华中科技大学电子与信息工程系2011年TI杯电子设计大赛项目总结报告

课程设计标题：基于 MSP430F149 和 DS18B20 的多路巡检温度测量与控制系统

班　　级：电信系提高班

作　　者：孙少康（组长）　U200814078
张羽　U200814119
何亚昕　U200813053
万俊超　U200813046
朱劭俊　U200812837

指导老师：　汪小燕

时　　间：2011 年 7 月 8 日

目录

1 硬件课程设计任务书

1.1 课程设计题目

基于 MSP430F149 和 DS18B20 的多路巡检温度测量与控制系统

1.2 课程设计目的

1) 熟悉数字温度传感器 DS18B20、nRF905 无线收发模块的工作原理、基本特点
2) 熟悉并掌握以 MSP430F449 为核心的 LSD-TEST430F44X 实验箱的电路图特点以及各个模块的功能特点
3) 熟悉并掌握 MSP430F149 单片机最小系统的特点和编程控制方法和键盘、LCD 显示，中断等的操作，掌握利用 IAR Embedded Workbench for MSP430 对可编程逻辑器件进行硬件下载的方法
4) 掌握 MSP430F149 的基本原理及利用 IAR Embedded Workbench for MSP430 进行可编程逻辑器件设计的方法
5) 掌握 MSP430F149 的基本原理及利用 IAR Embedded Workbench for MSP430 进行可编程逻辑器件设计的方法
6) 掌握软件程序和硬件 PCB 板进行系统联调的方法技巧

1.3 课程设计要求

1.3.1 基本要求部分

a) 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℃。
b) 具有输入控制功能，可由外界输入温度超限报警门限。
c) 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。
d) 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。
e) 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。

1.3.2 发挥部分

a) 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℃。
b) 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

℃。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。

c) 其他功能

2 内容摘要及关键词

2.1 内容摘要

单片机在检测和控制系统中得到了广泛的应用,温度是一个系统经常需要测量控制和保持的量,而温度是一个模拟量,不能直接与单片机交换信息,采用适当的技术将模拟的温度量转化为数字量在原理上虽然不困难但成本较高,还会遇到其它方面的问题。因此对单片机温度控制系统的研究有重要目的和意义。本文主要介绍了以低功耗单片机 MSP430F149、数字温度传感器 DS18B20、无线传输模块 nRF905、温度检测电路、声光报警电路；在描述了外围硬件电路的同时,还做了大量的软件工作,在 IAR 环境下用 C 语言编写了、按键、声光报警、LCD 显示等模块的程序。本设计实现了多路温度的巡检功能，并可通过按键设置温度上下限，超限时给出声光报警信号，另外还可以在 LCD 上显示各路温度、温度上下限值以及温度变化曲线。

2.2 关键词

MSP430F149 单片机 数字温度传感器 DS18B20 nRF905 温度采集 温度控制 LCD 显示

3 人员分工

孙少康　元器件购买，Proteus 仿真，文档
张羽　LCD、测温、无线模块软件编写及调试
何亚昕　元器件购买、LCD 硬件电路及软件调试、焊接
万俊超　Protel 仿真、测温模块硬件、焊接
朱劭俊　按键、报警模块代码编写及硬件连接调试

4 小组完成的功能

i. 有四路 DS18B20 温度检测系统，显示精度 0.01℃可以通过按键来切换测量模式。
ii. 无线传输测量结果
iii. 测量结果显示在 LCD 上，track 显示是第几路温度的数据。

iv. 可以使用按键调节温度门限的最低值和最高值，调节精度 1℃，当前路的温度不在此范围时，电路的蜂鸣器会鸣叫报警。

v. 测量温度时可在点阵 LCD 上看到温度曲线，分辨率 1℃，可以同时显示 16s 内的温度变化。

5 系统总体设计

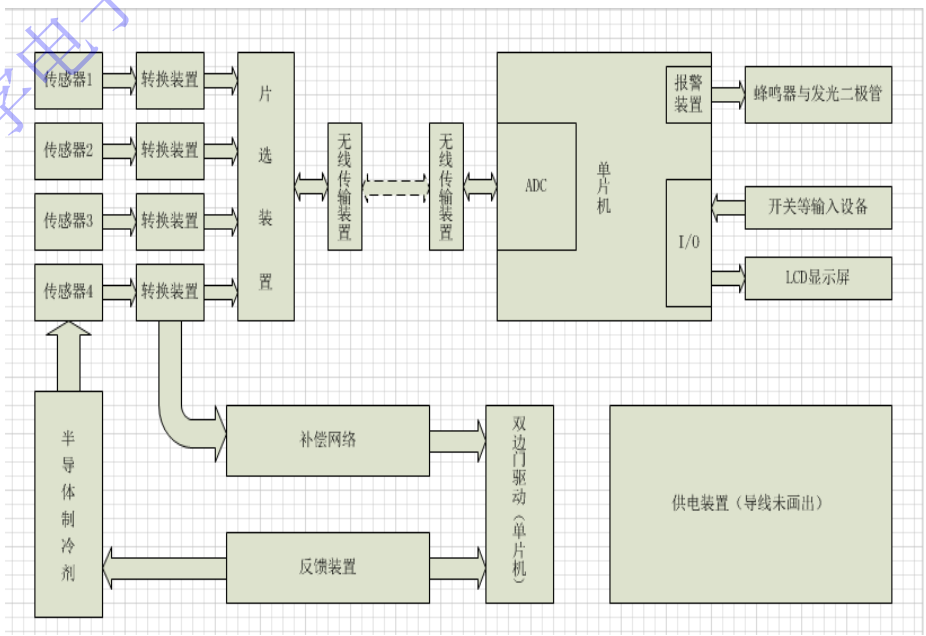
5.1 原始方案

5.1.1 原方案系统目标

以 TI 公司 MSP430F149 单片机为核心，辅以温度传感器 AD590、键盘控制、LCD 显示、数码管显示等器件组成的各个功能模块构成的半导体温控系统，多路模拟开关 CD4051 用于实现片选，实现一个两路由通过 MSP430F149 进行处理，由温度控制模块通过温度传感器 AD590、等实现恒温控制的半导体温控装置。系统利用矩阵式键盘实现功能模式和温度设定值的输入，温度设定的最小分度为 0.1℃。两路

输出信号都可以通过 BF10 蓝牙串口无线通信发送至接收端，并在 ZJM12864BSBD 液晶屏显示。当温度溢出设定的温度范围时，并且 LED 闪光报警。通过选择不同模式，当第一路环境温度超过设定温度时，也能通过报警电路发出声光报警

5.1.2 原方案系统框图



5.1.3 原方案简要原理

整机分三个主模块：处理器模块，恒温控制模块和输入输出模块。处理器模块主要由 MSP430F149 和相应接口构成，用于两路温度传感器的数据处理，预设温度的设定，温度信息的输出和超限报警信号输出。恒温控制模块主要由 TEC 和驱动电路 TLC 构成，用于根据 MSP430F449 输出的预设温度和实际温度调节控制黑箱温度。输入输出模块用于检测两路温度数据，显示预设温度、两路实测温度和报警，由温度传感器 AD590、LCD 和串口组成。两路温度传感器可以实现简单的多路巡检，在 LCD 屏幕上显示一段时间内的温度变化曲线。系统启动后，首先设定预设温度，MSP430 将其储存在 RAM 内并输出到 LCD 数码显示器。每一个时间间隔，MSP430 读入两路的温度数据，并在 LCD 上显示选择的一路温度和温度曲线，同时与预设温度进行比较，如果超限则报警。而 TLC 则通过 MSP430 反馈的黑箱温度和预设温度控制 TEC 工作，保持黑箱温度在一定范围内波动。预设温度、LCD 多路选择显示、报警终止等功能使用矩阵键盘进行输入选择并由 MSP430F149 进行处理。

5.1.4 处理器

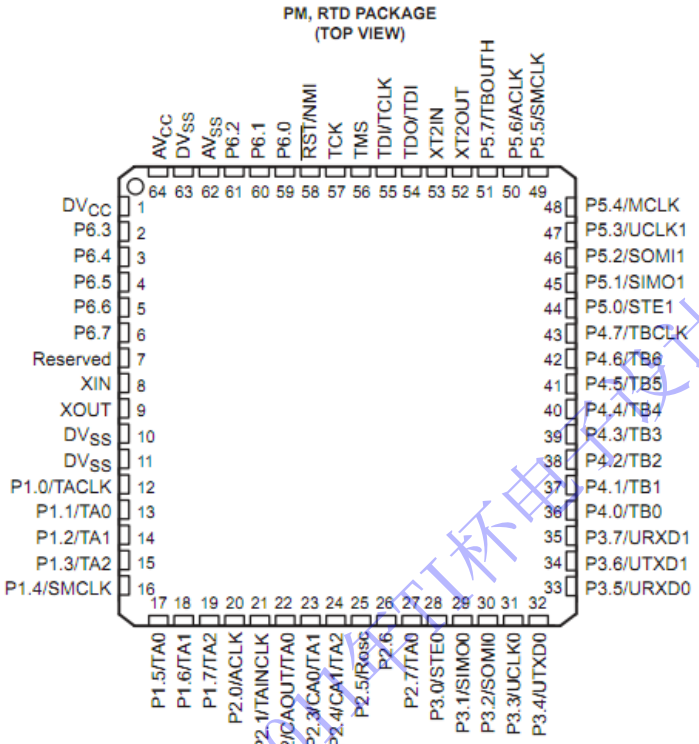
在单片机芯片的选择上，我们最后选择利用自己购买的一个 MSP430F149 最小系统芯片。

方案一：采用实验箱上的 MSP430F449 芯片。它需要的工作电压低，功耗低，带有 12 位 A/D 转换器，FLASH 存储器多达 60KB，RAM 多达 2KB。可通过编程方便地控制，而且实验箱内自带稳压电路。

方案二：自己单独购买一个 MSP430F149 芯片的最小系统。这样可以脱离实验箱，在设计 PCB 硬件板时可以把单片机加入其中，独立做出一个系统，对于硬件课设更有意义。但是这样需要自己设计安装调试稳压电路和其他接口电路，实验箱上配套的附件如液晶、按键、LED 等部分都难以利用，全部需要自己独立设计。

我们小组选择的是方案二，买了一个 MSP430F149 芯片的最小系统，然后为其设计了电源稳压电路，其他如测温电路、声光报警、按键、LCD 等接口部分也相应设计，目的是做出一个只需要外部输入电源就可以独立工作的系统，完全脱离实验箱，做出自己的特色来。

pin designation, MSP430F1471, MSP430F1481, MSP430F1491



5.1.5 温度传感器

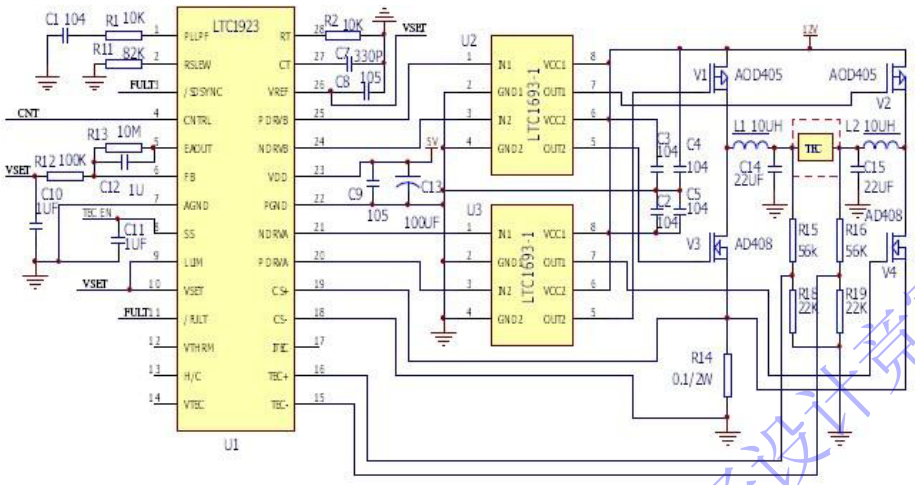
温度信号采用 AD590 传感器来采集。

AD590 将温度信号转化为电信号，然后将采集到的电信号进行放大，接着对模拟信号进行 A/D 转换，转换为数字量后经单片机处理在 LCD 上显示。

5.1.6 半导体制冷片

制冷片选择 TEC1-127-06,最大温差大于 65℃，最大温差电流 6A，最大温差电压 15.4V，最大致冷功率 51.4J/s，空气比热容为 1.300758J/(L*℃),假定黑箱积为 100mm×100mm×100mm=1L。木盒内温度上升或下降 15℃所需热量 Q 为：1.300758J/(L*℃) ×1L×15℃=19.5125J.理想情况下，该致冷片工作在最大致冷功率，可以达到题目中的要求。TEC 驱动器采用 LTC1923，利用 PWM 输出驱动 MOSFET 形成双极性电流全桥驱动器。控制精度可达 0.01℃，频率可达 1MHz，占空比可达 100%，高效，低噪声，低 EMI，双极性电流控制。

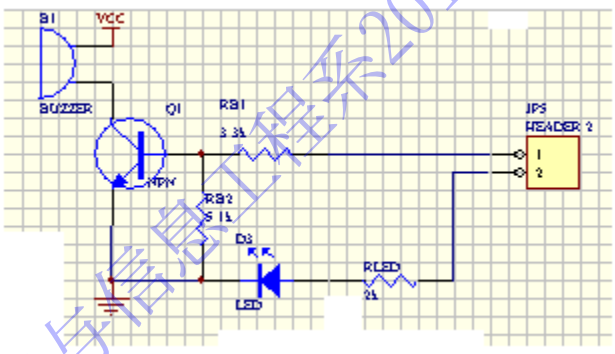
驱动电路：



半导体制冷片驱动电路

5.1.7 声光报警电路

设计声光报警电路是为了在测量温度超出设定温度阈值范围后给出声光报警信号。我们设计的声光报警电路如下所示：



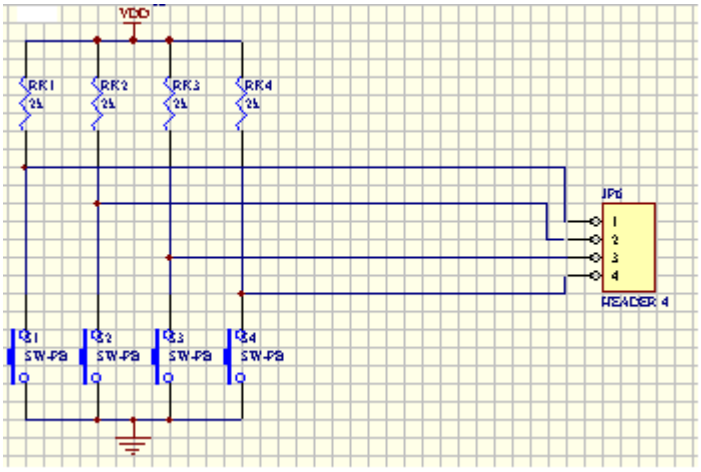
电路原理很简单，当系统发现测定温度超出设定的温度阈值范围后，输出一个高电平，使发光二极管 LED 点亮，同时由三极管驱动的蜂鸣器鸣响。

我们在面包板上插好电路后，输入一个高电平电压，发现发光二极管和蜂鸣器均正常工作，说明此电路设计正确，实际可用。

5.1.8 按键电路

设计按键电路的目的是可以在电路板上通过四个按键对系统部分功能进行控制，如通过按键设定阈值温度，转换 LCD 显示模式等。以每 250ms 速率进行扫描按键是否被按，进而达到从外界输入温度超限报警门限的要求。通过 MSP430F149 的 P1.0-P1.3 来控制键盘电路。

具体设计的电路如下所示：

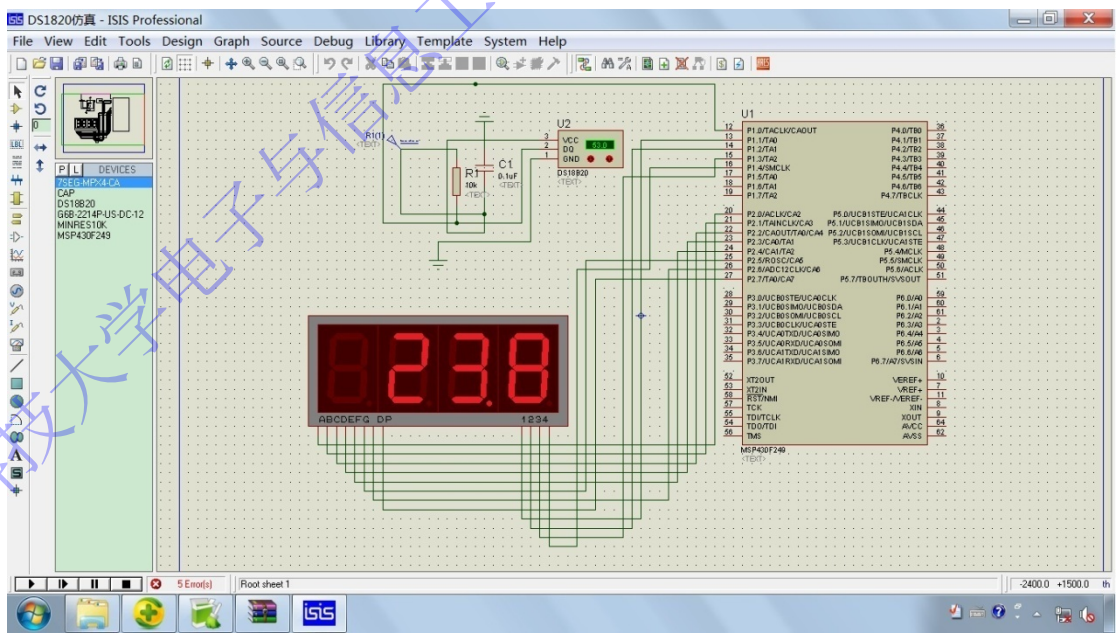


电路的工作原理是每当一个按键按下后，就会产生一个带有上升下降沿的脉冲电压信号，单片机通过检测此脉冲电压信号来在软件程序中进行相关操作。

我们在面包板上插好电路后，每当按下一个按键后，用示波器检测输出电压信号，都有一个明显的脉冲信号，说明此按键电路工作正常。

5.2初期仿真

MSP430 单片机仿真电路图，和仿真结果如下所示：



MSP430 单片机仿真电路图

仿真代码具体可参见附带的代码文件。

5.3 最终设计方案

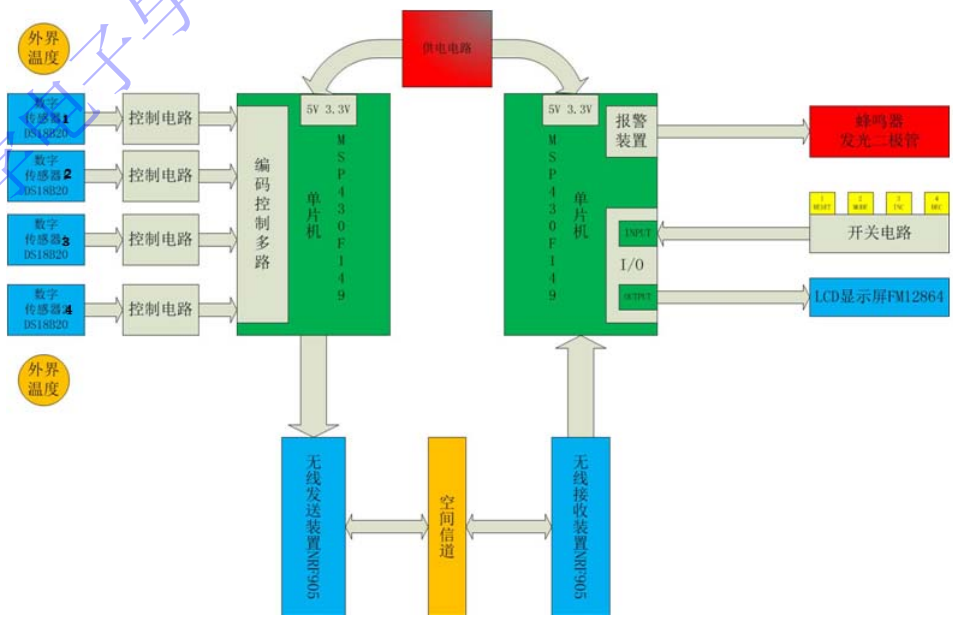
最终方案中由于时间限制，半导体恒温设备没有加入系统。半导体恒温控制系统需要使用专门的驱动芯片和电桥结构进行控制，但是由于 Protel99SE 中没有与之相关的器件库，使我们无法对其进行前期电路仿真，同时也不易得到相应的 PCB 图。后期因为时间关系，最终删除了恒温控制设备的设计方案。

同时由于模数转换与单片机的连接及程序较复杂，我们改用 DS18B20 温度传感器，它是美国 DALLAS 公司推出的单总线数字测温芯片。DS18B20 测温范围为－55℃～+125℃。本次设计中采用 12 为温度转换精度，使温度精度达到 0.0625℃，满足 0.1℃的设计要求。

无线传输模块，汪老师告诉我们蓝牙代码非开源，且成本较高，开发周期较长，我们改用无线收发模块 nRF905，nRF905 将所有高速射频协议集成在芯片内部，与微控制器相连部分只是简单的 SPI 口，数据传输的速率取决于 MCU 本身接口的速度。

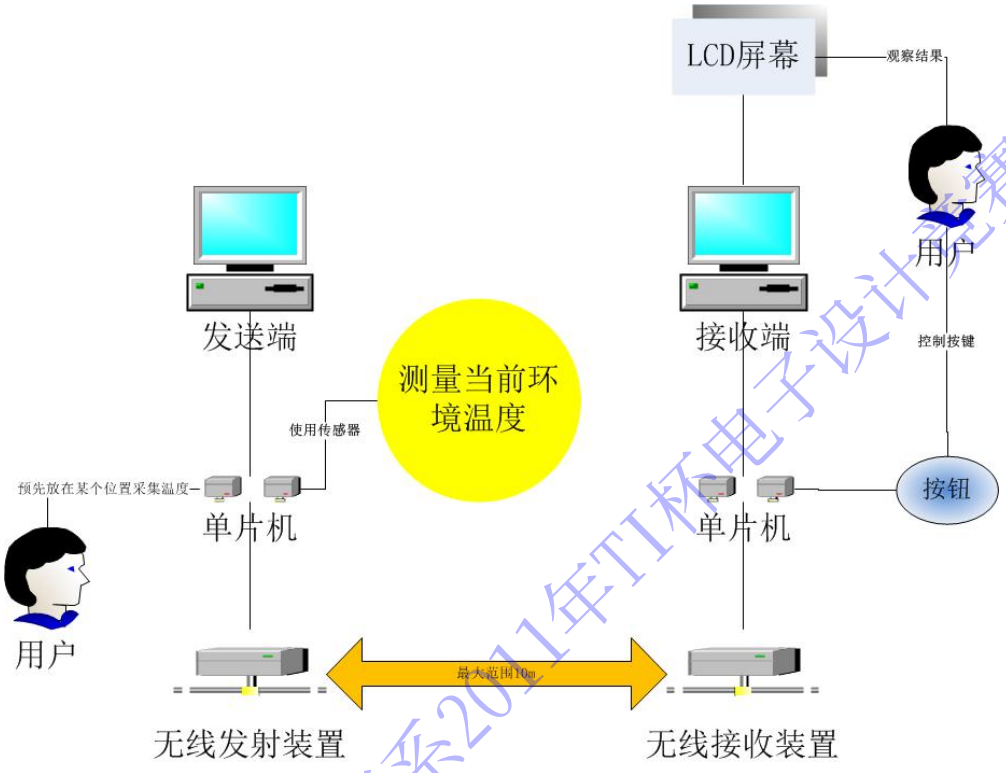
5.3.1 最终方案

综上，我们的系统最终由两片 MSP430F149 单片机，四路 DS18B20 温度传感器，点阵 LCD，无线收发模块 nRF905，矩阵键盘，蜂鸣报警和 PC 串口通信组成。其中 MSP430F149 和 DS18B20 为主要部分。前者控制整个系统的正常工作，后者作为温度传感器测量环境温度。其中可以外部控制的功能有四路温度传感器的切换，报警温度高低限的设定，这些功能由矩阵键盘的按键进行控制。同时，nRF905 实现从测温单元无线传输数据结果到接收端 MSP430F149，点阵 LCD 显示 16 秒内的温度变化情况及报警门限温度。最后一个模块是蜂鸣器电路，实现超限报警。



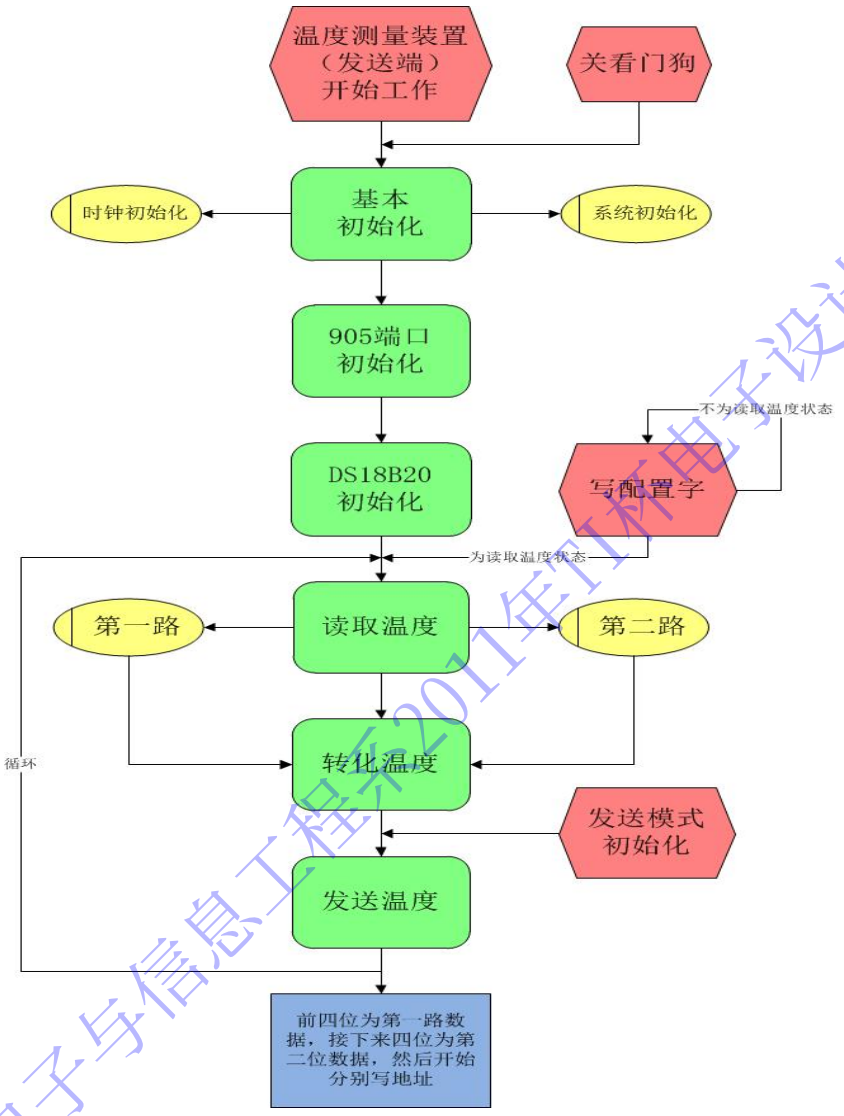
最终系统框图

5.3.2 系统应用场景

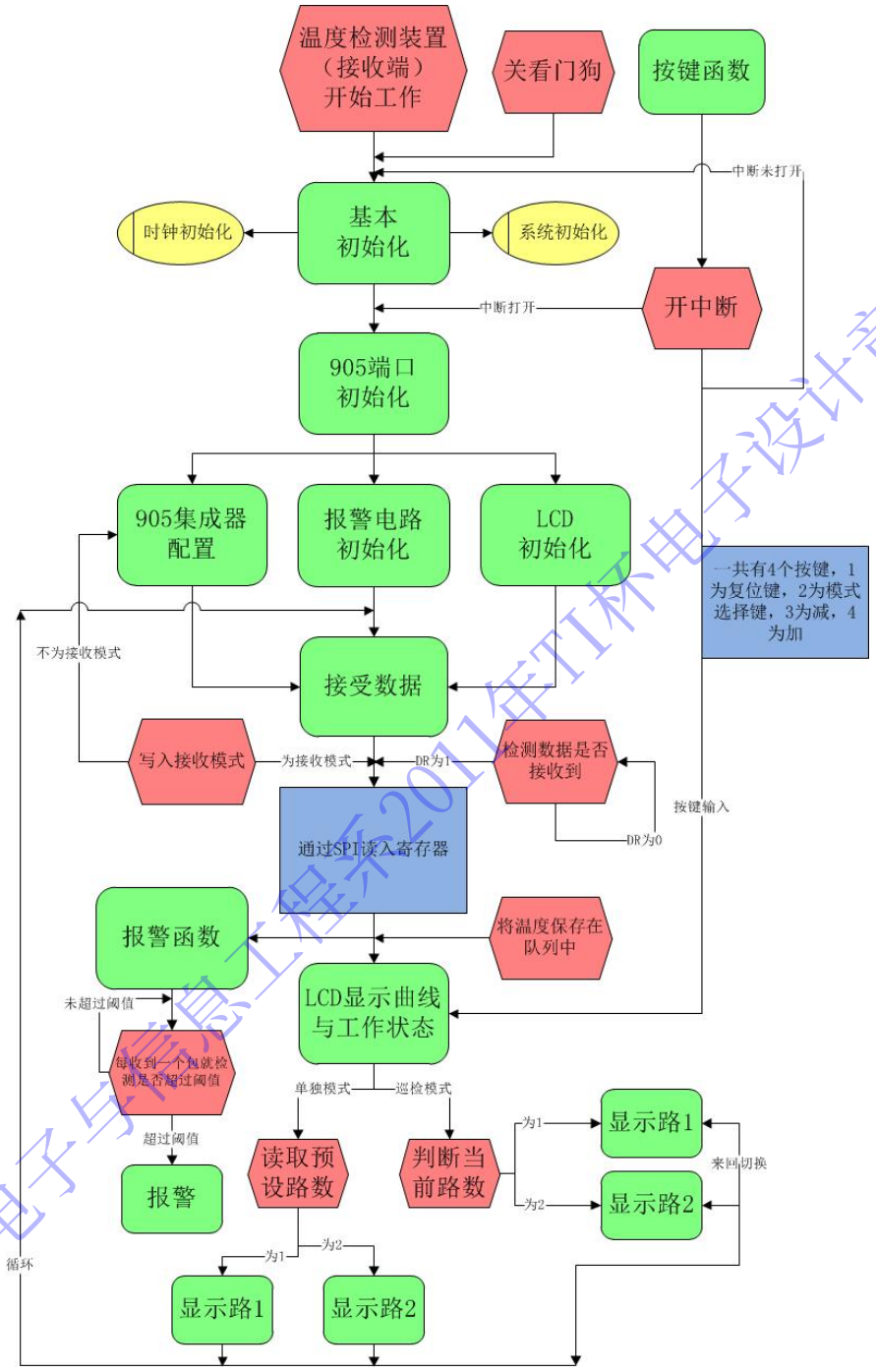


系统应用场景

5.3.3 系统程序流程图



发送端流程图



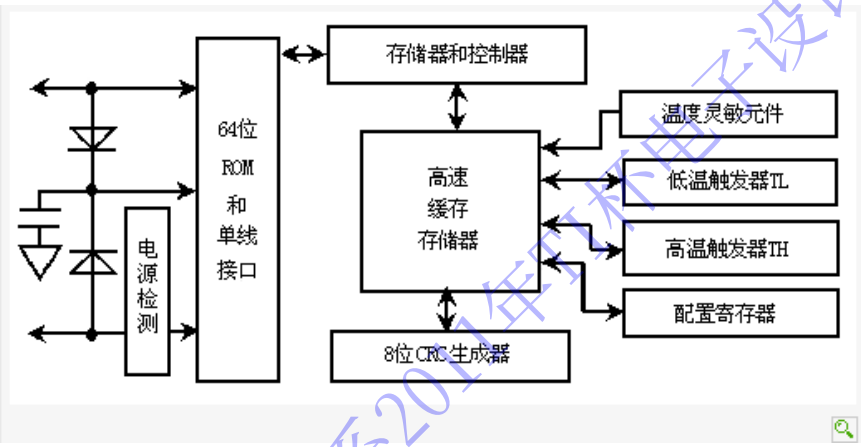
接收端流程图

5.3.4 DS18B20 驱动程序设计

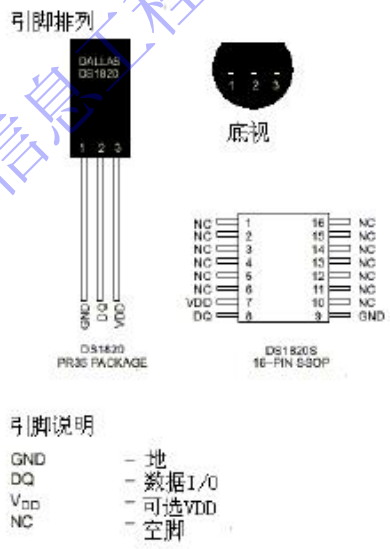
选择 DS18B20 的理由：DS18B20 是数字器件，与 PT100 热敏电阻等模拟器件相比，具有很多优点。一是电路连接简单，因为 DS18B20 采用单总线方式，除地线和 Vcc 外只需一根总线进行命令与数据的读写，不用进行 A/D 转换和零点校正等电路，简化了系统的复杂度。二是 DS18B20 提供 9-12 位精度，本次试验我们采用 12 位精度，精度可以达到 0.0625℃，完全满足 0.1

℃的设计要求，同时 DS18B20 自动进行矫正和 A/D 转换，直接输出数字量，准确性优于模拟器件。三是 DS18B20 内置光刻序列号，可以在一根总线上挂接最多八个器件进行多路巡检，本实验中因为只设计四路，所以没有采用单总线设计，但实际应用中在多路大量温度测量点的情况下，使用 DS18B20 可以有效节约 CPU 的端口资源利用。DS18B20 还内置报警功能，但本次试验没有使用。同时 DS18B20 支持多点组网功能，多个 DS18B20 可以并联在唯一的三线上，实现组网多点测温，这样就能够为以后的巡检功能做好铺垫。

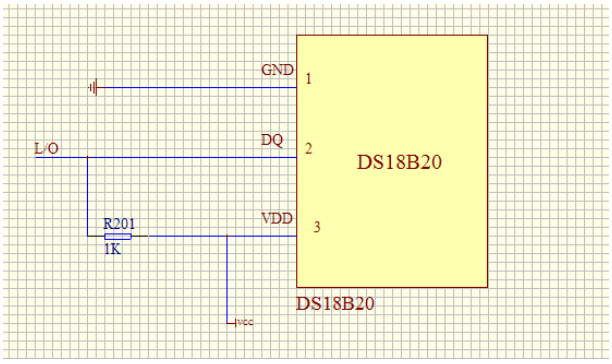
DS18B20 内部结构主要由四部分组成：64 位光刻 ROM 、温度传感器、非挥发的温度报警触发器 TH 和 TL、配置寄存器。DS18B20 的外形及管脚排列如下图：



其外包装如下：

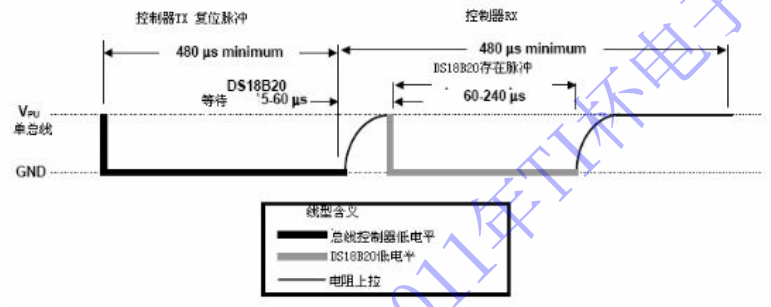


DS18B20 外包装



DS18B20 电路图

初始化时序图如下所示：

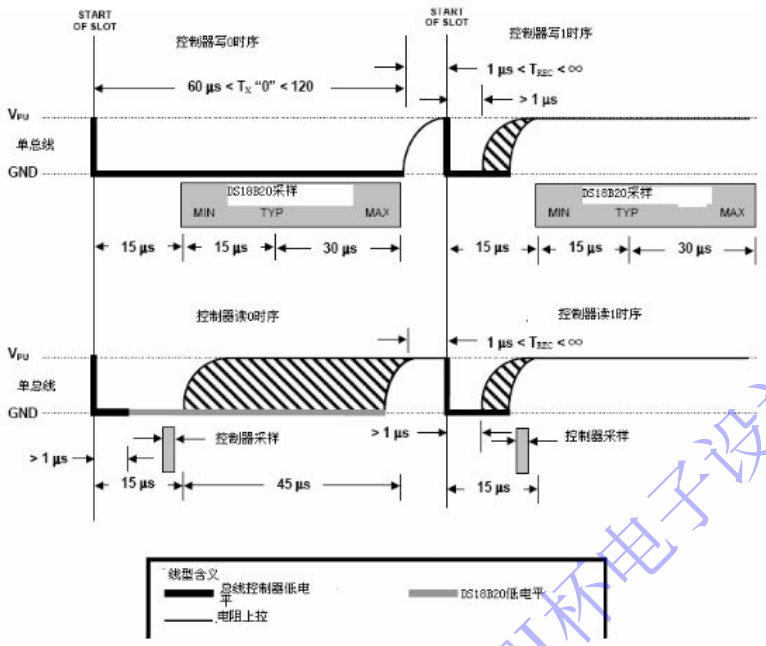


初始化时序图

根据 DS18B20 的初始化时序可以得到 DS18B20 的初始化函数：

```
//====================DS18B20 初始化====================
/*第一路*/
uchar Init_18B20_1()
{
    uchar Error;

    DQ_out_6;
    _DINT();
    DQ0_6;
    DelayNus(500);
    DQ1_6;
    DelayNus(55);
    DQ_in_6;
    _NOP();
    if(DQ_val_6)
    {
        Error = 1;          //第一路初始化失败
    }
    else
    {
        Error = 0;          //第一路初始化成功
    }
```

```
    DQ_out_6;
    DQ1_6;
    _EINT();

    DelayNus(400);

    return Error;
}
/*第二路*/
uchar Init_18B20_2()
{
    uchar Error;

    DQ_out_7;
    _DINT();
    DQ0_7;
    DelayNus(500);
    DQ1_7;
    DelayNus(55);
    DQ_in_7;
    _NOP();
    if(DQ_val_7)
    {
        Error = 1;          //第二路初始化失败
    }
    else
    {
        Error = 0;          //第二路初始化成功
    }
    DQ_out_7;
    DQ1_7;
    _EINT();

    DelayNus(400);

    return Error;
}
```



读写时序图

根据 DS18B20 的读写时序可以得到 DS18B20 的读写函数：

```
//==============================DS18B20 写函数==========
/*第一路*/
void Write_18B20_1(uchar wdata_1)
{
    uchar i_1;

    _DINT();
    for(i_1 = 0; i_1 < 8;i_1++)
    {
        DQ0_6;
        DelayNus(6);                //延时 6us
        if(wdata_1 & 0X01)      DQ1_6;
        else                     DQ0_6;
        wdata_1 >>= 1;
        DelayNus(50);               //延时 50us
        DQ1_6;
        DelayNus(10);               //延时 10us
    }
    _EINT();
}
/*第二路*/
void Write_18B20_2(uchar wdata_2)
{
    uchar i_2;

    _DINT();
```

```
    for(i_2 = 0; i_2 < 8;i_2++)
    {
        DQ0_7;
        DelayNus(6);                //延时 6us
        if(wdata_2 & 0X01)      DQ1_7;
        else                     DQ0_7;
        wdata_2 >>= 1;
        DelayNus(50);               //延时 50us
        DQ1_7;
        DelayNus(10);               //延时 10us
    }
    _EINT();
}
//=========================DS18b20 读函数==============
/*第一路*/
uchar Read_18B20_1()
{
    uchar i_1;
    uchar temp_1 = 0;

    _DINT();
    for(i_1 = 0;i_1 < 8;i_1++)
    {
        temp_1 >>= 1;
        DQ0_6;
        DelayNus(6);                //延时 6us
        DQ1_6;
        DelayNus(8);                //延时 9us
        DQ_in_6;
        _NOP();
        if(DQ_val_6)    temp_1 |= 0x80;
        DelayNus(45);               //延时 45us
        DQ_out_6;
        DQ1_6;
        DelayNus(10);               //延时 10us
    }
    _EINT();

    return  temp_1;
}
/*第二路*/
uchar Read_18B20_2()
{
    uchar i_2;
```

```
    uchar temp_2 = 0;

    _DINT();
    for(i_2 = 0;i_2 < 8;i_2++)
    {
        temp_2 >>= 1;
        DQ0_7;
        DelayNus(6);                //延时 6us
        DQ1_7;
        DelayNus(8);                //延时 9us
        DQ_in_7;
        _NOP();
        if(DQ_val_7)    temp_2 |= 0x80;
        DelayNus(45);               //延时 45us
        DQ_out_7;
        DQ1_7;
        DelayNus(10);               //延时 10us
    }
    _EINT();

    return   temp_2;
}
```

由于四路 DS18B20 采用双总线结构，无需读出序列号。

利用以上函数可以得到当前 DS18B20 的温度值:

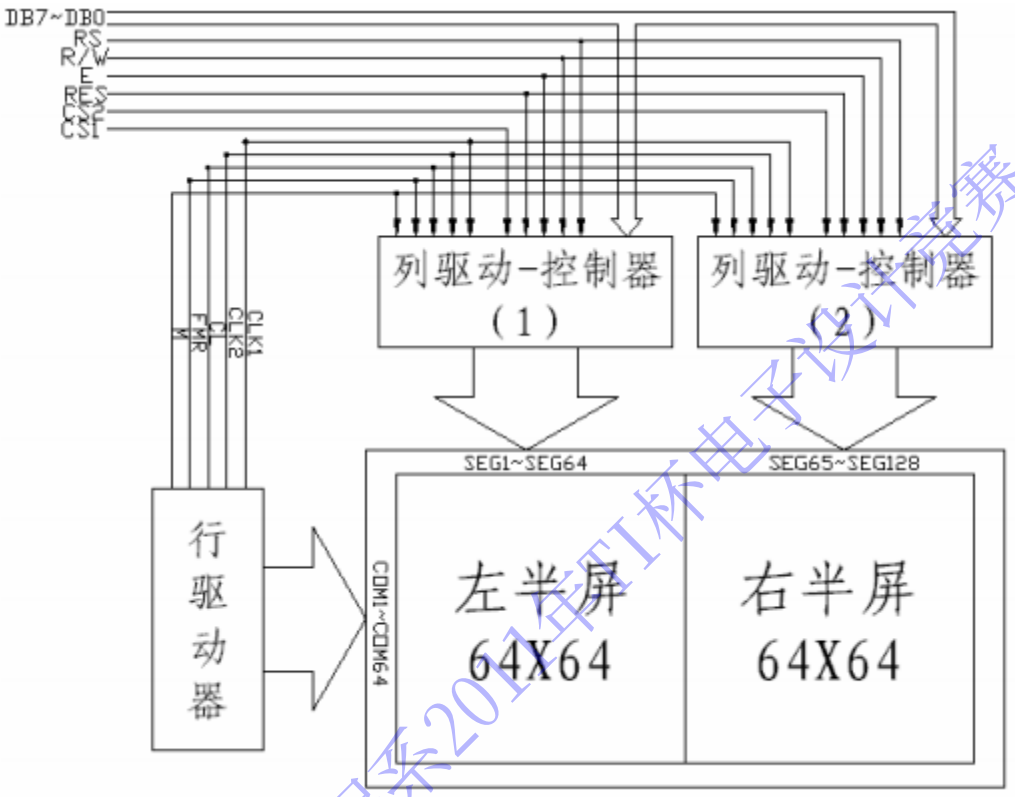
```
//=========================读温度=====================
/*第一路*/
void ReadTemp_1()
{
    temp_data_1[0] = Read_18B20_1();        //读低位
    temp_data_1[1] = Read_18B20_1();        //读高位
}
/*第二路*/
void ReadTemp_2()
{
    temp_data_2[0] = Read_18B20_2();        //读低位
    temp_data_2[1] = Read_18B20_2();        //读高位
}
//=======================完成一次温度转换和读取==========
/*第一路*/
void Do1Convert_1()
{
    uchar i_1;

    do
```

```
    {
        i_1 = Init_18B20_1();
    }
    while(i_1);
    Skip_1();
    Convert_1();
    for(i_1 = 20;i_1 > 0;i_1--)
        DelayNus(60000); //延时 800ms 以上，以完成温度转换
    do
    {
        i_1 = Init_18B20_1();
    }
    while(i_1);
    Skip_1();
    Read_SP_1();
    ReadTemp_1();
}
/*第二路*/
void Do1Convert_2()
{
    uchar i_2;

    do
    {
        i_2 = Init_18B20_2();
    }
    while(i_2);
    Skip_2();
    Convert_2();
    for(i_2 = 20;i_2 > 0;i_2--)
        DelayNus(60000); //延时 800ms 以上，以完成温度转换
    do
    {
        i_2 = Init_18B20_2();
    }
    while(i_2);
    Skip_2();
    Read_SP_2();
    ReadTemp_2();
}

//==================温度换算处理====================
//第一路
void work_temp_1()
```

```
{
        char n=0;
if(temp_data_1[1]>127)
        {
        temp_data_1[1]=(256-temp_data_1[1]);          //负值
        temp_data_1[0]=(256-temp_data_1[0]);
        n=1;
        }
        display_1[6]=((temp_data_1[0]&0xf0)>>4)|((temp_data_1[1]
&0x0f)<<4);
        display_1[5]=display_1[6]/100;                //百位
        display_1[4]=display_1[6]%100;                //
        display_1[2]=display_1[4]/10;                 //十位
        display_1[1]=display_1[4]%10;                 //个位
        switch    (temp_data_1[0]&0x0f)               //小数位
        {
        case 0x0f:display_1[0]=9;break;
        case 0x0e:display_1[0]=9;break;
        case 0x0d:display_1[0]=8;break;
        case 0x0c:display_1[0]=8;break;
        case 0x0b:display_1[0]=7;break;
        case 0x0a:display_1[0]=6;break;
        case 0x09:display_1[0]=6;break;
        case 0x08:display_1[0]=5;break;
        case 0x07:display_1[0]=4;break;
        case 0x06:display_1[0]=4;break;
        case 0x05:display_1[0]=3;break;
        case 0x04:display_1[0]=3;break;
        case 0x03:display_1[0]=2;break;
        case 0x02:display_1[0]=1;break;
        case 0x01:display_1[0]=1;break;
        case 0x00:display_1[0]=1;break;
        default:break;
        }
    if(n)          //负值时显示 aa,正直显示 dd
        {
        display_1[3]=0x11;                            //
        }
    else display_1[3]=0x22;
}
//第二路
void work_temp_2()
{
        char n=0;
```

```
if(temp_data_2[1]>127)
                {
                temp_data_2[1]=(256-temp_data_2[1]);                    //负值
                temp_data_2[0]=(256-temp_data_2[0]);
                n=1;
                }
                display_2[6]=((temp_data_2[0]&0xf0)>>4)|((temp_data_2[1]
&0x0f)<<4);
                display_2[5]=display_2[6]/100;                  //百位
                display_2[4]=display_2[6]%100;                  //
                display_2[2]=display_2[4]/10;                   //十位
                display_2[1]=display_2[4]%10;                   //个位
                switch    (temp_data_2[0]&0x0f)                 //小数位
                {
                case 0x0f:display_2[0]=9;break;
                case 0x0e:display_2[0]=9;break;
                case 0x0d:display_2[0]=8;break;
                case 0x0c:display_2[0]=8;break;
                case 0x0b:display_2[0]=7;break;
                case 0x0a:display_2[0]=6;break;
                case 0x09:display_2[0]=6;break;
                case 0x08:display_2[0]=5;break;
                case 0x07:display_2[0]=4;break;
                case 0x06:display_2[0]=4;break;
                case 0x05:display_2[0]=3;break;
                case 0x04:display_2[0]=3;break;
                case 0x03:display_2[0]=2;break;
                case 0x02:display_2[0]=1;break;
                case 0x01:display_2[0]=1;break;
                case 0x00:display_2[0]=1;break;
                default:break;
                }
        if(n)           //负值时显示 aa,正直显示 dd
                {
                display_2[3]=0x11;                              //
                }
        else display_2[3]=0x22;
}
```

5.3.5 点阵 LCD 驱动程序设计



点阵 LCD 电路

点阵 LCD 模块型号为 FM12864J，其接口电路如上图所示。它主要采用动态驱动原理由行驱动—控制器和列驱动器两部分组成了 128(列)×64(行)的全点阵液晶显示。此显示器采用了 COB 的软封装方式，通过导电橡胶和压框连接 LCD,使其寿命长，连接可靠，具有多功能指令，很容易与 MPU 相连。重要代码如下：

1.start 函数：

```
void delay(uint m)          //延迟函数

void SdData(uchar ddata)                    //送数据的函数
void SdCmd(uchar command)                   //送命令的函数

void Init()                                 //液晶的初始化如开显示
void InitPort()                             //端口的输出输入定义

void CleanScreen()}                         //清屏


void start()
{
   InitPort();
```

```
    Init();
    CleanScreen();
}
```

这是由主函数调用的初始液晶屏函数，进行必要的初始化工作，一般只运行一次。

其中：

```
void InitPort()             //端口初始化
{
    P4DIR=0xff;                    //定义为输出
    P4SEL=0X00;                    //普通 I/O 口

    //P2OUT=0X00;
    P6DIR=0xff;                    //定义为输出
    P6SEL=0X00;                    //普通 I/O 口
}

void Init()                 //初始化
{
    P6OUT&=~RST;   //   RST=0
    delay(150);
    P6OUT|=RST;    //   RST=1
    delay(100);
    P6OUT|=CS1;    //   CS1=1
    P6OUT|=CS2;    //   CS2=1
    //P2OUT&=~EL;

    SdCmd(disp_on);
    SdCmd(disp_z0);
}
```

其中，delay 延时函数，主要用来控制 LCD 刷新速率及 DS18B20 的数据转换速率：

```
void delay(uint m)      //延时函数
{
    uint   i,j;
    for(i=0;i<m;i++)
       for(j=0;j<109;j++)
          _NOP();
}

void SdData(uchar ddata)                //发送数据
{
    P6OUT&=~RS;
```

```
    P6OUT|=RW;              //RS=0，R/W=1,以便读液晶状态
    P4DIR=0x00;             //P4 口为输入口
    do
    {
      P6OUT|=E;             //E=1
      state=P4IN;
      P6OUT&=~E;            //E=0
    }
    while((state&0x01)!=0);
    P6OUT|=RS;              //rs=1
    P6OUT&=~RW;             //rw=0
    P4DIR=0xff;             //P4 口为输出口
    P4OUT=ddata;
    P6OUT|=E;               //E=1
    P6OUT&=~E;              //E=0
}

void SdCmd(uchar command)       //发送指令
{
    P6OUT&=~RS;
    P6OUT|=RW;
    P4DIR=0x00;
    do
    {
      P6OUT|=E;
      state=P4IN;
      P6OUT&=~E;
    }
    while((state&0x01)!=0);
    P6OUT&=~RW;
    P4DIR=0xff;
    P4OUT=command;
    P6OUT|=E;               //E=1
    P6OUT&=~E;              //E=0
}

void CleanScreen()              //清屏
{
    uchar       i,j,k;
    for(i=0;i<8;i++)
    {
      P6OUT|=CS1;
      P6OUT|=CS2;
      SdCmd(disp_y0);
```

```
        SdCmd(disp_x0+i);
        P6OUT|=CS1;
        P6OUT&=~CS2;
        for(j=0;j<64;j++)
          SdData(0x00);
        P6OUT&=~CS1;
        P6OUT|=CS2;
        for(k=64;k<128;k++)
          SdData(0x00);
      }
    }
```

2.display 函数：

LCD 显示函数：

```
    void display(float tem[], int thr_l, int thr_h, int select, int mode, int track,
uchar bmp[])
    {
      ledcon(tem, thr_l, thr_h, select, mode, track, bmp); /这里调用的图像生
成函数
      ShowPics(bmp);    //调用显示函数
      Delay(200);
    }
```

其中：

```
    void ShowPics(uchar *bmp)
    {
      uchar       i,j,k,y;
      for(i=0;i<8;i++)
      {
        P6OUT|=CS1;
        P6OUT|=CS2;
        SdCmd(disp_y0);
        SdCmd(disp_x0+i);
        P6OUT|=CS1;
        P6OUT&=~CS2;
        for(j=0;j<64;j++)
        {
          y = shift(bmp[i*128+j]);
          SdData(y);
        }
        P6OUT&=~CS1;
```

```
        P6OUT|=CS2;
        for(k=64;k<128;k++)
        {
          y = shift(bmp[i*128+k]);
          SdData(y);
        }
      }
    }


    uchar bmp[] = {……..}          //初始化的图像
```

这是由主函数调用的显示函数，根据主函数传出的参数来显示图像，会在主程序的运行之中不停的被调用。这里面还用了一个生成图像的子函数，下面将介绍。

3.void ledcon()函数：

```
    int judge(float tem[]);              //判断是否 reset，根据温度数组是否为全-1
    void readnum(float x, int num[]);    //将小数每位依次读出
    void numtoled(int x, int y, uchar bmp[]);    //传入位置，数值和数组来改变 8*4 的数字点阵
    void axistoled(float tem[],uchar bmp[]);     //根据温度数组显示曲线
    void clearpoint(uchar bmp[]);                //清空曲线上的点
    void cleariconn(uchar bmp[]);                //清光标

    void ledcon(float tem[], int thr_l, int thr_h, int select, int mode, int track, uchar bmp[])
    {
      int reset;

      reset = judge(tem);   //判断是否复位，根据温度数组是否为全-1

      if(reset==1)
      {
        numtoled(17,0,bmp);    //显示初始图像；
        numtoled(22,0,bmp);
        numtoled(30,0,bmp);
        numtoled(90,0,bmp);
        numtoled(95,0,bmp);
        numtoled(111,0,bmp);
        numtoled(116,0,bmp);

        bmp[164]=0x12;bmp[165]=0x29;bmp[166]=0x25;bmp[167]=0x12;
        bmp[168]=0x00;bmp[169]=0x1e;bmp[170]=0x21;bmp[171]=0x25;
```

```
        bmp[172]=0x16;

        clearicon(bmp);         //清光标
        numtoled(230,1,bmp);

        clearpoint(bmp);        //清空曲线上的点
    }
    else
    {
        if (index == 0)
            readnum(tem[NUM_OF_ROUND-1],num1);
        else
            readnum(tem[index-1],num1);
        numtoled(17,num1[0],bmp);
        numtoled(22,num1[1],bmp);
        numtoled(30,num1[2],bmp);
        //写温度值


        /*readnum((float)thr, num2);
        numtoled(97,num2[0],bmp);
        numtoled(102,num2[1],bmp);*/
        readnum((float)thr_l, num2);
        numtoled(90,num2[0],bmp);
        numtoled(95,num2[1],bmp);
         //把 threshold_low 写上去

        readnum((float)thr_h, num2);
        numtoled(111,num2[0],bmp);
        numtoled(116,num2[1],bmp);
         //把 threshold_high 写上去

        switch(mode)
        {
        case 1:

bmp[164]=0x12;bmp[165]=0x29;bmp[166]=0x25;bmp[167]=0x12;bmp[168]=0x
00;

bmp[169]=0x1e;bmp[170]=0x21;bmp[171]=0x25;bmp[172]=0x16;break;
        case 2:

bmp[164]=0x1e;bmp[165]=0x21;bmp[166]=0x21;bmp[167]=0x12;bmp[168]=0x
20;
```

```
bmp[169]=0x10;bmp[170]=0x0f;bmp[171]=0x10;bmp[172]=0x20;break;
        }    //模式——1 为单路 SG，2 为巡检 CY

        clearicon(bmp);
        switch(select)
        {
        case 0: bmp[137]=0x11; bmp[138]=0x0e; bmp[139]=0x04; break;
        case 1: bmp[196]=0x11; bmp[197]=0x0e; bmp[198]=0x04; break;
        case 2:
        case 3: //bmp[86] =0x11; bmp[87] =0x0e; bmp[88] =0x04;
          bmp[80] =0x11; bmp[81] =0x0e; bmp[82] =0x04;break;
        default: break;
        }    //写光标

        numtoled(230, track, bmp);   //把 track 写上去

        clearpoint(bmp);
        axistoled(tem, bmp);
        //生成当前温度和图
      }
    }
```

这个函数即是显示的核心函数，其中将几个位置的图像变化分别用不同的函数来生成。

整个函数的过程中只生成了一个 1024 长度的初始图像数组，每次都在上面改就是为了节省 ram 空间，也是为了是程序简单化，易于管理，修改，调用。

5.3.6 按键电路

我们组对按键输入是用中断的方式处理的，这种方式可以避免对端口的不停地扫描，我们觉得用中断的方式对系统资源的利用率更高，因此决定采用中断的方式处理按键输入。MSP430 的 P1 口用中断功能，我们把 P1.0~P1.3 作为系统的输入。端口初始化时把 P1.0~P1.3 作为输入，P1.4~P1.7 作为输出。同时把中断寄存器清零，然后打开相应管脚的中断功能，并设置对应的管脚由高到低电平跳变使相应的标志位置位。管脚初始化完成后，就可以对相应的输入处理，我们设置了一系列外部变量作为模式选择的标志位，因此，每次按键后产生一个中断，中断处理函数对相应的变量进行操作，就可以完成模式选择、报警阈值设置等操作。

具体相关程序代码如下所示：

```
//================================按键中断==============
int threshold_high,threshold_low;
```

```
int select;
int track;
int mode;                          //1 为单路显示，2 为巡检显示
int temperature_1, temperature_2;
float resultOfChannel_1[NUM_OF_ROUND];
float resultOfChannel_2[NUM_OF_ROUND];

void port1_init()
{
   P1OUT = 0x00;   //输出低电平
   P1DIR = 0XFF;
   P1DIR = P1DIR & ~BIT0 & ~BIT1 & ~BIT2 &~BIT3;
   P1IE = 0;
   P1IES = 0;
   P1IFG = 0;              //中断寄存器清零
   P1IE |= BIT0;
   P1IES |= BIT0;
   P1IE |= BIT1;
   P1IES |= BIT1;
   P1IE |= BIT2;
   P1IES |= BIT2;
   P1IE |= BIT3;
   P1IES |= BIT3;
}

#pragma vector = PORT1_VECTOR
__interrupt void keyprocessing()
{
   Delay(200);Delay(200);
   /*按键 1——复位*/
    if(P1IFG&BIT0)
   {
      P1IFG &= ~BIT0;
      for(int i=0; i<16; i++)
      {
         resultOfChannel_1[i] = -1;
         resultOfChannel_2[i] = -1;
      }
      P2DIR |= 0x08;
      LED4_1;
      Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
      LED4_0;
      Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
   }
```

```
        /*按键 2——选择修改项*/
        if(P1IFG&BIT1)
        {
          P1IFG &= ~BIT1;
          select = (select+1)%4;
          /*0--模式选择，1--track 选择，2--最低门限修改，3--最高门限修改
*/
          P2DIR |= 0x08;
          LED4_1;
          Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
          LED4_0;
          Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
        }
        /*按键 3——减键*/
        if(P1IFG&BIT2)
        {
          P1IFG &= ~BIT2;
          switch (select)
          {
          case 0: mode = 1; break;
          case 1: track = 1; break;
          case 2: threshold_low--; break;
          case 3: threshold_high--; break;
          default: break;
          }
          P2DIR |= 0x08;
          LED4_1;
          Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
          LED4_0;
          Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
        }
        /*按键 4——加键*/
        if(P1IFG&BIT3)
        {
          P1IFG &= ~BIT3;
          switch (select)
          {
          case 0: mode = 2; break;
          case 1: track = 2; break;
          case 2: threshold_low++; break;
          case 3: threshold_high++; break;
          default: break;
          }
          P2DIR |= 0x08;
```

```
            LED4_1;
            Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
            LED4_0;
            Delay(100);Delay(100);Delay(100);Delay(100);Delay(100);
        }
    }
```

5.3.7 声光报警电路

报警程序的原理比较简单，只要传感器输入的温度比设置的阈值大时，就通过 P1.4 输出低电平，驱动蜂鸣器和发光二极管，实现声光报警。

```
//报警
        if(temperature_1 >= threshold_high || temperature_2 >= threshold_high ||
           temperature_1 <= threshold_low || temperature_2 <= threshold_low)
        {
            BELL_0;   //打开蜂鸣器报警
                      Delay(200);Delay(200);
        }
        else
        {
            BELL_1 ;//关闭蜂鸣器报警
        }

          Delay(200);
```

5.3.8 数据无线传输模块

本模块是基于 nRF905 无线收发模块来进行单片机无线传输系统的设计。 nRF905 是挪威 Nordic VLSI 公司推出的单片射频收发器，工作电压为 1.9～3.6 V，32 引脚 QFN 封装(5×5 mm)，工作于 433／868／915 MHz 三个 ISM(工业、科学和医学)频道，频道之间的转换时间小于 650μs。nRF905 由频率合成器、接收解调器、功率放大器、晶体振荡器和调制器组成，不需外加声表滤波器，ShockBurstTM 工作模式，自动处理字头和 CRC(循环冗余码校验)，使用 SPI 接口与微控制器通信，配置非常方便。此外，其功耗非常低，以-10 dBm 的输出功率发射时电流只有 11 mA，工作于接收模式时的电流为 12.5 mA，内建空闲模式与关机模式，易于实现节能。收发流程图如下：

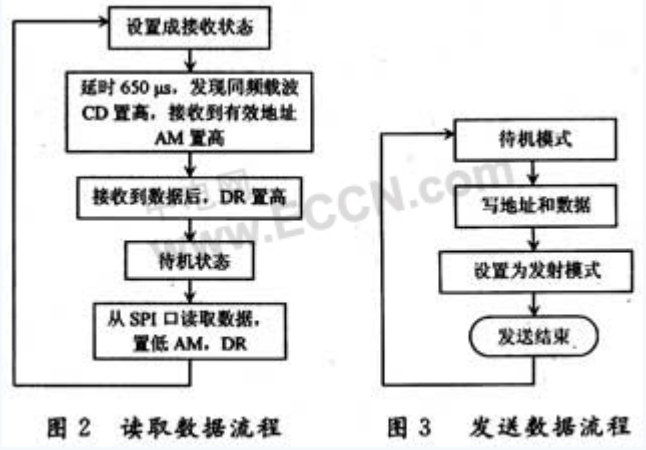


图 2　读取数据流程　　图 3　发送数据流程

具体代码如下：

发送：

```
//==========NRF905 装载地址+数据打包+数据发送==============
void TxPacket()
{
    uchar i;
    CSN_0;
    SpiWrite(WTP);                    // 待发数据装载命令
    for (i=0;i<4;i++)
    {
       SpiWrite(display_1[i]);
            //SpiWrite(0x01);
    }
          for (i=0;i<4;i++)
    {
       SpiWrite(display_2[i]);
            //SpiWrite(0x03);
    }
    CSN_1;                                          // 关闭 SPI
    Delay(1);
    CSN_0;                          // 打开 SPI
    SpiWrite(WTA);                      // 写入地址要和接收方地址一样
    for (i=0;i<4;i++)               // 4 字节地址
    {
       SpiWrite(TxAddress[i]);
    }
    CSN_1;                          //关闭 SPI
    TRX_CE_1;                       // Set TRX_CE high,start Tx data
transmission
    Delay(10);                      // while (DR!=1);
    TRX_CE_0;                       // Set TRX_CE low
}
```

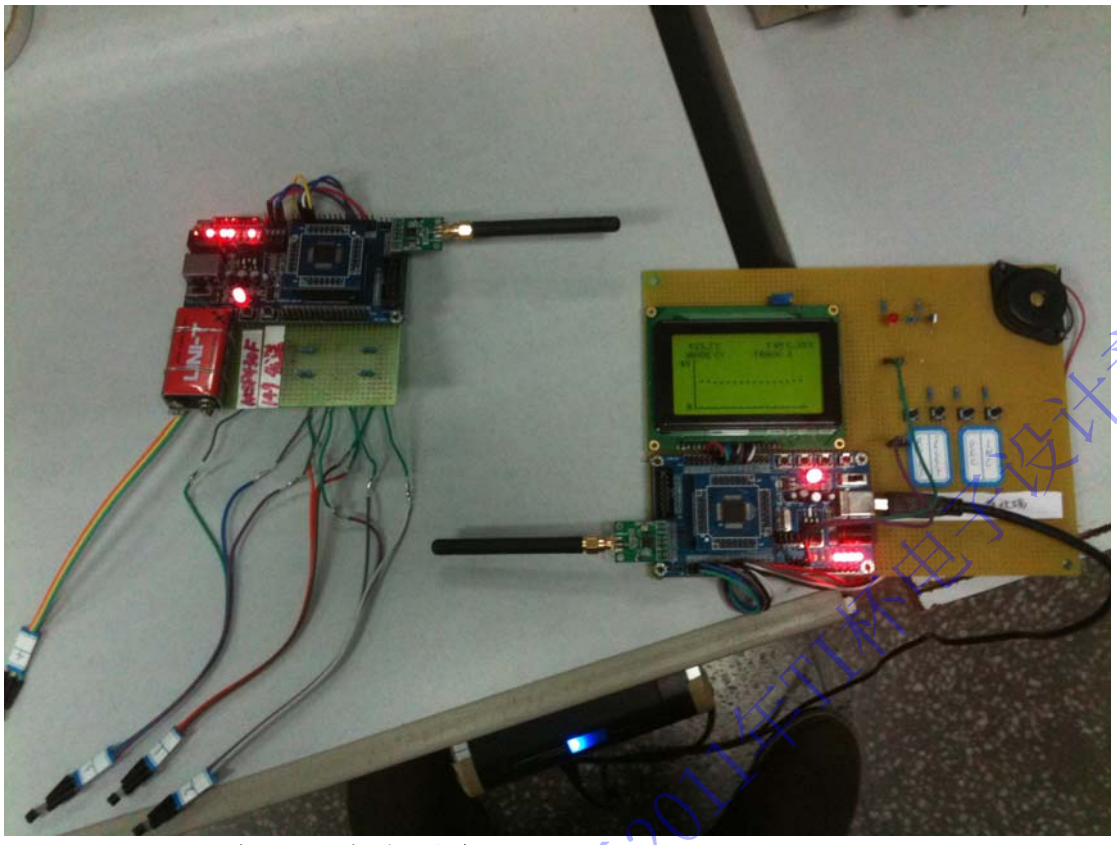
```
//============发送模式初始化-=======================
void SetTxMode()
{
    TRX_CE_0;
    TXEN_1;
    Delay(1);                                   // Delay for mode change(>=650us)
}
//=================数据发送=========================
void TX()
{
            SetTxMode();
    TxPacket();
}
```

接收：

```
//====================接收模式=======================
void SetRxMode(void)
{
    TXEN_0;
    TRX_CE_1;
    NRF905_Delay(1);                // delay for mode change(>=650us)
}
//================NRF905 数据接收流程===================
void    RX(void)
{
      SetRxMode();                  // Set nRF905 in Rx mode
      while (CheckDR()==0);
      NRF905_Delay(10);
      RxPacket();         // Recive data by nRF905
      NRF905_Delay(10);
}
```

5.4 整机调试

在焊接工作完成之后，我们开始对整机进行调试。经过长时间的整机调试，我们最终得出了如下图所示的结果。



有以下几点需要注意：

第一，测温模块关于延时的问题。在一开始的过程中，无论如何调试，接受端屏幕上都没有任何变化，一直显示为恒定值，实际上，这是由于延时不够所造成的，每当发送端发送数据给接受端时，都会在这之前附上一个初始化的过程，从初始化完成到数据接收完成并显示之间需要大概 750ms 的时间。如果延时设置过短，那么接收端实际上一直未正确地等待到数据，只能显示恒定的初始化的值。

第二，测温模块一开始在延时设置之后依然无法正常显示。实际上这是由于我们的操作不当造成的。我们在焊接板子的时候，一开始直接试图把 DS18B20 的引脚焊接在板子上，事实上，该数字温度传感器最大可测量温度只有 125℃，而焊锡的熔点则大于 180℃，所以芯片极有可能已被烧坏，在更换了新的芯片后，我们又不小心错误地接反了高低电平，导致输出依然不正常，不过最终检测出这个错误之后，测温模块已经可以正常工作了。另外，测温模块的 I/O 口上最好接一个上拉电阻，这样可以将电压供电范围拓宽。

第三，LCD 显示模块在最开始时始终无法点亮。实际上这是由于供电不足造成的，由于单片机可以采用 JTAG,USB,普通电源三种供电方式，而我们调试时经常使用 U 口供电，这导致实际上单片机上的 Vcc 引脚并没有标称的 5V，万用表显示只有 2.8V，于是不能点亮启动电压在 5V±10%就不足为怪了。换用普通电源供电后工作正常。

第四，LCD 的背光调节是由 3 脚和 18 脚及相连的滑动变阻器决定的，一开始我们按照说明书上的引脚判断 19 和 20 两脚就可以单独控制背光，但是实际上这样是行不通的，19 接高电平，20 接地只能点亮背光，并无法调节背光。

第五，按键开关的第一个按键 RESET 延迟极大，不像是正常时延导致的结果。经过对代码的分析，我们发现，在一个循环内部，我们错误地使用了

break，导致程序在这里结束时直接跳过了整个循环！改为 continue 之后 RESET 恢复正常，刷新速率较快，但是仍有问题。

第六，RESET 之后，我们发现，系统并未复位到一开始的状态，而是从最接近的一个采样点开始复位。我们再次阅读程序，发现了问题所在。DS18B20 是将温度信息存在了一个队列之中，而伴随着不断读取新的队列，队列当前位置的指针也在不停地前移，当 RESET 按下之后，数据的确被清置，但是指针却没有回到初始位置，所以显示上就出现了从最近的位置开始复位的现象。加上指针归零的语句之后复位完全正常了。

第七，报警电路只有二极管点亮，蜂鸣器不发声。起初我们认为是焊接过程中存在问题，导致蜂鸣器导线没有良好接触。后来我们发现，本来应该用于放大电平的三极管 9015 被我们焊成了一个错误的型号，导致蜂鸣器无法正常工作。更换 9015 后工作正常。

6 结束语

通过本次硬件课设，我们全组成员对于单片机的认识有了进一步的提高，从一开始接到任务之后，我们就立刻着手搜集各种资料，并展开对 430 单片机的学习，期中考试之前这段时间的突击学习对我们后来编程联调还是起到了很关键的促进作用。

本次实验也让我们有机会第一次完成一个成型系统的设计，达到了将所学转化为实际生产中所需的目标。事实上，我们这次对于芯片的选型并不是非常的成功，体现在很多芯片的重复购置上。我们首先选择自己焊接一个 MSP430F149 的最小系统，以为这样可以大大节省开销，而事实上并非如此，我们自己焊接的设备缺少稳定性，而且发生错误之后不知道问题究竟出在哪，因为许多引脚都没有引出，也没有相应的指示灯之类的标识，所以只好改换了由厂家定做的单片机系统，才得以完成了课设要求。在以后计划的过程中，我们必须吸取这样的教训，做到利益的最大化。

本次课设的完成离不开全组成员的共同努力，在最后的两周里，全体组员夜以继日地展开联调与编译的工作，十分辛苦，最后的联调成功是对全组人员的嘉奖。当然，课设中汪小燕老师给予了我们非常有帮助性的指导，让我们少走了许多弯路，比如用蓝牙进行无线传输的不可行，用 449 实验箱先仿真出结果以便修改等等。汪老师为我们争取到了几乎是全天在实验室里的时间，为我们提供转接口，烙铁等各类器材，对我们的实验起到了非常大的作用，在此特向汪老师表示感谢。

7 参考资料

[1] MSP430 系列单片机 C 语言程序设计与开发　北京航空航天大学出版社　胡大可

[2] MSP430 单片机 C 语言应用程序设计实例精讲　电子工业出版社　秦龙

[3] Protel 99SE 自学手册-实例应用篇　人民邮电出版社　刘朋
[4] MSP430 学习板实验指导书(IAR)
[5] MSP430x4xx Family Users Guide
[6] 利达尔试验箱附带光盘资料