LM75A 应用范例

1. LM75A 概述

LM75A 是一个高速 I^2C 接口的温度传感器，可以在-55℃～+125℃的温度范围内将温度直接转换为数字信号，并可实现 0.125℃的精度。MCU 可以通过 I^2C 总线直接读取其内部寄存器中的数据，并可通过 I^2C 对 4 个数据寄存器进行操作，以设置成不同的工作模式。LM75A 有 3 个可选的逻辑地址管脚，使得同一总线上可同时连接 8 个器件而不发生地址冲突。

LM75A 可配置成不同的工作模式。它可设置成在正常工作模式下周期性地对环境温度进行监控，或进入关断模式来将器件功耗降至最低。OS 输出有 2 种可选的工作模式：OS 比较器模式和 OS 中断模式，OS 输出可选择高电平或低电平有效。

正常工作模式下，当器件上电时，OS 工作在比较器模式，温度阈值为 80℃，滞后阈值为 75℃。

1.1 LM75A 管脚描述

LM75A 的管脚描述见图 1。

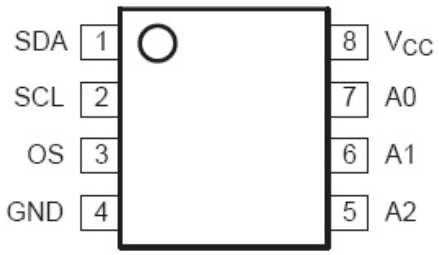


图 1 LM75A 管脚描述

SDA： I^2C 串行双向数据线，开漏口。
SCL： I^2C 串行时钟输入，开漏口。
OS： 过热关断输出。开漏输出。
GND： 地，连接到系统地。
A2： 用户定义的地址位 2。
A1： 用户定义的地址位 1。
A0： 用户定义的地址位 0。
V_{CC}： 电源。

1.2 LM75A 的功能特点

- 提供环境温度对应的数字信息，直接表示温度；
- 可以对某个特定温度作出反应，可以配置成中断或者比较器模式（OS 输出）；
- 高速 I^2C 总线接口，有 A2-A0 地址线，一条总线上最多可同时使用 8 个 LM75A；
- 低功耗设计，工作电流典型值为 250uA，掉电模式为 3.5uA；
- 测量的温度最大范围为-55℃～+125℃；
- 宽工作电压范围：2.8V～5.5V；
- 提供了良好的温度精度（0.125℃）；
- 可编程温度阈值和滞后设定点。

1.3 LM75A 内部寄存器

- **温度寄存器 Temp（地址 0x00）**

温度寄存器是一个只读寄存器，包含2个8位的数据字节，由一个高数据字节（MS）和一个低数据字节（LS）组成。这两个字节中只有11位用来存放分辨率为0.125℃的Temp数据（以二进制补码数据的形式），如表 1所示。对于8位的I^2C总线来说，只要从LM75A的“00地址”连续读两个字节即可（温度的高8位在前）。

表 1 温度寄存器

Temp MS 字节								Temp LS 字节							
MSB							LSB	MSB							LSB
B7	B6	B5	B4	B3	B2	B1	B0	B7	B6	B5	B4	B3	B2	B1	B0
Temp 数据（11 位）											未使用				
MSB										LSB					
D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X

根据11位的Temp数据来计算Temp值的方法：

1. 若D10＝0，温度值（℃）＝＋Temp数据）×0.125℃；
2. 若D10＝1，温度值（℃）＝－Temp数据的二进制补码）×0.125℃。

表 2给出了一些Temp数据和温度值的例子。

表 2 Temp 表

Temp 数据			温度值
11 位二进制数（补码）	3 位十六进制值	十进制值	℃
0111 1111 000	3F8h	1016	+127.000℃
0111 1110 111	3F7h	1015	+126.875℃
0111 1110 001	3F1h	1009	+126.125℃
0111 1101 000	3E8h	1000	+125.000℃
0001 1001 000	0C8h	200	+25.000℃
0000 0000 001	001h	1	+0.125℃
0000 0000 000	00h	0	0.000℃
1111 1111 111	7FFh	-1	-0.125℃
1110 0111 000	738h	-200	-25.000℃
1100 1001 001	649h	-439	-54.875℃
1100 1001 000	648h	-440	-55.000℃

- **配置寄存器（地址 0x01）**

配置寄存器为 8 位可读写寄存器，其位功能分配如表 3 所示。

表 3 配置寄存器位功能

B7	B6	B5	B4	B3	B2	B1	B0
保留			OS 故障队列		OS 极性	OS 比较/中断	关断

D7-D5： 保留，默认为0。

D4-D3： 用来编程OS故障队列。

00到11代表的值为1、2、4、6，默认值为0。

D2： 用来选择OS极性。

D2=0，OS低电平有效（默认）；

D2=1，OS高电平有效。

D1： 选择 OS 工作模式。

D1=0，配置成比较器模式，直接控制外围电路；

D1=1，OS 控制输出功能配置成中断模式，以通知 MCU 进行相应处理。

D0： 选择器件工作模式。

D0=0，LM75A 处于正常工作模式（默认）；

D0=1，LM75A 进入关断模式。

- **滞后寄存器 Thyst（0x02）**

滞后寄存器是读/写寄存器，也称为设定点寄存器，提供了温度控制范围的下限温度。每次转换结束后，Temp 数据（取其高 9 位）将会与存放在该寄存器中的数据相比较，当环境温度低于此温度的时候，LM75A 将根据当前模式（比较、中断）控制 OS 引脚作出相应反应。

该寄存器都包含 2 个 8 位的数据字节，但 2 个字节中，只有 9 位用来存储设定点数据（分辨率为 0.5℃的二进制补码），其数据格式如表 4 所示，默认为 75℃。

表 4 低/高报警温度寄存器数据格式

D15	D14---D8							D7	D6---D0
T8	T7	T6	T5	T4	T3	T2	T1	T0	未定义

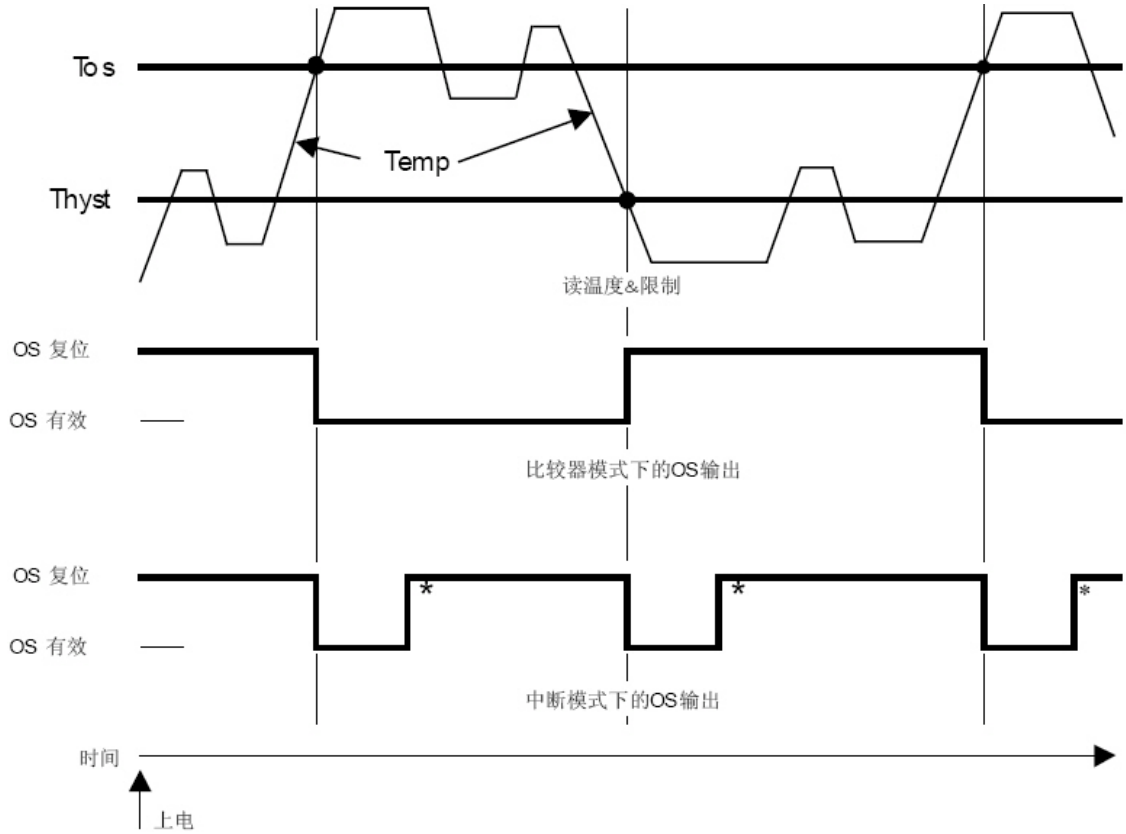
- **过温关断阈值寄存器 Tos（0x03）**

过温关断寄存器提供了温度控制范围的上限温度。每次转换结束后，Temp 数据（取其高 9 位）将会与存放在该寄存器中的数据相比较，当环境温度高于此温度的时候，LM75A 将根据当前模式（比较、中断）控制 OS 引脚作出相应反应。其数据格式如表 4 所示，默认为 80℃。

1.4 OS 输出

OS输出为开漏输出口。为了观察到这个输出的状态，需要接一个外部上拉电阻，其阻值应当足够大（高达200kΩ），以减少温度读取误差。OS输出可通过编程配置寄存器的B2位设置为高或低有效。

如图 2 所示，为 LM75A 在不同模式下 OS 引脚对温度作出的响应。OS 设为低有效。



*＝OS 可通过读寄存器或使器件进入关断状态来复位。

假设故障队列在每个Tos和Thyst交叉点处都被满足。

图 2 LM75A 温度响应

可以看出，当 LM75A 工作在比较器模式时，当温度高于 Tos 时，OS 输出低电平。此时采取了降温措施，启动降温设备（如风扇），直到温度再降到 Thyst，则停止降温，因此在这种模式下，LM75A 可以直接控制外部电路来保持环境温度；而在中断模式，则在温度高于 Tos 或低于 Thyst 时产生中断。

注意：在中断模式下，只有当 MCU 对 LM75A 进行读操作后，其中断信号才会消失（图中 OS 变为高电平）。

1.5 I^2C 串行接口

在主控器的控制下，LM75A可以通过SCL和SDA作为从器件连接到I^2C总线上。主控器必须提供SCL时钟信号，可以通过SDA读出器件数据或将数据写入到器件中。注意：必须在SCL和SDA端分别连接一个外部上拉电阻，阻值大约为10kΩ。

LM75A从地址（7位地址）的低3位可由地址引脚A2、A1和A0的逻辑电平来决定。地址的高4位预先设置为‘1001’。表 5给出了器件的完整地址，从表中可以看出，同一总线上可连接8个器件而不会产生地址冲突。

由于输入管脚SCL、SDA、A2-A0内部无偏置，因此在任何应用中它们都不能悬空（这一点很重要）。

表 5 地址表

MSB						LSB
1	0	0	1	A2	A1	A0

2. 使用 TKS-58B 仿真器做 LM75A 温度采集实验

2.1 实验内容简介

利用 TKS-58B 仿真标准 51 内核单片机，使用模拟 I^2C 总线对 LM75A 温度寄存器进行读操作，通过仿真器提供的变量观察窗口的得到当前环境温度数据。

2.2 实验硬件介绍

LM75A 工作电压为 3.0V～5.5V，采用 5.0V 直接供电。设计中只采用了一个 LM75A，所以 I^2C 的地址线 A0-A2 接地即可，其 I^2C 地址为 0x90。由于 SDA、SCL 为开漏口，所以分别需要增加一个 4.7K～10K 的上拉电阻，本实验中取 8.2K。

注：由于使用 TKS-58B 仿真 MCU，仿真器内部提供了保证芯片运行的单片机最小系统，因此在此不作赘述。

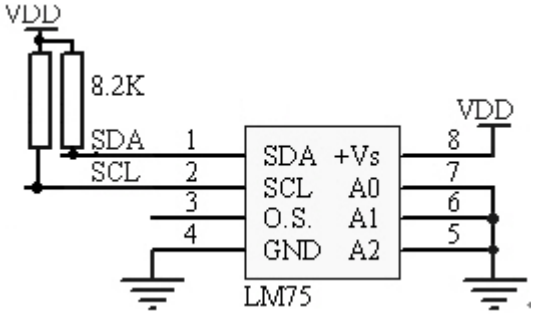


图 3 温度测量电路

2.3 实验参考例程

本程序主要实现的功能：使用模拟 I^2C 总线方式读温度寄存器（子地址为 0x00）的值。程序中调用了标准 51 的模拟 I^2C 软件包，主要使用 IRcvStr（向有子地址器件读取多字节数据）函数，读取 LM75A 温度寄存器的数据。

实验参考例程见程序清单 1。

程序清单 1 LM75A 温度测量例程

```
//*************************************
//File:       LM75A.c
//功能:       读 LM75A 采集的温度值
//*************************************
#include    "VIIC_C51.h"
#include    <intrins.h>
#define     uchar unsigned char

uchar       data LM75A=0x90;                    // LM75A 的 I2C 地址
uchar       data MRD[2]={0};                    //接收缓冲区

void  main(void)
{
    while(1)
```

```
    {
        IRcvStr(LM75A,0x00,MRD,2);          //读取 LM75A 当前温度值
    }
}
```

标准 51 单片机模拟 I^2C 软件包请参见附录中的程序清单 2。

2.4 观察实验结果

工程编译通过后，使用 TKS-58B 对其进行硬件仿真，仿真界面如图 4 所示。

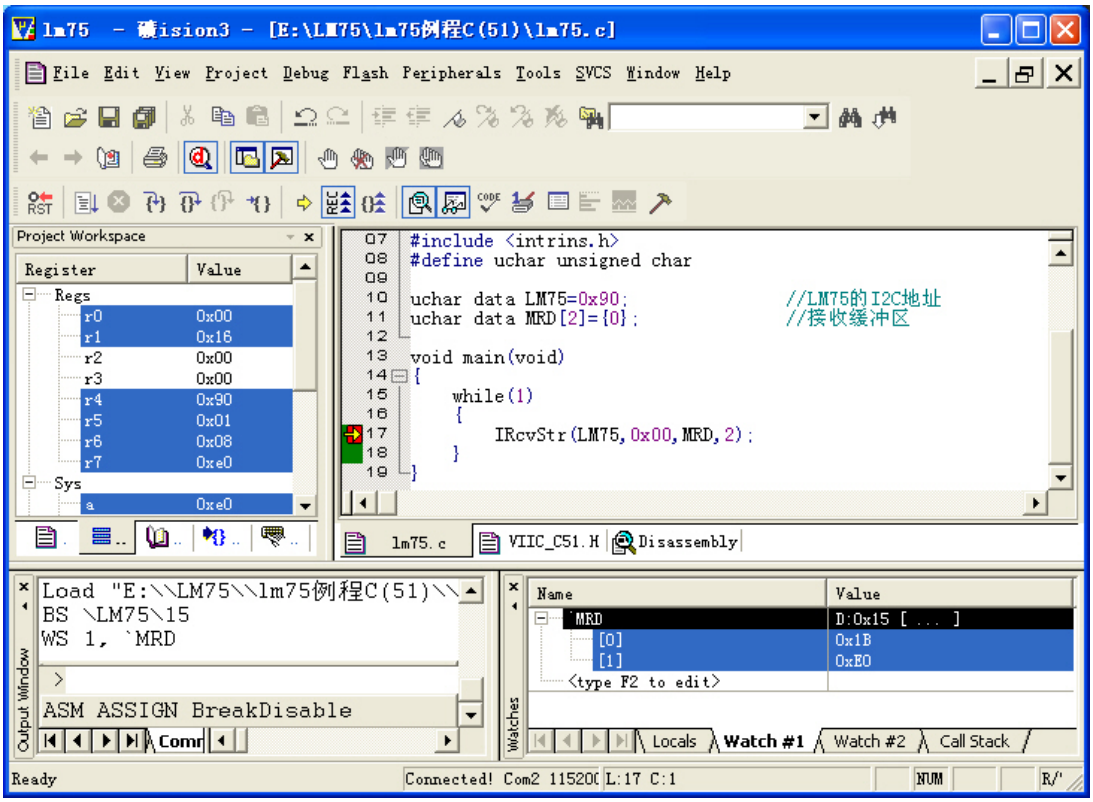


图 4　仿真界面

用户可通过选择菜单“View → Watch & Call Stack Window”打开变量观察窗口，并将 MRD 添加到 Watch #1 中。运行程序到断点处，可观察到 2 次读取到的温度寄存器的值，如图 5 所示。

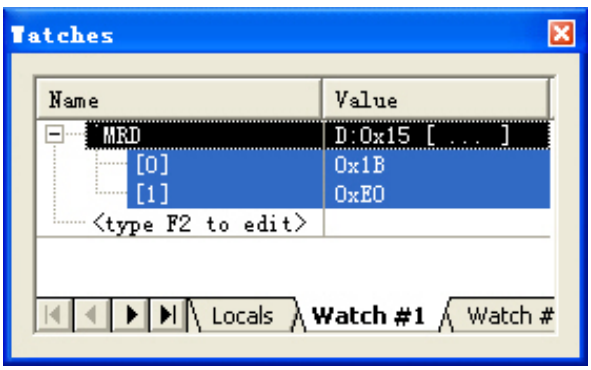


图 5　变量观察窗口

由图 5，MRD[0]为 0x1b，MRD[1]为 0xe0，对照表 1 及其相应计算公式，可很容易地计算出当前温度为+27.875℃。

注：

LM75A 其它工作模式的操作，与本实验类似，用户可参考相关数据手册《LM75A 数字温度传感器和温度监控器》，同样可在 www.zlgmcu.com 网站中下载得到。

附录：

程序清单 2 模拟 I^2C 软件包

```
/*********************************************************************
                          VIIC_C51.h
  此程序是 I2C 操作平台（主方式的软件平台）的底层的 C 子程序,如发送数据
及接收数据,应答位发送,并提供了几个直接面对器件的操作函数，它很方便的
与用户程序连接并扩展.....
      注意:函数是采用软件延时的方法产生 SCL 脉冲,固对高晶振频率要作
一定的修改....(本例是 1us 机器周期,即晶振频率要小于 12MHZ)
*********************************************************************/
#include <reg764.h>                    /*头文件的包含*/
#include <intrins.h>

#define   uchar unsigned char          /*宏定义*/
#define   uint   unsigned int

#define   _Nop()   _nop_()             /*定义空指令*/

sbit SDA=P1^3;                         /*模拟 I2C 数据传送位*/
sbit SCL=P1^2;                         /*模拟 I2C 时钟控制位*/
bit ack;                               /*应答标志位*/

/*********************************************************************
                          起动总线函数
函数原型: void   Start_I2c();
功能:          启动 I2C 总线,即发送 I2C 起始条件.
*********************************************************************/
void Start_I2c()
{
  SDA=1;                               /*发送起始条件的数据信号*/
  _Nop();
  SCL=1;
  _Nop();                              /*起始条件建立时间大于 4.7us,延时*/
  _Nop();
```

```
  _Nop();
  _Nop();
  _Nop();
  SDA=0;                                /*发送起始信号*/
  _Nop();                               /* 起始条件锁定时间大于 4μs*/
  _Nop();
  _Nop();
  _Nop();
  _Nop();
  SCL=0;                                /*钳住 I2C 总线，准备发送或接收数据 */
  _Nop();
  _Nop();
}

/*****************************************************************
                        结束总线函数
函数原型: void   Stop_I2c();
功能:          结束 I2C 总线,即发送 I2C 结束条件.
*****************************************************************/
void Stop_I2c()
{
  SDA=0;                                /*发送结束条件的数据信号*/
  _Nop();                               /*发送结束条件的时钟信号*/
  SCL=1;                                /*结束条件建立时间大于 4μs*/
  _Nop();
  _Nop();
  _Nop();
  _Nop();
  _Nop();
  SDA=1;                                /*发送 I2C 总线结束信号*/
  _Nop();
  _Nop();
  _Nop();
  _Nop();
}

/*****************************************************************
                    字节数据传送函数
函数原型: void   SendByte(uchar c);
功能:   将数据 c 发送出去,可以是地址,也可以是数据,发完后等待应答,并对
      此状态位进行操作.(不应答或非应答都使 ack=0  假)
      发送数据正常，ack=1; ack=0 表示被控器无应答或损坏。
*****************************************************************/
void   SendByte(uchar c)
```

```
{
 uchar BitCnt;

 for(BitCnt=0;BitCnt<8;BitCnt++)          /*要传送的数据长度为 8 位*/
     {
      if((c<<BitCnt)&0x80)SDA=1;          /*判断发送位*/
         else   SDA=0;
      _Nop();
      SCL=1;                              /*置时钟线为高，通知被控器开始接收数据位*/
       _Nop();
       _Nop();                            /*保证时钟高电平周期大于 4μs*/
       _Nop();
       _Nop();
       _Nop();
      SCL=0;
     }

     _Nop();
     _Nop();
     SDA=1;                              /*8 位发送完后释放数据线，准备接收应答位*/
     _Nop();
     _Nop();
     SCL=1;
     _Nop();
     _Nop();
     _Nop();
     if(SDA==1)ack=0;
         else ack=1;                     /*判断是否接收到应答信号*/
     SCL=0;
     _Nop();
     _Nop();
}

/*****************************************************************
                      字节数据传送函数
函数原型: uchar   RcvByte();
功能:   用来接收从器件传来的数据,并判断总线错误(不发应答信号)，
       发完后请用应答函数。
*****************************************************************/
uchar   RcvByte()
{
  uchar retc;
  uchar BitCnt;
```

```
  retc=0;
  SDA=1;                                  /*置数据线为输入方式*/
  for(BitCnt=0;BitCnt<8;BitCnt++)
       {
         _Nop();
         SCL=0;                           /*置时钟线为低，准备接收数据位*/
         _Nop();
         _Nop();                          /*时钟低电平周期大于 4.7μs*/
         _Nop();
         _Nop();
         _Nop();
         SCL=1;                           /*置时钟线为高使数据线上数据有效*/
         _Nop();
         _Nop();
         retc=retc<<1;
         if(SDA==1)retc=retc+1;           /*读数据位,接收的数据位放入 retc 中  */
         _Nop();
         _Nop();
       }
  SCL=0;
  _Nop();
  _Nop();
  return(retc);
}

/*****************************************************************
                        应答子函数
原型:   void Ack_I2c(bit a);

功能:主控器进行应答信号,(可以是应答或非应答信号)
*****************************************************************/
void Ack_I2c(bit a)
{

  if(a==0)SDA=0;                          /*在此发出应答或非应答信号  */
          else SDA=1;
  _Nop();
  _Nop();
  _Nop();
  SCL=1;
    _Nop();
    _Nop();                               /*时钟低电平周期大于 4μs*/
    _Nop();
    _Nop();
```

```
    _Nop();
 SCL=0;                                  /*清时钟线，钳住 I2C 总线以便继续接收*/
    _Nop();
    _Nop();
}

/*****************************************************************
                        向无子地址器件发送字节数据函数
函数原型: bit   ISendByte(uchar sla,ucahr c);
功能:       从启动总线到发送地址，数据，结束总线的全过程,从器件地址 sla.
             如果返回 1 表示操作成功，否则操作有误。
注意：      使用前必须已结束总线。
*****************************************************************/
bit ISendByte(uchar sla,uchar c)
{
    Start_I2c();                        /*启动总线*/
    SendByte(sla);                      /*发送器件地址*/
      if(ack==0)return(0);
    SendByte(c);                        /*发送数据*/
      if(ack==0)return(0);
  Stop_I2c();                           /*结束总线*/
  return(1);
}

/*****************************************************************
                        向有子地址器件发送多字节数据函数
函数原型: bit   ISendStr(uchar sla,uchar suba,ucahr *s,uchar no);
功能:       从启动总线到发送地址，子地址,数据，结束总线的全过程,从器件
            地址 sla，子地址 suba，发送内容是 s 指向的内容，发送 no 个字节。
             如果返回 1 表示操作成功，否则操作有误。
注意：      使用前必须已结束总线。
*****************************************************************/
bit ISendStr(uchar sla,uchar suba,uchar *s,uchar no)
{
    uchar i;

    Start_I2c();                        /*启动总线*/
    SendByte(sla);                      /*发送器件地址*/
      if(ack==0)return(0);
    SendByte(suba);                     /*发送器件子地址*/
      if(ack==0)return(0);

    for(i=0;i<no;i++)
     {
```

```
        SendByte(*s);                       /*发送数据*/
          if(ack==0)return(0);
        s++;
      }
  Stop_I2c();                               /*结束总线*/
   return(1);
}

/*****************************************************************
                        向无子地址器件读字节数据函数
函数原型: bit   IRcvByte(uchar sla,ucahr *c);
功能:       从启动总线到发送地址，读数据，结束总线的全过程,从器件地
            址 sla，返回值在 c.
              如果返回 1 表示操作成功，否则操作有误。
注意：      使用前必须已结束总线。
*****************************************************************/
bit IRcvByte(uchar sla,uchar *c)
{
    Start_I2c();                            /*启动总线*/
    SendByte(sla+1);                        /*发送器件地址*/
      if(ack==0)return(0);
    *c=RcvByte();                           /*读取数据*/
      Ack_I2c(1);                           /*发送非就答位*/
   Stop_I2c();                              /*结束总线*/
   return(1);
}

/*****************************************************************
                        向有子地址器件读取多字节数据函数
函数原型: bit   ISendStr(uchar sla,uchar suba,ucahr *s,uchar no);
功能:       从启动总线到发送地址，子地址,读数据，结束总线的全过程,从器件
            地址 sla，子地址 suba，读出的内容放入 s 指向的存储区，读 no 个字节。
              如果返回 1 表示操作成功，否则操作有误。
注意：      使用前必须已结束总线。
*****************************************************************/
bit IRcvStr(uchar sla,uchar suba,uchar *s,uchar no)
{
    uchar i;

    Start_I2c();                            /*启动总线*/
    SendByte(sla);                          /*发送器件地址*/
      if(ack==0)return(0);
    SendByte(suba);                         /*发送器件子地址*/
      if(ack==0)return(0);
```

```
    Start_I2c();
    SendByte(sla+1);
        if(ack==0)return(0);

    for(i=0;i<no-1;i++)
      {
       *s=RcvByte();                    /*发送数据*/
        Ack_I2c(0);                     /*发送就答位*/
       s++;
      }
    *s=RcvByte();
      Ack_I2c(1);                       /*发送非应位*/
  Stop_I2c();                           /*结束总线*/
   return(1);
}
                                        /*      完毕          */
```