地 址				引 脚 地 址			方向位
	DA3	DA2	DA1	DA0	A2	A1	A0	R/W

7位从器件地址为“DA3、DA2、DA1、DA0”和“A2、A1、A0”，其中“DA3、DA2、DA1、DA0”为器件地址，即器件固有地址编码，出厂时就已给定。“A2、A1、A0”为引脚地址，由器件“A2、A1、A0”为引脚地址，由器件引脚A2、A1、A0在电路中接高电平或接地决定（见图10-11）。

数据方向位（R/W*）规定了总线上的单片机（主器件）与从器件数据传送方向。R/W*=1，表示主器件接收（读）。R/W*=0，表示主器件发送（写）

6．数据传送格式

I2C总线每传送一位数据都与一个时钟脉冲对应，传送的每一帧数据均为一字节。但启动I2C总线后传送的字节数没有限制，只要求每传送一个字节后，对方回答一个应答位。在时钟线为高电平期间，数据线的状态就是要传送的数据。

数据线上数据改变须在时钟线为低电平间完成。在数据传输期间，只要时钟线为高电平，数据线都必须稳定，否则数据线上任何变化都当作起始或终止信号。

I2C总线数据传送必须遵循规定的数据传送格式。图10-10所示为一次完整的数据传送应答时序。由总线规范，起始信号表明一次数据传送的开始，其后为寻址字节。在寻址字节后是按指定读、写的数据字节与应答位。在数据传送完成后主器件都必须发送终止信号。在起始与终止信号之间传输的数据字节数由主器件（单片机）决定，理论上讲没有字节限制。

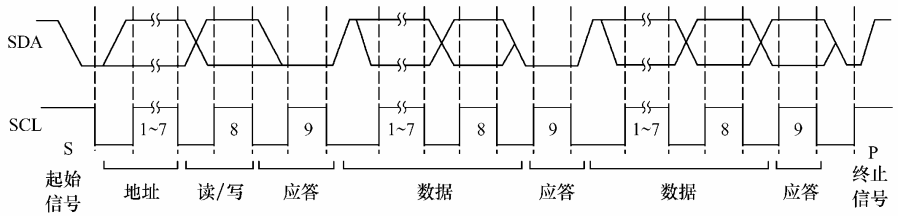

图10-10　I²C总线一次完整的数据传送应答时序

由上述数据传送格式可看出：

（1）无论何种数据传送格式，寻址字节都由主器件发出，数据字节传送方向则由寻址字节中方向位来规定。

（2）寻址字节只表明从器件的地址及数据传送方向。从器件内部的n个数据地址，由器件设计者在该器件的I2C总线数据操作格式中，指定第1个数据字节作为器件内的单元地址指针，并且设置地址自动加减功能，以减少从器件地址的寻址操作。

（3）每个字节传送都必须有应答信号（A/A*）相随。

（4）从器件接收到起始信号后都必须释放数据总线，使其处于高电平，以便主器件发送从机地址。

10.3.3 AT89S51的I2C总线扩展系统

许多公司都推出带有I2C接口的单片机及各种外围扩展器件，常见的有ATMEL的AT24C××系列存储器、PHILIPS的PCF8553（时钟/日历且带有256×8 RAM）和PCF8570（256×8 RAM）、MAXIM的MAX117/118（A/D转换器）和MAX517/518/519（D/A转换器）等。

I2C系统中主器件通常由带有I2C接口的单片机担当。从器件必须带有I2C总线接口。AT89S51单片机没有I2C接口，可利用并行I/O口线结合软件来模拟I2C总线时序，可使AT89S51不受没有I2C接口限制。因此，在许多应用中，都将I2C总线模拟传送作为常规设计方法。

图10-11为AT89S51单片机与具有I2C总线器件的扩展接口电路。图中，AT24C02为E2PROM芯片，PCF8570为静态256×8 RAM，PCF8574为8位I/O接口，SAA1064为4位LED驱动器。虽然各种器件的原理和功能有很大的差异，但它们与AT89S51单片机连接是相同的。

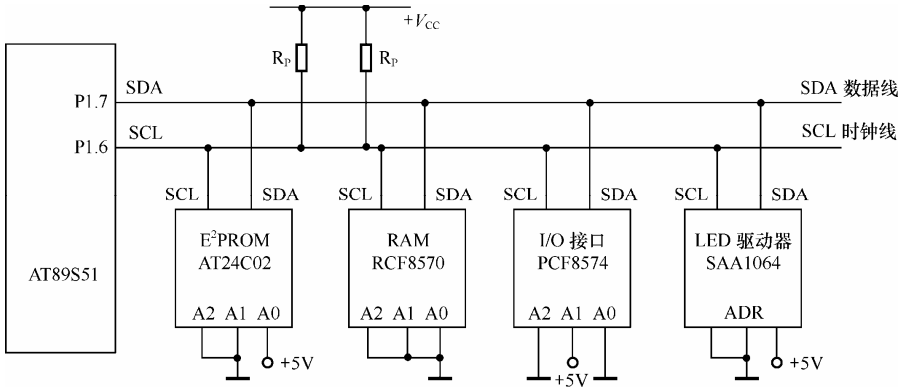

图10-11　AT89S51单片机扩展I²C总线器件的接口电路

10.3.4　I2C总线数据传送的模拟

由于AT89S51没有I2C接口，通常用I/O口线结合软件来实现I2C总线上的信号模拟。AT89S51为单主器件方式下，没有其他主器件对总线竞争与同步，只存在着主器件单片机对I2C总线上各从器件的读、写操作。

1．典型信号模拟

为保证数据传送的可靠性，标准I2C总线数据传送有严格的时序要求。I2C总线的起始信号、终止信号、应答/数据“0”及非应答/数据“1”的模拟时序见图10-12～图10-17。

对于终止信号，要保证有大于4.7μs信号建立时间。终止信号结束时，要释放总线，使SDA、SCL维持在高电平上，在大于4.7μs后才可进行第1次起始操作。在单主器件系统中，为防止非正常传送，终止信号后SCL可设置在低电平。

对发送应答位、非应答位来说，与发送数据“0”和“1”的信号定时要求完全相同。只要满足在时钟高电平大于4.0μs期间，SDA线上有确定的电平状态即可。

2．典型信号及字节收发的模拟子程序

AT89S51单片机在模拟I2C总线通信时，需编写以下5个函数：总线初始化、起始信号、终止信号、应答/数据“0”以及非应答/数据“1”函数。

（1）总线初始化函数。初始化函数的功能是将SCL和SDA总线拉高以释放总线。参考程序如下：

```
#include <reg51.h>
#include <intrins.h>          //包含函数_nop_（）的头文件
sbit  sda=P1^0;               //定义I2C模拟数据传送位
sbit  scl=P1^1;               //定义I2C模拟时钟控制位
```



```
void init( )                  //总线初始化函数
{
    scl=1;                    //scl为高电平
    _nop_ ( );                //延时约1μs
    sda=1;                    //sda为高电平
    delay5us();               //延时约5μs
}
```

（2）起始信号S函数。图10-12的起始信号S，要求一个新的起始信号前总线的空闲时间大于4.7μs，而对于一个重复的起始信号，要求建立时间也须大于4.7μs。图10-12为起始信号的时序波形在SCL高电平期间SDA发生负跳变。起始信号到第1个时钟脉冲负跳沿的时间间隔应大于4μs。

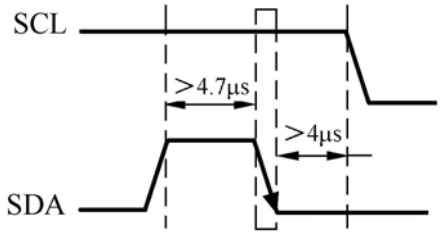

图10-12　起始信号S的模拟时序

起始信号S的函数如下：

```
void start(void)                    //起始信号函数
{
   scl=1;
   sda=1;
   delay5us();
   sda=0;
   delay4us();
   scl=0;
}
```


（3）终止信号P函数。图10-13为终止信号P的时序波形。在SCL高电平期间SDA的一个上升沿产生终止信号。

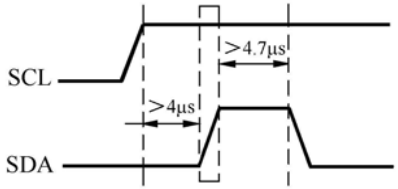

图10-13　终止信号P的模拟时序

终止信号函数如下：

```
void stop(void)              //终止信号函数
{
scl=0;
sda=0;
delay4us();
scl=1;
delay4us();
sda=1;
delay5us();
sda=0;
}
```

（4）应答位函数。发送接收应答位与发送数据“0”相同，即在SDA低电平期间SCL发生一个正脉冲，产生图10-14的模拟时序。

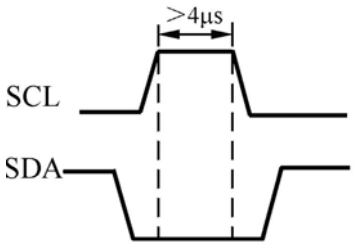

图10-14　发送应答位的模拟时序

发送接收应答位的函数如下：

```
void Ack(void )
{
    uchar i;
    sda=0;
    scl=1;
    delay4us();
    while((sda==1)&&(i<255))i++;
    scl=0;
    delay4us();
}
```


SCL在高电平期间，SDA被从器件拉为低表示应答。命令行中的（SDA = 1）和（i<255））相与，表示若在这一段时间内没有收到从器件的应答，则主器件默认从器件已收到数据而不再等待应答信号，要是不加该延时退出，一旦从器件没有发应答信号，程序将永远停在这里，而在实际中是不允许这种情况发生。

（5）非应答位/数据“1”函数。发送非应答位与发送数据“1”相同，即在SDA高电平期间SCL发生一个正脉冲，产生图10-15的模拟时序。

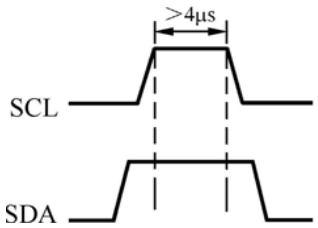

图10-15　非应答位/数据“1”的模拟时序

发送非接收应答位/数据“1”的函数如下：

```
void NoAck(void )
{
sda=1;
scl=1;
delay4us();
scl=0;
sda=0;
}
```

3．字节收发的子程序

除上述典型信号模拟外，在I2C总线数据传送中，经常使用单字节数据的发送与接收。

（1）发送1字节数据子程序

下面是模拟I2C数据线由SDA发送1字节数据（可以是地址，也可是数据），发送完后等待应答，并对状态位ack进行操作，即应答或非应答都使ack=0。发送数据正常ack=1，从器件无应答或损坏，则ack=0。发送1字节数据参考程序如下：

```
void SendByte(uchar data)
      {
      uchar i,temp;
      temp=data;
      for(i=0; i <8; i++)
            {
            temp= temp<<1;          //左移一位
            scl=0;
```



```
      delay4us();
      sda=Cy;
      delay4us();
      scl=1;
      delay4us();
      }
scl=0;
delay4us();
sda=1;
delay4us();
```


串行发送1字节时，需把该字节8位逐位发出，“temp=temp<<1;”就是将temp中的内容左移1位，最高位将移入Cy位中，然后将Cy赋给SDA，进而在SCL的控制下发送出去。

（2）接收1字节数据子程序

下面是模拟从I2C的数据线SDA接收从器件传来的1字节数据的子程序：

```
void rcvbyte()
{
      uchar i,temp;
      scl=0;
      delay4us();
      sda=1;
```

```
        for(i=0; i <8; i++)
        {
                scl=1;
                delay4us();
                temp=（temp<<1）| sda;
                scl=0;
                delay4us();
        }
        delay4us();
        return temp;
}
```

同理，串行接收1字节时，需将8位一位一位地接收，然后再组合成1个字节。“temp =（temp<<1）| SDA；”是将变量temp左移1位后与SDA进行逻辑“或”运算，依次把8位数据组合成1字节来完成接收。

11.3.5 利用I2C总线扩展E2PROM AT24C02的IC卡设计

通用存储器IC卡由通用存储器芯片封装而成，由于结构和功能简单，成本低、使用方便，因此在各个领域都得到了广泛的应用。

目前用于IC卡的通用存储器芯片多为E2PROM，且采用I2C总线接口，典型代表为ATMEL公司的I2C接口的AT24Cxx系列。该系列具有AT24C01/02/04/08/16等型号，它们封装形式、引脚及内部结构类似，只是容量不同，分别为128B/256B/512B/1KB/2KB。下面以AT24C02为例，介绍单片机通过I2C总线对AT24C02/进行读写，即IC卡的制作。

1．AT24C02芯片简介

（1）封装与引脚

AT24C02的封装形式有双列直插（DIP）8脚式和贴片8脚式两种，无论何种封装，其引脚功能都是一样的。AT24C02的DIP形式引脚如图10-16所示，引脚功能见表10-1。

（2）存储单元的寻址

AT24C02存储容量为256B，分为32页，每页8B。有两种寻址方式：芯片寻址和片内子地址寻址。

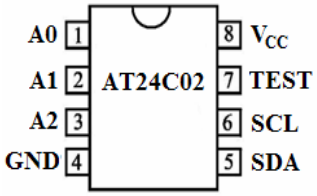

图10-16 AT24C02的DIP引脚

表10-1 AT24C02的引脚功能

引脚	名称	功　能
1～3	A0、A1、A2	可编程地址输入端
4	GND	电源地
5	SDA	串行数据输入/输出端
6	SCL	串行时钟输入端
7	TEST	硬件写保护控制引脚，当 TEST=0，正常进行读/写操作。TEST=1，对部分存储区域只能读，不能写（写保护）。
8	V_{CC}	+ 5V 电源

① 芯片寻址。AT24C02芯片地址固定为1 010，它是I2C总线器件的特征编码，其地址控制字的格式为1 010 A2A1A0 R/W* 。A2A1A0引脚接高、低电平后得到确定的3位编码，与1 010形成7位编码，即为该器件的地址码。由于A2A1A0共有8种组合，故系统最多可外接8片AT24C02，R/W*是对芯片的读/写控制位。

② 片内子地址寻址。在确定了AT24C02芯片的7位地址码后，片内的存储空间可用1字节的地址码进行寻址，寻址范围为00H～FFH，即可对片内的256个单元进行读/写操作。

（3）写操作

AT24C02有两种写入方式，字节写入与页写入。

① 字节写入方式。单片机（主器件）先发送启动信号和1字节的控制字，从器件发出应答信号后，单片机再发送1字节的存储单元子地址（AT24C02内部单元的地址码），单片机收到AT24C02应答后，再发送8位数据和1位终止信号。

② 页写入方式。单片机先发送启动信号和1字节控制字，再发送1字节存储器起始单元地址，上述几个字节都得到AT24C02应答后，就可发送最多1页的数据，并顺序存放在已指定的起始地址开始的相继单元，最后以终止信号结束。

（4）读操作

AT24C02读操作也有两种方式，即指定地址读和指定地址连续读。

① 指定地址读方式。单片机发送启动信号后，先发送含有芯片地址的写操作控制字，AT24C02应答后，单片机再发送1字节的指定单元地址，AT24C02应答后再发送1个含有芯片地址读操作控制字，此时如AT24C02应答，被访问单元的数据就会按SCL信号同步出现在SDA线上，供单片机读取。

② 指定地址连续读方式。指定地址连续读方式是单片机收到每个字节数据后要做出应答，只有AT24C02检测到应答信号

后，其内部的地址寄存器就自动加1指向下一个单元，并顺序将指向单元的数据送到SDA线上。当需要结束读操作时，单片机接收到数据后，在需要应答的时刻发送一个非应答信号，接着再发送一个终止信号即可。

【例10-2】单片机通过I2C串行总线扩展1片AT24C02，实现单片机对存储器AT24C02的读、写。由于Proteus元件库中没有AT24C02，可用FM24C02芯片代替，即在Proteus中“关键字”对话框元件查找栏中输入“24C02”，就会在左侧的元件列表中显示，然后在元件列表中选择即可。

AT89S51与FM24C02的接口原理电路见图10-17。

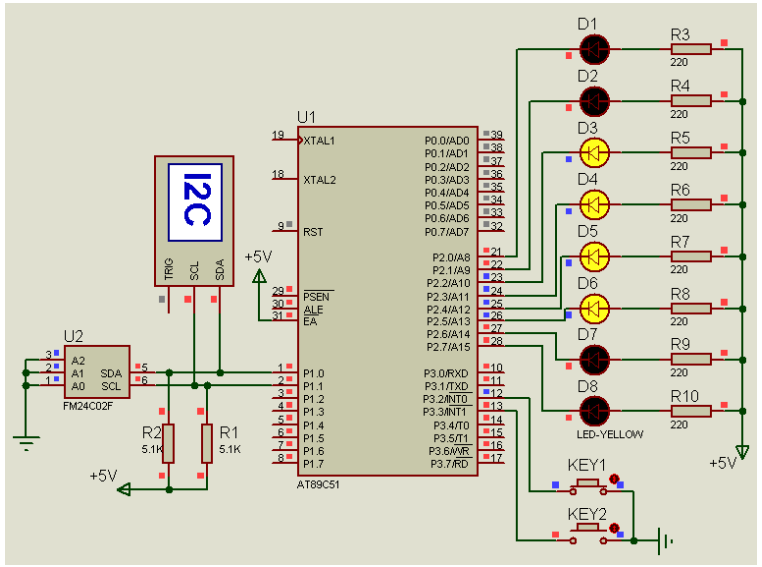

图10-17　AT89S51与AT24C02接口的原理电路

图中KEY1作为外部中断0的中断源，当按下KEY1，单片机通过I2C总线发送数据0xaa给AT24C02，等发送数据完毕后，将0xaa送P2口通过LED显示。

KEY2作为外部中断1的中断源，当按下KEY2时，单片机通过I2C总线读AT24C02，等读数据完毕后，将读出的最后数据0x55送P2口的LED显示出来。

最终显示的仿真效果是：按下KEY1，标号为D1～D8的8个LED中D3、D4、D5、D6灯亮，如图10-17所示。按下KEY2，则D1、D3、D5、D7灯亮。

Proteus提供的I2C调试器是调试I2C系统的得力工具，使用I2C调试器的观测窗口可观察I2C总线上的数据流，查看I2C总线发送的数据，也可作为从器件向I2C总线发送数据。

在原理电路中添加I2C调试器具体操作：先单击图4-2左侧工具箱中的虚拟仪器图标，此时在预览窗口中显示出各种虚拟仪器选项，单击“I2C DEBUGGER”项，并在原理编辑窗口单击鼠标左键，就会出现I2C调试器符号，如图10-18所示。然后把I2C调试器的“SDA”端和“SCL” 端分别连接在I2C总线的“SDA”和“SCL”线上。

仿真运行时，用鼠标右键单击I2C调试器符号，出现下拉菜单，单击“Terminal”选项，即可出现I2C调试器的观测窗口，如图10-18所示。从观测窗口上可看到按一下KEY1时，出现在I2C总线上的数据流。

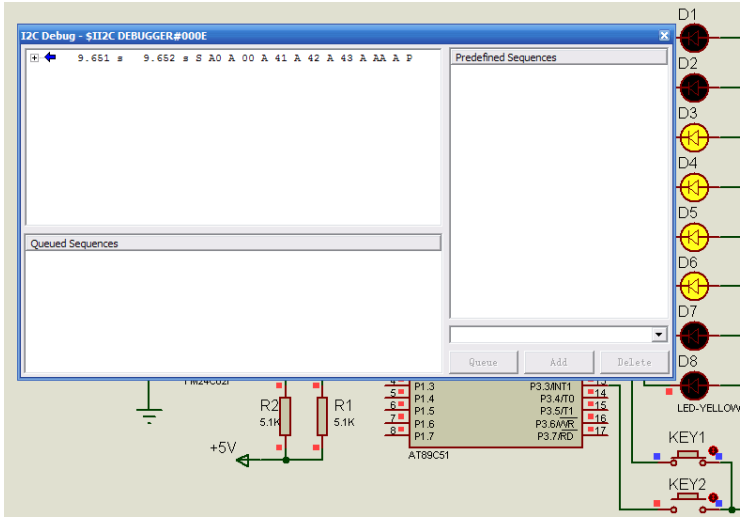

图10-18 I2C调试器的观测窗口

本例参考程序：

```
#include "reg51.h"
#include "intrins.h"                    //包含有函数_nop_()的头文件
#define uchar unsigned char
#define uint unsigned int
#define out P2                          //发送缓冲区的首地址
sbit scl=P1^1;
sbit sda=P1^0;
sbit key1=P3^2;
sbit key2=P3^3;
uchar data mem[4]_at_ 0x55;             //发送缓冲区的首地址
uchar mem[4]={0x41,0x42,0x43,0xaa};  //欲发送的数据数组
uchar data rec_mem[4] _at_ 0x60 ;       //接收缓冲区的首地址
void start(void);                               //起始信号函数
```

```
void stop(void);                                //终止信号函数
void sack(void);                        //发送应答信号函数
bit rack(void);                                 //接收应答信号函数
void ackn(void);                        //发送无应答信号函数
void send_byte(uchar);                          //发送一个字节函数
uchar rec_byte(void);                           //接收一个字节函数
void write(void);                               //写一组数据函数
void read(void);                                //读一组数据函数
void delay4us(void);                            //延时4μs
void main(void)                 //主函数
{
EA=1;EX0=1;EX1=1;               //总中断开，外中断0与外中断0允许中断
while(1);
}
void ext0()interrupt 0          //外中断0中断函数
{
write();                        //调用写数据函数
```

```
}
void ext1()interrupt 2          //外中断1中断函数
{
read();                         //调用读数据函数
}
void read(void)                 //读数据函数
{
uchar i;
bit f;
start();                        //起始函数
send_byte(0xa0);                //发从机的地址
f=rack();               //接收应答
if(!f)
{
start();                        //起始信号
send_byte(0xa0);
f=rack();
send_byte(0x00);                //设置要读取从器件的片内地址
f=rack();
```

```
if(!f)
{
        or(i=0;i<3;i++)
        {
                rec_mem[i]=rec_byte();
                sack();
        }
        rec_mem[3]=rec_byte();ackn();
        }
    }
}
stop();out=rec_mem[3];while(!key2);
}
void write(void)                                    //写数据函数
{
uchar i;
bit f;
start();
```

```
send_byte(0xa0);
f=rack();-
if(!f){
        send_byte(0x00);
        f=rack();
if(!f){
    for(i=0;i<4;i++)
     {
        send_byte(mem[i]);
        f=rack();
        if(f)break;
    }
      }
  }
stop();out=0xc3;while(!key1);
}
```

```
   void start(void)                                 //起始信号
   {
   scl=1;
   sda=1;
   delay4us();
   sda=0;
   delay4us();
   scl=0;
   }
   void stop(void)                                  //终止信号
   {
   scl=0;
   sda=0;
   delay4us();
   scl=1;
   delay4us();
```

```
sda=1;
delay5us();
sda=0;
}
bit rack(void)                                      //接收一个应答位
{
bit flag;
scl=1;
delay4us();
flag=sda;
scl=0;
return(flag);
}
void sack(void)                                     //发送接收应答位
{
sda=0;
delay4us();
scl=1;
```

```
delay4us();
scl=0;
delay4us();
sda=1;
delay4us();
}
void ackn(void)                          //发送非接收应答位
{
sda=1;
delay4us();
scl=1;
delay4us();
scl=0;
delay4us();
sda=0;
```



```
}
uchar rec_byte(void)                     //接收一个字节
{
uchar i,temp;
for(i=0;i<8;i++)
{
        temp<<=1;
        scl=1;
delay4us();
                temp|=sda;
                scl=0;
                delay4us();
}
return(temp);
}
void send_byte(uchar temp)               //发送一个字节
```



```
{
uchar i;
scl=0;
for(i=0;i<8;i++)
{
        sda=(bit)(temp&0x80);
        scl=1;
        delay4us();
        scl=0;
        temp<<=1;
}
sda=1;
}
void delay4us(void)                      //延时4μs
{
_nop_();_nop_();_nop_();_nop_();
}
```