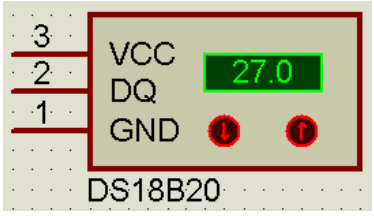
東華理工大學
EAST CHINA INSTITUTE OF TECHNOLOGY
luoyong199092@163.com

DS18B20 原理及应用实例

基于 Proteus 仿真

前言：本文详细介绍了 DS18B20 原理，并在后面举例说明了其在单片机中的应用，所举例子包含 Proteus 仿真电路图，源程序，程序注释详细清楚。

1、DS18B20简介：DS18B20温度传感器是DALLAS公司生产的1-wire式单总线器件，具有线路简单，体积小的特点，用它组成的温度测量系统线路非常简单，只要求一个端口即可实现通信。温度测量范围在－55℃～+125℃之间，分辨率可以从9~12位选择，内部还有温度上、下限报警设置。每个DS18B20芯片都有唯一的序列号，所以可以利用多个DS18B20同时连接在同一条总线上，组成多点测温系统。但最多只能连接8个，如果数量过多，会使供电电源电压过低，从而造成信号传输的不稳定。



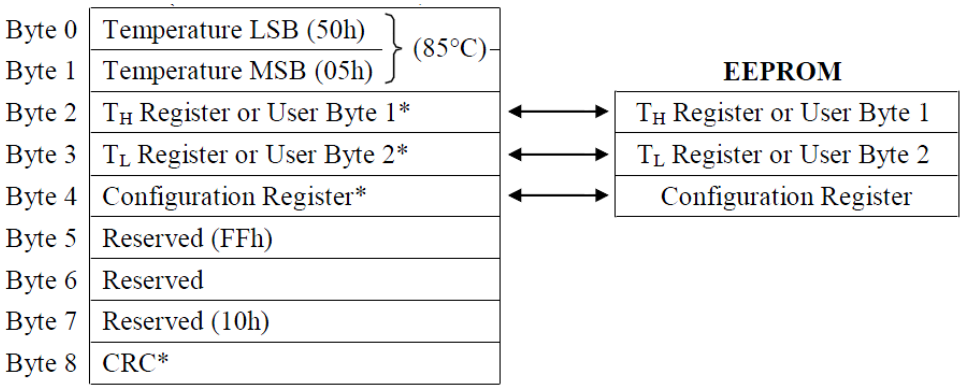
2、DS18B20 结构：如右图所示，DS18B20 有三只引脚，VCC、DQ 和 GND。DQ 为数字信号输入/输出端（DQ 一般接控制器（单片机）的一个 I/O 口上，由于单总线为开漏所以需要外接一个 4.7K 的上拉电阻）；GND 为电源地；VDD 为外接供电电源输入端（在寄生电源接线方式时接地）。 DS18B20 内部结构主要由四部分组成：64 位光刻 ROM、温度传感器、非挥发的温度报警触发器 TH 和 TL、配置寄存器。

光刻ROM中的64位序列号是出厂前被光刻好的，它可以看作是该DS18B20的地址序列码。64位光刻ROM的排列是：开始8位是产品类型标号，接着的48位是该DS18B20自身的序列号，最后8位是前面56位的CRC校验码(循环冗余校验码)。光刻ROM的作用是使每一个DS18B20都各不相同，这样就可以实现一根总线上挂接多个DS18B20的目的。

温度传感器可完成对温度的测量，以12位转化为例，用16位符号扩展的二进制补码读数形式提供。

RAM数据暂存器，数据在掉电后丢失，DS18B20共9个字节RAM，每个字节为8位。如下图所示，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暂存器的第0和第1个字节。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，第2、3字节是用户EEPROM（常用于温度报警值储存）的镜像，用户用来设置最高报警和最低报警值，在上电复位时其值将被刷新。第4个字节则是用户第3个EEPROM的镜像，用来配置9~12位的转换精度（即分辨率）。第5、6、7个字节为计算机自身使用。第8个字节为前8个字节的CRC码。EEPROM 非易失性记忆体，用于存放长期需要保存的数据，上下限温度报警值和校验数据，DS18B20共3位EEPROM，并在RAM都存在镜像，以方便用户操作。

寄存器内容	字节地址
温度值低位 （LS Byte）	0
温度值高位 （MS Byte）	1
高温限值（TH）	2
低温限值（TL）	3
配置寄存器	4
保留	5
保留	6
保留	7
CRC校验值	8



配置寄存器：配置寄存器只有一个字节，如下图所示，低五位一直都是"1"，最高位为测试模式位，用于设置DS18B20在工作模式还是在测试模式，在出厂时该位被设置为0，用户不要去改动。R1和R0用来设置分辨率（出厂时被设置为12位），由R1，R0的不同组合可以配置为9位，10位，11位，12位的温度显示。这样就可以知道不同的温度转化位所对应的转化时间，四种配置的分辨率分别为0.5℃，0.25℃，0.125℃和0.0625℃。

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

温度的决定配置图如下图所示：

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

3、温度的读取：

DS18B20 在出厂时已配置分辨率为 12 位，如下图所示，读取温度时共读取 16 位，前 5 个数字为符号位，当前 5 位全为 1 时，读取的温度为负数；当前 5 位全为 0 时，读取的温度为正数，以把低 11 位的 2 进制转化为 10 进制后再乘以 0.0625 便为所测的温度。从下图还可以看出，低字节（LS BYTE）的低 4 位为小数位，高字节（MS BYTE）的低 3 位加上低字节高 4 位组成整数位。当符号位 S=0 时，直接将二进制位转换为十进制；当 S=1 时，先将补码变为原码，再计算十进制值。

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

举例如下：

温度/数据关系 表 2

温度 ℃	数据输出（二进制）	数据输出（十六进制）
+125	0000 0111 1101 0000	07D0h
+85	0000 0101 0101 0000	0550h
+25. 0625	0000 0001 1001 0001	0191h
+10. 125	0000 0000 1010 0010	00A2h
+0. 5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
- 0. 5	1111 1111 1111 1000	FFF8h
- 10. 125	1111 1111 0101 1110	FF5Eh
- 25. 0625	1111 1110 0110 1111	FE6Fh
- 55	1111 1100 1001 0000	FC90h

4、DS18B20 的初始化：

根据 DS18B20 的通讯协议，主机（单片机）控制 DS18B20 完成温度转换必须经过三个步骤：每一次读写之前都要对 DS18B20 进行复位操作，复位成功后发送一条 ROM 指令，最后发送 RAM 指令，这样才能对 DS18B20 进行预定的操作。复位要求主 CPU 将数据线下拉 500 微秒，然后释放，当 DS18B20 收到信号后等待 16～60 微秒左右，后发出 60～240 微秒的存在低脉冲，主 CPU 收到此信号表示复位成功。

（1） 先将数据线 DQ 置高电平“1”。

（2） 延时（该时间要求的不是很严格，但是尽可能的短一点）

（3） 数据线拉到低电平“0”。

（4） 延时 750 微秒（该时间的时间范围可以从 480 到 960 微秒）。

（5） 数据线拉到高电平“1”。

（6） 延时等待（如果初始化成功则在 15 到 60 微妙时间之内产生一个由 DS18B20 所返回的低电平“0”。据该状态可以来确定它的存在，但是应注意不能无限的进行等待，不然会使程序进入死循环，所以要进行超时控制）。

（7） 若 CPU 读到了数据线上的低电平“0”后，还要做延时，其延时的时间从发出的高电平算起（第（5）步的时间算起）最少要 480 微秒。

（8） 将数据线再次拉高到高电平“1”后结束。

初始化程序如下：

```
uchar Init_DS18B20()
{
    uchar status; //status 为 DS18B20 返回的状态
    DQ = 1;
    Delay(8);
    DQ = 0;
    Delay(90);
    DQ = 1;
```

```
    Delay(8);
    status=DQ;
    Delay(100);
    DQ = 1;
    return status;
}
```

5、DS18B20 的写操作：

（1） 数据线先置低电平“0”。
（2） 延时确定的时间为 15 微秒。
（3） 按从低位到高位的顺序发送字节（一次只发送一位）。
（4） 延时时间为 45 微秒。
（5） 将数据线拉到高电平。
（6） 重复上（1）到（6）的操作直到所有的字节全部发送完为止。
（7） 最后将数据线拉高。

写一字节程序如下：

```
void WriteOneByte(uchar dat)    //写字节 dat
{
    uchar i;
    for(i=0;i<8;i++)
    {
        DQ = 0;
        DQ = dat& 0x01;
        Delay(5);
        DQ = 1;
        dat >>= 1;
    }
}
```

6、DS18B20 的读操作:

（1）将数据线拉高“1”。
（2）延时 2 微秒。
（3）将数据线拉低“0”。
（4）延时 3 微秒。
（5）将数据线拉高“1”。
（6）延时 5 微秒。
（7）读数据线的状态得到 1 个状态位，并进行数据处理。
（8）延时 60 微秒。

读一字节程序如下：

```
uchar ReadOneByte()
{
    uchar i,dat=0;
    DQ = 1;
    _nop_();
    for(i=0;i<8;i++)
    {
```

```
            DQ = 0;
            dat >>= 1;
            DQ = 1;
            _nop_();
            _nop_();
            if(DQ)
            dat |= 0X80;
            Delay(30);
            DQ = 1;
        }
        return dat;
}
```

7、ROM 和 RAM 指令：

上面已经提到，主机（单片机）控制 DS18B20 完成温度转换必须经过三个步骤：每一次读写之前都要对 DS18B20 进行复位操作，复位成功后发送一条 ROM 指令，最后发送 RAM 指令，这样才能对 DS18B20 进行预定的操作。

其中，ROM 指令有 5 条，为 8 位长度，功能是对片内的 64 位光刻 ROM 进行操作。ROM 指令分别是读 ROM 数据、指定匹配芯片、芯片搜索、跳跃 ROM、报警芯片搜索。

ROM 指令表：

指 令	约定代码	功 能
读 ROM	33H	读 DS1820 温度传感器 ROM 中的编码（即 64 位地址） （只有当总线上只存在一个 DS18B20 的时候才可以使用此指令，如果挂接不止一个，当通信时将会发生数据冲突。）
符合 ROM（指定匹配芯片）	55H	发出此命令之后，接着发出 64 位 ROM 编码，访问单总线上与该编码相对应的 DS1820 使之作出响应，为下一步对该 DS1820 的读写作准备。
搜索 ROM	0FOH	用于确定挂接在同一总线上 DS1820 的个数和识别 64 位 ROM 地址。为操作各器件作好准备。（在芯片初始化后，搜索指令允许总线上挂接多芯片时用排除法识别所有器件的 64 位 ROM。）
跳过 ROM	0CCH	忽略 64 位 ROM 地址，直接向 DS1820 发温度变换命令。适用于单片工作。（这条指令使芯片不对 ROM 编码做出反应，在单芯片的情况之下，为了节省时间则可以选用此指令。如果在多芯片挂接时使用此指令将会出现数据冲突，导致错误出现。）
告警搜索命令	0ECH	执行后只有温度超过设定值上限或下限的片子才做出响应。（只要芯片不掉电，报警状态将被保持，直到再一次测得温度值达不到报警条件为止。）

RAM 操作指令共 6 条，同样为 8 位，存储器操作指令分别是写 RAM 数据、读 RAM 数据、将 RAM 数据复制到 EEPROM、温度转换、将存储器操作指令：在 ROM 指令发送给 EEPROM 中的报警值复制到 RAM、工作方式切换。RAM 操作指令的功能是命令 DS18B20 作什么样的工作，是芯片控制的关键。

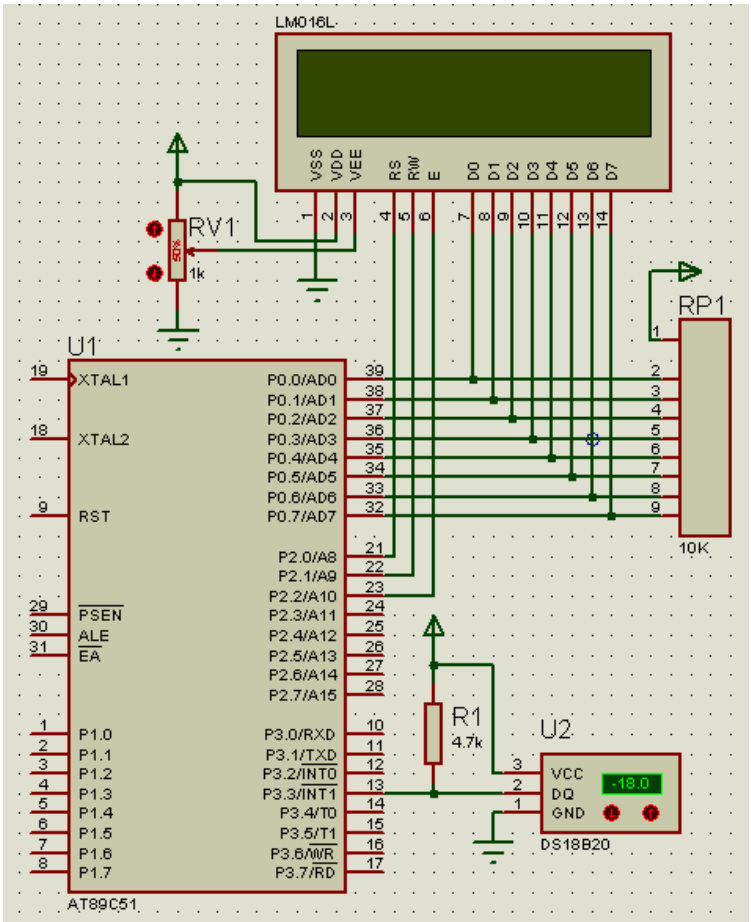
RAM 指令表：

指 令	约定代码	功 能
温度变换	44H	启动 DS1820 进行温度转换，12 位转换时最长为 750ms（9 位为 93.75ms）。结果存入内部 9 字节 RAM 的第 0，1 字节中。
读暂存器	BEH	读内部 RAM 中 9 字节的内容（**从地址 0 读起，每次读一字节，中途可停止，只读简单温度值则读前 2 个字节即可**）
写暂存器	4EH	发出向内部 RAM 的 3、4 字节写上、下限温度数据命令，紧跟该命令之后，是传送两字节的数据。
复制暂存器	48H	将 RAM 中第 3 、4 字节的内容复制到 EEPROM 中。
重调 EEPROM	B8H	将 EEPROM 中内容恢复到 RAM 中的第 3 、4 字节。
读供电方式	B4H	读 DS1820 的供电模式。寄生供电时 DS1820 发送“ 0 ”，外接电源供电 DS1820 发送“ 1 ”。

8、DS18B20 应用举例（一）

如下图所示：DQ 通过 4.7K 上拉电阻外接正电源(由于单总线为开漏所以需要外接一个 4.7K 的上拉电阻)，并连接单片机 P3.3 口。

本例中，1602LCD 显示 DS18B20 所测量的外部温度，调节 DS18B20 模拟改变外界温度时，新的温度将刷新显示在 LCD 上。



C 程序如下：

由于本例仅保存一位小数，温度小数位对照表 df_Table[]将 0000~1111 对应的 16 个不同的小数进行四舍五入，例如，当读取的温度低字节低 4 位为 0101 时，对应的温度应为 $2^{-2}+2^{-4}=0.3125\approx0.3$，因此数组第 5 个元素（对应 0101）的值为 3；又如，如果低 4 位为 0110，对应的温度为 $2^{-2}+2^{-3}=0.375\approx0.4$，因此，数组第 6 个元素（对应 0110）取值为 4。

```
#include <reg52.h>
#include <intrins.h>
#define uint unsigned int
#define uchar unsigned char
#define delay4us() {_nop_();_nop_();_nop_();_nop_();}   //12MHZ 系统频率下，延时 4us

sbit DQ = P3^3;
sbit LCD_RS = P2^0;
sbit LCD_RW = P2^1;
sbit LCD_EN = P2^2;

uchar code Temp_Disp_Title[]={"Current Temp : "};              //1602 液晶第一行显示内容
uchar Current_Temp_Display_Buffer[]={" TEMP:          "};   //1602 液晶第二行显示内容

uchar code df_Table[]={ 0,1,1,2,3,3,4,4,5,6,6,7,8,8,9,9 }; //温度小数位对照表

uchar CurrentT = 0;        //当前读取的温度整数部分
uchar Temp_Value[]={0x00,0x00};     //从 DS18B20 读取的温度值
uchar Display_Digit[]={0,0,0,0};     //待显示的各温度数位
bit DS18B20_IS_OK = 1;       //DS18B20 正常标志

void DelayXus(uint x)   //延时 1
{
    uchar i;
    while(x--)
    {
        for(i=0;i<200;i++);
    }
}

bit LCD_Busy_Check()   //LCD 忙标志，返回值为 1602LCD 的忙标志位，为 1 表示忙
{
    bit result;
    LCD_RS = 0;
    LCD_RW = 1;
    LCD_EN = 1;
    delay4us();
    result = (bit)(P0&0x80);
    LCD_EN=0;
```

```
    return result;
}

void Write_LCD_Command(uchar cmd)   //1602LCD 写指令函数
{
    while(LCD_Busy_Check());
    LCD_RS = 0;
    LCD_RW = 0;
    LCD_EN = 0;
    _nop_();
    _nop_();
    P0 = cmd;
    delay4us();
    LCD_EN = 1;
    delay4us();
    LCD_EN = 0;
}

void Write_LCD_Data(uchar dat)   //1602LCD 写数据函数
{
    while(LCD_Busy_Check());
    LCD_RS = 1;
    LCD_RW = 0;
    LCD_EN = 0;
    P0 = dat;
    delay4us();
    LCD_EN = 1;
    delay4us();
    LCD_EN = 0;
}

void LCD_Initialise()     //1602LCD 初始化
{
    Write_LCD_Command(0x01);
    DelayXus(5);
    Write_LCD_Command(0x38);
    DelayXus(5);
    Write_LCD_Command(0x0c);
    DelayXus(5);
    Write_LCD_Command(0x06);
    DelayXus(5);
}

void Set_LCD_POS(uchar pos)     //1602LCD 设置显示位置
```

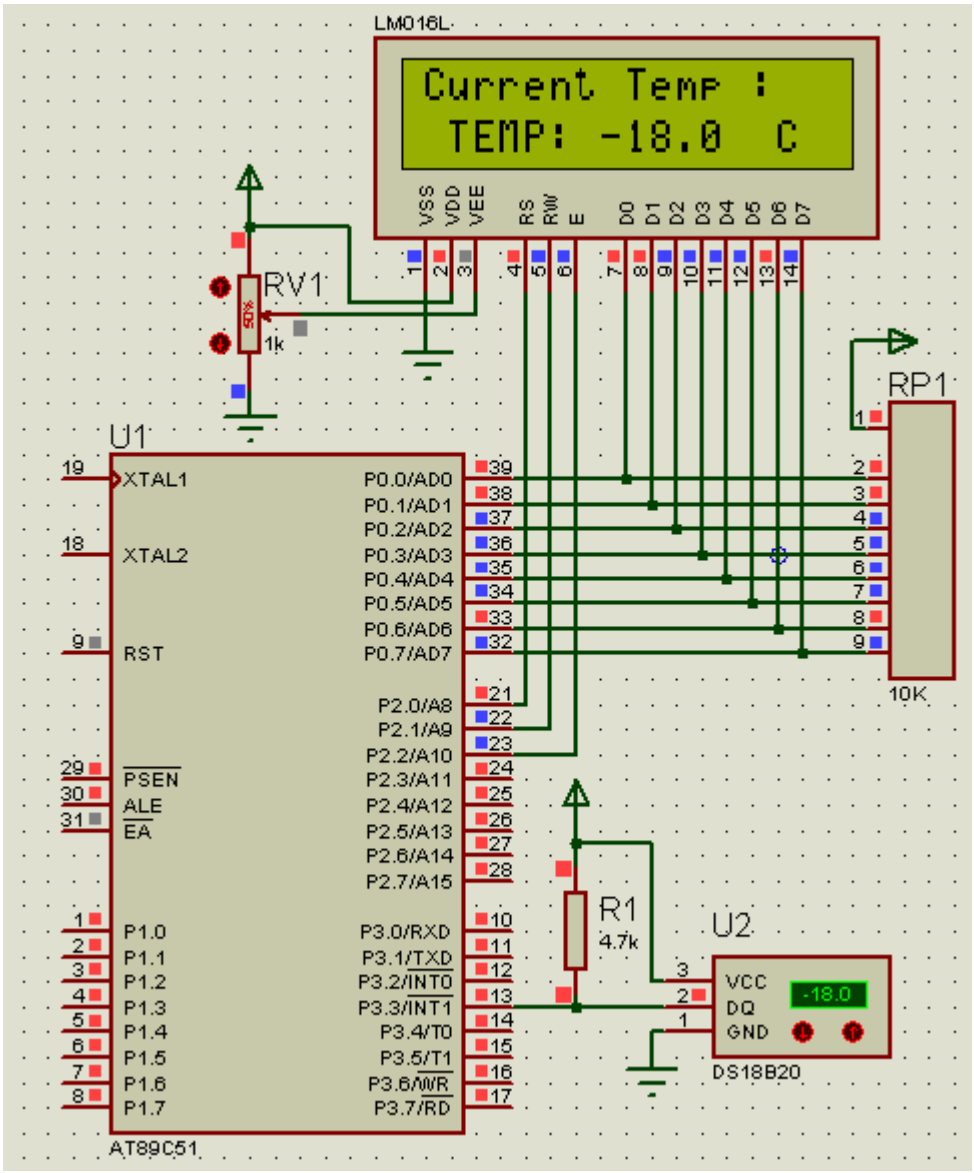
```
{
    Write_LCD_Command(pos|0x80);
}

void Delay(uint x)     //延时 2
{
    while(x--);
}

uchar Init_DS18B20()  //初始化（或者说复位）DS18B20
{
    uchar status;
    DQ = 1;
    Delay(8);
    DQ = 0;
    Delay(90);
    DQ = 1;
    Delay(8);
    status=DQ;Delay(100);
    DQ = 1;
    return status;
}

uchar ReadOneByte()       //从 DS18B20 读一字节数据
{
    uchar i,dat=0;
    DQ = 1;
    _nop_();
    for(i=0;i<8;i++)
    {
          DQ = 0;
          dat >>= 1;
          DQ = 1;
          _nop_();
          _nop_();
          if(DQ)
          dat |= 0X80;
          Delay(30);
          DQ = 1;
    }
    return dat;
}
```

```
void WriteOneByte(uchar dat)     //从 DS18B20 写一字节数据
{
    uchar i;
    for(i=0;i<8;i++)
    {
        DQ = 0;
        DQ = dat& 0x01;
        Delay(5);
        DQ = 1;
        dat >>= 1;
    }
}

void Read_Temperature()   //从 DS18B20 读取温度值
{
    if(Init_DS18B20()==1)      //DS18B20 故障
        DS18B20_IS_OK=0;
    else
    {
        WriteOneByte(0xcc);   //跳过序列号命令
        WriteOneByte(0x44);   //启动温度转换命令
       Init_DS18B20();   //复位 DS18B20（每一次读写之前都要对 DS18B20 进行复位操作）
        WriteOneByte(0xcc);   //跳过序列号命令
        WriteOneByte(0xbe); //读取温度寄存器
        Temp_Value[0] = ReadOneByte(); //读取温度低 8 位（先读低字节，再读高字节，）
        Temp_Value[1] = ReadOneByte();//读取温度高 8 位   （每次只能读一个字节）
        DS18B20_IS_OK=1;    //DS18B20 正常
    }
}

void Display_Temperature()   //在 1602LCD 上显示当前温度
{
    uchar i;
    uchar t = 150, ng = 0;   //延时值与负数标志
    if((Temp_Value[1]&0xf8)==0xf8)   //高字节高 5 位如果全为 1，则为负数，为负数时取反
    {                                //加 1，并设置负数标志为 1
        Temp_Value[1] = ~Temp_Value[1];
        Temp_Value[0] = ~Temp_Value[0]+1;
        if(Temp_Value[0]==0x00)     //若低字节进位，则高字节加 1
            Temp_Value[1]++;
        ng = 1;   //设置负数标志为 1
    }
    Display_Digit[0] = df_Table[Temp_Value[0]&0x0f];     //查表得到温度小数部分
```

```
    //获取温度整数部分（低字节低 4 位清零，高 4 位右移 4 位）+（高字节高 5 位清零，
    //低三位左移 4 位）
    CurrentT = ((Temp_Value[0]&0xf0)>>4) | ((Temp_Value[1]&0x07)<<4); /
    //将温度整数部分分解为 3 位待显示数字
    Display_Digit[3] = CurrentT/100;
    Display_Digit[2] = CurrentT%100/10;
    Display_Digit[1] = CurrentT%10;
    //刷新 LCD 缓冲        //加字符 0 是为了将待数字转化为字符显示
    Current_Temp_Display_Buffer[11] = Display_Digit[0] + '0';
    Current_Temp_Display_Buffer[10] = '.';
    Current_Temp_Display_Buffer[9]   = Display_Digit[1] + '0';
    Current_Temp_Display_Buffer[8]   = Display_Digit[2] + '0';
    Current_Temp_Display_Buffer[7]   = Display_Digit[3] + '0';
    if(Display_Digit[3] == 0)   //高位为 0 时不显示
        Current_Temp_Display_Buffer[7]   = ' ';
if(Display_Digit[2] == 0&&Display_Digit[3]==0) //高位为 0，且次高位为 0，则次高位不显示
        Current_Temp_Display_Buffer[8]   = ' ';
//负号显示在恰当位置
if(ng)
    {
        if(Current_Temp_Display_Buffer[8]   == ' ')
            Current_Temp_Display_Buffer[8]   = '-';
        else if(Current_Temp_Display_Buffer[7]   == ' ')
            Current_Temp_Display_Buffer[7]   = '-';
        else
            Current_Temp_Display_Buffer[6]   = '-';
    }
    Set_LCD_POS(0x00);   //第一行显示标题
    for(i=0;i<16;i++)
    {
        Write_LCD_Data(Temp_Disp_Title[i]);
    }
    Set_LCD_POS(0x40); //第二行显示当前温度
    for(i=0;i<16;i++)
    {
        Write_LCD_Data(Current_Temp_Display_Buffer[i]);
    }
  //显示温度符号
    Set_LCD_POS(0x4d);
    Write_LCD_Data(0x00);
    Set_LCD_POS(0x4e);
    Write_LCD_Data('C');
}
```

```
void main()     //主函数
{
    LCD_Initialise();
    Read_Temperature();
    Delay(50000);
    Delay(50000);
    while(1)
    {
        Read_Temperature();
        if(DS18B20_IS_OK)
            Display_Temperature();
        DelayXus(100);
    }

}
```

Proteus 仿真运行结果如下：



9、上述 Proteus 仿真文件下载地址：http://luoyong199092.qjwm.com/

10、参考文献

[1]彭伟.单片机 C 语言程序设计实训 100 例.北京:电子工业出版社.2009

[2]贾振国，许琳.智能化仪器仪表原理及应用.北京:中国水利水电出版社.2011