单片机控制与应用实验报告

（提交电子版即可，姓名和学号一定要写）

姓名：王家寅
学号：21160833

实验五 重量测量（液晶显示）

原理总结

（该实验涉及的基本原理及其在实验中的使用方法）

该实验的大体流程可以分为如下几步:

1. 通过传感器元件获得重量的模拟信号（电压），经过 ADC0803 转换成数字信号
2. 在后台程序进行数字信号的逻辑处理，找出数字信号与重量的换算关系式以及调零方法
3. 将数据显示到 64*64*2 的液晶显示屏上。
4.

其中，关于液晶屏的一些参数设置及处理：

5.3 软件功能说明

5.3.1 指令表

指令名称	控制信号		控制代码							
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
显示开关设置	0	0	0	0	1	1	1	1	1	D
显示起始行设置	0	0	1	1	L5	L4	L3	L2	L1	L0
页面地址设置	0	0	1	0	1	1	1	P2	P1	P0

列地址设置	0	0	0	1	C5	C4	C3	C2	C1	C0
读取状态字	0	1	BUSY	0	ON/OFF	RESET	0	0	0	0
写显示数据	1	0	数据							
读显示数据	1	1	数据							

5.3.2 控制时序表

CS1	CS2	RS	R/W	E	DB7~DB0	功能
X	X	X	X	0	高阻	总线释放
1	1	0	0	下降沿	输入	写指令代码
1	1	0	1	1	输出	读状态字
1	1	1	0	下降沿	输入	写显示数据
1	1	1	1	1	输出	读显示数据

关于 ADC 的一些控制字:

ADC_POWER：ADC 电源控制位，0 关 1 开。

SPEED1,SPEED0：模数转换器速度控制位，控制 A/D 转换所需时间。

ADC_FLAG：模数转换结束标志位，AD 转换完后，ADC_FLAG=1，一定要软件清 0。

ADC_START：模数转换器（ADC）转换启动控制位，1 开始转换，转换结束后为 0。

CHS2/CHS1/CHS0：模拟输入通道选择，选择使用 P1.0~P1.7 作为 A/D 输入。

ADC_RES、ADC_RESL： A/D 转换结果寄存器，是特殊功能寄存器，用于保存 A/D 转换结果。

IE：中断允许寄存器（可位寻址）

EA：CPU 的中断开放标志，EA=1，CPU 开放中断，EA=0，CPU 屏蔽所有中断申请。

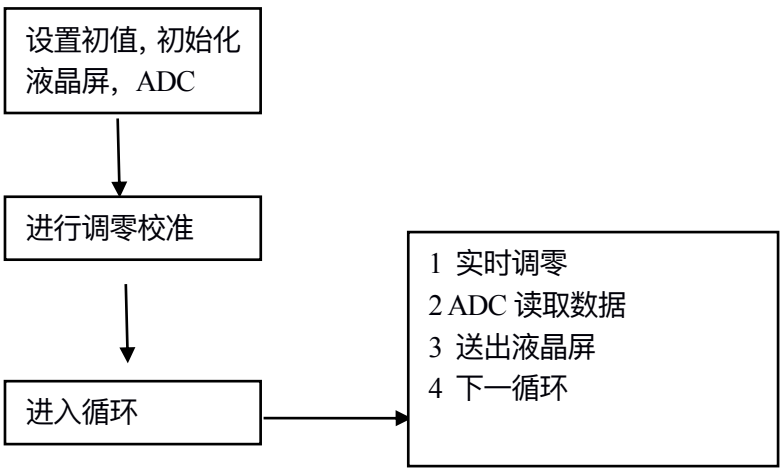
EADC：A/D 转换中断允许位。1 允许 0 禁止。

IPH：中断优先级控制寄存器高（不可位寻址）。

IP：中断优先级控制寄存器低（可位寻址）。

程序分析

（程序设计的思路、程序代码+注释）




```
#include <reg52.h>
#include <intrins.h>

#define uchar unsigned char
#define uint unsigned int
```

```
sbit CS1=P1^7;///左半边
sbit CS2=P1^6;///右半边
sbit E=P3^3;///使能信号
sbit RW=P3^4;///读写操作选择
sbit RS=P3^5;///寄存器选择(数据/指令)
sbit RES=P1^5;///复位 低电平有效
sbit BUSY=P2^7;//当前为运行状态

/**Declare SFR associated with the ADC */
sfr ADC_CONTR = 0xBC;    ///ADC control register
sfr ADC_RES = 0xBD;    ///ADC hight 8-bit result register
sfr ADC_LOW2 = 0xBE;    ///ADC low 2-bit result register
sfr P1ASF = 0x9D;    ///P1 secondary function control register: P1 口模拟功能控制寄存器
sfr AURX1 = 0xA2; ///AURX1 中的 ADRJ 位用于转换结果寄存器的数据格式调整控制

/**Define ADC operation const for ADC_CONTR*/
#define ADC_POWER 0x80    ///ADC power control bit
#define ADC_FLAG 0x10    ///ADC complete flag
#define ADC_START 0x08    ///ADC start control bit
#define ADC_SPEEDLL 0x00    ///540 clocks
#define ADC_SPEEDL 0x20    ///360 clocks
#define ADC_SPEEDH 0x40    ///180 clocks
#define ADC_SPEEDHH 0x60    ///90 clocks

uchar ch = 0;    ///ADC channel NO.0


uchar code zima[20][32]=
{
0x00,0x00,0xC0,0xE0,0x30,0x10,0x08,0x08,0x08,0x08,0x08,0x18,0x30,0xE0,0xC0,0x00,
0x00,0x00,0x07,0x0F,0x18,0x10,0x20,0x20,0x20,0x20,0x20,0x10,0x18,0x0F,0x07,0x00,///*"0"*0/

0x00,0x00,0x00,0x10,0x10,0x10,0x10,0xF0,0xF8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x20,0x20,0x20,0x20,0x3F,0x3F,0x20,0x20,0x20,0x20,0x00,0x00,0x00,///*"1"*1/

0x00,0x00,0x60,0x50,0x10,0x08,0x08,0x08,0x08,0x08,0x08,0x98,0xF0,0x70,0x00,0x00,
0x00,0x00,0x20,0x30,0x28,0x28,0x24,0x24,0x22,0x22,0x21,0x20,0x30,0x18,0x00,0x00,///*"2"*2/

0x00,0x00,0x30,0x30,0x08,0x08,0x88,0x88,0x88,0x88,0x58,0x70,0x30,0x00,0x00,0x00,
0x00,0x00,0x18,0x18,0x20,0x20,0x20,0x20,0x20,0x20,0x31,0x11,0x1F,0x0E,0x00,0x00,///*"3"*3/

0x00,0x00,0x00,0x00,0x00,0x80,0x40,0x20,0x10,0xF0,0xF8,0xF8,0x00,0x00,0x00,0x00,
0x00,0x04,0x06,0x05,0x05,0x04,0x24,0x24,0x24,0x3F,0x3F,0x3F,0x24,0x24,0x24,0x00,///*"4"*4/
```

```
0x00,0x00,0x00,0xC0,0x38,0x88,0x88,0x88,0x88,0x88,0x88,0x88,0x08,0x08,0x00,0x00,
0x00,0x00,0x18,0x29,0x21,0x20,0x20,0x20,0x20,0x20,0x30,0x11,0x1F,0x0E,0x00,0x00,///*"5"*5/

0x00,0x00,0x80,0xE0,0x30,0x10,0x98,0x88,0x88,0x88,0x88,0x88,0x98,0x10,0x00,0x00,
0x00,0x00,0x07,0x0F,0x19,0x31,0x20,0x20,0x20,0x20,0x20,0x20,0x11,0x1F,0x0E,0x00,///*"6"*6/

0x00,0x00,0x30,0x18,0x08,0x08,0x08,0x08,0x08,0x88,0x48,0x28,0x18,0x08,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x38,0x3E,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,///*"7"*7/

0x00,0x00,0x70,0x70,0xD8,0x88,0x88,0x08,0x08,0x08,0x08,0x98,0x70,0x70,0x00,0x00,
0x00,0x0C,0x1E,0x12,0x21,0x21,0x20,0x21,0x21,0x21,0x23,0x12,0x1E,0x0C,0x00,0x00,///*"8"*8/

0x00,0xE0,0xF0,0x10,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x18,0x10,0xF0,0xC0,0x00,0x00,
0x00,0x00,0x11,0x33,0x22,0x22,0x22,0x22,0x22,0x32,0x11,0x1D,0x0F,0x03,0x00,0x00,///*"9"*9/

0x08,0x08,0x0A,0xEA,0xAA,0xAA,0xAA,0xFF,0xA9,0xA9,0xA9,0xE9,0x08,0x08,0x08,0x00,
0x40,0x40,0x48,0x4B,0x4A,0x4A,0x4A,0x7F,0x4A,0x4A,0x4A,0x4B,0x48,0x40,0x40,0x00,///*"  重
"*10/

0x40,0x40,0x40,0xDF,0x55,0x55,0x55,0xD5,0x55,0x55,0x55,0xDF,0x40,0x40,0x40,0x00,
0x40,0x40,0x40,0x57,0x55,0x55,0x55,0x7F,0x55,0x55,0x55,0x57,0x50,0x40,0x40,0x00,///*"量"*11/

0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0xC0,0xC0,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,///*":"*12/

0x00,0x04,0x04,0xE4,0x24,0x24,0x24,0x3F,0x24,0x24,0x24,0xE4,0x04,0x04,0x00,0x00,
0x00,0x00,0x80,0x43,0x31,0x0F,0x01,0x01,0x01,0x3F,0x41,0x43,0x40,0x40,0x70,0x00,///*"克"*13/

};



void send_byte(uchar dat ,uchar cs1,uchar cs2);
void send_all(uint page,uint lie,uint offset);
void delay(uint x);
void init_adc();
void init_yejing();
void calibrate();
int get_ad_result();
void clearscreen();

int cweight;
int weight;
```

```
void main()
{
    init_yejing();
    init_adc();
    calibrate();///校准

    while(1)
    {
        weight=(get_ad_result()-cweight)/2-50;
          weight += weight/10;///真实重量
        clearscreen();

        send_all(1,1,10);///重
        send_all(1,2,11);///量
        send_all(1,3,12);///:
        send_all(4,3,weight/100);///百
        send_all(4,4,(weight/10)%10);///十
        send_all(4,5,weight%10);///个
        send_all(4,6,13);///克

        delay(50000);

    }

}

void init_yejing()
{
    send_byte(192,1,1);///设置起始行
    send_byte(63,1,1);///打开显示开关
}

void send_byte(uchar dat,uchar cs1,uchar cs2)
{
    P2=0xff;
    CS1=cs1; CS2=cs2;
    RS=0; RW=1; E=1;//读状态字
    while(BUSY) ;

    ///送数据或控制字
    E=0;
```

```
    RS=!(cs1&&cs2),RW=0;//写指令代码
    P2=dat;
    E=1; delay(3);


   E=0;//总线释放

    CS1=CS2=0;
}

void send_all(uint page,uint lie,uint offset)
{
    uint i,j,k=0;
    for(i=0;i<2;++i)
    {
        send_byte(184+i+page,1,1);///选择页面    184-页面地址设置
        send_byte(64+lie*16-(lie>3)*64,1,1);///选择列号
        for(j=0;j<16;++j)
            send_byte(zima[offset][k++],lie<4,lie>=4);///送数
    }
}

void init_adc()
{
    P1ASF = 1; ///Set   P1.0 as analog input port
    AURX1 |= 0X04; ///AURX1 中的 ADRJ 位用于转换结果寄存器的数据格式调整控制

    ADC_RES = ADC_LOW2 = 0; ///Clear previous result

    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;    ///ch=0 ADC
channel NO.0
    delay(4);   ///ADC power-on delay and Start A/D conversion
}

int get_ad_result()
{
    int ADC_result;
    ADC_RES = ADC_LOW2 = 0; ///Clear previous result
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ch | ADC_START;
    _nop_(); _nop_(); _nop_(); _nop_(); _nop_(); _nop_(); ///Must wait before inquiry
    while (!(ADC_CONTR & ADC_FLAG)); ///Wait complete flag
    ADC_result = (ADC_RES & 0x03) *256 + ADC_LOW2;///ADC_RES 中存高 2 位
    ADC_CONTR &= ~ADC_FLAG; ///Close ADC flag 位置 0
    return ADC_result;   ///Return ADC result
```

```
}

void calibrate() //校正
{
    cweight=(get_ad_result()-0)/2;
}

void delay(uint x)
{
    while(x--) ;
}

void clearscreen()
{
    int i,j;
    for(i=0;i<8;++i)
    {
        send_byte(184+i,1,1);///页
        send_byte(64,1,1);///列
            for(j=0;j<64;++j)
                {
                    send_byte(0x00,0,1);
                    send_byte(0x00,1,0);
                }
    }
}
```

思考题:

1. 调零的原理:
   硬件：通过电路或者改变压名电阻调零。
   软件：对采集的数据进行科学处理进行调零。
2. A/D 和 D/A 转换原理
   A/D 转换：逐次逼近法。逐次逼近式 AD 转换器与计数式 A/D 转换类似，只是数字量由“逐次逼近寄存器 SAR”产生。SAR 使用“对分搜索法”产生数字量，以 8 位数字量为例，SAR 首先产生 8 位数字量的一半，即 10000000B,试探模拟量 Vi 的大小，若 Vo>Vi，清除最高位，若 Vo<Vi,保留最高位。在最高位确定后，SAR 又以对分搜索法确定次高位，即以低 7 位的一半 y1000000B(y 为已确定位) 试探模拟量 Vi 的大小。在 bit6 确定后，SAR 以对分搜索法确定 bit5 位，即以低 6 位的一半 yy100000B(y 为已确定位) 试探模拟量的大小。重复这一过程，直到最低位 bit0 被确定，转换结束。
3. IC 总线在信号通讯过程中的应用
   主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时被寻址的器件被认为是从器件。

问题分析

（实验过程中遇到的问题及解决方法）

问题：压敏电阻测量变化问题

解决：调零时动态读取数据进行调零。

实验六 直流电机脉宽调制调速

原理总结

（该实验涉及的基本原理及其在实验中的使用方法）

本实验的液晶显示部分与实验五相同，不再赘述，关键在于直流电机转速的控制。

设计思想：在开关闭合时，持续给出脉冲信号使电机加速。

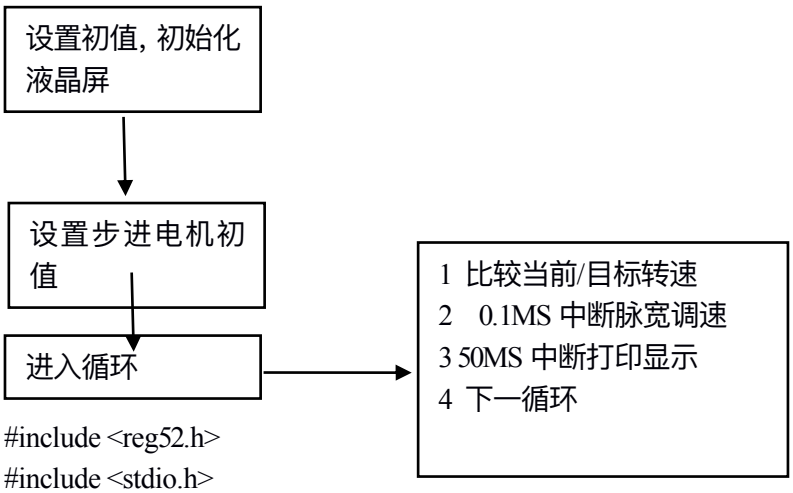
在开关断开时，依靠惯性旋转。

定时记录转速数据显示。

或者按开关记录转速数据显示。

程序分析

（程序设计的思路、程序代码+注释）




```
#include <reg52.h>
#include <stdio.h>
#include <intrins.h>

#define TIMER 10
#define MAXSPEED 180
#define MIDSPEED 90
#define MINSPEED 50

sbit CS1=P1^7;
sbit CS2=P1^6;
sbit RST=P1^5;
sbit E=P3^3;
sbit RW=P3^4;
sbit RS=P3^5;
sbit BUSY=P2^7;
sbit RESET=P2^4;
sbit KEY1=P3^6;
sbit KEY2=P3^7;
sbit OUT=P1^1;

sfr P4=0xc0;
sfr P4SW=0xbb;
sbit DCLK=P4^4;
sbit LED=P4^5;
```

```
int i,j,k,a,b,c,d,temp;
int timer=TIMER,count=0,countS=0,currentSpeed=0,objSpeed=0;
int SUM=0,N=30,M=256;
char code ledCode[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
char code mubiao[4][16]={

    {0x00,0x00,0xFE,0x22,0x22,0x22,0x22,0x22,0x22,0x22,0x22,0x22,0xFE,0x00,0x00,0x00},

    {0x00,0x00,0xFF,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0xFF,0x00,0x00,0x00},/*"?",0
*/

    {0x10,0x10,0xD0,0xFF,0x90,0x10,0x20,0x22,0x22,0x22,0xE2,0x22,0x22,0x22,0x20,0x00},

    {0x04,0x03,0x00,0xFF,0x00,0x13,0x0C,0x03,0x40,0x80,0x7F,0x00,0x01,0x06,0x18,0x00},/*"?",1
*/
                        };
char code dangqian[4][16]={

    {0x00,0x40,0x42,0x44,0x58,0x40,0x40,0x7F,0x40,0x40,0x50,0x48,0xC6,0x00,0x00,0x00},

    {0x00,0x40,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0xFF,0x00,0x00,0x00},/*"?",0
*/


    {0x08,0x08,0xE8,0x29,0x2E,0x28,0xE8,0x08,0x08,0xC8,0x0C,0x0B,0xE8,0x08,0x08,0x00},

    {0x00,0x00,0xFF,0x09,0x49,0x89,0x7F,0x00,0x00,0x0F,0x40,0x80,0x7F,0x00,0x00,0x00},/*"?",1
*/
                        };
char                                                                    code
number[10][16]={{0x00,0xE0,0x10,0x08,0x08,0x10,0xE0,0x00,0x00,0x0F,0x10,0x20,0x20,0x10,0x0F,
0x00},/*"0",0*/

    {0x00,0x10,0x10,0xF8,0x00,0x00,0x00,0x00,0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00},/*"1",1
*/

    {0x00,0x70,0x08,0x08,0x08,0x88,0x70,0x00,0x00,0x30,0x28,0x24,0x22,0x21,0x30,0x00},/*"2",2
*/

    {0x00,0x30,0x08,0x88,0x88,0x48,0x30,0x00,0x00,0x18,0x20,0x20,0x20,0x11,0x0E,0x00},/*"3",3
*/

    {0x00,0x00,0xC0,0x20,0x10,0xF8,0x00,0x00,0x00,0x07,0x04,0x24,0x24,0x3F,0x24,0x00},/*"4",4
*/
```

```
	{0x00,0xF8,0x08,0x88,0x88,0x08,0x08,0x00,0x00,0x19,0x21,0x20,0x20,0x11,0x0E,0x00},/*"5",5
*/

	{0x00,0xE0,0x10,0x88,0x88,0x18,0x00,0x00,0x00,0x0F,0x11,0x20,0x20,0x11,0x0E,0x00},/*"6",6
*/

	{0x00,0x38,0x08,0x08,0xC8,0x38,0x08,0x00,0x00,0x00,0x00,0x3F,0x00,0x00,0x00,0x00},/*"7",7
*/

	{0x00,0x70,0x88,0x08,0x08,0x88,0x70,0x00,0x00,0x1C,0x22,0x21,0x21,0x22,0x1C,0x00},/*"8",
8*/

	{0x00,0xE0,0x10,0x08,0x08,0x10,0xE0,0x00,0x00,0x00,0x31,0x22,0x22,0x11,0x0F,0x00},/*"9",9
*/};

void wait(unsigned int count){
	while(count--){
		_nop_();
	}
}

void send2LED(char temp){
	for(d=0;d<8;d++){
		DCLK=0;
		LED=temp & 0x80;
		DCLK=1;
		temp<<=1;
	}
}

void outLed(int num){
	a=num/100;
	b=(num-a*100)/10;
	c=num-a*100-b*10;
	send2LED(ledCode[c]);
	send2LED(ledCode[b]);
	send2LED(ledCode[a]);
}


void checkReady(bit cs){
	CS1= cs==0 ? 1:0;
	CS2= cs==1 ? 1:0;
```

```
        P2=0xff;
        E=1;
        RS=0;
        RW=1;
        while(BUSY==1);
        E=0;
        CS2=0;
        CS1=0;
}
void write_command(bit cs,char com){
        checkReady(cs);
        CS1= cs==0 ? 1:0;
        CS2= cs==1 ? 1:0;
        RS=0;
        RW=0;
        E=1;
        P2=com;
        wait(20);
        E=0;
        CS1=0;
        CS2=0;
}
void cls(bit cs){
        for(i=0;i<8;i++){
                write_command(cs,0xb8+i);
                for(j=0;j<64;j++){
                        checkReady(cs);
                        CS1= cs==0 ? 1:0;
                        CS2= cs==1 ? 1:0;
                        E=1;
                        RS=1;
                        RW=0;
                        P2=0x00;
                        wait(20);
                        E=0;
                }
        }
}
void write_bytes(bit cs,char * buffer,int len){
        for(i=0;i<len;i++){
                checkReady(cs);
                CS1= cs==0 ? 1:0;
                CS2= cs==1 ? 1:0;
                E=1;
```

```
            RS=1;
            RW=0;
            P2=*(buffer+i);
            wait(20);
            E=0;
        }
    }
    void outChina(bit cs,char x,char y,char china[][16]){
        write_command(cs,x+0xb8);
        write_command(cs,y+0x40);
        write_bytes(cs,*china,16);

        write_command(cs,x+1+0xb8);
        write_command(cs,y+0x40);
        write_bytes(cs,*(china+1),16);
    }
    void outNumber(bit cs,char x,char y,int num){
        write_command(cs,x+0xb8);
        write_command(cs,y+0x40);
        write_bytes(cs,number[num],8);
        write_command(cs,x+1+0xb8);
        write_command(cs,y+0x40);
        write_bytes(cs,number[num]+8,8);
    }
    void showNow(int num){
        a=num/1000;
        b=(num-a*1000)/100;
        c=(num-a*1000-b*100)/10;
        d=num-a*1000-b*100-c*10;
        outNumber(1,4,43,d);
        outNumber(1,4,33,c);
        outNumber(1,4,23,b);
        outNumber(1,4,13,a);
    }
    void showTarget(int num){
        a=num/1000;
        b=(num-a*1000)/100;
        c=(num-a*1000-b*100)/10;
        d=num-a*1000-b*100-c*10;
        outNumber(0,4,43,d);
        outNumber(0,4,33,c);
        outNumber(0,4,23,b);
        outNumber(0,4,13,a);
    }
```

```
void initLCD(){
	cls(1);
	write_command(0,0x3f);
	cls(0);
	write_command(1,0x3f);
	outChina(0,1,5,mubiao);
	outChina(0,1,35,mubiao+2);
	outChina(1,1,5,dangqian);
	outChina(1,1,35,dangqian+2);
}
void ex_int0() interrupt 0
{
	count=count+1;
}
int getStepLen(int num){
	if(num<6){
		return 0;
	}
	if(num<10){
		return 1;
	}
	temp=num/100;
	temp=(num-temp*100)/10;
	temp=temp/3*2+1;
	if((N-temp)<1){
		return 1;
	}
	return temp;
}
void t0_int0() interrupt 1
{
	TR0=0;

	timer=timer-1;
	if(timer==0){
		if(countS==0){
			countS=count;
			currentSpeed=count*2;
		}else{
			currentSpeed=count+countS;
			countS=0;
		}
		timer=TIMER;
		count=0;
```

```
            if(currentSpeed<objSpeed){
                N=N+getStepLen(objSpeed-currentSpeed);
            }
            if(currentSpeed>objSpeed){
                N=N-getStepLen(currentSpeed-objSpeed);
            }
        }

        TH0=0x3c;
        TL0=0xb0;
        TR0=1;
}
void t1_int0() interrupt 3
{
        TR1=0;

        SUM=SUM+N;
        if((!KEY1)&&(!KEY2)){
            OUT=0;
        }else{
            if(!KEY1){
                objSpeed=MAXSPEED;
            }
            if(!KEY2){
                objSpeed=MINSPEED;
            }
            if(KEY1 && KEY2){
                objSpeed=MIDSPEED;
            }
            if(SUM>M){
                OUT=0;
                SUM=SUM-M;
            }else{
                OUT=1;
            }
        }

        TH1=0xff;
        TL1=0x9c;
        TR1=1;
}
void initTimer(){
        P4SW=0x30;
```

```
    TMOD=0x11;
    TH0=0x3c;
    TL0=0xb0;
    TH1=0xff;
    TL1=0x9c;

    IT0=1;
    EA=1;
    ET0=1;
    ET1=1;
    EX0=1;

    TR0=1;
    TR1=1;
}
void waitL(int count){
    k=0xffff;
    while(count--){
        while(k--);
    }
}
void main(){
    initLCD();
    initTimer();
    while(1){
        if((!KEY1)&&(!KEY2)){
            outLed(currentSpeed);
        }else{
            showTarget(objSpeed);
            showNow(currentSpeed);
        }
        waitL(0x05);
    }
}
```

思考题:

1. 脉宽调速和电压调速的区别。
   脉宽调速：通过输出高频 0/1 信号调速，需要外围器件少，适用于单片机，不适用于平滑调速。
   电压调速：通过改变电压模拟量调速，实现无级平滑调速，损耗小，经济性强。
2. 累加进位法
   设置累加变量 X，每次+N，如果》M，则输出 1，减去 M。否则输出 0.占空比：N/M,1 的比例为 N/M。
3. 转速测量误差及减小方法

物理：减少摩擦力
计算：保持 40 转左右的速度，过快超出传感器测量范围，会产生误差。

问题分析

（实验过程中遇到的问题及解决方法）
问题：转速测量不准确问题
解决：由于传感器测速上限，过快的转速无法测量，通过编程限制转速上限。

实验七 温度测量与控制（二选一）

原理总结

（该实验涉及的基本原理及其在实验中的使用方法）

本实验使用的 DS18B20 是单总线数字温度计，测量范围从—55℃到+125℃，增量值为 0.5 ℃。

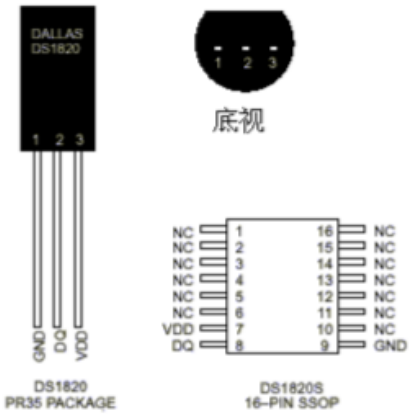
用于贮存测得的温度值的两个 8 位存贮器 RAM 编号为 0 号和 1 号。1 号存贮器存放温度值的符号，如果温度为负（℃），则 1 号存贮器 8 位全为 1，否则全为 0。 0 号存贮器用于存放温度值的补码 LSB(最低位)的 1 表示 0.5℃ 。

将存贮器中的二进制数求补再转换成十进制数并除以 2，就得到被测温度值。

温度检测与控制系统由加热灯泡，温度二极管，温度检测电路，控制电路和继电器组成。温度二极管和加热灯泡封闭在一个塑料保温盒内，温度二极管监测保温盒内的温度，用温控实验板内部的 A/D 转换器 ADC7109 检测二极管两端的电压，通过电压和温度的关系，计算出盒内空气的实际温度。

本实验使用 STC89C516RD+单片机实验板。单片机的 P1.4 与 DS18B20 的 DQ 引脚相连，进行数据和命令的传输。 单片机的 P1.1 连接热电阻。当 P1.1 为高电平时，加热热电阻。

引脚排列



DS1820
PR35 PACKAGE

DS1820S
16-PIN SSOP

引脚说明

GND	– 地
DQ	– 数据I/O
V_{DD}	– 可选VDD
NC	– 空脚

说明

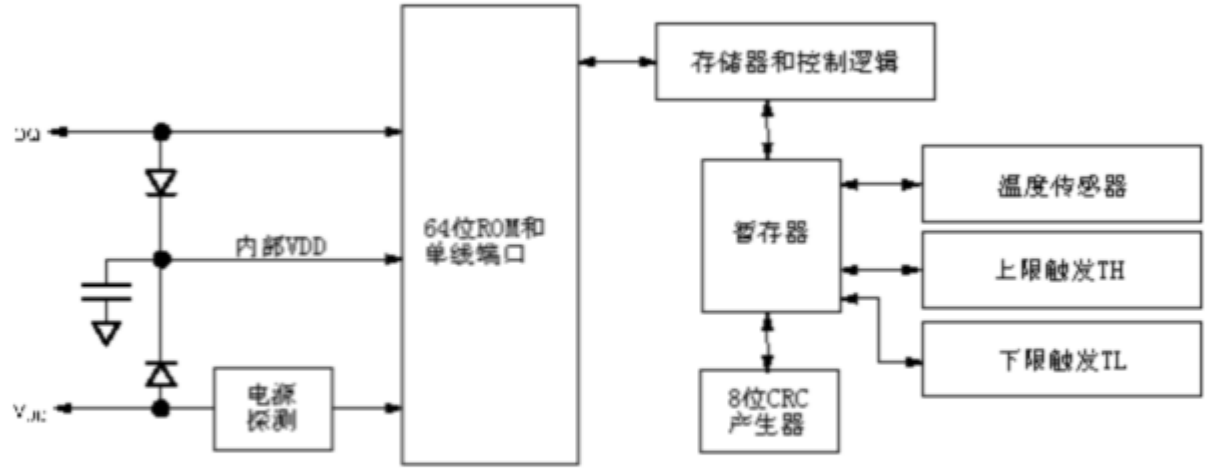
DS1820 数字温度计以 9 位数字量的形式反映器件的温度值。

DS1820 通过一个单线接口发送或接收信息，因此在中央微处理器和 DS1820 之间仅需一条连接线（加上地线）。用于读写和温度转换的电源可以从数据线本身获得，无需外部电源。

因为每个 DS1820 都有一个独特的片序列号，所以多只 DS1820 可以同时连在一根单线总线上，这样就可以把温度传感器放在许多不同的地方。这一特性在 HVAC 环境控制、探测建筑物、仪器或机器的温度以及过程监测和控制等方面非常有用。

图 1 的方框图示出了 DS1820 的主要部件。DS1820 有三个主要数字部件：1）64 位激光 ROM，2）温度传感器，3）非易失性温度报警触发器 TH 和 TL。器件用如下方式从单线通讯线上汲取能量：在信号线处于高电平期间把能量储存在内部电容里，在信号线处于低电平期间消耗电容上的电能工作，直到高电平到来再给寄生电源（电容）充电。DS1820 也可用外部 5V 电源供电。

DS1820方框图（图1）

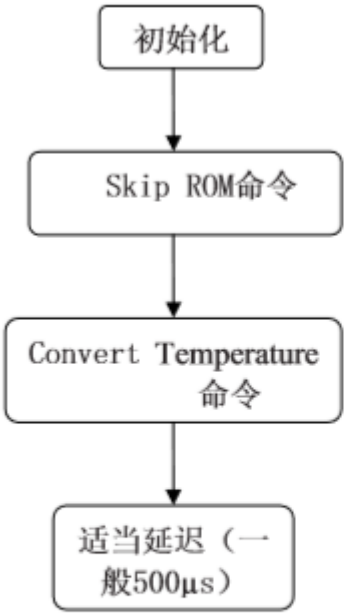


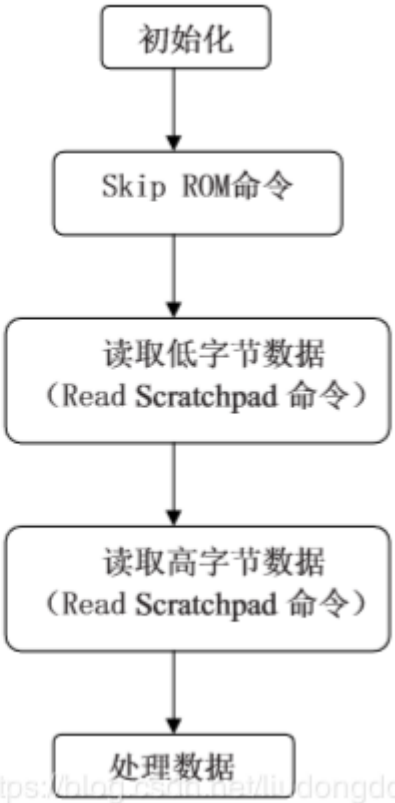
DS1820 依靠一个单线端口通讯。在单线端口条件下，必须先建立 ROM 操作协议，才能进行存储器和控制操作。因此，控制器必须首先提供下面 5 个 ROM 操作命令之一：1）读 ROM，2）匹配 ROM，3）搜索 ROM，4）跳过 ROM，5）报警搜索。这些命令对每个器件的激光 ROM 部分进行操作，在单线总线上挂有多个器件时，可以区分出单个器件，同时可以向总线控制器指明有多少器件或是什么型号的器件。成功执行完一条 ROM 操作序列后，即可进行存储器和控制操作，控制器可以提供 6 条存储器和控制操作指令中的任一条。

一条控制操作命令指示 DS1820 完成一次温度测量。测量结果放在 DS1820 的暂存器里，用一条读暂存器内容的存储器操作命令可以把暂存器中数据读出。温度报警触发器 TH 和 TL 各由

处理数据

DS18B20 的高速暂存存储器共有 9 个字节。当温度转换命令发布后，经转换所得的温度值以二字节补码形式存放在高速暂存存储器的第 0 和第 1 个字节。这是 12 位精度转换后得到的 16 位数据，其中前面 5 位为符号位。如果测得的温度大于或等于 0，这 5 位为 0，只要将测到的数值乘于 0.0625 即可得到实际温度；如果温度小于 0，这 5 位为 1，测到的数值需要取反加 1 再乘于 0.0625 即可得到实际温度。单片机可通过单线接口读到该数据，读取时低位在前，高位在后。



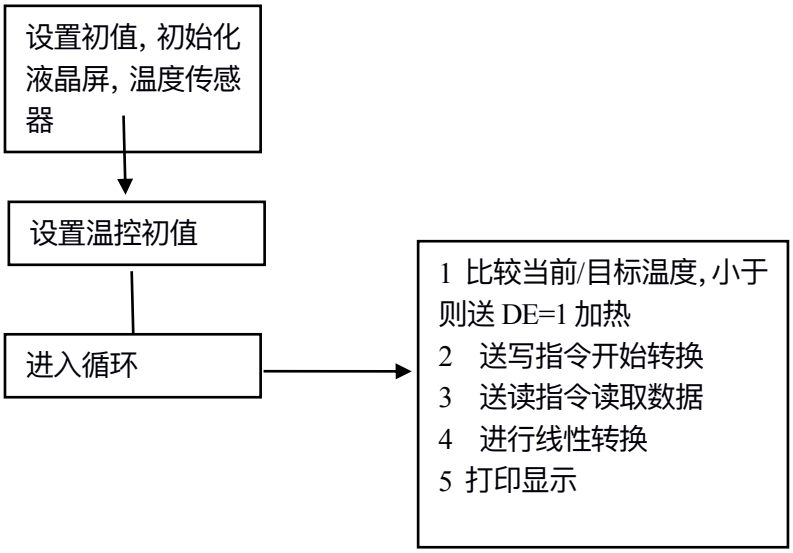


写操作（包括写指令）具体过程如下：通过单总线采取移位的方式来向DS18B20写入数据，按照从低位到高位的顺序每次一位的方式写进去，需要满足写时间间隙的要求。在写数据时间隙的前15μs数据总线需要是被单片机拉置低电平，而后则将是芯片对总线数据的采样时间，采样时间在15～60μs，采样时间内如果单片机将总线拉高则表示写“1”，如果单片机将总线拉低则表示写“0”。每一位的发送都应该有一个至少 15μs的低电平起始位，随后的数据“0”或“1”应该在45μs 内完成。整个位的发送时间应该保持在60～120uS，否则不能保证通信的正常。

读操作（包括读数据）具体过程如下：也是通过移位的方法从DS18B20中读取数据，按照从低位到高位的顺序每次一位的方式读入，需要满足读时间间隙的要求。读时间隙时控制时的采样时间应该更加的精确才行，读时间隙时也是必须先由单片机产生至少 1μs的低电平，表示读时间的起始。随后在总线被释放后的15μs 中DS18B20 会发送内部数据位，这时单片机如

程序分析

(程序设计的思路、程序代码+注释)




```
#include <reg52.h>
#include <intrins.h>

#define uchar unsigned char
#define uint unsigned int


uchar code zima[20][32]=
{
0x00,0x00,0xC0,0xE0,0x30,0x10,0x08,0x08,0x08,0x08,0x08,0x18,0x30,0xE0,0xC0,0x00,
0x00,0x00,0x07,0x0F,0x18,0x10,0x20,0x20,0x20,0x20,0x20,0x10,0x18,0x0F,0x07,0x00,///*"0"*0/

0x00,0x00,0x00,0x10,0x10,0x10,0x10,0xF0,0xF8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x20,0x20,0x20,0x20,0x3F,0x3F,0x20,0x20,0x20,0x20,0x00,0x00,0x00,///*"1"*1/

0x00,0x00,0x60,0x50,0x10,0x08,0x08,0x08,0x08,0x08,0x08,0x98,0xF0,0x70,0x00,0x00,
0x00,0x00,0x20,0x30,0x28,0x28,0x24,0x24,0x22,0x22,0x21,0x20,0x30,0x18,0x00,0x00,///*"2"*2/

0x00,0x00,0x30,0x30,0x08,0x08,0x88,0x88,0x88,0x88,0x58,0x70,0x30,0x00,0x00,0x00,
0x00,0x00,0x18,0x18,0x20,0x20,0x20,0x20,0x20,0x20,0x31,0x11,0x1F,0x0E,0x00,0x00,///*"3"*3/

0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x40,0x20,0x10,0xF0,0xF8,0xF8,0x00,0x00,0x00,
0x00,0x04,0x06,0x05,0x05,0x04,0x24,0x24,0x24,0x3F,0x3F,0x3F,0x24,0x24,0x24,0x00,///*"4"*4/

0x00,0x00,0x00,0xC0,0x38,0x88,0x88,0x88,0x88,0x88,0x88,0x88,0x08,0x08,0x00,0x00,
0x00,0x00,0x18,0x29,0x21,0x20,0x20,0x20,0x20,0x20,0x20,0x30,0x11,0x1F,0x0E,0x00,0x00,///*"5"*5/
```

```
0x00,0x00,0x80,0xE0,0x30,0x10,0x98,0x88,0x88,0x88,0x88,0x88,0x98,0x10,0x00,0x00,
0x00,0x00,0x07,0x0F,0x19,0x31,0x20,0x20,0x20,0x20,0x20,0x20,0x11,0x1F,0x0E,0x00,///*"6"*6/

0x00,0x00,0x30,0x18,0x08,0x08,0x08,0x08,0x08,0x88,0x48,0x28,0x18,0x08,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x38,0x3E,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,///*"7"*7/

0x00,0x00,0x70,0x70,0xD8,0x88,0x88,0x08,0x08,0x08,0x08,0x98,0x70,0x70,0x00,0x00,
0x00,0x0C,0x1E,0x12,0x21,0x21,0x20,0x21,0x21,0x21,0x23,0x12,0x1E,0x0C,0x00,0x00,///*"8"*8/

0x00,0xE0,0xF0,0x10,0x08,0x08,0x08,0x08,0x08,0x08,0x18,0x10,0xF0,0xC0,0x00,0x00,
0x00,0x00,0x11,0x33,0x22,0x22,0x22,0x22,0x22,0x32,0x11,0x1D,0x0F,0x03,0x00,0x00,///*"9"*9/

0x08,0x08,0x0A,0xEA,0xAA,0xAA,0xAA,0xFF,0xA9,0xA9,0xA9,0xE9,0x08,0x08,0x08,0x00,
0x40,0x40,0x48,0x4B,0x4A,0x4A,0x4A,0x7F,0x4A,0x4A,0x4A,0x4B,0x48,0x40,0x40,0x00,///*"  重
"*10/

0x40,0x40,0x40,0xDF,0x55,0x55,0x55,0xD5,0x55,0x55,0x55,0xDF,0x40,0x40,0x40,0x00,
0x40,0x40,0x40,0x57,0x55,0x55,0x55,0x7F,0x55,0x55,0x55,0x57,0x50,0x40,0x40,0x00,///*"量"*11/

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0xC0,0xC0,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,///*":"*12/

0x00,0x04,0x04,0xE4,0x24,0x24,0x24,0x3F,0x24,0x24,0x24,0xE4,0x04,0x04,0x00,0x00,
0x00,0x00,0x80,0x43,0x31,0x0F,0x01,0x01,0x01,0x3F,0x41,0x43,0x40,0x40,0x70,0x00,///*"克"*13/

0x10,0x21,0x86,0x70,0x00,0x7E,0x4A,0x4A,0x4A,0x4A,0x4A,0x7E,0x00,0x00,0x00,0x00,0x00,
0x02,0xFE,0x01,0x40,0x7F,0x41,0x41,0x7F,0x41,0x41,0x7F,0x41,0x41,0x7F,0x40,0x00,///*"温",14*/

0x00,0x00,0xFC,0x04,0x24,0x24,0xFC,0xA5,0xA6,0xA4,0xFC,0x24,0x24,0x24,0x04,0x00,
0x80,0x60,0x1F,0x80,0x80,0x42,0x46,0x2A,0x12,0x12,0x2A,0x26,0x42,0xC0,0x40,0x00,///*"度",15*/

};



sbit CS1=P1^7;///左半边
sbit CS2=P1^6;///右半边
sbit E=P3^3;///使能信号
sbit RW=P3^4;///读写操作选择
sbit RS=P3^5;///寄存器选择(数据/指令)
sbit RES=P1^5;///复位 低电平有效
sbit BUSY=P2^7;

sbit De=P1^1; ///加热
```

```
sbit DQ=P1^4; ///DS18B20 单数据总线
uchar TPH,TPL; ///温度值高位  低位
unsigned int t; ///温度值
unsigned int t1=30; ///目标温度值

sbit swh1=P3^6;
sbit swh2=P3^7;
uchar flag1=0;
uchar flag2=0;



void send_byte(uchar dat ,uchar cs1,uchar cs2);
void send_all(uint page,uint lie,uint offset);
void delay(uint x);
void init_yejing();
void clearscreen();

void DelayXus(uchar n); ///微秒级延时
void ow_rest(); ///复位
void write_byte(char dat);
unsigned char read_bit(void);




void main(void)
{
	init_yejing();

	t=0;

	while(1)
	{


		if(swh1==0)
		{
			flag1=1;
		}
		if(swh1==1 && flag1==1)
		{
```

```
            t1++;
            flag1=0;
        }


        if(swh2==0)
            flag2=1;
        if(swh2==1 && flag2==1)
        {
            t1--;
            flag2=0;
        }



if(t<t1)
De=1;
else De=0;
ow_rest(); ///设备复位

write_byte(0xCC); ///跳过 ROM 命令

write_byte(0x44); ///开始转换命令
while (!DQ); ///等待转换完成

ow_rest(); ///设备复位
write_byte(0xCC); ///跳过 ROM 命令
write_byte(0xBE); ///读暂存存储器命令
TPL = read_bit(); ///读温度低字节
TPH = read_bit(); ///读温度高字节


t=TPH; ///取温度高位
t<<=8; ///高位 8 位
t|=TPL; ///加上温度低位
t*=0.625; ///实际温度  可直接显示

  t=t/10;


 send_all(1,1,14);///温
send_all(1,2,15);///度
send_all(1,3,12);///:
```

```
        send_all(4,2,t1/10);///十
      send_all(4,3,t1%10);///个

      send_all(4,5,t/10);///十
      send_all(4,6,t%10);///个

        delay(50000);

        clearscreen();




    }
}



void DelayXus(uchar n)
{
    while (n--)
    {
        _nop_();
        _nop_();
    }
}

unsigned char read_bit(void)///读位
{
    uchar i;
    uchar dat = 0;
    for (i=0; i<8; i++) ///8 位计数器
    {
        dat >>= 1;
        DQ = 0; ///开始时间片
        DelayXus(1); ///延时等待
        DQ = 1; ///准备接收
        DelayXus(1); ///接收延时
        if (DQ) dat |= 0x80; ///读取数据
        DelayXus(60); ///等待时间片结束
    }
    return dat;
}
```

```
void ow_rest()///复位
{
    CY = 1;
    while (CY)
    {
        DQ = 0; ///送出低电平复位信号
        DelayXus(240); ///延时至少 480us
        DelayXus(240);
        DQ = 1; ///释放数据线
        DelayXus(60); ///等待 60us
        CY = DQ; ///检测存在脉冲,DQ 为 0 转换完成
        DelayXus(240); ///等待设备释放数据线
        DelayXus(180);
    }
}

void write_byte(char dat)///写字节
{
    uchar i;
    for (i=0; i<8; i++) ///8 位计数器
    {
        DQ = 0; ///开始时间片
        DelayXus(1); ///延时等待
        dat >>= 1; ///送出数据
        DQ = CY;
        DelayXus(60); ///等待时间片结束
        DQ = 1; ///恢复数据线
        DelayXus(1); ///恢复延时
    }
}




void init_yejing()
{
    send_byte(192,1,1);///设置起始行
    send_byte(63,1,1);///打开显示开关
}

void send_byte(uchar dat,uchar cs1,uchar cs2)
{
```

```
    P2=0xff;
    CS1=cs1; CS2=cs2;
    RS=0; RW=1; E=1;
    while(BUSY) ;

    ///送数据或控制字
    E=0;
    RS=!(cs1&&cs2),RW=0;
    P2=dat;
    E=1; delay(3); E=0;

    CS1=CS2=0;
}

void send_all(uint page,uint lie,uint offset)
{
    uint i,j,k=0;
    for(i=0;i<2;++i)
    {
        send_byte(184+i+page,1,1);///选择页面
        send_byte(64+lie*16-(lie>3)*64,1,1);///选择列号
        for(j=0;j<16;++j)
            send_byte(zima[offset][k++],lie<4,lie>=4);///送数
    }
}


void delay(uint x)
{
    while(x--) ;
}

void clearscreen()
{
    int i,j;
    for(i=0;i<8;++i)
    {
        send_byte(184+i,1,1);///页
        send_byte(64,1,1);///列
            for(j=0;j<64;++j)
                {
                    send_byte(0x00,0,1);
                    send_byte(0x00,1,0);
                }
```

```
    }
}
```

思考题：

1. 进行精确的延时的程序有几种方法？各有什么优缺点？

答：实现延时通常有两种方法：一种是硬件延时，要用到定时器/计数器，这种方法可以提高 CPU 的工作效率，也能做到精确延时；另一种是软件延时，这种方法主要采用循环体进行

（1）使用定时器/计数器实现精确延时

单片机系统一般常选用 11.059 2 MHz、12 MHz 或 6 MHz 晶振。第一种更容易产生各种标准的波特率，后两种的一个机器周期分别为 1 μs 和 2 μs，便于精确延时。本程序中假设使用频率为 12 MHz 的晶振。最长的延时时间可达 216=65 536 μs。若定时器工作在方式 2，则可实现极短时间的精确延时；如使用其他定时方式，则要考虑重装定时初值的时间（重装定时器初值占用 2 个机器周期）。

在实际应用中，定时常采用中断方式，如进行适当的循环可实现几秒甚至更长时间的延时。

（2）软件延时与时间计算

可以在 C 文件中通过使用带_NOP_( )语句的函数实现，定义一系列不同的延时函数，如 Delay10us( )、Delay25us( )、Delay40us( )等存放在一个自定义的 C 文件中，需要时在主程序中直接调用。

2. 参考其他资料，了解 DS18B20 的其他命令用法。

ROM操作命令有：

指令	代码	解释
Read ROM (读ROM)	[33H]	此命令允许总线主机读DS18B20的8位产品系列编码，唯一的48位序列号，以及8位的CRC。
Match ROM (匹配ROM)	[55H]	此命令后继以64位的ROM数据序列，允许总线主机对多点总线上特定的DS18B20寻址。
Skip ROM (跳过ROM)	[CCH]	在单点总线系统中，此命令通过允许总线主机不提供64位ROM编码而访问存储器操作来节省时间。
Search ROM (搜索ROM)	[F0H]	此命令允许总线控制器用排除法识别总线上的所有从机的64位编码。
Alarm search (告警搜索)	[ECH]	此命令的流程与搜索ROM命令相同。但是，仅在最近一次温度测量出现告警的情况下，DS18B20才对此命令作出响应。

存储器操作命令如下：

命令	代码	解释
Write Scratchpad (写暂存存储器)	[4EH]	此命令向DS18B20的暂存器中写入数据，开始位置在地址2。接下来写入的两个字节将被存到暂存器中的地址位置2和3。
Read Scratchpad (读暂存存储器)	[BEH]	此命令读取暂存器的内容。读取将从字节0开始，一直进行下去，直到第9（字节8，CRC）字节读完。

Copy Scratchpad (复制暂存存储器)	[48H]	此命令把暂存器的内容拷贝到DS18B20的EPROM存储器里，即把温度报警触发字节存入非易失性存储器里。
Convert Temperature (温度变换)	[44H]	这条命令启动一次温度转换而无需其他数据。
Recall EPROM (重新调出)	[B8H]	此命令把贮存在EPROM中温度触发器的值重新调至暂存存储器。
Read Power Supply (读电源)	[B4H]	对于在此命令发送至DS18B20之后所发出的第一读数据的时间片，器件都会给出其电源方式的信号："0"=寄生电源供电，"1"=外部电源供电。

问题分析

（实验过程中遇到的问题及解决方法）