硬件课程设计报告

课　　题：温度测量与巡检装置

班　　级：电信提高 08 级

作　　者：胡千里　马 林

　　胡 晨　吴东骏

指导老师：汪小燕

课设评价：

课设成绩：

目 录

1. 实验任务

设计并制作一个温度测量与多路无线巡检装置，可测量传感器感知的接触表面温度并进行诸如巡检等功能扩展。

1.1 基本要求

1.1.1 温度测量范围为 0℃～45℃，具有温度数码显示功能，分辨率为 0.1℃。具有输入控制功能，可由外界输入温度超限报警门限。
1.1.2 具有温度超限报警功能，当温度超出指定温度，必须给出声或光提示信号。
1.1.3 进行多路温度巡检，显示当前巡检传感器的温度测量值；可根据输入选择工作在巡检或指定传感器测量模式。
1.1.4 能够实现多路温度巡检数据的无线传送以及接收无线控制指令进行温度测量并回送测量数据。

1.2 发挥部分

1.2.1 能记录并实时显示温度调节过程的曲线，显示的误差绝对值小于 2℃。
1.2.2 进行温度的自动调节控制，可调节范围为 5℃～35℃，最小设定分度为 1℃。
1.2.3 温度控制范围可由外界输入，当温度达到某一设定值并稳定后，装置接触表面的温度波动范围控制在±5℃以内。要求温度调控达到稳定状态时，必须给出声或光提示信号。
1.2.4 其他功能

2. 设计目标

2.1 近端单片机

实现 LED 把段数码管实时显示阈值温度。阈值温度包括上限截止温度和下线截止温度。阈值温度的精确度达小数点后一位。

能够调整上下限阈值温度，调整的精确度达小数点后一位小数。

实现 LCD7 位半数码管实时显示采集到的温度数据。采集的温度包括远端单片机采集的一路温度数据和二路温度数据，并且能在近端单片机显示温度来源的路数。显示的实时温度精确度达小数点后两位小数。

实现 LCD 显示器显示两路温度曲线以及直角坐标和坐标所能显示的温度范围，还能显示所显示温度曲线的路数来源，可以更改显示屏所显示的温度曲线。最重要的是，更换所显示的路数的时候，温度曲线能显示该路之前的温度曲线，而不

是只显示更换路数之后的温度曲线。

能够利用 NRF905 实现温度数据的收发，与远端单片机 MSP430F149 实现数据通信。

能够利用 5*3 矩阵键盘实现按键输入，并且能通过所输入的案件，达到以下目的：

1. 阈值温度的调整：调整上下限截止温度，并且所能调整的精度达到 0.1 摄氏度；
2. 更改所采集的温度路数，并以曲线和数值的形式显示在 LCD 与 LED 上；
3. 更换模式：温度阈值输入与工作模式切换；

能够使用蜂鸣器/LED 灯达到温度超过阈值报警的功能。

2.2 远端单片机

能够利用 NRF905 实现温度数据的收发，与近端单片机 MSP430F449 实现数据通信。

能够利用两路 DS19B20 实现多路温度采集，并通过 NRF905 将采集到得温度传送回近端单片机 MSP430F449，并且以分辨率达 0.01℃的形式显示出来，以及以连续温度曲线的形式显示出来。

能够利用 TEC 达到可移动温度控制的目的。其中温度控制模块一般成本比较高，所以这里重点降低成本.实现的方法是:

1. 利用电路,达到只使用 1 片温控芯片 TEC 同时达到制冷与加热的目的;
2. 利用 3.6 元地 9V 电池代替价格达 100 元的不可移动电源设备达到可移动目的与低成本的目的。

温度测量范围大于老师所要求的 0~45℃。

3. 系统实现功能

3.1基本功能实现情况：

3.1.1 两路 DS18B20 连接在远端单片机 MSP430F149，采集温度数据，温度测量范围包含 0℃~45 摄氏度，采集温度精确度为 0.0625℃，在近端单片机 MSP430F449 用 LCD 显示采集温度的路，以及温度，温度分辨率为 0.1 的℃。具有输入控制功能，

可由外界输入温度超限报警门限，报警方式为蜂鸣器声音报警，以及 LED 灯闪烁把噢噢经。

3.1.2　具有温度超限报警功能，初始阈值温度设为 15.0℃~33.5℃，当温度超出指定温度，蜂鸣器鸣叫，LED 灯闪烁，声光同时报警。

3.1.3　远端 MSP430F149 单片机连接两路温度巡检，显示当前巡检传感器的温度测量值；可根据近端 MSP430F449 单片机连接的按键 4，选择显示指定路采集的温度。巡检或指定传感器测量模式。

3.1.4　通过近端与远端单片机各连接一个 nRF905,实现两路温度巡检数据的无线传送以及接收无线控制指令进行温度测量并回送测量数据。

3.2 发挥部分实现情况：

3.2.1　近端单片机 MS430F449 连接(YXD12864)LCD 显示屏，能记录并实时显示温度调节过程的曲线, 每隔 600mS 跟新一个温度点,显示的温度即为采集获得的温度数据,LCD 显示分辨率为 0.5℃,所以显示的误差绝对值小于 2℃。

3.2.2　按键 1,2，1 为阈值升高 0.1℃，2 为阈值降低 0.1℃，3 为选择更改上限或是下限温度阈值，设定温度阈值。

4. 装置使用说明

使用前配置好 MSP430F449 的外围电路，把各个拨码开关拨到正确的位置，把 MSP430F449 的串口 0 外接计算机，并在计算机上配置好端口。

对 MSP430F449 加电，在电脑上运行程序。这时可以看到 MSP430F449 单片机上绘制的温度曲线和字符 LCD 上显示的第几路和温度值。初始默认显示第一路的温度，LED 显示关闭，高低报警阈值分别为 33.5℃和 15℃。

字符 LCD、点阵 LCD 随环境温度变化即时更新。按一下矩阵键盘的 KEY3 一下，会发现 LED 闪动。左三位显示温度的下限，右三位显示温度的上限，此时停止 DS18B20 的温度采集，字符 LCD、点阵 LCD 和电脑上的温度显示不变。按 KEY1 一次可以增加温度上限，按 KEY2 一次可以减少温度上限。再按一下 KEY3 就可以使用 KEY1 和 KEY2 调节温度下限。再按一下 KEY3 温度正常采集。

当温度正常采集时，按下 KEY4 可以切换测温路数，并会在字符 LCD 和点阵 LCD 上显示。切换时点阵 LCD 或重新运行程序的温度显示曲线会自动重新初始化。

当前路温度值超过设定的上下限温度范围时蜂鸣器会自动报警，LED 当温度重新落入正常范围或按下 KEY1 键修改上下限温度时蜂鸣器停止报警。

按键	功能
KEY1	温度阈值+0.1℃
KEY2	温度阈值-0.1℃
KEY3	上限温度改变模式/下限温度改变模式/温度采集模式
KEY4	第一路/第二路温度采集转换

图 1. 按键模块 4 大按键的功能

5. 系统设计方案

5.1原计划设计框图

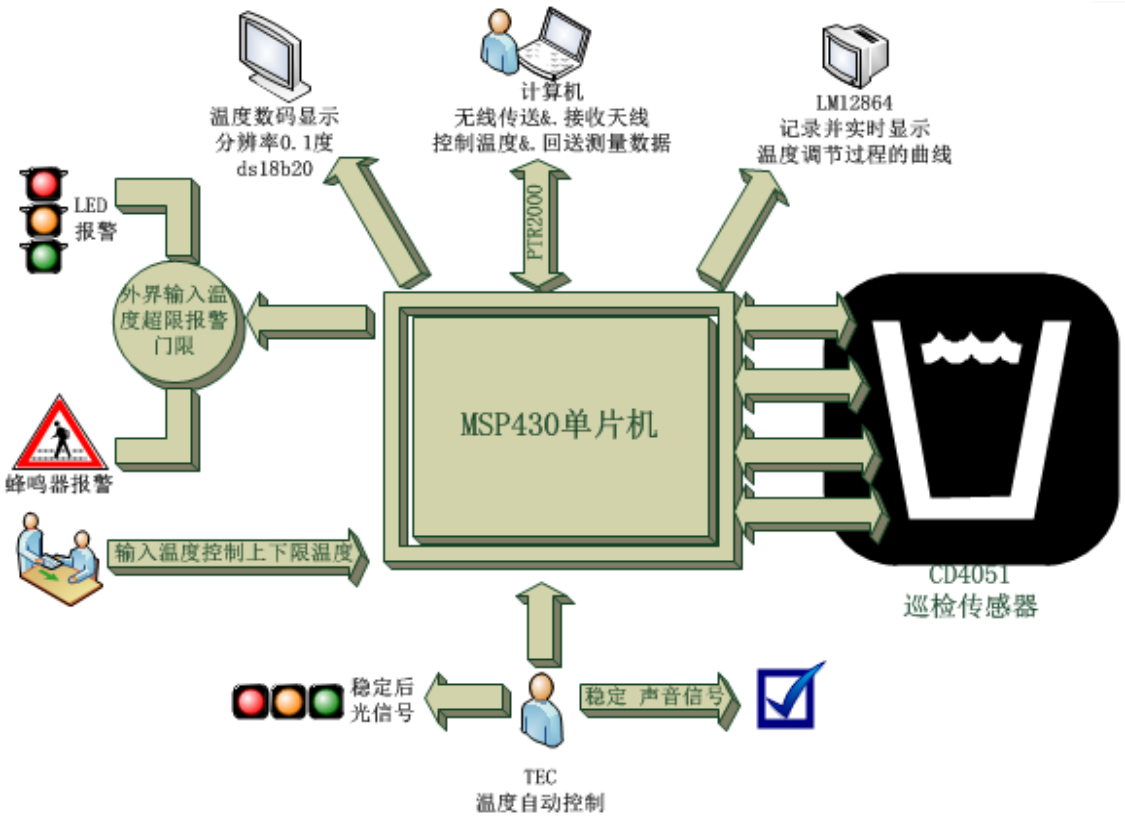

图 2. 原计划设计系统图

5.2最终实现设计框图

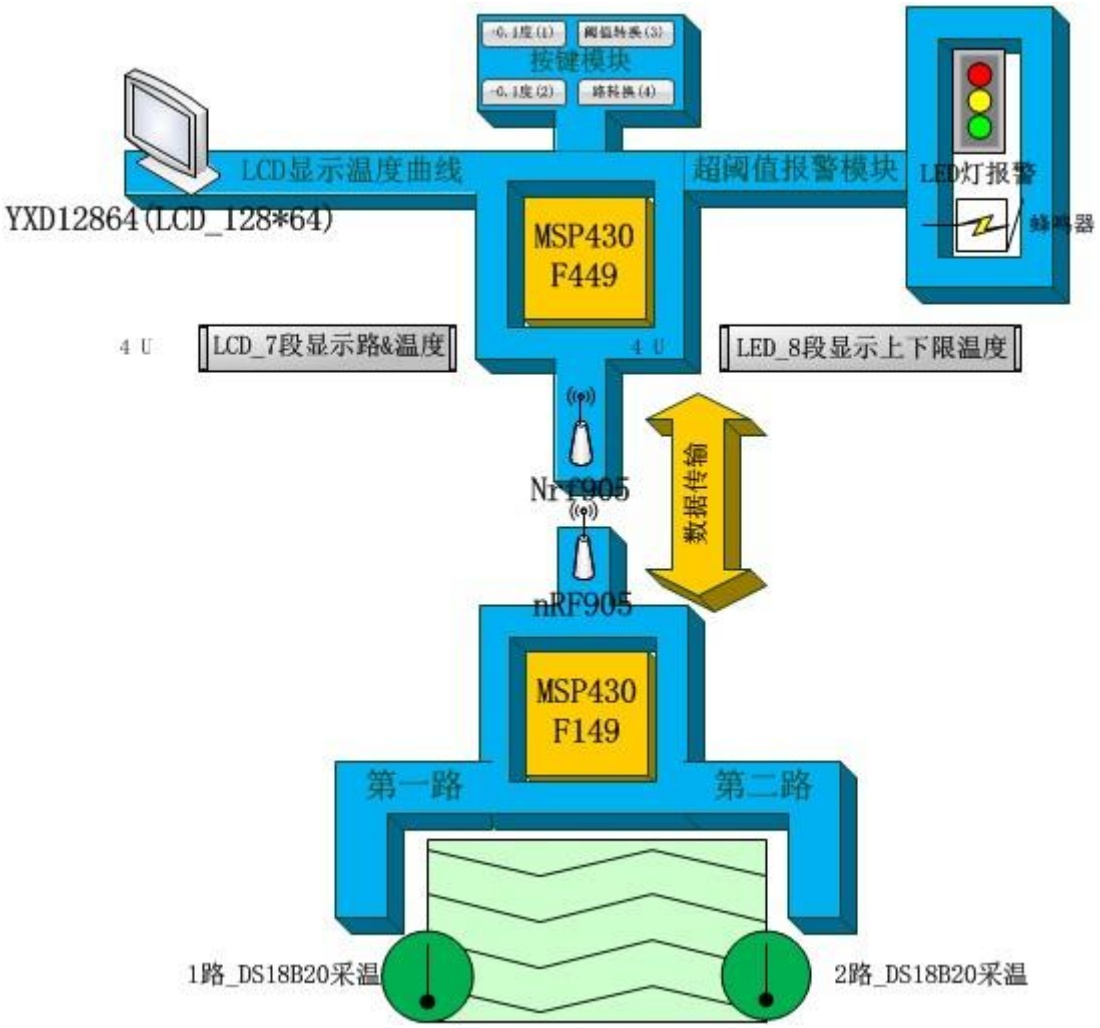

图 3. 最终计划设计系统框图

5.3开发环境

5.3.1 硬件&价格清单

货名	数量	价格/(元)
MSP430F449 试验箱(包括 LCD 屏,LED 数码管,LCD 数码管,5*3 矩阵按键,报警灯)	1	/
MSP430F149(远端单片机)	1	75
nRF905(无线收发器)	2	55+55
DS18B20(采温器件)	5	16（2 片）+20（3 片）
通用版	4	6+7+7+10
蜂鸣器(声音报警)	2	1+1
杜邦线	1	8
LCD(128*64)	1	60
SPD-05UDC-SL-C(继电器)	1	3.5
TEC1-12705(温控元件)	1	46
导线	2 米	1
LED 灯	3	0.6
9V 碳性电池	3	3.6+4+4
9V 碳性电池电池帽	3	1.5*3
音乐芯片	1	2
总花费	-	390.2

图 4. 硬件清单&价格

5.3.2 软件

Visual C++ 6.0 编程软件
IAR Embedded Workbench KickStart for MSP430 V4.21
Microsoft VISIO
Proteus Professional

6. 系统硬件&软件设计与实现

6.1按键模块

6.1.1 按键模块主要元件&功能

利尔达_嵌入式开发系统试验箱(MSP430F449 &.5*3 按键 &.LCD 数码显示 &.LCD 液晶屏显示&.LED 数码管显示*6 &.LED 灯报警)。

6.1.2 按键模块原理

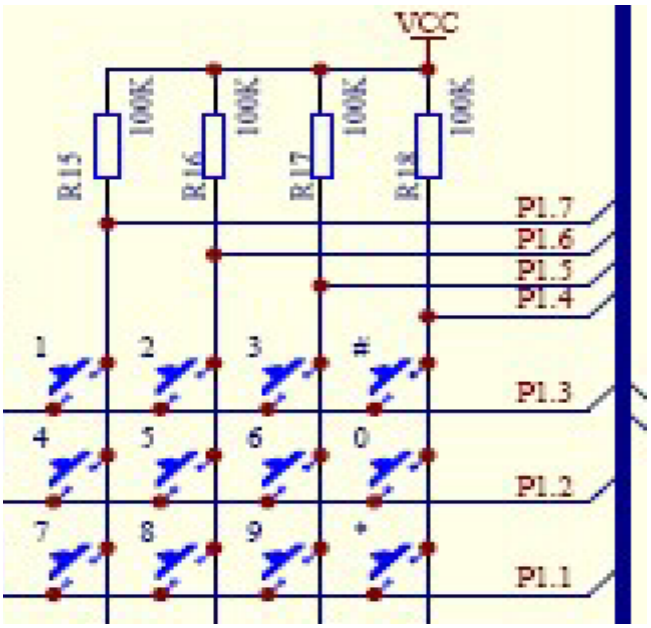

图 5. 按键原理图

按键模块分 2 大部分：

1. 判断键盘中有无键按下：

将全部行线 P1.1-P1.3 置低电平，然后检测列线的状态，只要有一列的电平为低，则表示键盘中有键被按下，而且闭合的键位于低电平线与 4 根行线相交叉的 4 个按键之中；若所有列线均为高电平，则表示键盘中无键按下。

2. 判断闭合键所在的位置：

在确认有键按下后，即可进入确定具体闭合键的过程。

依次将行线置为低电平（即在置某根行线为低电平时，其它线为高电平），当确定某根行线为低电平后，再逐行检测各列线的电平状态，若某列为低，则该列线与置为低电平的行线交叉处的按键就是闭合的按键。

实际程序中为：依次将 P1.1，P1.2，P1.3 唯一输出低电平，每将一位唯一输出低电平时，依次检测 P1.4，P1.5,P1.6,P1.7 电平，若检测到高电平，则跳过，继续检测下一个，并且将设置的变量 NUM++。

这样就可以得到 NUM 与矩阵键盘上按键的一一对应关系，再由 KEY_TABLE 的一一对应关系得到按键的数值。

以下是每个键按下对应的唯一的逻辑关系表：

1（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	2（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	3（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	#（仅 P1.2 输出低电平，且 P1.4 检测到低电平）
4（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	5（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	6（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	0（仅 P1.2 输出低电平，且 P1.4 检测到低电平）
7（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	8（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	9（仅 P1.1 输出低电平，且 P1.4 检测到低电平）	*（仅 P1.1 输出低电平，且 P1.4 检测到低电平）

图 6. 确定按键号的逻辑条件

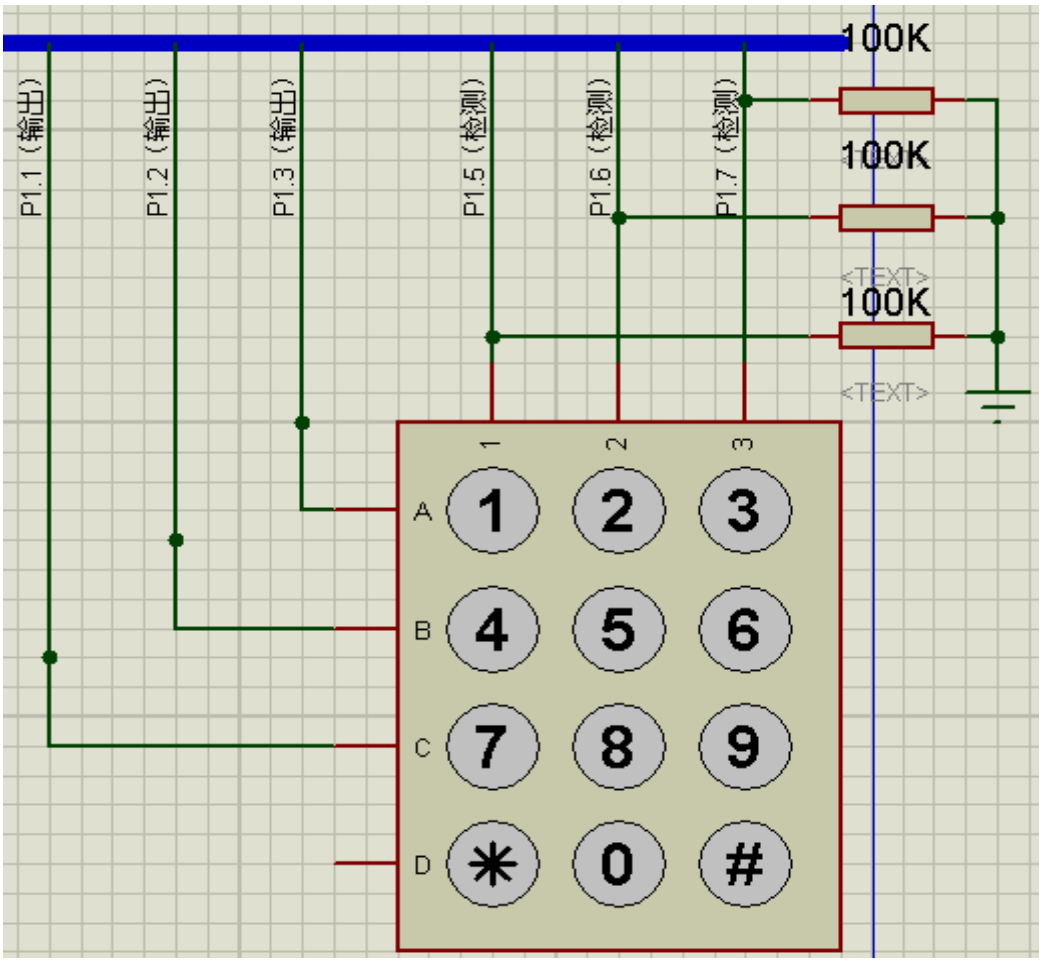

图 7. 矩阵按键 Proteus 仿真电路图

6.1.3 软件实现的思路框架

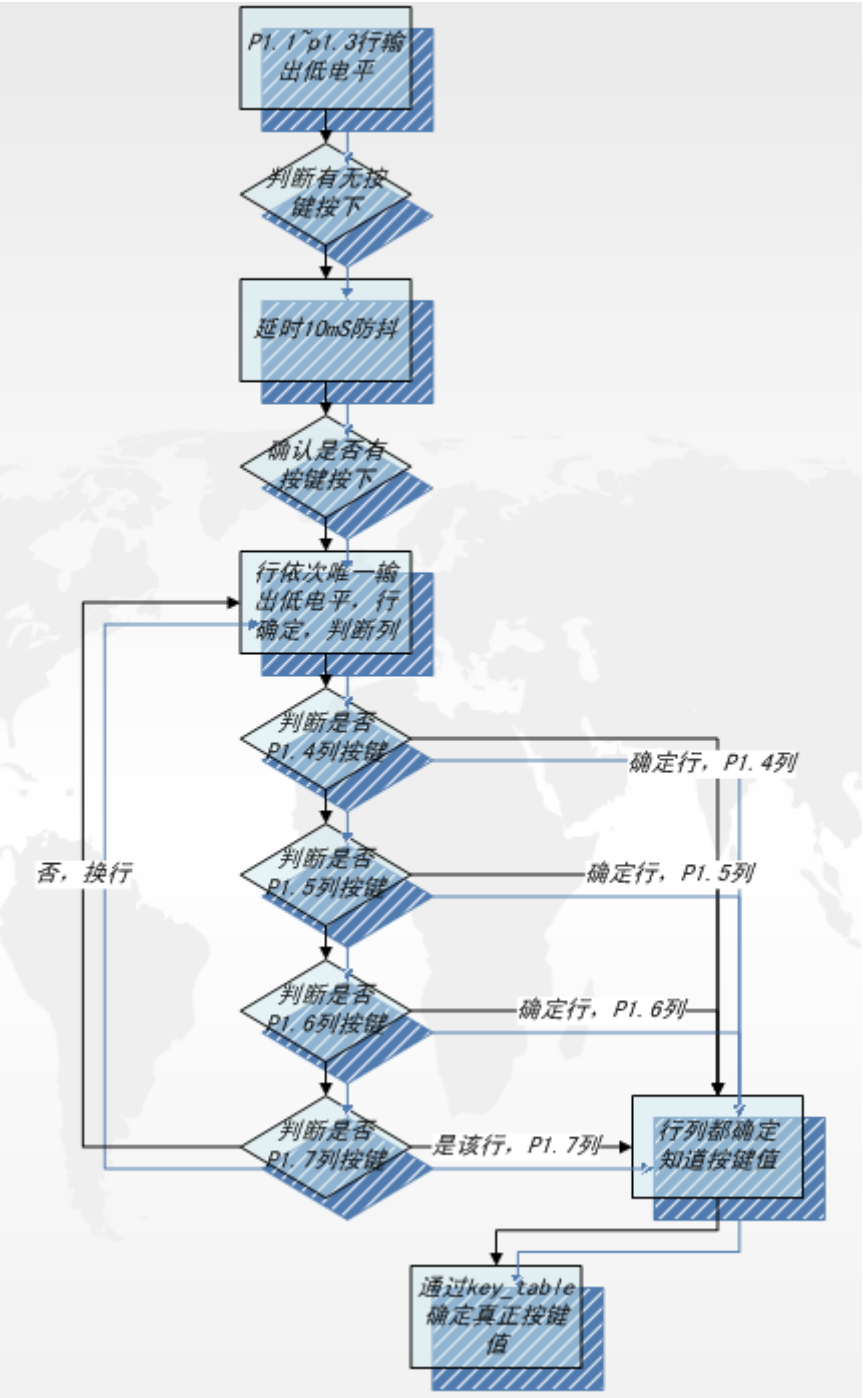

图 8. 按键部分软件流程图

6.1.4 主要子函数的具体代码

以下为键盘输入的代码实现：

```
void scan_key_deal(void)
```

```
{
        unsigned char counter;                          //行扫描
        unsigned char buffer;                           //p1 输入寄存器
        unsigned char com_byte;                         //行扫描寄存器
        unsigned char key_num;                          //键值查表
        P1DIR=0x0f;                                     //p1.4~~p1.7 键盘列扫描输入，
键盘行扫描输出
        com_byte=0xff;                                  //行扫描寄存器
        P1OUT&=~0x0E;                  //P1.1,P1.2,P1.3 皆低电平输出，判断有无键按下
        buffer=P1IN;
**********************分界线以上为判断有无按键按下**********************
***************分界线以下为当按键按下时，判断按键的键值***************
if(0xf0!=(buffer&0xf0))                                 //如果有键按下
  {
    P1OUT|=0x0E;                                        //P1.1,P1.2,P1.3 皆高电平输出
    key_num=0x00;                                       //key_num 为待查表得键值，先归零
    com_byte=0x02;                                      //从 p1.1 对应的第一行开始扫描
    for(counter=0x03;counter>0;counter--)               //此循环为 P1.1,P1.2,P1.3 行依次扫
描，确定被按下的按键
       {
        P1OUT&=~com_byte;                 //行 P1.1~P1.3 依次唯一低电平输出
        if(!(P1IN&BIT4))                  //行确定的情况下，判断按键值是否在 p1.4 列
            break;
            key_num++;
        if(!(P1IN&BIT5))                  //行确定的情况下，判断按键值是否在 p1.5 列
            break;
            key_num++;
        if(!(P1IN&BIT6))                  //行确定的情况下，判断按键值是否在 p1.6 列
            break;
            key_num++;
        if(!(P1IN&BIT7))                  //行确定的情况下，判断按键值是否在 p1.7 列
            break;
            key_num++;
        com_byte=com_byte<<1;             //左移一位，换上一行
       }
     flag|=key_press;                     //键盘按下标志,置为非零
     buffer=P1IN;
     key_buff=key_table[key_num];         //key_num 查表得到真正的键值
     return;
   }
}
```

真正的键值	0	1	2	3	4	5	6	7	8	9	#	*
key_num	4	11	10	9	7	6	5	3	2	1	8	0

图 9. Key_num 与真正按键值的对应表 1(PS, 实际只用到了加粗字体的 4 个按键)

1(key_num=11)	**2(key_num=10)**	**3(key_num=9)**	#(key_num=8)
4(key_num=7)	5(key_num=6)	6(key_num=5)	0(key_num=4)
7(key_num=3)	8(key_num=2)	9(key_num=1)	*(key_num=0)

图 10. Key_num 与真正按键值的对应表 2(PS, 实际只用到了加粗字体的 4 个按键)。

6.2温度控制模块

6.2.1 温度控制主要元件&功能

图 11. TEC1-12705 实物图

TEC1-12705：当直流电流通过 TEC（Thermoelectric Cooler）时，电流产生的热量会从 TEC 的一侧传到另一侧，在 TEC 上产生″热″侧和″冷″侧，表现为 TEC 一端吸热，一端放热，这就是 TEC 的加热与致冷原理。

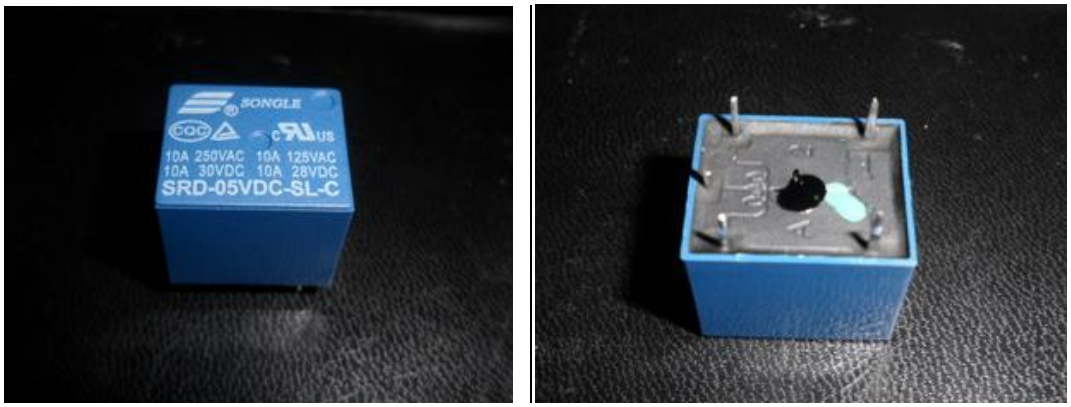

图 12. SRD-05VDC-SL-C 继电器实物图

SRD-05VDC-SL-C(继电器)：当输入量(激励量)的变化达到规定要求时，在电气输出电路中使被控量发生预定的阶跃变化的一种电器。

图 13. 9V 电池实物图(图中 2 电池我们买的 2 个电池外形一摸一样)

9V 电池 2 个：直流供电，两个电池各负责一个方向的电流。

6.2.2 温度控制模块原理

在远端单片机 MSP430F149，用 deltT 表示设定的需要达到的温度值与实际温度的差值。然后根据不同 deltT 所处的范围，选择所对应的处理：

deltT(温度差值)	(-∞，-10)	(-10，-6)	(-6，-3)	(-3.-1)	(-1，1)	(1,3)	(3,6)	(6,10)	(10，+∞)
if else 语句选择	超强加热	强加热	中加热	弱加热	保温	弱制冷	中制冷	强制冷	超强制冷
加热时间	50%	40%	35%	30%	25%	20%	15%	10%	1%
制冷时间	50%	60%	65%	70%	75%	80%	85%	90%	99%

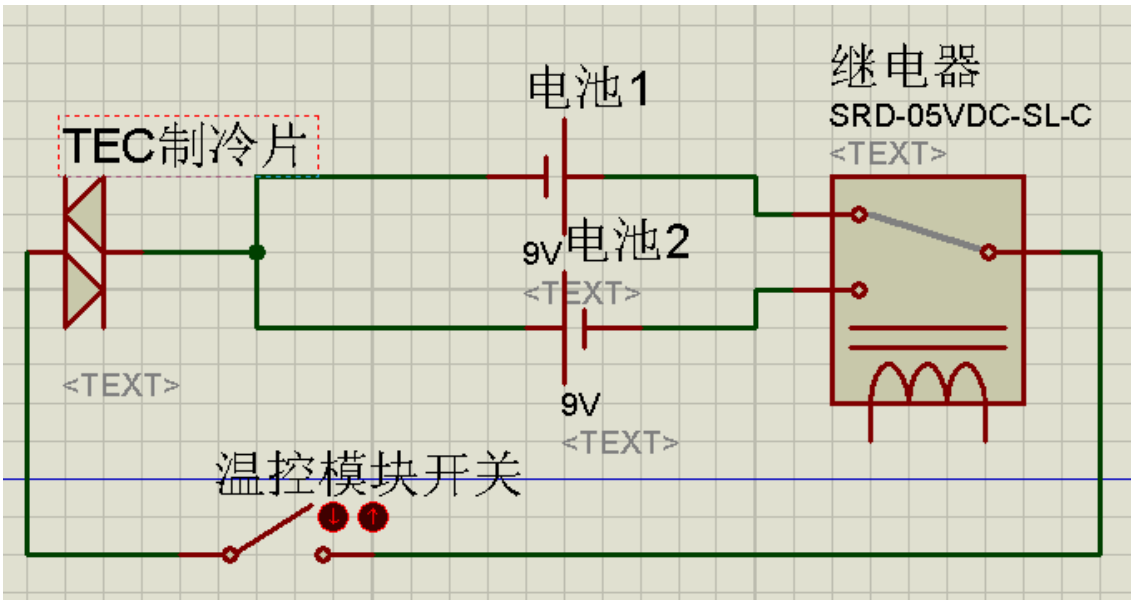

图 15. 制冷模块 Proteus 仿真电路图

6.2.3 软件实现的思路框架

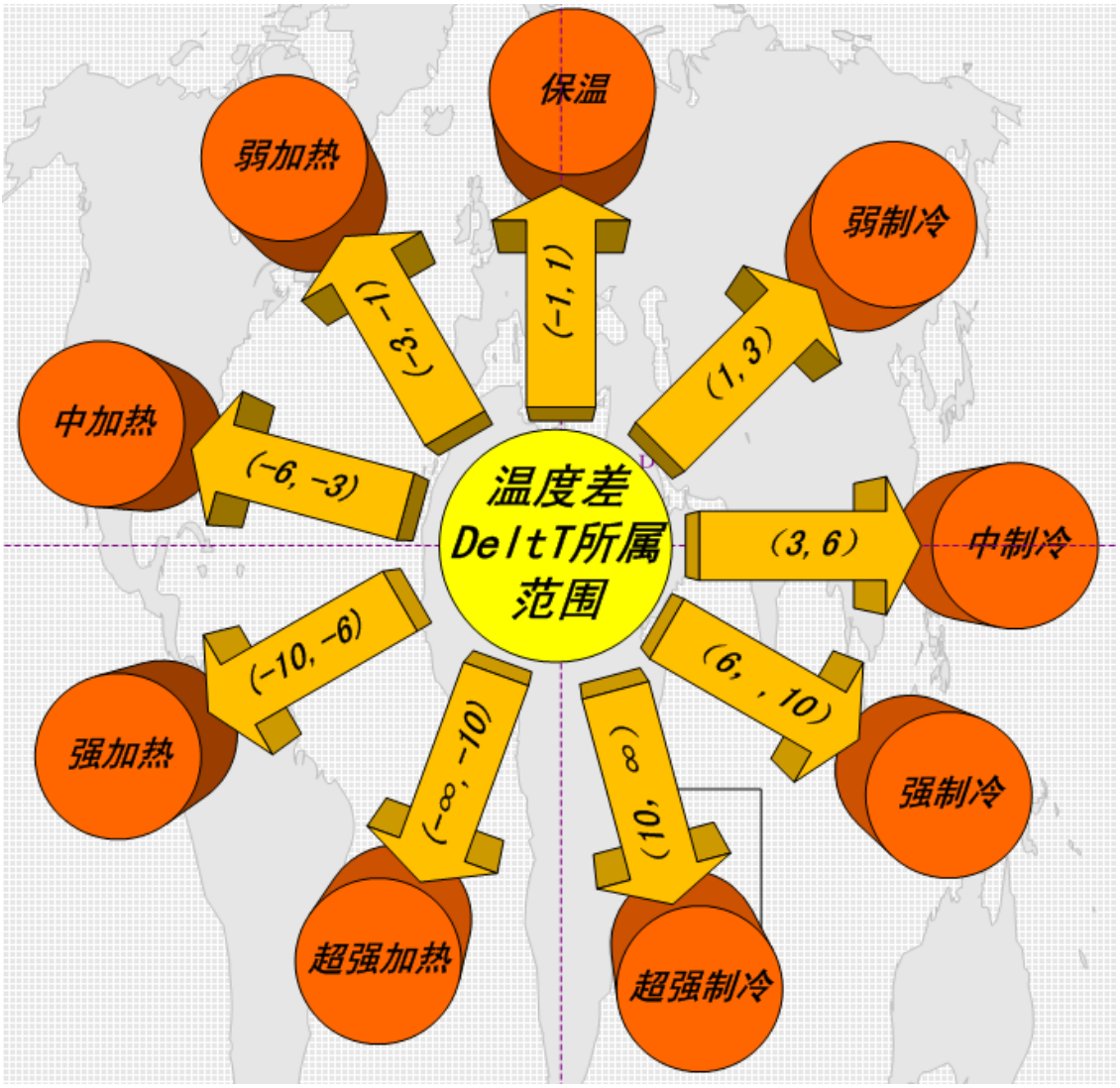

图 16. 温控部分软件流程图

6.2.4 函数的主要代码

```
Float   deltT;                          //设定温度与实际温度的温度差
while(1)
{
    if(deltT>10)
        Cooling4();                     //超强制冷
    else if(6<deltT&deltT<=10)
        Cooling3();                     //强制冷
    else if(3<deltT&deltT<=6)
```

```
        Cooling2();                         //中制冷
    else if(1<deltT&deltT<=3)
        Cooling1();                         //弱制冷
    else if(-1<deltT&deltT<=1)
        Keeping();                          //保温
    else if (-3<deltT&deltT<=-1)
        Heating1();                         //弱加热
    else if (-6<deltT&deltT<=-3)
        Heating2();                         //中加热
    else if (-10<deltT&deltT<=-6)
        Heating3();                         //强加热
    else
        Heating4();                         //超强加热
}
```

6.3LED/LCD 数码管显示模块

6.3.1 数字管显示模块主要元件&功能

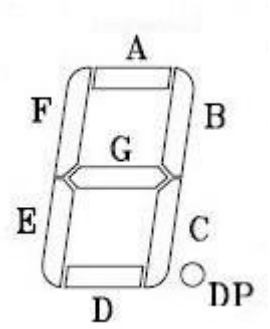

图 17.

LED 八段显示数码管：LED 数码管实际上是由七个发光管组成 8 字形构成的，加上小数点就是 8 个。这些段分别由字母 a,b,c,d,e,f,g,dp 来表示。当数码管特定的段加上电压后，这些特定的段就会发亮，以形成我们眼睛看到的字样了。
LCD 七位半显示数码屏：类似 LED8 段数码管,只是 LCD 不是发光二极管点亮,而是用 LCD 显示屏显示,它的码元转换表与 LED 数码管不同.

6.3.2 数字管显示模块原理

led 数码管是由多个发光二极管封装在一起组成“8”字型的器件，引线已在内部连接完成，

只需引出它们的各个笔划，公共电极。

led 数码管常用段数一般为 7 段有的另加一个小数点，然后按 7 段发光二极管点亮的不同小段，以显示 LED 1，2，3，4，5，6，8，9,0 数字等等。

LCD 显示代码表									LCD_table
0	1	2	3	4	5	6	7	8	9
0x7b	0x12	0x4f	0x1f	0x36	0x3d	0x7d	0x13	0x7f	0x3f

图 18.LCD 显示代码表

LED 显示代码表(共阴极)									LED_table
0	1	2	3	4	5	6	7	8	9
0xd7	0x14	0xcd	0x5d	0x1e	0x5b	0xdb	0x15	0xdf	0x5f

图 19.LED 显示代码表

6.3.3 软件实现的思路框架

6.3.4 主要子函数的具体代码

```
*****************************LED 显示函数*****************************
void led_display(unsigned char i)
{
        P3OUT=i;                                    // P3 位选
        P4OUT=0x02;                                 // P4.1 控制 P3 为位选
        P4OUT&=~0x02;                               // P4.1 清零

        P3OUT=~led_disp_bit;                        // P3 输出 8 段显示码
        P4OUT=0x01;                                 //P4.1 控制 P3 为显示码
        P4OUT&=~0x01;                               //P4 清零
        P3OUT=0x00;                                 //P3 清零,LED 灯全灭
}

*****************************LCD 显示函数*****************************
void display_lcd()
{
    LCDMEM[7]=LCDMEM[5]=lcd_table[10];              //
    if(dsnum==0x00)                                 // 若所选显示的温度为 1 路数据
      LCDMEM[6]=lcd_table[1];                       // 最左边显示 1,表示显示以路
```

```
    Else                                                    // 若所选显示的温度为 2 路数据
      LCDMEM[6]=lcd_table[2];                               // 最左边显示 2,表示显示二路

      LCDMEM[4]=lcd_table[abs(ResultTemp)/10000];    // 显示温度百位位数字
      LCDMEM[3]=lcd_table[abs(ResultTemp)/1000%10]; // 显示温度十位数字
      LCDMEM[2]=lcd_table[abs(ResultTemp)%1000/100]+0x80;//温度个位,0X80 是小数点
      LCDMEM[1]=lcd_table[abs(ResultTemp)%100/10];  // 显示温度 0.1 位数字
      LCDMEM[0]=lcd_table[abs(ResultTemp)%100%10];  // 显示温度 0.01 位数字
}
```

6.4温度超限报警模块

6.4.1 报警模块主要元件&功能

图 20.蜂鸣器

蜂鸣器: 蜂鸣器是一种一体化结构的电子讯响器，采用直流电压供电。

6.4.2 报警模块原理

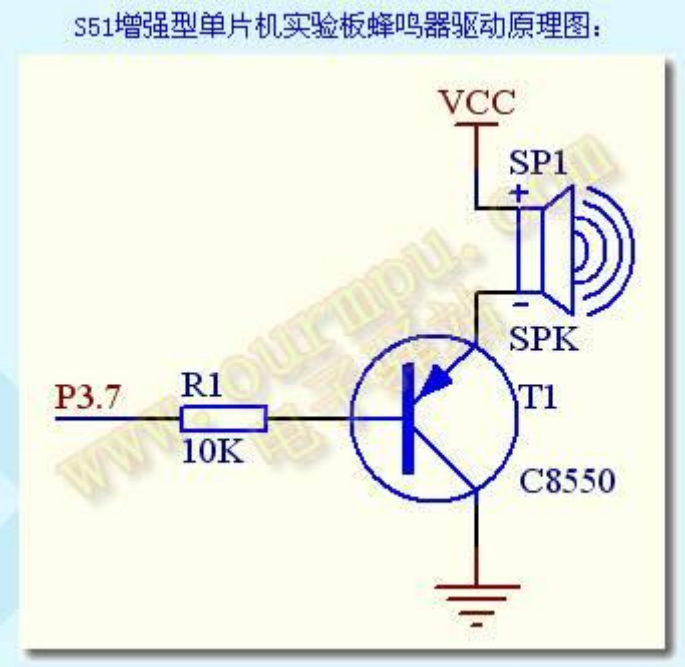

图 21.

采集温度>阈值温度上限或采集温度<阈值温度下限,输出两个高电平,驱动蜂鸣器鸣叫，LED 红灯亮。

其余时间，连接 LED 绿灯的引脚输出高电平。

6.4.3 报警模块思路框架

当一路或者二路所采集的温度，超过与之温度上限或者低于阈值温度下限时，蜂鸣器鸣叫，LED 灯红灯亮；温度恢复到阈值以内时，蜂鸣器停止鸣叫，LED 等绿灯亮。

6.4.4 主要子函数的具体代码

```
void alarm(void)          //报警函数
{
  P4DIR|=BIT3;            //连接蜂鸣器 P4.3 置为输出
  P4OUT|=BIT3;            //连接蜂鸣器 P4.3 输出高电平
  DelayNus(60000);        //延时,持续鸣叫
  DelayNus(60000);
}
```

6.5LCD 点阵显示温度曲线模块

6.5.1 LCD 点阵显示模块主要元件&功能

本系统使用的是ZJM12864BSBD点阵，ZJM12864BSBD 是一低功耗的点阵图形式LCD 模块。其特点如下：

1）显示格式：128 点（列） × 64 点（行）；
2）显示类型：STN 黄绿模式、半反半透、6:00 视角、正向显示；
3）驱动方式：1/64 占空比；
4）易与8、16 位的MPU 相连；
5）多功能指令；
6）加电自动复位；
7）控制芯片：KS0107B、KS0108B；
8）工作电压：＋3.3.0V ± 0.5V。

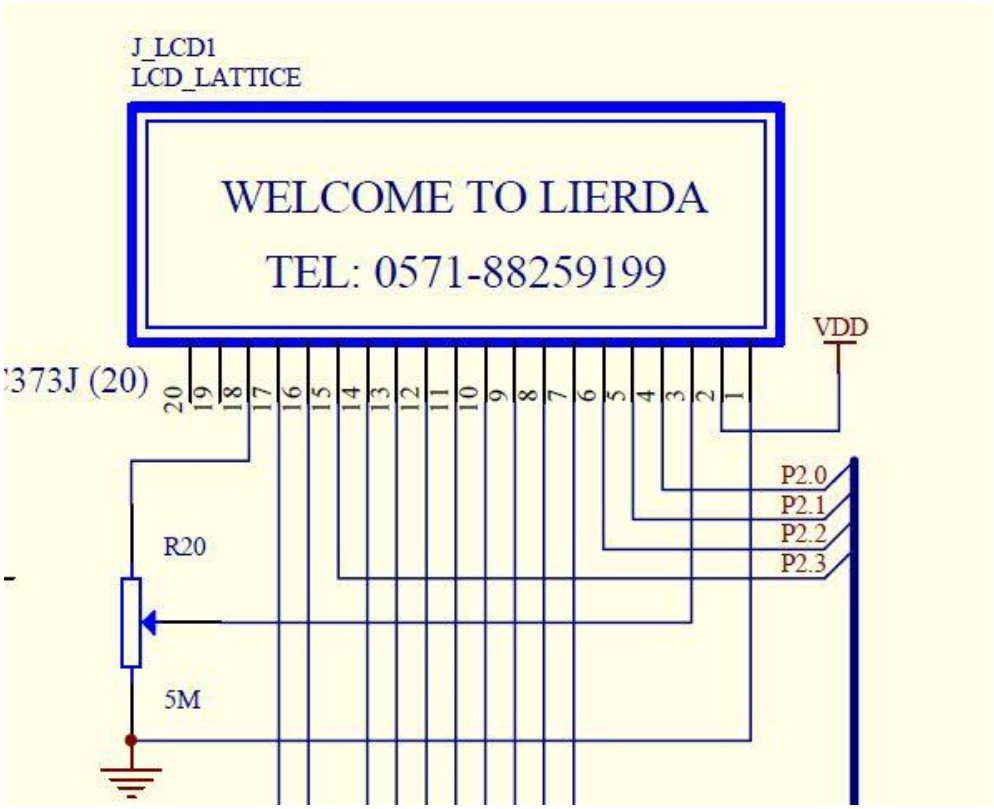

图 22.

其接口电路如上图所示。其中，数据端口使用的是 P3 端口。点阵 LCD 模块型号为 ZJM12864BSBD，这是一款低功耗的点阵图形式 LCD，显示格式为 128 点(列)×64 点(行)，具有多功能指令，很容易与 MPU 相连。其接口电路如上图所示。

方框图下图所示：

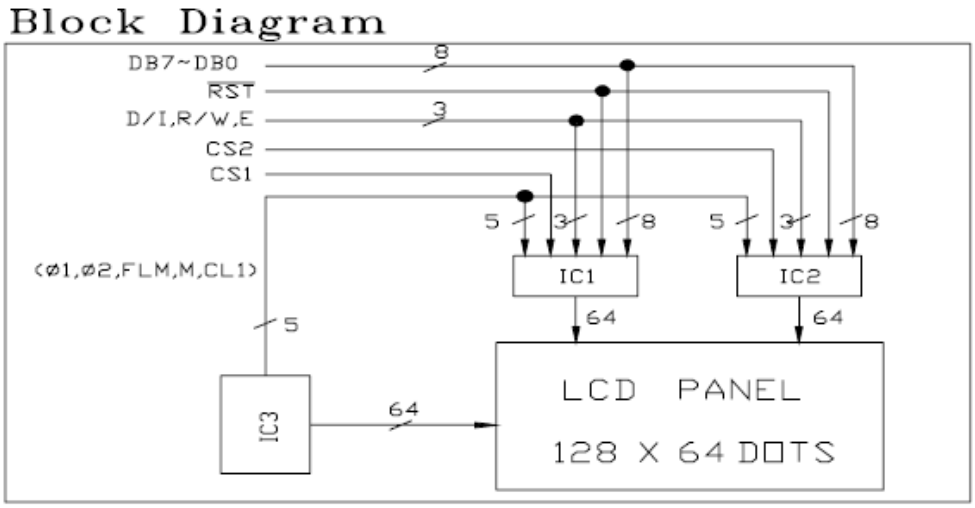

图 23.

可以看到，该 LCD 由两片 64*64 的点阵构成。LCD 的时序如下：

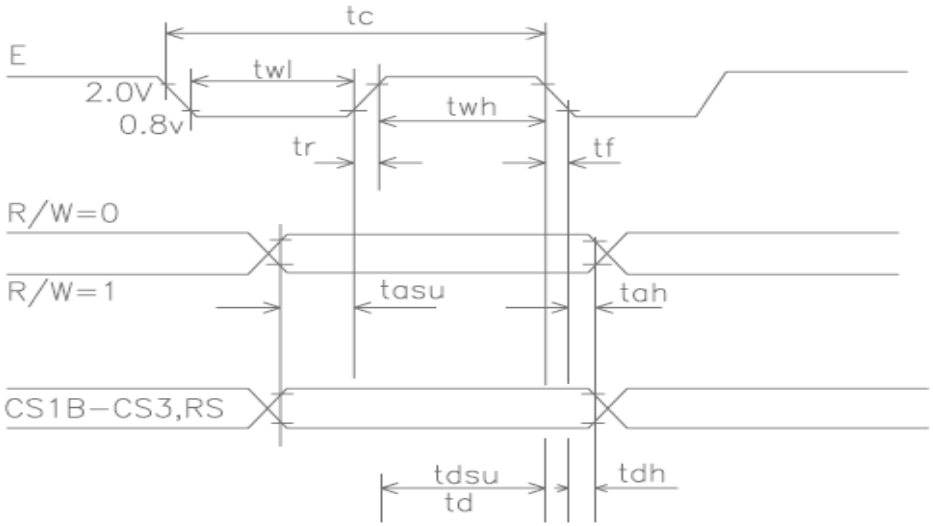

图 24.

其中，E 信号为时能信号，RS 为数据、指令选择信号，RW 为读写选择信号。

引脚功能如表 3 所示：

引脚号	符　号	电　平	功　能
1	V_{ss}	0	地
2	V_{dd}	5.0V	逻辑电压
3	V_0	—	LCD 驱动电压调节
4	RS	H / L	H：数据输入　L：指令输入
5	R/W	H / L	H：数据读出　L：数据写入
6	E	H、H→L	使能信号
7	DB_0	H / L	数据总线
8	DB_1	H / L	
9	DB_2	H / L	
10	DB_3	H / L	
11	DB_4	H / L	
12	DB_5	H / L	
13	DB_6	H / L	
14	DB_7	H / L	
15	CS1	H	片选信号 1
16	CS2	H	片选信号 2
17	RST	L	复位信号
18	VEE	—	-10V 输出端
19	NC	—	—
20	NC	—	—

图 25.　ZJM12864BSBD 引脚功能

6.5.2　LCD 点阵管显示模块原理

控制和显示指令如表 4 所示。

指　令	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	功　能
显示开/关	0	0	0	0	1	1	1	1	1	0/1	0：显示关 1：显示开
设定显示起始地址	0	0	0	1	段地址（0 ~ 63）						在段地址中设定Y显示地址
页地址设定(X地址)	0	0	1	0	1	1	1	页(0 ~ 7)			设定页地址计数器中 RAM 地址
显示起始行	0	0	1	1	显示起始地址（0 ~ 63）						设定屏幕显示的起始行
读状态	0	1	BUSY	0	ON/OFF	RESET	0	0	0	0	读状态: BUSY 0:允许输入指令 1:忙 ON/OFF 0:显示开 1:显示关 RESET 0: 正常 1: 复位
写显示数据	1	0	写　数　据								向 RAM 中写数据(Y地址自动加1)
读显示数据	1	1	读　数　据								RAM 中的数据读到数据总线上.

图 26.　控制与显示指令

LCD 的地址分布如下：

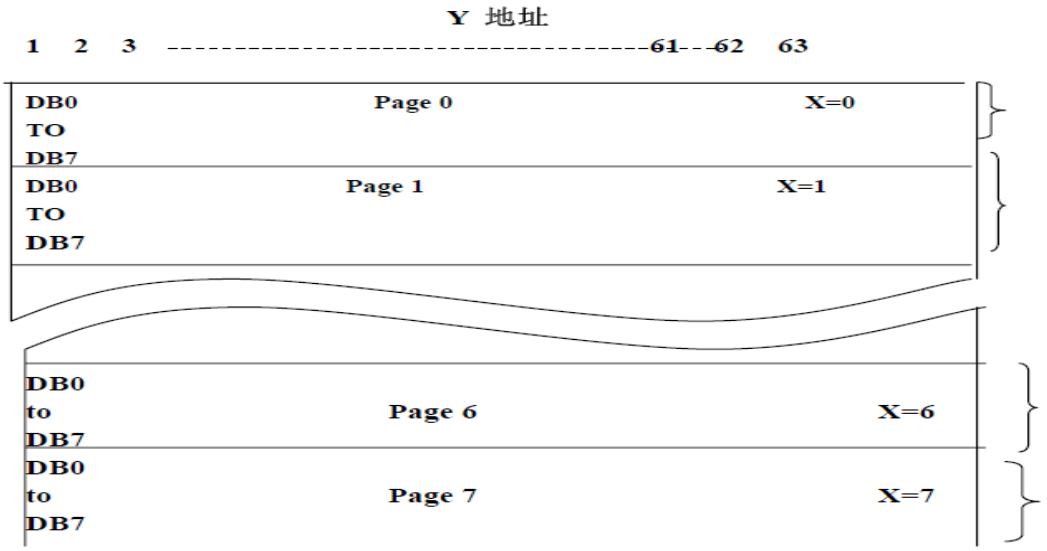

图 27.

由点阵方框图可知 JC1、JC2 控制点阵的列，每个控制64 列，共控制128 列；JC3 控制行，控制64 行，故组成128*64 的点阵。CS1、CS2 为JC1、JC2 的片选线；从点阵分布图可知，64 行分为8 页，每页8 行。则确定64*64 的某一点上的坐标为先确定列，在确定页，最后确定某一页的某一行。结合控制和指令表，得出在点阵中的某一个点上让其显示的步骤如下：

1) 先选则JC1 或JC2 或两个都选；
2) 选择数据类型（数据OR 指令、输入OR 输出）；
3) 显示起始行地址；
4) 显示开关；
5) 设定显示中Y 起始地址；
6) 设定显示中页地址。

送显示数据时先设定 X 地址。由于 Y 地址自动加 1，因此在显示一页的数据时，只需要设定一次 Y 地址，设为起始地址，即为 0xB8。

1.汉字的显示

要显示汉字，首先需要知道汉字对应的点阵码。下面是汉字的点阵码示意图：

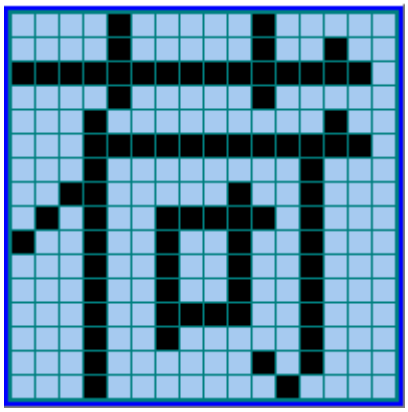

图 28.

这是一个 16*16 的点阵汉字。采用纵向 8 点下高位的扫描方式。可以看到，第一个数字是 0x04。整个汉字的点阵码是：

0x04,0x04,0x84,0xF4,0x2F,0x24,0x24,0x24,
0x24,0xA4,0x2F,0x24,0xE4,0x36,0x24,0x00,

0x02,0x01,0x00,0xFF,0x00,0x00,0x3F,0x11,

0x11,0x1F,0x41,0x80,0x7F,0x00,0x00,0x00

由于要显示的汉字是 16*16，而 LCD 的一页只有 8 行，因此一个汉字需要在两页上显示。于是先显示汉字的上半部分，然后显示下半部分。具体说来，先送一个页地址，接下来送一个列地址，然后发送显示数据。然后处理汉字的第二部分。

2.曲线的显示

显示曲线时，将右片视为一个整体。定义一个一维数据数组，大小为 512（8 页 64 列）。要显示的内容是一条分界线，Y 轴，X 轴和采样数据。点阵与数组的关系如下：

0	1	2	……	……
64	65			
128				
……				
……				

图 29

显示分界线，即将位置为 0，64，128……的数据赋值为 0xFF。显示 Y 轴，即将 72,136……的数据赋值为 0xFF。显示 X 轴，即将 392~439 的数据赋值为 0x80。

显示采集的数据时，首先算出数据在哪一页，然后算出在页中的哪一行。从而得到显示位置和显示数据。

6.5.3 软件实现的思路框架

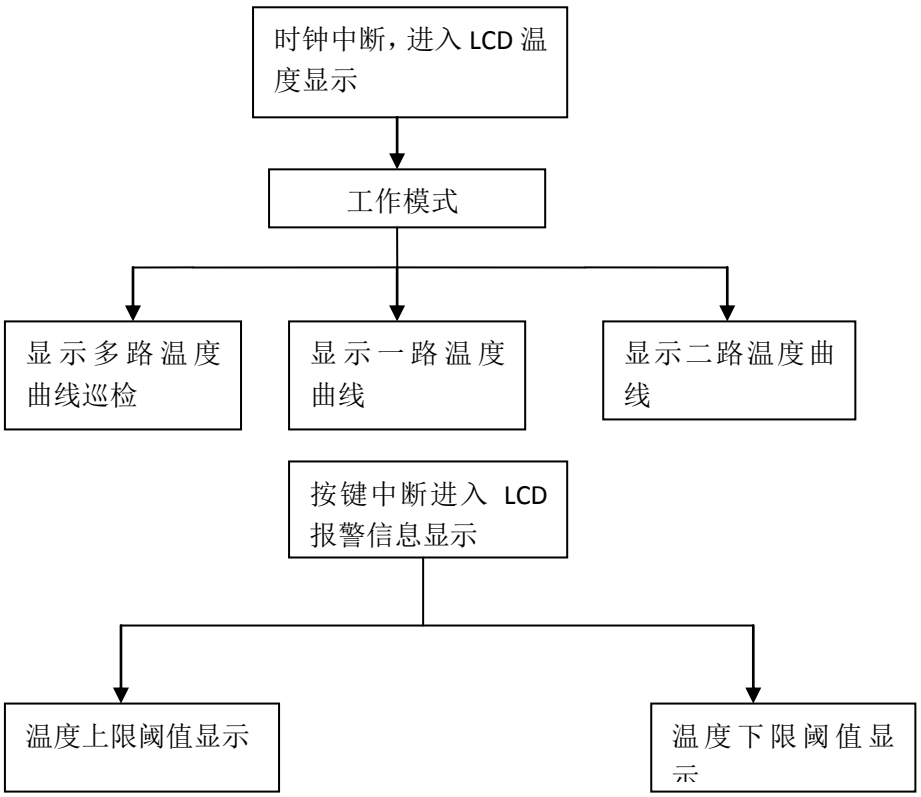

图 30.

6.5.4 主要子函数的具体代码

1.LCD 初始化函数：
void port_ini(void)
2.LCD 写命令函数：
  void write_command(BYTE nByte)
3.写 char 函数
void write_char(BYTE nByte, BYTE CS1, BYTE CS2)
4.清屏函数
void clear_lcd(void)
5.显示汉字函数
void display_hz(BYTE *chr, BYTE nRow, BYTE nCol)
6.显示图标函数
void display_16_16_icon(BYTE *chr, BYTE nRow, BYTE nCol)
7.显示温度点函数
void Display_64_1(BYTE *chr, BYTE nCol)
8.显示汉字函数
void display_num(BYTE *chr, BYTE nRow, BYTE nCol)
9.显示数字函数
void display_axisnum(void)
10. 显示纵坐标轴
void display_axisv(void)
11.显示横坐标
void display_axish(void)
3. 时钟中断 LCD 温度显示曲线函数

```
#pragma vector=BASICTIMER_VECTOR
__interrupt void basic_timer (void)
{
      display_lcd();

       if((!channelSwitchCount) && (!chSwitchMode))
            channel=~channel;

          unsigned char i=0;
            for(i=0;i<89;i++)
            {
               temprature_1[i]=temprature_1[i+1];
               temprature_2[i]=temprature_2[i+1];
            }
             temprature_1[89]=t_1*2;
```

```
            temprature_2[89]=t_2*2;
          display_info_disp();
                    if(channel) display_16_16_icon((BYTE *)&ER[0],1,110);
             else display_16_16_icon((BYTE *)&YI[0],1,110);
          if(channel==0)
               for(i=0;i<90;i++)
               {
                  BYTE CHARX[8]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
                  unsigned char a=7-(temprature_1[i]-40)/8;
                  unsigned char b=(temprature_1[i]-40)%8;
                  CHARX[a]=0X80;
                  CHARX[a]>>=b;
                  CHARX[7]|=0x80;
                  Display_64_1((BYTE *)CHARX,i+15);
               }
        else
            for(i=0;i<90;i++)
               {
                  BYTE CHARX[8]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
                  unsigned char a=7-(temprature_2[i]-40)/8;
                  unsigned char b=(temprature_2[i]-40)%8;
                  CHARX[a]=0X80;
                  CHARX[a]>>=b;
                  CHARX[7]|=0x80;
                  Display_64_1((BYTE *)CHARX,i+15);
               }
      if(!chSwitchMode)
                  channelSwitchCount= (channelSwitchCount<9)?(channelSwitchCount+1):0;
}
```

6.6 温度测量模块

6.6.1 温度测量模块主要元件&功能

温度测量模块，我用的是 DS18B20 芯片来实现温度的实时采集。下面是该器件的实体图片:

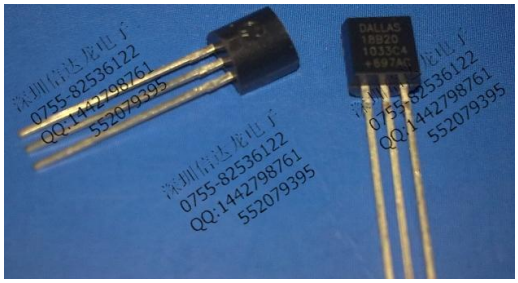

图 31. DS18B20

由于该器件是采用独特的单线接口方式，DS18B20 在与微处理器连接时仅需要一条口线即可实现微处理器与 DS18B20 的双向通讯，这样极大的节省了我们的单片机有限的引脚资源。并且是电路设计与连接特别的简单。在硬件上简化了电路。这是我选择该器件的原因之一。

第二个原因是：测量结果直接输出数字温度信号，以"一 线总线"串行传送给 CPU，同时可传送 CRC 校验码，具有极强的抗干扰纠错能力，这样我们处理接收到的数据特别方便，只需要通过一定的函数就可以把数字温度信号转换成实时温度值。

第三个原因是因为他比较便宜，相比其它器件，它体积小，而且只需要 7 元钱即可买一个。通过上述三个方面的考虑，我决定用 DS18B20 来作为我温度数据测量与采集器件。

1.DS18B20 的外形及管脚排列如下图 32:

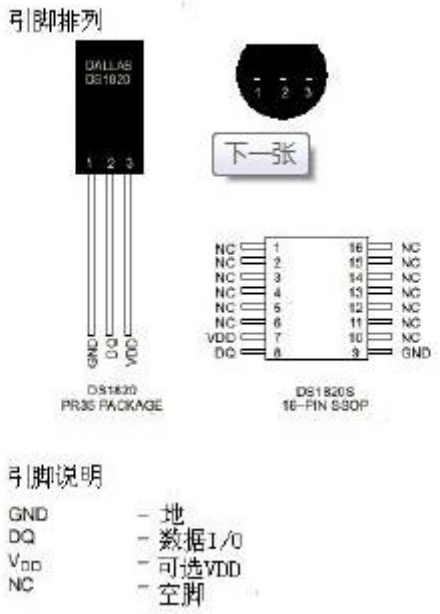

图 32.

6.6.2 温度测量模块原理

DS18B20 内部结构图如下图，图 33：

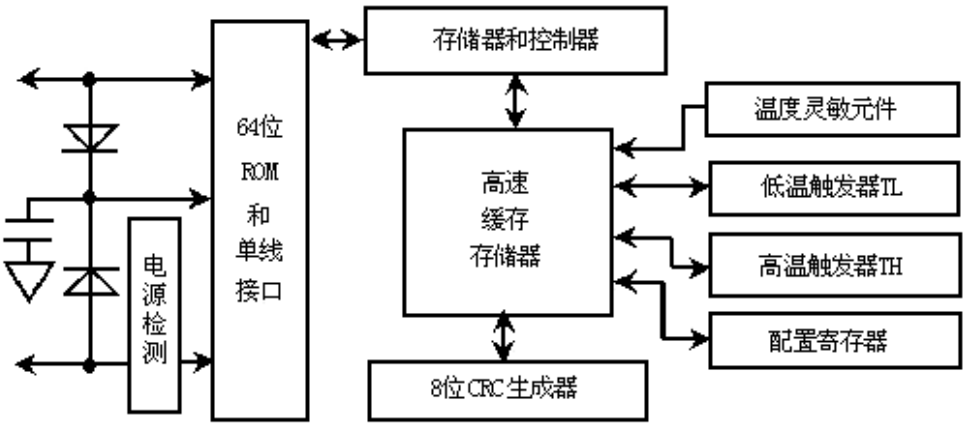

图 33

3.DS18B20 工作模式

DS18B20 的读写时序和测温原理与 DS1820 相同，只是得到的温度值的位数因分辨率不同而不同，且温度转换时的延时时间由 2s 减为 750ms。 DS18B20 测温原理如图 3 所示。图中低温度系数晶振的振荡频率受温度影响很小，用于产生固定频率的脉冲信号送给计数器 1。高温度系数晶振 随温度变化其振荡率明显改变，所产生的信号作为计数器 2 的脉冲输入。计数器 1 和温度寄存器被预置在－55℃所对应的一个基数值。计数器 1 对 低温度系数晶振产生的脉冲信号进行减法计数，当计数器 1 的预置值减到 0 时，温度寄存器的值将加 1，计数器 1 的预置将重新被装入，计数器 1 重 新开始对低温度系数晶振产生的脉冲信号进行计数，如此循环直到计数器 2 计数到 0 时，停止温度寄存器值的累加，此时温度寄存器中的数值即 为所测温度。图 3 中的斜率累加器用于补偿和修正测温过程中的非线性，其输出用于修正计数器 1 的预置值。

6.6.3 软件实现的思路框架&具体代码

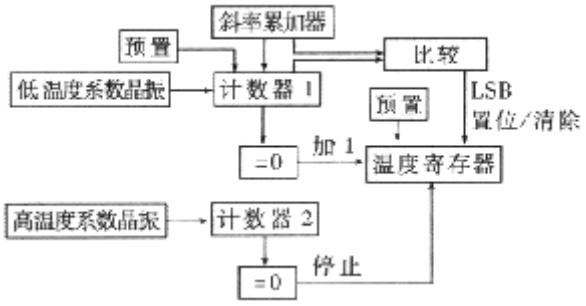

图 34

4.单总线数据传输协议

单总线通信协议中存在两种写时隙：写 0 写 1。主机采用写 1 时隙向从机写入 1，而写 0 时隙向从机写入 0。所有写时隙至少要 60us，且在两次独立的写时隙之

间至少要 1us 的恢复时间。两种写时隙均起始于主机拉低数据总线。产生 1 时隙的方式：主机拉低总线后，接着必须在 15us 之内释放总线，由上拉电阻将总线拉至高电平；产生写 0 时隙的方式为在主机拉低后，只需要在整个时隙间保持低电平即可（至少 60us）。在写时隙开始后 15~60us 期间，单总线器件采样总电平状态。如果在此期间采样值为高电平，则逻辑 1 被写入器件；如果为 0，写入逻辑 0。下图为写时隙（包括 1 和 0）时序

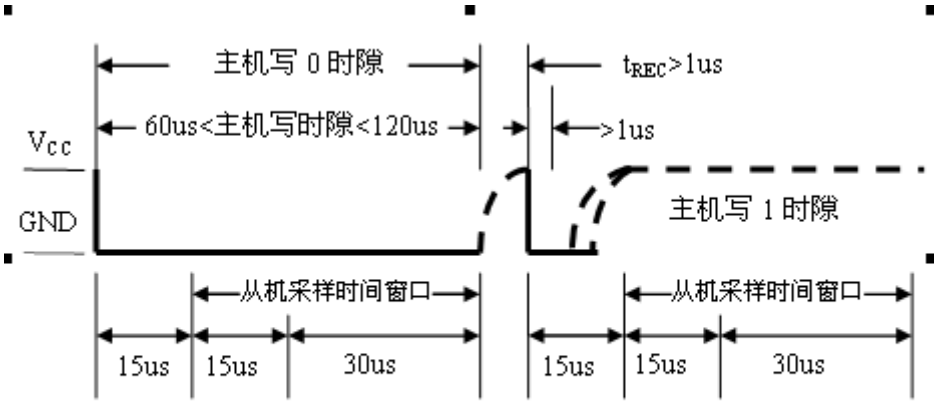

图 1-1 单总线通信协议中写时隙时序图

上图中黑色实线代表系统主机拉低总线，黑色虚线代表上拉电阻将总线拉高。下面是写函数：

```
void ds18b20_1_write(char Date)
{
   char i;
      DS18B20_1_OUT;//将 P6.7 置为输出端口
       for(i=0;i<8;i++)//将一个 BYTE 的 8 为一次由低位到高位，依次写到总线上
       {
          OUT_DS18B20_1_0;
         // DS1820_delay(6);
          if(Date & 0x01) //判断 Date 最低位是否为 0，如果是则 P6.7 输出 1，否则输出 0
          {
             OUT_DS18B20_1_1;
          }
          else
          {
             OUT_DS18B20_1_0;
          }
          DS1820_delay(4);
          OUT_DS18B20_1_1;
          Date>>=1;
       }
          DS1820_delay(8);
}
```

对于读时隙，单总线器件仅在主机发出读时隙时，才向主机传输数据。所有主

机发出读数据命令后，必须马上产生读时隙，以便从机能够传输数据。所有读时隙至少需要 60us，且在两次独立的读时隙之间至少需要 1us 恢复时间。每个读时隙都由主机发起，至少拉低总线 1us。在主机发出读时隙后，单总线器件才开始在总线上发送 1 或 0。若从机发送 1，则保持总线为高电平；若发出 0，则拉低总线。

当发送 0 时，从机在读时隙结束后释放总线，由上拉电阻将总线拉回至空闲高电平状态。从机发出的数据在起始时隙之后，保持有效时间 15us，因此主机在读时隙期间必须释放总线，并且在时隙起始后的 15us 之内采样总线状态。

下图给出读时隙（包括 0 或 1）时序

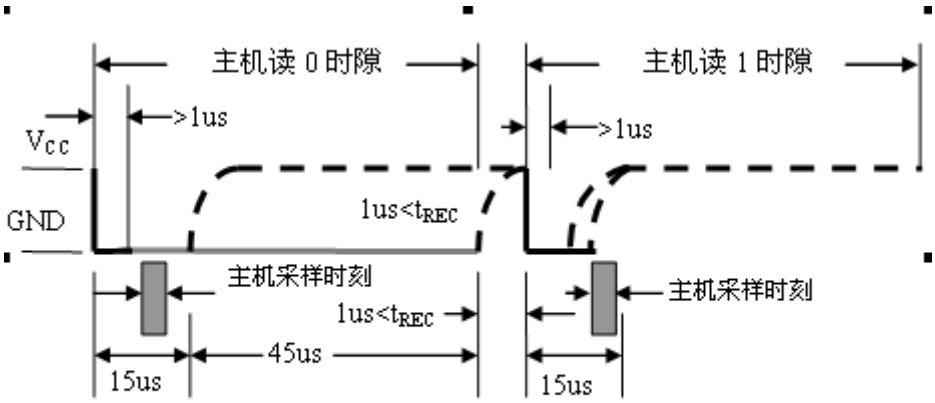

图 1-2 单总线通信协议中读时隙时序图

图中黑色实线代表系统主机拉低总线，灰色实线代表总局拉低总线，而黑色的虚线则代表上拉电阻总线拉高。下面为读操作的代码：

```
char ds18b20_1_read(void)
{
   char i;
   char temp=0;
   for(i=0;i<8;i++)//循环读取总线上的电平信息，高电平为 1，低电平为 0，并且从低电平开始读
   {
    DS18B20_1_OUT;//将 P6.7 置为输出
    OUT_DS18B20_1_0;// 将 P6.7 输出 0
    temp>>=1;//将 temp 变量右移 1 为
    OUT_DS18B20_1_1;
    DS18B20_1_IN;
    if(P6IN&BIT7)//判断总线是否为高电平，如果为高，则将 1 赋给 temp 变量最高位，否则最高位
赋 0
    {
      temp|=0x80;
    }
      DS1820_delay(4);
      DS18B20_1_OUT;
      DS1820_delay(5);
   }
return (temp);//返回读取到的 1 个 byte
}
```

单总线上所有的通信都是以初始化序列开始的，初始化序列包括主机发出的复位脉冲及从机的应答脉冲，这一过程如图所示，黑色实线代表系统主机拉低总线，灰色实线代表从机拉低总线，而黑色的虚线则代表上拉电阻将总线拉高。

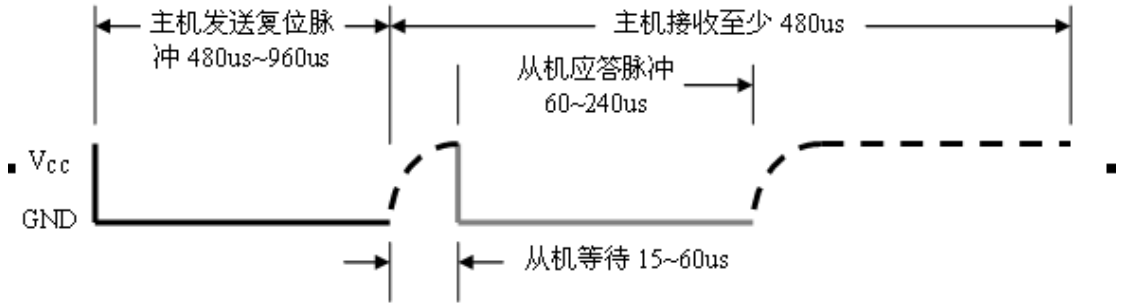

图 4-5 初始化过程中的复位与应答脉冲

系统主设备发送端发出的复位脉冲是一个 480~960us 的低电平，然后释放总线进入接收状态。此时系统总线通过 4.7K 的上拉电阻接至 vcc 高电平，时间约为 15~60us，接在接受端的设备就开始检测 io 引脚上的下降沿以及监视在脉冲的到来。主设备处于这种状态下的时间至少 480us。

作为从设备在接收到系统主设备发出的复位脉冲之后，向总线发出一个应答脉冲，表示从设备已准备好，可根据各种命令发送或接收数据。通常情况下，器件等待 15~60us 即可发送应答脉冲。下面是初始化函数的代码：

```
void ds18b20_1_reset(void)
{

  DS18B20_1_OUT; //将 P6.7 引脚置为输出状态
  OUT_DS18B20_1_1;//这一句和下一句一起表示给出一个复位脉冲
  OUT_DS18B20_1_0;
  DS1820_delay(2500);   //等待 500us
  OUT_DS18B20_1_1;//将总线拉为高电平
  DS18B20_1_IN;      //设置为输入，用来检测从机的应答脉冲
  DS1820_delay(10);//等待 2us
  while(P6IN&BIT7);//检测是否有应答脉冲，知道检测到应答脉冲才会往下面运行
  DS18B20_1_OUT;//将 P6.7 置为输出
  OUT_DS18B20_1_1; //将 P6.7 置为高电平
  DS1820_delay(250);
}
```

5.温度数据格式

如下图所示,这是 12 位转化后得到的 12 位数据，存储在 18B20 的两个 8 比特的 RAM 中，二进制中的前面 5 位是符号位，如果测得的温度大于 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。例如+125℃的数字输出为 07D0H，+25.0625℃的数字输出为 0191H，-25.0625℃的数字输出为 FE6FH，-55℃的数字输出为 FC90H 。

	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

温度转换函数如下：

```
float change_1(void)
{
    temperature=(TxRxBuf[1]&0x0f);
    temperature<<=8;
    temperature|=TxRxBuf[0];
    Temper=temperature*0.0625;
    return Temper;
}
```

此函数将温度转换成 float 型，用来给 lcd 显示提供数据。

6.7 无线收发模块：

6.7.1 无线收发模块主要元件&功能

一、 模块简介

RF905SE（外置天线）

图 35. RF905SE（外置天线, 天线又分垂直与水平） 板子尺寸 32*18mm

NRF905 无线模块特点：

(1) 433Mhz 开放 ISM 频段免许可证使用

(2) 最高工作速率 50kbps，高效 GFSK 调制，抗干扰能力强，特别适合工业控制场

合
(3) 125 频道，满足多点通信和跳频通信需要
(4) 内置硬件 CRC 检错和点对多点通信地址控制
(5) 低功耗 1.9 - 3.6V 工作，待机模式下状态仅为 2.5uA
(6) 收发模式切换时间 < 650us
(7) 模块可软件设地址，只有收到本机地址时才会输出数据（提供中断指示)，可直接接各种单片机使用，软件编程较为方便
(8) TX Mode: 在+10dBm 情况下，电流为 30mA; RX Mode: 12.2mA
(9) 标准 DIP 间距接口，便于嵌入式应用
(10)RFModule-Quick-DEV 快速开发系统，含开发板

二、接口电路管脚说明

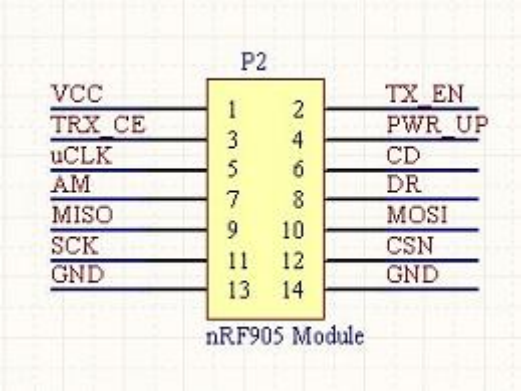

图 36. NRF905 引脚图

管脚	名称	管脚功能	说明
1	VCC	电源	电源+3.3~3.6V DC
2	TX_EN	数字输入	TX_EN=1 TX 模式 TX_EN= 0 RX 模式
3	TRX_CE	数字输入	
4	PWR_UP	数字输入	
5	uCLK	时钟输出	废弃不用，向后兼容
6	CD	数字输出	载波检测
7	AM	数字输出	地址匹配
8	DR	数字输出	接收或发射数据完成
9	MISO	SPI 接口	SPI 输出
10	MOSI	SPI 接口	SPI 输入
11	SCK	SPI 时钟	SPI 时钟
12	CSN	SPI 使能	SPI 使能
13	GND	地	接地
14	GND	地	接地

图 37. NRF905 引脚功能表

说明：

(1) VCC 脚接电压范围为 3.3V~3.6V 之间，不能在这个区间之外，超过 3.6V 将会烧毁模块。推荐电压 3.3V 左右。
(2) 除电源 VCC 和接地端,其余脚都可以直接和普通的 5V 单片机 IO 口直接相连，无需电平转换。3V 左右的单片机亦适用。
(3) 硬件上面没有 SPI 的单片机也可以控制本模块，用普通单片机 IO 口模拟 SPI 不需要单片机 SPI 模块介入，只需添加代码模拟 SPI 时序即可。
(4) 13 脚、14 脚为接地脚,需要和母板的逻辑地连接起来。

三、模块引脚和电气参数说明
RF905 模块使用 Nordic 公司的 nRF905 芯片开发而成。
RF905 单片无线收发器工作在 433/868/915MHZ 的 ISM 频段。由一个完全集成的频率调制器，一个带解调器的接收器，一个功率放大器，一个晶体震荡器和一个调节器组成。
ShockBurst 工作模式的特点是自动产生前导码和 CRC，可以通过 SPI 接口进行编程配置。
电流消耗很低，在发射功率为＋10dBm 时发射电流为 30mA，接收电流为 12.5mA。通过空闲时段使芯片进入 POWERDOWN 模式，可以很容易实现低功耗设计。

RF905SE 模块性能参考数据

参数	数值	单位
最低工作电压	3.0	V
最大发射功率	10	dBm
最大数据传输率曼切斯特编码	50	Kbps
输出功率为-10 dBm 时工作电流	9	mA
接收模式时工作电流	12.5	mA
温度范围	-40 to +85	℃
典型灵敏度	-100	dBm
POWERDOWN 模式时工作电流	2.5	μ A

图 38. NRF905 参考数据

RF905SE 模块工作电压与最大发射增益参考数据

工作电压(模块 VCC 供电电压)	模块最大发射增益(dBm)
+3.3V	+7.3dBm
+3.6V	+10dBm

图 39. NRF905 最大发射增益参考数据

四、工作方式
RF905 一共有四种工作模式, 其中有两种活动 RX/TX 模式和两种节电模式。

活动模式
ShockBurst RX， ShockBurst TX
节电模式

POWERDOWN 和 SPI 编程
STANDBY 和 SPI 编程

nRF905 工作模式由 TRX_CE、TX_EN、PWR_UP 的设置来设定。

PWR_UP	TRX_CE	TX_EN	工作模式
0	X	X	掉电和 SPI 编程
1	0	X	Standby 和 SPI 编程
1	1	0	ShockBurst RX
1	1	1	ShockBurst TX

图 40. NRF905 工作模式

4.1 ShockBurst 模式

ShockBurstTM 收发模式下，使用片内的先入先出堆栈区，数据低速从微控制器送入，但高速发射，这样可以尽量节能，因此，使用低速的微控制器也能得到很高的射频数据发射速率。与射频协议相关的所有高速信号处理都在片内进行，这种做法有三大好处：尽量节能；低的系统费用(低速微处理器也能进行高速射频发射)；数据在空中停留时间短，抗干扰性高。ShockBurstTM 技术同时也减小了整个系统的平均工作电流。

在 ShockBurstTM 收发模式下，RF905 自动处理字头和 CRC 校验码。在接收数据时，自动把字头和 CRC 校验码移去。在发送数据时，自动加上字头和 CRC 校验码，当发送过程完成后，DR 引脚通知微处理器数据发射完毕。

4.1.1 ShockBurst TX 发送流程

典型的 RF905 发送流程分以下几步：

A. 当微控制器有数据要发送时，通过 SPI 接口，按时序把接收机的地址和要发送的数据送传给 RF905，SPI 接口的速率在通信协议和器件配置时确定;

B. 微控制器置高 TRX_CE 和 TX_EN，激发 RF905 的 ShockBurstTM 发送模式;

C. RF905 的 ShockBurstTM 发送：

(1) 射频寄存器自动开启；

(2) 数据打包(加字头和 CRC 校验码)；

(3) 发送数据包；

(4) 当数据发送完成，数据准备好引脚被置高；

D. AUTO_RETRAN 被置高，RF905 不断重发，直到 TRX_CE 被置低；

E. 当 TRX_CE 被置低，RF905 发送过程完成，自动进入空闲模式。

注意：ShockBurstTM 工作模式保证，一旦发送数据的过程开始，无论 TRX_EN 和 TX_EN 引脚是高或低，发送过程都会被处理完。只有在前一个数据包被发送完毕，RF905 才能接受下一个发送数据包。

4.1.2 ShockBurst RX 接收流程

接收流程

A. 当 TRX_CE 为高、TX_EN 为低时，RF905 进入 ShockBurstTM 接收模式；

B. 650us 后，RF905 不断监测，等待接收数据；

C. 当 RF905 检测到同一频段的载波时，载波检测引脚被置高；
D. 当接收到一个相匹配的地址，AM 引脚被置高；
E. 当一个正确的数据包接收完毕， RF905 自动移去字头、地址和 CRC 校验位，然后把 DR 引脚置高
F. 微控制器把 TRX_CE 置低，nRF905 进入空闲模式；
G. 微控制器通过 SPI 口，以一定的速率把数据移到微控制器内；
H. 当所有的数据接收完毕，nRF905 把 DR 引脚和 AM 引脚置低；
I. nRF905 此时可以进入 ShockBurstTM 接收模式、ShockBurstTM 发送模式或关机模式。
当正在接收一个数据包时，TRX_CE 或 TX_EN 引脚的状态发生改变，RF905 立即把其工作模式改变，数据包则丢失。当微处理器接到 AM 引脚的信号之后， 其就知道 RF905 正在接收数据包，其可以决定是让 RF905 继续接收该数据包还是进入另一个工作模式。

4.1.3 节能模式

RF905 的节能模式包括关机模式和节能模式。
在关机模式，RF905 的工作电流最小，一般为 2.5uA。进入关机模式后，RF905 保持配置字中的内容，但不会接收或发送任何数据。 空闲模式有利于减小工作电流，其从空闲模式到发送模式或接收模式的启动时间也比较短。在空闲模式下，RF905 内部的部分晶体振荡器处于工作状态。

五、配置 RF905 模块

所有配置字都是通过 SPI 接口送给 RF905。SIP 接口的工作方式可通过 SPI 指令进行设置。当 RF905 处于空闲模式或关机模式时，SPI 接口可以保持在工作状态。

5.1 SPI 接口寄存器配置

SPI 接口由状态寄存器、射频配置寄存器、发送地址寄存器、发送数据寄存器和接收数据寄存器 5 个寄存器组成。
状态寄存器包含数据准备好引脚状态信息和地址匹配引脚状态信息；
射频配置寄存器包含收发器配置信息，如频率和输出功能等；
发送地址寄存器包含接收机的地址和数据的字节数；
发送数据寄存器包含待发送的数据包的信息，如字节数等；
接收数据寄存器包含要接收的数据的字节数等信息。

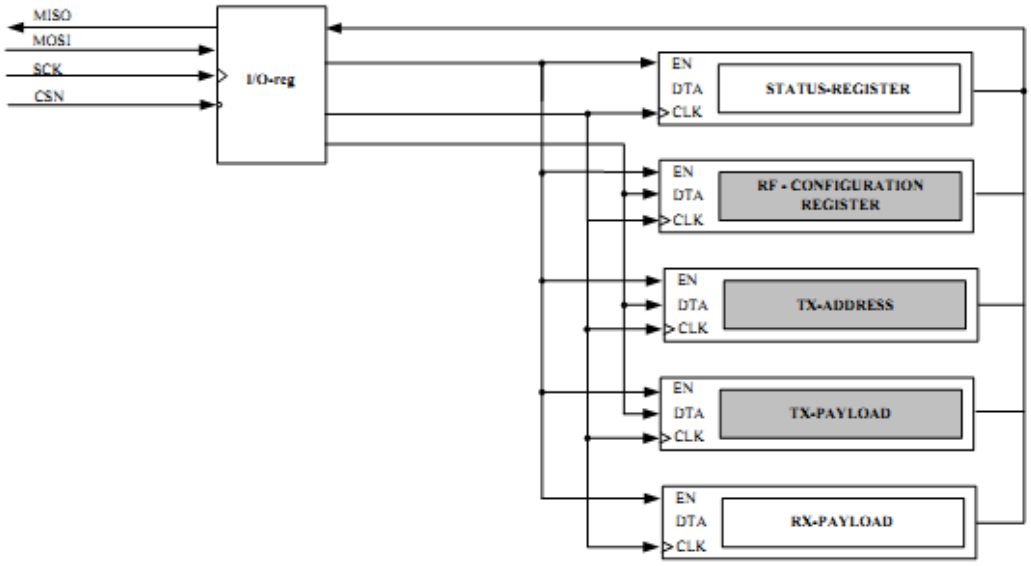

Figure 5. SPI – interface and the five internal registers.

状态寄存器 Status-Register
寄存器包含数据就绪 DR 和地址匹配 AM 状态
RF 配置寄存器 RF-Configuration Register
寄存器包含收发器的频率,输出功率等配置信息
发送地址 TX-Address
寄存器包含目标器件地址字节长度由配置寄存器设置
发送有效数据 TX-Payload
寄存器包含发送的有效 ShockBurst 数据包数据字节长度由配置寄存器设置
接收有效数据 RX-Payload
寄存器包含接收到的有效 ShockBurst 数据包。数据字节长度由配置寄存器设置在配置寄存器中。有效数据由数据准备就绪 DR 指示。

5.2 SPI 指令设置
当 CSN 为低时, SPI 接口开始等待一条指令。任何一条新指令均由 CSN 的由高到低的转换开始。用于 SPI 接口的有用命令见下表：

SPI 串行接口指令设置

SPI 串行接口指令		
指令名称	指令格式	操作
W_CONFIG(WC)	0000AAAA	写配置寄存器 AAAA 指出写操作的开始字节,字节数量取决于 AAAA 指出的开始地址
R_CONFIG(RC)	0001AAAA	读配置寄存器 AAAA 指出读操作的开始字节,字节数量取决于 AAAA 指出的开始

		地址
W_TX_PAYLOAD(WTP)	00100000	写 TX 有效数据 1-32 字节写操作全部从字节 0 开始
R_TX_PAYLOAD(RTP)	00100001	读 TX 有效数据 1-32 字节读操作全部从字节 0 开始
W_TX_ADDRESS(WTA)	00100010	写 TX 地址 1-4 字节写操作全部从字节 0 开始
R_TX_ADDRESS(RTA)	00100011	读 TX 地址 1-4 字节读操作全部从字节 0 开始
R_RX_PAYLOAD(RRP)	00100100	读 RX 有效数据 1-32 字节读操作全部从字节 0 开始
CHANNEL_CONFIG(CC)	1000 pphc cccc cccc	快速设置配置寄存器中 CH_NO,HFREQ_PLL 和 PA_PWR 的专用命令 CH_NO=ccccccccc, HFREQ_PLL=h, PA_PWR=pp

图 41

5.3 SPI 时序

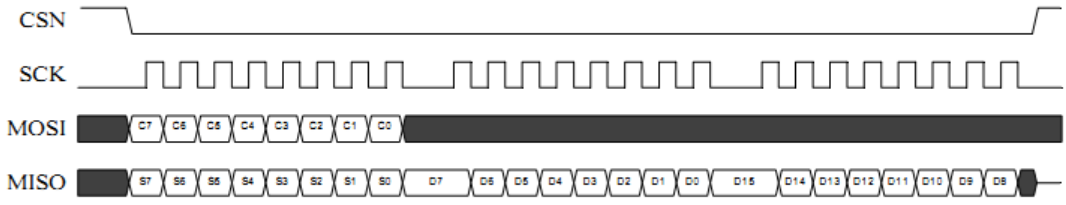

Figure 6. SPI read operation.

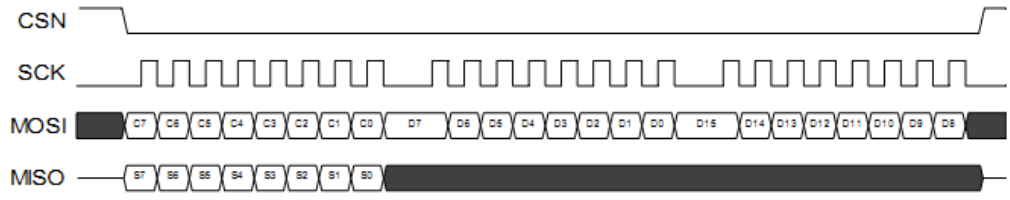

Figure 7. SPI write operation.

6.7.2 无线模块 NRF905 收发显示框架&具体代码

6.1 [nRF905 配置寄存器]

RF-Configuration-Register(R/W)		
字节#	内容位[7：0]，MSB=BIT[7]	初始化值
0	Bit[7：0] CH NO[7:0]	0110_1100
1	Bit[7:6]没用，AUTO_RETRAN，RX_RED_PWR，PA_PWR[1:0]，HFREQ_PLL，CH_NO[8]	0000_0000
2	Bit[7] 没用，TX_AFW[2:0]， Bit[3] 没用， RX_AFW[2:0]	0100_0100
3	Bit[7:6]没用，RX_PWR[5:0]	0010_0000
4	Bit[7:6]没用，TX_PWR[5:0]	0010_0000
5	RX 地址 0 字节	E7
6	RX 地址 1 字节	E7
7	RX 地址 2 字节	E7
8	RX 地址 3 字节	E7
9	CRC_模式， CRC 校验允许，XOF[2:0]， UP_CLK_EN，UP_CLK_FREQ[1:0]	1110_0111

图 42

字节 0：

[7:0] CH_NO[7:0]：

连同字节 1 的 CH_NO[8]和 HFREQ_PLL 控制 905 的载波频段

参考设置：

Operating frequency HFREQ_PLL CH_NO

430.0 MHz [0] [001001100]

433.1 MHz [0] [001101011]

433.2 MHz [0] [001101100]

434.7 MHz [0] [001111011]

862.0 MHz [1] [001010110]

868.2 MHz [1] [001110101]

868.4 MHz [1] [001110110]

869.8 MHz [1] [001111101]

902.2 MHz [1] [100011111]

902.4 MHz [1] [100100000]

927.8 MHz [1] [110011111]

载波频率的计算公式：

$$f_{OP} = (422.4 + (CH\_NO/10)) \cdot (1 + HFREQ\_PLL)\ MHz$$

字节 1：

[0] CH_NO [8] ：参见字节 0

[1] HFREQ_PLL ：

0 - 器件工作在 433MHZ 频段

1 - 期间工作在 868/915MHZ 频段

[3:2] PA_PWR ：

输出功率

00 -10dBm （默认）

01 -2dBm

10 +6dBm

11 +10dBm

[4] RX_RED_PWR ：

降低接收模式电流消耗至 1.6mA，灵敏度降低。

0 - 正常模式 （默认）

1 - 低功耗模式

[5] AUTO_RETRAN：

自动重发 TX 寄存器中的数据包，如果 TRX_CE 和 TX_EN 被设置为高。

0 - 不重发数据包 （默认）

1 - 自动重发数据包

[7:6] 保留

字节 2

[2:0] RX_AWF [2:0] ：

RX 地址宽度

001 - 1 字节 RX 地址宽度 （默认）

100 - 4 字节 RX 地址宽度

[3] 保留

[6:4] TX_AWF [2:0] ：

TX 地址宽度

001 - 1 字节 TX 地址宽度

100 - 4 字节 TX 地址宽度

[7] 保留

字节 3

[5:0] RX_PW [5:0] ：

RX 接收有效数据宽度

000001 - 1 字节 RX 有效数据宽度

000010 - 2 字节 RX 有效数据宽度

……

100000 - 32 字节 RX 有效数据宽度

[7:6] 保留

字节 4

[5:0] TX_PW [5:0] ：

TX 发送有效数据宽度

000001 - 1 字节 TX 有效数据宽度

000010 - 2 字节 TX 有效数据宽度

……
100000 - 32 字节 TX 有效数据宽度
[7:6] 保留

字节 5 ： RX 地址 0 字节
字节 6 ： RX 地址 1 字节
字节 7 ： RX 地址 2 字节
字节 8 ： RX 地址 3 字节

字节 9
[1:0] UP_CLK_FREQ [1:0]：
输出时钟频率
00 - 4MHZ
01 - 2MHZ
10 - 1MHZ
11 - 500KHZ
[2] UP_CLK_EN ：
输出时钟使能
0 - 没有外部时钟
1 - 外部时钟信号使能 （默认）
[5:3] XOF [2:0] ：
晶体振荡器频率，必须依据外部晶体的标称频率设置
(无线模块上 905 芯片外接晶振的频率)
000 - 4MHZ
001 - 8MHZ
010 - 12MHZ
011 - 16MHZ
100 - 20MHZ （默认）
[6] CRC_EN ：
CRC 校验允许
0 - 部允许
1 - 允许 （默认）
[7] CRC_MODE ：
CRC 模式
0 - 8 位 CRC 校验位
1 -16 位 CRC 校验位 （默认）

程序中的相关代码段：

```
/*nRF905 寄存器配置参数*/
typedef struct RFConfig
{
  uchar n;
  uchar buf[10];
}RFConfig;
```

```
code RFConfig RxTxConf =
{
  10, 0x4c, 0x0c, 0x44, 0x20, 0x20, 0xcc, 0xcc, 0xcc,0xcc, 0x58
};
```

6.2 [通过 SPI 接口向 nRF905 配置寄存器读写配置信息]

[SPI 概念] SPI 外围串行接口由四条线构成：
MOSI 主机输出从机输入 （主机写操作）
MISO 主机输入从机输出 （主机读操作）
SCK 串行时钟信号，由主机控制
CSN 片选信号，低电平有效

代码中 nRF905 SPI 接口指令的宏定义

```
//（以下操作全部从对应寄存器的字节 0 开始）
#define   WC      0x00      // 写配置寄存器（ RF-Configuration Register）
#define   RC      0x10      // 读配置寄存器（ RF-Configuration Register）
#define WTP   0x20     // 向 TX-Payload 寄存器写入发送有效数据
#define RTP   0x21    // 从 TX-Payload 寄存器读取发送有效数据
#define WTA   0x22    // 向 TX-Address 寄存器写入发送地址
#define RTA   0x23    // 从 TX-Address 寄存器读取发送地址
#define RRP   0x24    // 从 RX-Payload 寄存器读取接收到的有效数据
```

```
//<SPI 写操作 代码>
void SpiWrite(uchar   byte)
{
  uchar i;
  DATA_BUF=byte;      // 将需要发送的数据写入缓存
  for (i=0;i<8;i++)     // 循环 8 次发送一个字节的数据
  {
    if (flag)       // flag = DATA_BUF^7;
      MOSI=1;
    else
      MOSI=0;
    SCK=1;         // SCK 高电平
    DATA_BUF=DATA_BUF<<1; // 左移一位,为下一位的发送做准备
    SCK=0;         // SCK 低电平
  }
```

```
}
```
步骤一：MOSI 线准备好需要发送的数据位
步骤二：SCK 置高，器件读取 MOSI 线上的数据
步骤三：SCK 置低，准备发送数据的下一位
以上步骤循环执行 8 次，通过 SPI 向器件发送数据完成！

注意：数据的传输时，高位在前，低位在后。

```
//<SPI 读操作  代码>
uchar SpiRead(void)
{
 uchar i;
 for (i=0;i<8;i++)     //循环 8 次发送一个字节的数据
 {
  DATA_BUF=DATA_BUF<<1; //左移一位,准备接收下一位数据
  SCK=1;         // SCK  高电平
  if (MISO)
    flag1=1;     // flag1 = DATA_BUF^0;
  else
    flag1=0;
  SCK=0;         // SCK 低电平
 }
 return DATA_BUF;     // DATA_BUF  为接收到的完整数据
}
```
步骤一：MISO 线准备好需要发送的数据位
步骤二：SCK 置高，主机读取 MISO 线上的数据
步骤三：SCK 置低，准备接收数据的下一位
以上步骤循环执行 8 次，通过 SPI 从器件上读数据完成！

注意：数据的传输时，高位在前，低位在后。

```
//<主机通过 SPI 接口向 905 配置寄存器写入信息>
void Config905(void)
{
 uchar i;
 CSN=0;          // CSN 片选信号，SPI 使能
 SpiWrite(WC);       //  向 905 芯片写配置命令
 for (i=0;i<RxTxConf.n;i++)   //  循环写入配置信息
 {
  SpiWrite(RxTxConf.buf[i]); //RxTxConf 保存预先设置好的配
```

置信息
 }
 CSN=1;   // 结束 SPI 数据传输
}

步骤一：CSN 置低电平，SPI 接口开始等待第一条指令
步骤二：调用 SpiWrite 函数，向器件发送 WC 信号，准备写入配置信息
步骤三：反复调用 SpiWrite 函数，向器件配置寄存器写入配置信息
步骤四：CSN 置高电平，结束 SPI 通讯。
  nRF905 配置完成！

```
//<设置器件为发送模式>
void SetTxMode(void)
{
 TX_EN=1;
 TRX_CE=0;
 Delay(1);         // delay for mode change(>=650us)
}
//<设置器件为接收模式>
void SetRxMode(void)
{
 TX_EN=0;
 TRX_CE=1;
 Delay(1);         // delay for mode change(>=650us)
}


//使用 nRF905 发送数据
void TxPacket(void)
{
 uchar i;
 CSN=0;
 SpiWrite(WTP);       // Write payload command
 for (i=0;i<32;i++)
 {
  SpiWrite(TxBuf[i]);   // 写入 32 直接发送数据
 }
 CSN=1;          // 关闭 SPI,保存写入的数据
 Delay(1);
 CSN=0;          // SPI 使能,准备写入地址信息
 SpiWrite(WTA);       // 写数据至地址寄存器
 for (i=0;i<4;i++)     // 写入 4 字节地址
{
```

```
   SpiWrite(RxTxConf.buf[i+5]);
  }
  CSN=1;          // 关闭 SPI
  TRX_CE=1;        // 进入发送模式,启动射频发送
  Delay(1);        // 进入 ShockBurst 发送模式后,芯片保证数据发送完成后返回 STANDBY 模式
  TRX_CE=0;
}
```

步骤一：通过 SpiWrite 函数发送 WTP 命令，准备写入 TX 有效数据
步骤二：循环调用 SpiWrite 向 TX-Payload 寄存器写入 TX 有效数据（中间夹有 CSN 电平变化）
步骤三：延时
步骤四：通过 SpiWrite 函数发送 WTA 命令，准备写入 TX 地址
步骤五：循环调用 SpiWrite 向 TX-Address 寄存器写入 TX 地址
步骤六：TRX_CE=1; 开始发送数据,延时，nRF905 数据发送完成。
当 nRF905 接收到一条完成的信息时，会将 DR 引脚置高。

```
//这段代码做了较大的简化，只留下必要的部分逻辑
void RxPacket(void)
{
  uchar i;
  TRX_CE=0;        // 设置 905 进入待机模式
  CSN=0;          // 使能 SPI
  SpiWrite(RRP);       // 准备读取接收到的数据
  for (i=0;i<32;i++)
  {
    RxBuf[i]=SpiRead();   // 通过 SPI 接口从 905 芯片读取数据
  }
  CSN=1;          // 禁用 SPI
  while(DR||AM);

  TRX_CE=1;
}
```

步骤一：TRX_CE=0; 必须将此引脚置低，使 905 进入 standby 模式
步骤二：发送 RRP 指令
步骤三：循环调用 SpiRead 函数，读取接收到的数据
步骤四：等待 DR 和 AM 引脚复位为低电平
（中间夹有 CSN 电平变化）
数据包接收完成！

注：
AM 地址匹配，接收到有效地址，被置高
DR 接收到有效数据包，并解码后，被置高
当所有有效数据被读取后，nRF905 将 AM 和 DR 置低

9. 结束语

参加 TI 杯硬件课程设计，我们小组 4 人在第一周分工处理 4 大模块：按键模块，数码管显示模块，LCD 曲线显示模块。各组员不仅深入理解了自己所负责的模块的功能，实现原理，和硬件结构，更是在整机联调的过程中对整个。

这次硬件课程设计同时锻炼了我软硬件两方面的能力，同时也提高了我发现问题和解决问题的能力：

首先这次硬件课程设计给了我一个了解单片机的机会。虽然很早就听过 430 等型号的单片机，也在微机原理课程上听老师讲解过 CPU 和单片机的区别，但一直没有对单片机原理和应用做深入了解。这次实验，我们组没有一个选修 MSP430 嵌入式课程，整个代码部分从头学起，逐渐对其他类型的功能、时序、模块驱动和接口程序有了一定的了解，学会了使用 C 语言对 MSP430 进行编程并使用 JTAG 接口下载调试。

其次，这次试验让我对已经学过的课程有了更深刻的理解，如模拟电路基础、数字电路基础和微机原理，并让我把在之前实验课上学到的实验技巧和知识运用在了此次课程设计之中，让不同课堂上学到的知识融会贯通，一起帮助我完成了这次试验。

最后，这次课程设计让我解决问题的能力有了提高。从最开始模块之间的矛盾，LED 非该模块问题闪烁；中期，NRF905 不能工作，全员齐上阵，通力排难；到后期，元器件损坏，4 次紧急前往广埠屯购买器件。我们最终解决了这样一个一个问题。在最后的整机联调过程中，我们更是遇到许多新的问题，如模块时序冲突，不同管脚性质不一等，我们反复的调试下载和修改，终于提前两天解决了所有问题，完成了系统预期的功能。

开始的并行推进保证了在小组在首次参与 TI 杯类硬件比赛性质课程，不熟悉流程，工作重点不清晰的条件下，不断出现各种意想不到问题困单的情况下，依然能保证进度。从第一天，全组成员在 303 实验室只呆一下午，到第一周周五后，全组成员将除睡觉之外所有时间都安排在实验室，两周的参与过程中，大家不仅在知识上和硬件的了解，处理上有了一个质的飞跃，更重要的是，大家能通力合作，互相帮助，对整个实验有了兴趣，对小组其他成员有了责任心，自愿地将自己所有的精力都投入硬件课设的参与过程。

总体而言，这次硬件课程设计非常具有挑战性，我们组在没有 1 人选修 MSP430 嵌入式选修课的情况下，对 430 从头学起。在这个过程中，困难是层出不穷，但大家在经历前 3 天的磨合期后，所有组员能将除睡觉之外的所有精力投入参与过程，这是其他任何组都看不到的现象。通力合作，自己任务完成后，将别人完成有困难的任务，当作自己的任务一起协助完成，为其他组员，敢于奉献，用于担起集体的责任，这是除了软硬件知识，洞悉开发实现流程外，我们组最大的收获。

10. 参考文献

1. 利达尔试验箱附带光盘资料
2. MSP430 学习板实验指导书(IAR)

3. MSP430x4xx Family Users Guide

11. 附录

人员分工

胡晨: 初步设计蜂鸣器报警电路;
按键模块代码实现;
LED/LCD 数码管显示模块(合作完成);
温控模块(独立尝试,最后完成,但温控不达标);
单向发送模块(与其他 3 人合作完成);
小组报告&展示 PPT 的起草,整合工作,最后确定;
Proteus 电路图绘制及整合，visio 前后期系统框图绘制;
4 次购买元器件。

胡千里: 负责整体软件框架结构，程序流程，以及数据结构的设计
结合 MSP430 单片机的特点，实现整个系统的低功耗解决方案
与无线发送模块配合，完成无线接收模块的实现
改进 3×5 键盘矩阵的扫描处理方案
段式 LED 显示功能实现与改进
段式 LCD 显示功能实现
各模块代码的整合与软件系统整体调试以及 Bug 修复
负责 MSP430 引脚分配，辅助电路接线及焊接

吴东骏: 初期资料收集工作;
了解以及购买元器件;
LCD 点阵显示模块;
nRF905 无线接收模块;
报警模块与该模块的调试;
电路焊接工作;
最后程序代码整合。

马林: 初期资料收集工作;
组织组员，分配任务，调动组内资源;
开题报告定稿，19 周中期报告 PPT 制作;
无线发模块;
温度测量;
整个下位机的调试，模块整合工作;（此任务工作量较大）
电路整合，引脚的确定;
几乎所有的电路焊接工作。

注：以上列出的每一项任务，所花费的时间均大于半个工作日。小于半个工作日完成的若干任务皆未计入分工。