华侨大学
Huaqiao university

课程设计论文

题　　目	基于 18B20 及 LCD1602 的两点温度测量显示装置
院(系)别	机电及自动化学院
专　　业	测控技术与仪器
级　　别	2017
学　　号	1711211025
姓　　名	庞 德 霖
指导老师	杜 建 华

2021 年 1 月

摘　要

温度作为一个重要的物理量，和我们的生活息息相关。测量温度成为了一个重要的话题。温度测量发展到至今，技术已经相当成熟。我们生活中许多方面都涉及到温度测量显示，如给鸡蛋孵化时欲要测温，给人量体温时需要测温，空调在运行中也是不断地在测温与显示。

在当下的科技发展状况下，使用较多的还是数字温度传感器，其中本设计所用到的 DS18B20 型数字温度传感器是生活中使用较为广泛的一种传感器，DS18B20 是一款高精度单总线温度测量芯片。温度传感器的测温范围为-55°C 到+125℃根据用户需要通过配置寄存器可以设定数字转换精度和测温速度。

本文基于 DS18B20 温度传感器和 LCD1602 液晶显示器设计一个两点温度检测装置,其具有实时监测温度和具有可调上下限报警温度阈值的功能。整个设计编程基于 keil 软件，硬件仿真软件在 Proteus 仿真软件下进行。

关键词：DS18B20，LCD1602，单片机，温度检测

ABSTRACT

As an important physical quantity, temperature is closely related to our lives. Measuring temperature has become an important topic. The technology of temperature measurement has been quite mature until now. Many aspects of our lives involve temperature measurement and display. For example, we need to measure temperature when hatching eggs, and we need to measure temperature when measuring human body temperature. The air conditioner is constantly measuring and displaying temperature during operation.

In the current state of technological development, digital temperature sensors are more commonly used. The DS18B20 digital temperature sensor used in this design is a sensor that is widely used in life. DS18B20 is a high-precision single-bus temperature measurement chip . The temperature measurement range of the temperature sensor is -55° C to +125° C. According to user needs, the digital conversion accuracy and temperature measurement speed can be set through the configuration register.

This article designs a two-point temperature detection device based on the DS18B20 temperature sensor and LCD1602 liquid crystal display, which has the functions of real-time monitoring of temperature and adjustable upper and lower alarm temperature thresholds. The entire design programming is based on keil software, and the hardware simulation software is carried out under the Proteus simulation software.

Keywords： DS18B20，LCD1602，MCU，temperature check

目　　录

第一章 绪论

1.1 本题设计背景

温度测量以及融入在了我们生活中的方方面面，达到航天飞机燃油箱的温度实时检测，小到为我们测量体温的测温枪，我们时时刻刻都在和温度打交道，因此如何准确、有效、迅速地测量一个点的温度或者一个空间的温度成为了人们日常需求。如今温度测量方式已经发展得十分成熟，从只能凭靠感觉到借助外物，从物理的热障冷缩到热电偶、热电阻的电量变化，从模拟量到数字量，每一步都是人类智慧的结晶。

生活中我们常常需要实时检测一个地方的温度，并且希望当温度不在设定的范围时就会报警，从而避免安全事故的发生以及财产的损失，本课题基于该环境下，通过DS18B20[1]数字温度传感器测量温度，通过 LCD1602[2]液晶显示器实时的显示测量的温度，通过按钮设定报警阈值 的上下限，实现实时检测两点的温度，并且当任何一点的温度值设置超过上下限时就会发生报警。

1.2 温度测量发展

温度是表征物体冷热程度的物理量[3]。在人类社会的生产、科研和日常生活中，温度的测量占有重要地位。但是温度不能直接测量，而是借助于某种物体的某种物理参数随温度冷热不同而明显变化的特性进行间接测量。

温度的测量方法，通常按感温元件是否与被测物接触而分为接触式测量和非接触式测量两大类。接触式测量应用的温度传感器具有结构简单、工作稳定可靠及测量精度高等优点，如膨胀式温度计、热电阻传感器等。非接触式测量应用的温度传感器，具有测量温度高，不干扰被测物温度等优点，但测量精度不高，如红外线高温传感器、光纤高温传感器等。

在当下的科技发展状况下，使用较多的还是数字温度传感器[6]，其中本设计所用到的 DS18B20 型数字温度传感器是生活中使用较为广泛的一种传感器，DS18B20 温度传感器是 DALLAS 公司生产的采用 1-Wire 总线技术的典型作品。他可以将被测温度直接转化成数字量，因此单片机可以方便的通过串行总线实现读取。另外，由于 1-Wire 具有成本低、节省 I/O 口、抗干扰能力强、便于总线扩展和维护等特点。其转换精度高，能满足大部分情况下的需求，实现的功能强大，可以设置转换的分辨率等，且价格便宜，非常适合用于环境不苛刻且精度要去不高的情况。

1.3 总体概述

本设计在 Keil4 与 Proteus8[10]的仿真环境下，用两个 DS18B20 实时检测温度，并同时通过 LCD1602 液晶显示器将其显示出来，同时还有设置温度上下限的功能，实现用户自定义，让设计更加的实用与人性化。

使用说明：当用户为装置上电时，液晶显示屏将开始实时的显示传感器所在的两点的温度，当用户按下 SET 开关时，液晶显示器显示设置模式，用户可以通过按“+1”按钮、“-1”按钮、“+10”按钮、“-10”按钮实现设置温度的转变，通过“Change”按钮来选择是设置温度上限还是温度下限，设置完毕后在按下“SET”按钮回到温度实时显示模式。当两点温度中的任意一点超出设定值时，将会启动 LED 闪烁与蜂鸣器发生报警。

第二章 硬件设计

2.1 硬件设计总览

硬件设计是通过 Proteus8 进行仿真且调试的，Proteus 软件是英国 Lab 公司出版的 EDA 工具软件(该软件中国总代理为广州风标电子技术有限公司)。它不仅具有其它 EDA 工具软件的仿真功能，还能仿真单片机及外围器件。它是目前最好的仿真单片机及外围器件的工具。

本设计的硬件系统中，操作系统采用 AT89C51 单片机[13]，数字传感器采用 Proteus8 自带的元件库中的 DS18B20，液晶显示器采用的是 Proteus8 自带的元件库中的 LM016L，其余外围电路均采用常见的元器件。整体电路设计如图 2.1 所示。完整的仿真电路图见附录一。

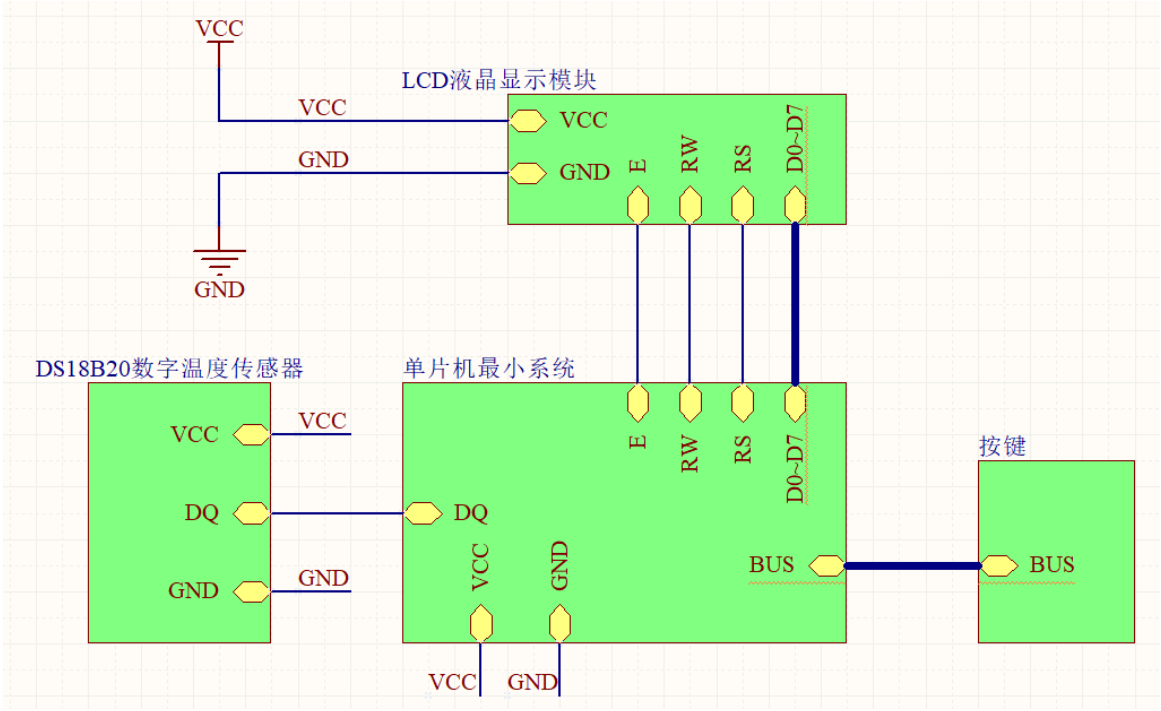


图 2.1 整体设计原理图

2.2 单片机最小系统

单片微型计算机简称单片机。它把组成微型计算机的中央处理器(CPU)、随机存取存储器(RAM)、只读存储器(ROM)、I/O 接口电路、定时/计数器及串行通信接口等功能部件制作在一块集成芯片中，构成一个完整的微型计算机。国际上通常称单片机为微控制器(MCU)，又称为嵌入式微控制器(EMCU)。单片机拥有体积小、成本低、应用广泛、易于产品化等特点，能方便地组成各种智能化的控制设备和仪器，且其应用十分广泛，通常用于工业控制、仪器仪表、家用电器等方面。

单片机的最小系统电路由单片机及其复位电路和外接晶振电路和电源组成。其电路如图 2.2 所示。在晶振电路中，晶振 X1 采用无缘晶振，C1 与 C2 根据晶振厂家提供的数据尽量选择较小，有利于单片机起震。C4 是电解电容，其值通常较大，用于滤除电源中的高频信号，提供稳定电压。在复位电路中，电阻和二极管用于上电复位，按钮用于实现按键复位。

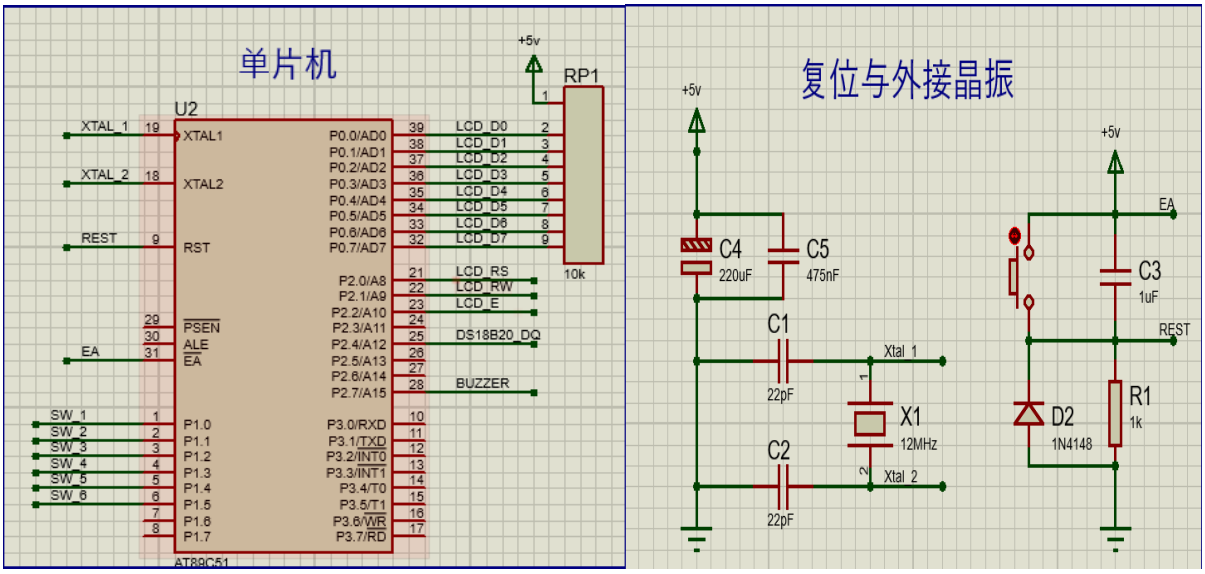


图 2.2　单片机最小系统电路

2.3 DS18B20 数字温度传感器

2.3.1 器件特性描述

DS18B20[15]是一款高精度单总线温度测量芯片。温度传感器的测温范围为-55° C 到+125° C 根据用户需要通过配置寄存器可以设定数字转换精度和测温速度。芯片内置 5byte 非易失性存储单元供用户使用，3byte 用于高低温报警及配置精度，另外 2byte 用于保存用户自定义信息,非易失存储写周期需 10ms。在-10° C 到+85° C 范围内最大误差为±0.5° C，在全温度范围内最大误差为±1° C。18B20 具有寄生供电和外部供电两种工作方式，其中寄生供电可以通过数据线供电，不需要外部供电。

DS18B20 使用单总线协议，总线通讯通过一根控制信号线实现。控制线需要一个弱上拉电阻这样所有的器件都通过三态或者开漏极端口（就是 DS18B20 的 DQ 引脚）连接到总线上。在这个总线系统中，单片机（主机）通过每个器件的唯一 64 位编码识别并寻址总线上的器件。因为每个器件都有唯一的编码，实际上图 2.3 是 DS18B20 的整体原理框图。64 位 ROM 存储了器件的唯一序列码。暂存器包含了两个字的温度寄存器，存储来自于温度传感器的数字输出。另外，暂存器提供了 2 个字节一高一低两个报警触发阈值寄存器（TH 和 TL）以及 1 个字节的配置寄存器，配置寄存器允许用户设定温度数字转换的

分辨率为 9,10，11 或 12 位。上面提到的 3 个字节和 2 个字节的用户可编程 EEPROM 是非易失性存储，器件掉电时数据不会丢失。

挂在总线上并可以被寻址的设备数量是无限的。NS18B20 的另一个特点是其可以不需要外部供电运行。这种情况下是当总线为高的时候，通过单总线在 DQ 引脚上的上拉电阻给器件供电的。总线高信号对一个内部电容（Cp）充电，然后在总线低的时候，内部电容就会维持对器件供电。这种从单总线获取电源的方法被称为“寄生供电”。另一个选择，NS18B20 也可以通过外部 VDD 供电。

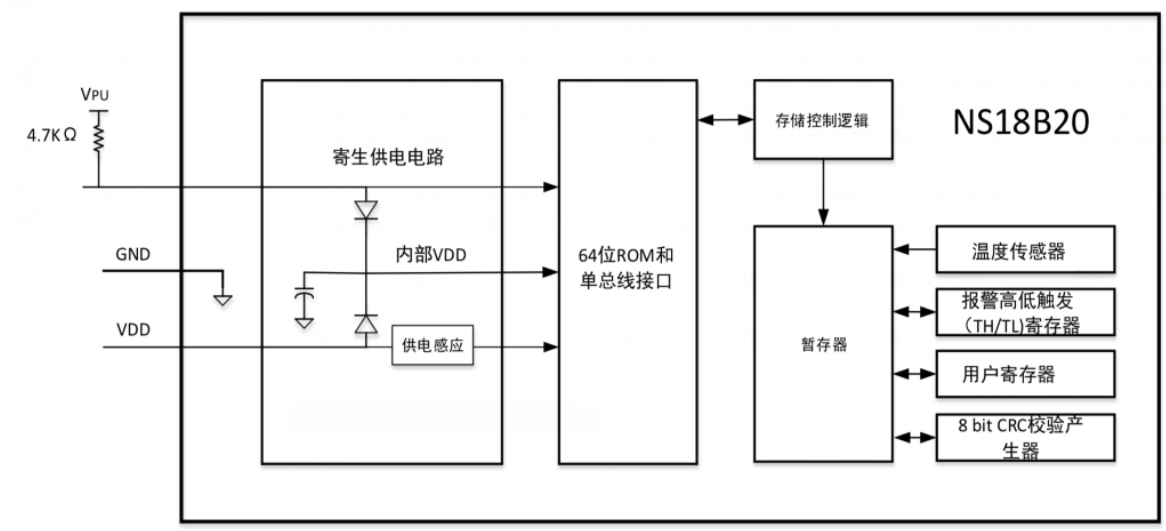


图 2.3　18B20 原理框图

DS18B20 的工作性能如下：

- 1-Wire 数据通信；
- 可以数据线供电，电压范围 3~5.5V；
- 最高 12 位分辨率；
- 12 位分辨率时的最大工作周期；
- 可选择寄生工作方式；
- 检测温度范围为-55℃~+125℃；
- 为测温度在-10~+85℃时，精度为±0.5℃；
- 内置 E2PROM，限温报警功能；
- 64 位光刻 ROM，内置产品序列号，方便多机挂接；
- 封装形式多样；
- 负压特性，电源极性接反时，芯片不会烧毁。

DS18B20 的核心功能是直接数字测温传感器。温度传感器的分辨率具有 9,10，11,12 位，可以根据用户配置，对应的温度分度分别是 0.5° C,0.25° C,0.125° C,和 0.0625° C。上电后的默认的分辨率是 12 位。

2.3.1 封装及使用方法

DS18B20 有 8 引脚 SO 封装、8 引脚 μSOP 封装以及 3 引脚 TO-92 封装 3 种形式。图 2.4 给出了 DS18B20 芯片 TO-92 封装和 SO/ μSOP 封装及引脚分布。

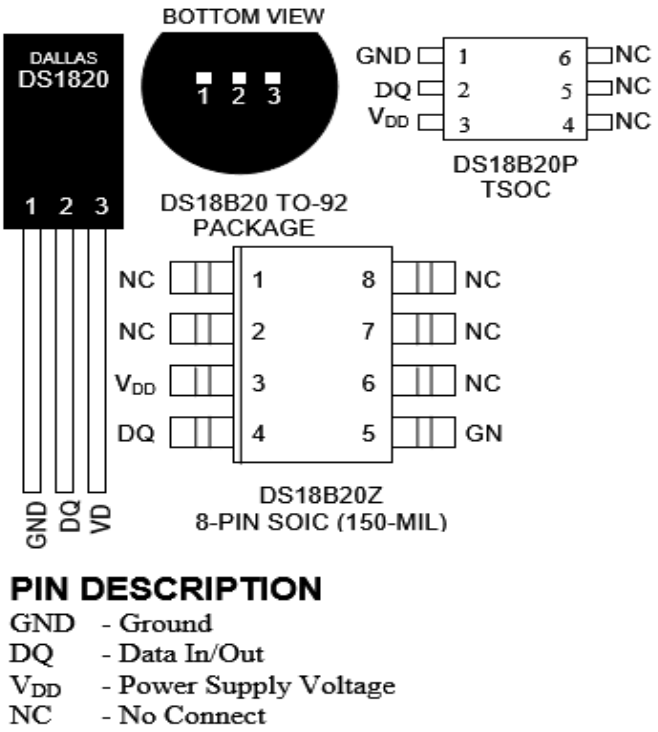


图 2.4　DS18B20 封装形式

本设计采用的是 3 引脚的 DS18B20，且采用外接电源的方式，VCC 和 GND 外接电源，DQ 数据端直接连接单片机用于数据传输。其硬件仿真接法如图 2.5 所示。

其中将两个传感器的 DQ 端放在一根总线上，只要严格的遵循 DS18B20 的协议，那么可以在这根总线上接很多的温度传感器，因为每个传感器在生产出来是内部就有一个全世界独一无二的编号，只要我们在发送匹配 ROM 命令的时候区分开来那么就可以在一根总线上使用多个 DS18B20。

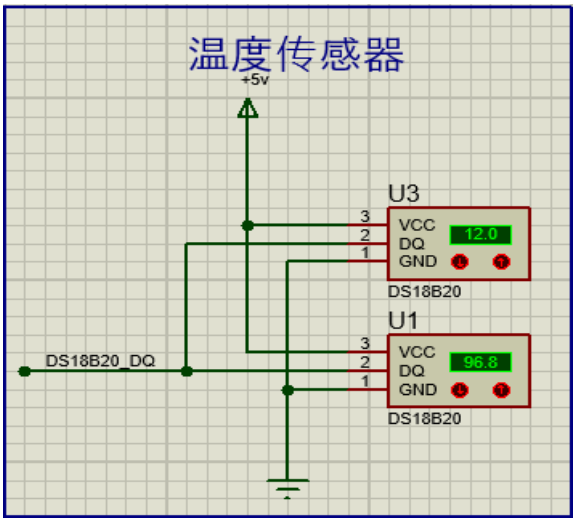


图 2.5　18B20 原理框图

DS18B20 的使用方法十分简单，只需要严格的遵循它的协议。每次访问都需要严格的按照三步顺序，步骤一：初始化；步骤二：发送 ROM 指令（ROM Commdand），后面可以跟随数据传输；步骤三：DS18B20 的功能指令，后面可以会跟随必要的数据切换。值得注意的是 DS18B20 和单片机就这一根数据线相连，因此每次访问 DS18B20 都需要严格的遵循这个顺序，其中任何一个步骤丢失了都会导致 DS18B20 不响应。

2.4 LCD1602 液晶显示器

LCD1602 的应用比较普遍，市面上字符型液晶绝大多数是基于 HD44780 液晶芯片的。由于字符型液晶的控制原理完全相同，因此 HD44780 读写的控制程序可以很方便地应用于市面上大部分的字符型液晶中。字符型 LCD 通常有 14 条引脚线或 16 条引脚线的 LCD，多出来的 2 条线是背光电源线 Vcc(15 脚)和地线 GND(16 脚)，其控制原理与 14 脚的 LCD 完全一样。本设计仿真时采用的是 14 引脚的 LCD。其引脚功能见表 2.1 所示。在仿真电路中的接法如图 2.6 所示。

表 2.1　LCD1602 引脚功能

引脚号	符号	功能
1	VSS	电源地
2	VDD	电源+5V
3	VEE	对比度调整电压。接正电源时对比度最弱，接地时对比度最强，对比度过高时会产生“鬼影”，使用时可以通过一个 10kQ 的电位器调整对比度
4	RS	寄存器选择：1 代表数据寄存器；0 代表指令寄存器
5	R/W	读、写操作：1 代表读；0 代表写
6	E	使能信号：1 时读取信息；1→0(下降沿)时执行命令
7~14	D0~D7	数据总线

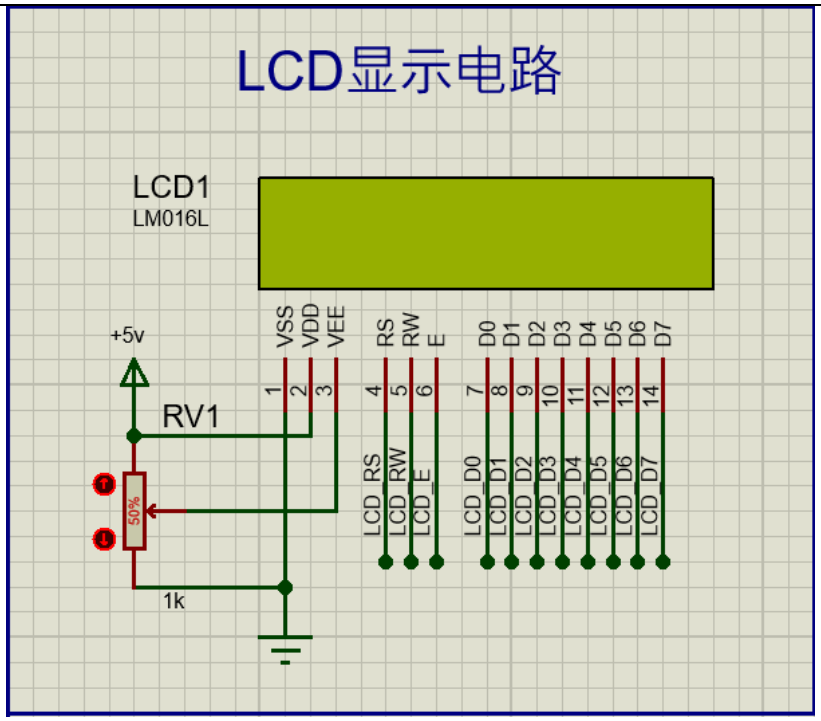


图 2.6　LCD1602 硬件电路

2.5 键盘及报警装置

本设计采用的键盘是普通的按钮，采用 6 个按钮来分别实现控制不同的功能。键盘电路如图 2.7 所示。电路中，按钮一边接地另一边接 I/O 口，当按钮按下时，单片机检测出低电平，从而驱动报警电路相应。报警电路采用一个 PNP 管驱动蜂鸣器，同时用灌电流的方法接一个 LED 发光二极管，当检测温度超出上下限时，单片机就会驱动 LED 闪烁和蜂鸣器发生。报警电路如图 2.8 所示。

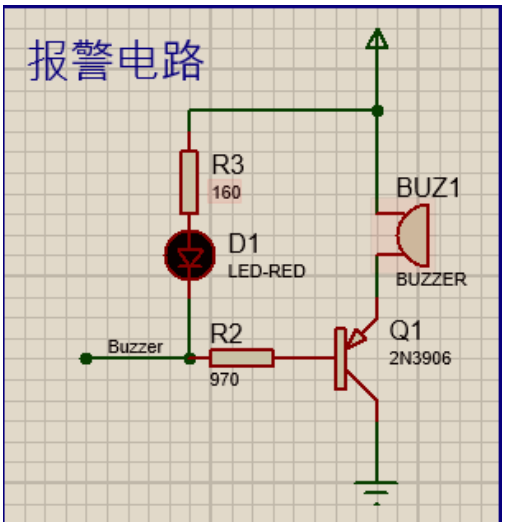


图 2.8　报警电路

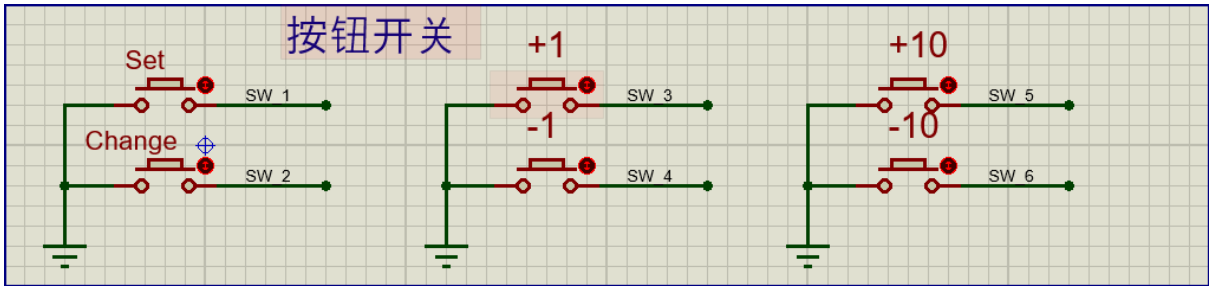


图 2.7　键盘电路

第三章 软件设计

3.1 软件设计环境

本设计的编程软件采用的是 keil4 软件，KeilC51 是美国 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统，与汇编相比，C 语言在功能上、结构性、可读性、可维护性上有明显的优势，因而易学易用。Keil 提供了包括 C 编译器、宏汇编、链接器、库管理和一个功能强大的仿真调试器等在内的完整开发方案，通过一个集成开发环境将这些部分组合在一起。

Keil 使用接近于传统 C 语言的语法来开发。与汇编相比，C 语言易学易用，而且大大提高了工作效率和项目开发周期。还能在关键的位置嵌入汇编，使程序达到接近于汇编的工作效率。KeilC51 标准 C 编译器为 8051 微控制器的软件开发提供了 C 语言环境，同时保留了汇编代码高效、快速的特点。C51 编译器的功能不断增强，更加贴近 CPU 本身及其他的衍生产品。C51 已被完全集成到其开发环境中，这个集成开发环境包括：编译器、汇编器、实时操作系统、项目管理器、调试器。Keil 可为它们提供单一而灵活的开发环境。

本设计采用更加人性化的方法，采用多个 C 文件放在一个工程文件中，且编写时遵循大量编程工作者的阅读习惯，增强代码的可读性，方便他人阅读、借用。

3.2 LCD1602 液晶显示程序

LCD1602 自带 80 个字节的字符存储功能，其包含了常用英文及其他字符的字模，以及可供用户自行载入的寄存器，用于用户自定义字符。我们若想要显示该字符只需要输入其对应的地址即可，常用的字符通常为其 ASCⅡ码。

其 RS、R/W、E 端口为控制端，这三个端口的状态决定 LCD 的当前状态，其功能表见表 3.1.

表 3.1 LCD1602 控制信号真值表

RS	R/W	E	功能
0	0	下降沿	写指令
0	1	1	读忙标志和 AC 值
1	0	下降沿	写数据
1		1	读数据

通常我们使用 LCD 是只需要对其写入数据，因此我们就可以根据其控制信号的真值表和写操作的时序图（图 3.1）来编写我们所需要的函数。

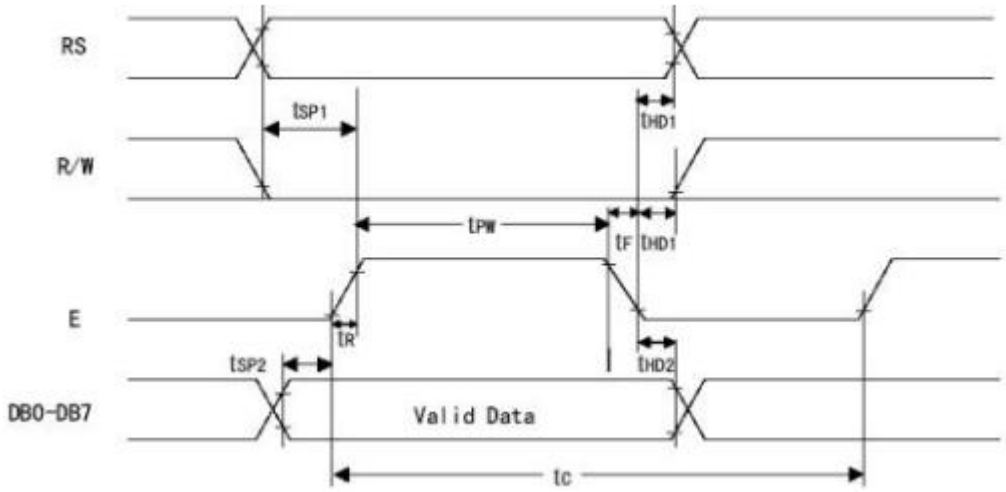


图 3.1　LCD 写操作时序图

3.2.1 写命令函数

写命令函数如图 3.2 所示。将 RS、R/W、E 三个端口的电平均拉低，通过数据总线 P0 口写入命令，然后将 E 端口拉高，延时 1 毫秒后拉低，产生一个下降沿信号使命令生效。

```
extern void lcd_write_com(uchar com)    /*LCD写命令函数*/
{
      delay1ms(1);
      LCD_RS = 0;
      LCD_RW = 0;
      LCD_E = 0;
      P0 = com;
      delay1ms(1);
      LCD_E = 1;
      delay1ms(1);
      LCD_E = 0;
}
```

图 3.2　LCD 写命令函数

3.2.2 写数据函数

写数据函数如图 3.3 所示。将 RS 置为高电平，R/W 和 E 置为低电平，通过数据总线 P0 口写入数据，然后将 E 端口拉高，延时 1 毫秒后拉低，产生一个下降沿信号使数据生效。

```
static void lcd_write_dat(uchar dat)    /*LCD写数据函数*/
{
      delay1ms(1);;
      LCD_RS = 1;
      LCD_RW = 0;
      LCD_E = 0;
      P0 = dat;
      delay1ms(1);
      LCD_E = 1;
      delay1ms(1);
      LCD_E = 0;
}
```

图 3.3　LCD 写数据函数

3.2.3 初始化函数

LCD1602 内部一共由 11 条指令，具体指令可以参考其相应的书籍。我们可以根据其内置的指令代码，将对应的代码输入后即可将 LCD 设置为不同的显示模式。LCD 初始化函数如图 3.4 所示。向 LCD 写入十六进制的 01 将其显示的内容清楚，写入 38 将其设置为两行均显示，且每个显示的位置为 5*7 点阵，其与单片机的数据总线为 8 根，写入 06 是设置当输入一个数据是，内部的指针直接+1 指向下一位，这样在写入数据时就不需要频繁改地址。

```
extern void lcd_init()          /*LCD初始化设置*/
{
    lcd_write_com(0x01);        //清屏
    delay1ms(1);
    lcd_write_com(0x38);        //设置显示两行、每个字符为5*7点阵、数据线为8根
    delay1ms(1);
    lcd_write_com(0x06);        //输入后地址指针右移
    delay1ms(1);
}
```

图 3.4　LCD 初始化函数

3.2.4 显示程序

本设计由于可以检测温度，当温度超出上下限时就会报警，但由于一块 LCD 显示的位置不够用，因此我们采用分时复用的方法，将其分为显示方式 1 和显示方式 2，显示方式 1 为实时显示温度，显示方式 2 为设置温度的上下限模式，用于用户修改上下限的温度。具体程序如图 3.5 所示。

程序根据返回值确定当前是显示模式 1 还是显示模式 2，其功能用于显示一些固定的字符，增强可用性。当显示模式为 2 时，即为用户设置模式，此时将通过程序使光标打开并闪烁，使用户更好的设置温度。

```
extern void lcd_dis_mod(uchar x)        /*显示方式设置，1为实时显示，2为设置模式*/
{
    if(x==1)
    {
        lcd_write_com(0x0c);        //显示打开、光标关闭
        delay1ms(1);
        lcd_write_com(0x80);
        lcd_write_dat('A');
        lcd_write_dat(':');
        lcd_write_com(0x8A);
        lcd_write_dat(0xdf);         //显示摄氏温度字符°
        lcd_write_dat('C');
        lcd_write_com(0xc0);
        lcd_write_dat('B');
        lcd_write_dat(':');
        lcd_write_com(0xcA);
        lcd_write_dat(0xdf);         //显示摄氏温度字符°
        lcd_write_dat('C');
    }
    if(x==2)
    {
        lcd_write_com(0x0f);        //显示打开、光标打开并闪烁
        delay1ms(1);
        lcd_write_com(0x80);
        lcd_write_dat('H');
        lcd_write_dat(':');
        lcd_write_com(0x86);
        lcd_write_dat(0xdf);         //显示摄氏温度字符°
        lcd_write_dat('C');
        lcd_write_com(0xc0);
        lcd_write_dat('L');
        lcd_write_dat(':');
        lcd_write_com(0xc6);
        lcd_write_dat(0xdf);         //显示摄氏温度字符°
        lcd_write_dat('C');
    }
}
```

图 3.5　显示模式程序

确定好显示模式后，接下来就时显示温度函数，显示温度函数会根据返回的温度值来进行显示。显示温度程序如图 3.6 所示。根据返回值 x 确定当前测量的温度是哪个传感器的，获取到温度值后，由于要显示小数点后两位，因此获取到的值是真实值乘 100 后的值，因此通过求余、取模的方法，将每一位的数字都读取出来，存放在一个数组当中，然后显示。并且设置灭零效果。具体函数中还会根据当前的显示模式确认显示传感器的温度还是上下限的温度，具体请参考附录二。

```
extern void lcd_disp_18b20(uchar x)     /*温度值显示函数*/
{
    uchar flagdat;
    disdata[0]=tvalue/10000+0x30;//百位数
    disdata[1]=tvalue%10000/1000+0x30;//十位数
    disdata[2]=tvalue%1000/100+0x30;//个位数
    disdata[3]=tvalue%100/10+0x30;//小数点后一位
    disdata[4]=tvalue%10+0x30;//小数点后两位

    if(tflag==0)
        flagdat=0x20;//正温度不显示符号
    else
        flagdat=0x2d;//负温度显示负号:-

    if(disdata[0]==0x30)
    {
        disdata[0]=0x20;//如果百位为0，不显示
        if(disdata[1]==0x30)
        {
            disdata[1]=0x20;//如果百位为0，十位为0也不显示
        }
    }

    if(x==1)
    {
        lcd_write_com(0x82);     //显示地址位置
    }
    if(x==2)
    {
        lcd_write_com(0xc2);
    }
    lcd_write_dat(flagdat);//显示符号位
    lcd_write_dat(disdata[0]);//显示百位
    lcd_write_dat(disdata[1]);//显示十位
    lcd_write_dat(disdata[2]);//显示个位
    lcd_write_dat(0x2e);//显示小数点
    lcd_write_dat(disdata[3]);//显示小数点后一位
    lcd_write_dat(disdata[4]);//显示小数点后两位
}
```

图 3.6　显示温度程序

3.3 DS18B20 温度检测程序

前面说到，每个 DS18B20 传感器在生产出来时都有一个独一无二的 64 位的序列号，最低的 8 位 ROM 编码包含了 18B20 的单总线系列代码：28h。接下来的 48 位包含一个唯一的序列码。最高的 8 位包含了由前面 56 位 ROM 编码产生的循环冗余校验码。其序列号形式如图 3.7 所示。当使用多个传感器接在一根总线上时，我们就需要对其发送 64 位序列号，只有当序列号匹配的传感器才会响应。

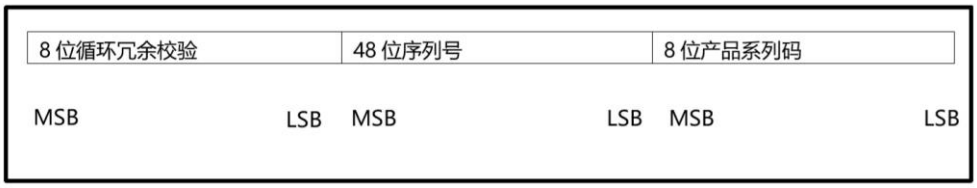


图 3.7　DS18B20 序列号形式

在 Proteus 中设置好序列号后通过 CRC 计算器，将其冗余校验码计算出来，这样完整的 ROM 码就计算出来了。

本设计中，两个传感器的序列号分别为：

28 30 C5 B8 00 00 00 8E

28 31 C5 B8 00 00 00 B9

3.3.1 初始化函数

在每次访问 DS18B20 的第一步我们都需要对其初始化，初始化我们根据其复位的电平时序图（图 3.8）来编写。

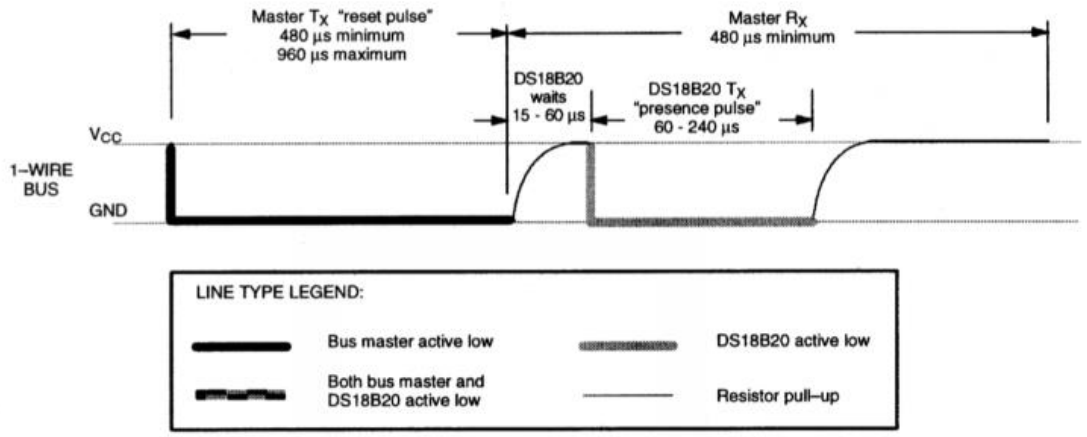


图 3.8　DS18B20 初始化时序

初始化序列包括一个由主机发出的复位脉冲和紧跟着的由从设备发出的应　答脉冲，如图 13 所示。当 18B20 发送应答脉冲以响应复位脉冲时，即向主机表明它已经挂在总线上并且准备完毕。在初始化序列过程中，主机通过将单总线拉低至少 480μs 来发出复位脉冲。随后主机释放总线并进入接收模式。当总线被释放后，5kΩ上拉电阻会把总线拉高。在 NS18B20 检测到这个上升沿后，它会等待 15μs 至 60μs，然后通过把单总线拉低 60μs 至 240μs 来发出应答脉冲。其具体编写程序如图 3.9 所示。

```
static void ds18b20reset()      /*ds1820复位函数*/
{
    uchar x=0;
    LCE_DQ = 1;          //DQ复位
    delay5us(4);         //延时
    LCE_DQ = 0;          //DQ拉低
    delay5us(100);       //精确延时大于480us
    LCE_DQ = 1;          //DQ拉高
    delay5us(40);
}
```

图 3.9　DS18B20 初始化程序

3.3.2 读/写时隙

主机在写时隙期间向 18B20 写入数据，在读时隙期间从 18B20 读取数据。每个时隙期间在单总线上发生 1-bit 的数据传输。其读写时隙的时序图如图 3.10 所示。

要产生写 1 时隙,主机把单总线拉低之后必须在 15μs 内释放单总线。总线被释放后，5kΩ上拉电阻就会把总线拉高。而要产生写 0 时隙的话，主机把单总线拉低之后必须在整个时隙期间都继续保持拉低状态（至少 60μs）。

在主机发起写时隙后，18B20 会在 15μs 到 60μs 的时间窗口内对总线采样。如果在这个采样时间窗口内总线为高，逻辑 1 就被写入 18B20，反之逻辑 0 就会被写入。

所有读时隙必须持续至少 60μs，并且两个读时隙之间的恢复时间不少于 1μs。读时隙的产生是通过主机拉低单总线至少 1μs 然后释放总线来实现的。主机发起读时隙之后，18B20 将开始在总线上传输逻辑 1 或逻辑 0。18B20 通过保持总线为高来发送 1，反之通过拉低总线来发送 0。当传输 0 的时候，18B20 会在时隙行将结束时释放总线，之后总线会被上拉电阻拉回高电平空闲状态。18B20 的输出数据在启动时隙的下降沿后 15μs 之内有效。所以，主机必须在时隙启动之后 15μs 之内释放总线并采样总线状态。说明了在一个读时隙内 TINIT，TRC 和 TSAMPLE 的总和必须少于 15μs。显示了系统的时间裕量可以通过以下方法最大化：保持 TINIT 和 TRC 越短越好，同时把主机采样窗口置于 15μs 读时隙的末尾。

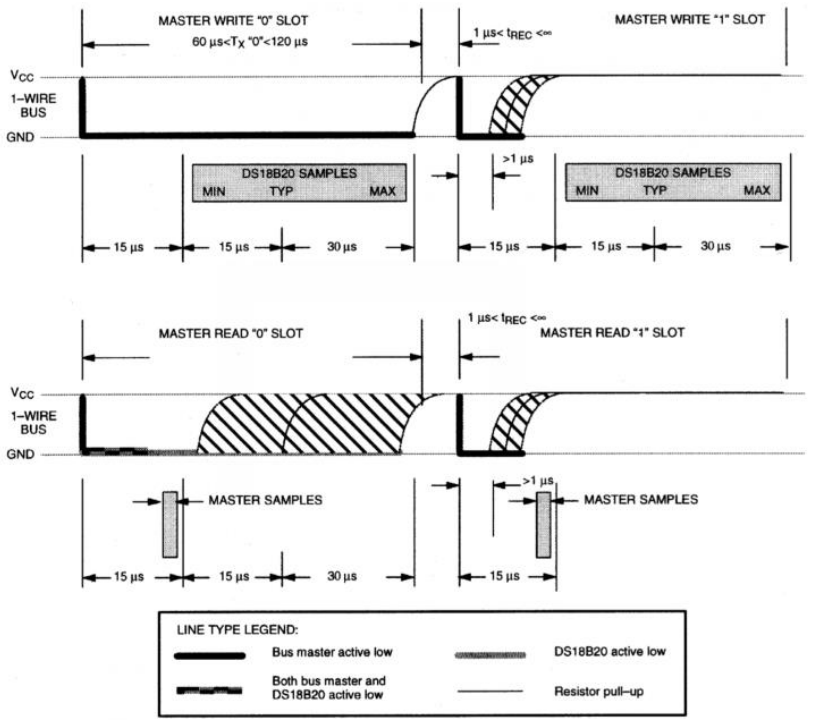


图 3.10　DS18B20 读写时隙时序图

在知道读写时隙的原理后，我们就可以根据此来编写 DS18B20 的读写程序，程序内容如图 3.11 所示。因为每次都是由低字节先发送，因此每读取一位后将数据右移一位，如果为高电平则面补 1，如果为低电平则补 0，循环 8 次后我们就可以从 DS18B20 中读取或者写一个字节的数据。

```
static uchar ds18b20_read()    /*读一个字节数据*/
{
    uchar i = 0;
    uchar dat = 0;
    for (i=0;i<8;i++)
    {
        LCE_DQ = 0; //给脉冲信号
        dat>>=1;
        LCE_DQ = 1; //给脉冲信号
        if(LCE_DQ)
        dat|=0x80;
        delay5us(10);
    }
    return(dat);
}

static void ds18b20_write(uchar dat)   /*写一个字节数据*/
{
    uchar i=0;
    for (i=0; i<8; i++)
    {
        LCE_DQ = 0;
        LCE_DQ = dat&0x01;
        delay5us(10);
        LCE_DQ = 1;
        dat>>=1;
    }
}
```

图 3.11　DS18B20 读写程序

3.3.3 ROM 指令和功能指令

DS18B20 所带的 ROM 指令和功能指令有许多条，在次只列出本设计所用到的指令。

当主机检测到应答脉冲后就可以发布 ROM 指令了。这些指令的运行机制是基于每个从设备唯一的 64 位 ROM 编码。如果有多个从设备挂在单总线上，主机就可以借此单独寻址特定的从设备。这些指令也使得主机可以确定总线上挂有多少设备，这些设备各是什么类型，以及是否有任何设备触发温度报警条件等。ROM 指令总共有 5 种，每种都是 8 位长度。主机在发布 18B20 的功能指令之前，必须发布一种合适的 ROM 指令。

Match ROM[55h]

该指令后跟 64 位的 ROM 编码序列，它能够让主机在多点或单点系统上寻址出一片特定的 NS18B20。只有 64 位 ROM 编码完全匹配的 NS18B20 才会响应主机发出的功能指令。所有其它从设备将等待下一个复位脉冲。根据这条指令编写的匹配 ROM 码函数如图 3.12 所示。首先将 8 个字节的 ROM 码存放于一个数组当中，每当调用这个函数就将这 64 位依次的发送给 DS18B20。

Skip ROM[CCh]

主机可以通过该指令同时寻址总线上所有从设备而无需发送任何 ROM 编码。例如，主机可以命令总线上所有 18B20 同时执行温度转换，方法是只需发布 Skip ROM 指令，然后跟着一个 Convert T[44h]指令。

需要注意的是，Read Scratchpad[BEh]指令只能在单一从设备挂在总线上时才能跟随在 Skip ROM 指令后。这种情况下，主机无需发送 64 位 ROM 编码而直接读取从设备数据，这样可以节约时间。但是如果总线上有超过一个从设备，那么 Skip ROM 指令后面跟随 Read Scratchpad 指令就会导致数据冲突，原因是同时有多个从设备试图传送数据。

```
static void b20_Matchrom(uchar x)  /*匹配ROM函数*/
{
    char j;
    ds18b20_write(0x55);     //发送匹配ROM命令
    if (x==1)
    {
        for(j=0;j<8;j++)
        ds18b20_write(str1[j]); //发送18B20的序列号，先发送低字节
    }
    if(x==2)
    {
        for(j=0;j<8;j++)
        ds18b20_write(str2[j]); //发送18B20的序列号，先发送低字节
    }
}
```

图 3.12　DS18B20 匹配 ROM 函数

在使用 ROM 指令寻址到一个希望与之通讯的 18B20 之后，主机可以发布一条 18B20 的功能指令。这些指令允许主机从 18B20 的暂存器中读写数据，启动温度转换以及查询供电模式，如下文所述。

Convert T[44h]

该指令启动一次温度转换过程,转换得到的温度数据存储在暂存器中的 2 个字节内，然后 NS18B20 返回至低功耗空闲状态。如果从设备处于寄生供电模式下，那么该指令发布后最多 10μs 之内，主机必须对单总线进行拉高，并且持续整个温度转换期间（tCONV）。如果 18B20 是由外部电源供电的，那么主机可以在 Convert T 指令后发布读时隙，然后 18B20 就会响应 0 或者 1 来表示温度转换正在进行中或者已经完成。寄生供电模式不适用于这种查询机制，因为单总线在整个转换过程中都保持高电平。

Write Scratchpad[4Eh]

该指令允许主机向 18B20 暂存器 Scratchpad1 中写入 3 个字节数据。第一个字节被写入 TH 寄存器（Scratchpad1 的字节 2），第二个字节被写入 TL 寄存器（字节 3），第三个字节被写入配置寄存器（字节 4）。数据传输由最低位开始。所有 3 个字节必须在主机发出复位信号前写入，否则数据可能损坏。其 Scratchpad1 结构如图 3.13 所示。

Read Scratchpad[BEh]

该指令允许主机读取暂存器 Scratchpad1 中的内容。数据传输始于字节 0 的最低位并持续遍历整个暂存器直至第 9 个字节（字节 8——循环冗余校验码）被读取。如果只需要读取暂存器中的部分数据，主机可以随时发布一个复位信号来终止读取操作。

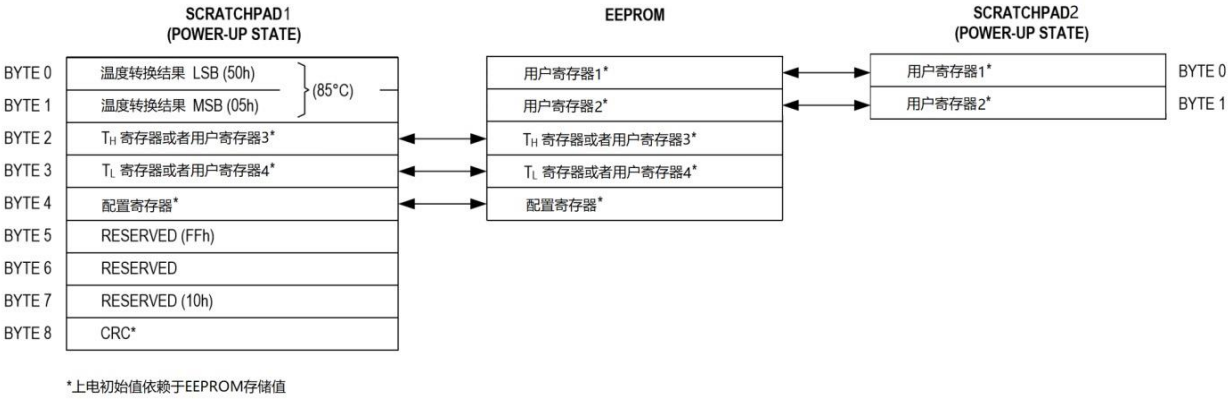


图 3.13　DS18B20 Scratchpad1

在 Scratchpad1 中第 2 个字节用于存放 DS18B20 内置的报警温度上限，第 3 个字节用于存放 DS18B20 内置的报警温度下限，字节 4 是配置寄存器，组织形式如图 3.14。用户可以通过设定表 3.2 中的 R0 和 R1 位来配置 18B20 的转换分辨率。上电默认值是 R0=1 和 R1=1（12 位分辨率）。位 0 到位 4 再加上位 7 是内部保留位，不可被改写。

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

图 3.14　DS18B20 Scratchpad1 字节 4 组织形式

表 3.2　DS18B20 分辨率设置

R1	R0	RESOLUTION (BITS)
0	0	9
0	1	10
1	0	11
1	1	12

3.3.4 测量温度程序

了解了 DS18B20 的使用规则后帮我们就可以开始编写测量温度的程序，具体程序如图 3.15 所示。

```
extern uchar read_temp(uchar z)    /*读取温度值并转换函数*/
{
    uchar th,tl;
    float tt;
/*************************************************************************/
    ds18b20reset();         //初始化
//  ds18b20_write(0xCC);          //发送Skip Rom命令，跳过匹配直接转换温度
    if(z==1)
    {
        b20_Matchrom(1);  //匹配ROM1
    }
    if(z==2)
    {
        b20_Matchrom(2);  //匹配ROM2
    }
    ds18b20_write(0x44);          //发送Convert T命令，开始转换温度
/*************************************************************************/
    ds18b20reset();         //初始化
    if(z==1)
    {
        b20_Matchrom(1);  //匹配ROM1
    }
    if(z==2)
    {
        b20_Matchrom(2);  //匹配ROM2
    }
    ds18b20_write(0x4E);          //发送write scratchpad命令,向18B20写三个字节，分别为报警上限、下限以及转换分辨率设置
    ds18b20_write(0xFF);          //由于不用18B20内部的报警，所以将上下限设置为最高与最低
    ds18b20_write(0x00);
    ds18b20_write(0x7F);          //设置转换精度—1F：9位——3F：10位——5F：11位——7F：12位—
/*************************************************************************/

     ds18b20reset();       //初始化
     if(z==1)
     {
         b20_Matchrom(1);  //匹配ROM1
     }
     if(z==2)
     {
         b20_Matchrom(2);  //匹配ROM2
     }
     ds18b20_write(0xbe);     //读取温度
     tl = ds18b20_read();     //低8位
     th = ds18b20_read();     //高8位
     tvalue = th;
     tvalue<<=8;
     tvalue=tvalue|tl;
     if(tvalue<0x0fff)
         tflag=0;
     else
     {
         tvalue=~tvalue+1;  //补码取反加一
         tflag=1;
     }
     tt=tvalue*0.0625;       //乘最小位的权值转换为10进制数值
     tvalue=tt*100;          //要显示两位小数，乘100用于取模提取数字
     return(tvalue);
 }
```

图 3.15　测量数据程序

每次询问 DS18B20 之前都严格遵循三个步骤：初始化、ROM 命令、功能指令。第一次询问时发送 Convert T 命令，令其开始转换温度。第二次询问时发送 Write Scratchpad 命令，后接三个字节，目的是设置传感器的转换分辨率。最后一次访问发送 Read Scratchpad 命令读取其温度。

读取完温度后，由于其温度是以补码的形式存在的，因此加一个判断，如果是负数就取反加一，求出其绝对值，在设置一个标志位用于判断但前温度是正数还是负数。且将最终结果×100，目的是为了提取每一位的数字至小数点后两位。

3.4 主程序

编写完两个主要器件的函数后，接下来就是报警程序和按键检测程序，为了防止篇幅过长，这里不在赘述，具体程序见附录二。最终通过主程序来调用上述所描述的函数，从而达到我们所需要的要求。主程序的流程框图如图 3.16 所示。

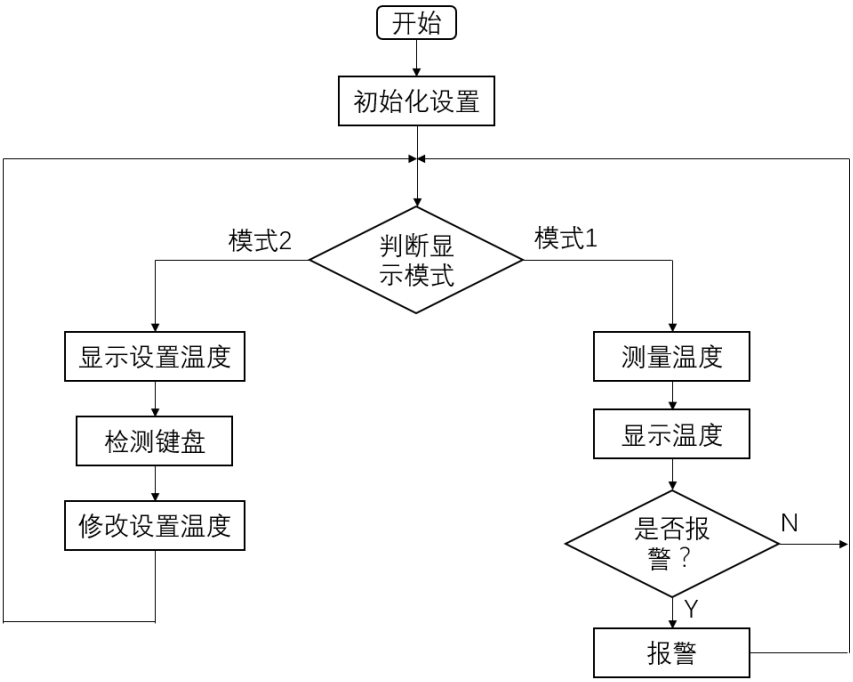


图 3.16　主程序流程框图

上电时，默认进入显示模式 1，即实时显示传感器的温度，当检测到“SET”按钮按下时进入设置模式，此时将显示默认的温度上下限，用户可以根据自己的需求修改，当用户修改完毕后，再次按下“SET”按钮回到模式 1，这时当两个传感器的任何一点检测温度超出设定的范围时就会报警，回到正常温度就会取消报警。

至此，本设计的功能就已经全部实现。主函数程序如图 3.17 所示.

```
void main(void)
{
    lcd_init();//LCD初始化显示
    while(1)
    {
        keypadscan();
        if(dismod_change==1)
        {
            lcd_dis_mod(1);
            while(dismod_change)   //在此状态重复读取温度值并显示
            {
                read_temp(1);//读取温度
                buz_led_alarm();
                lcd_disp_18b20(1);//显示
                read_temp(2);//读取温度
                buz_led_alarm();
                lcd_disp_18b20(2);//显示
                keypadscan();
            }
        }
        if(dismod_change==0)       //在此状态循环检测键盘设置上下限
        {
            lcd_dis_mod(2);
            while(!dismod_change)
            {
                keypadscan();
                if(temp_change)              //判断当前设置是上限还是下限用于提醒用户
                {
                    lcd_write_com(0x81);     //让光标闪烁
                }
                else
                    lcd_write_com(0xc1);
            }
        }
    }
}
```

图 3.17　主程序

第四章 系统仿真调试

在 Keil 中编写完成后，设置频率和仿真单片机的晶振频率一致后，生成 Hex 文件，将其导入 Proteus 里的单片机中就可以对程序进行仿真调试了。经调试后程序的功能与预想的情况符合。

仿真开始，LCD 实时显示传感器的温度，仿真效果如图 4.1 所示。

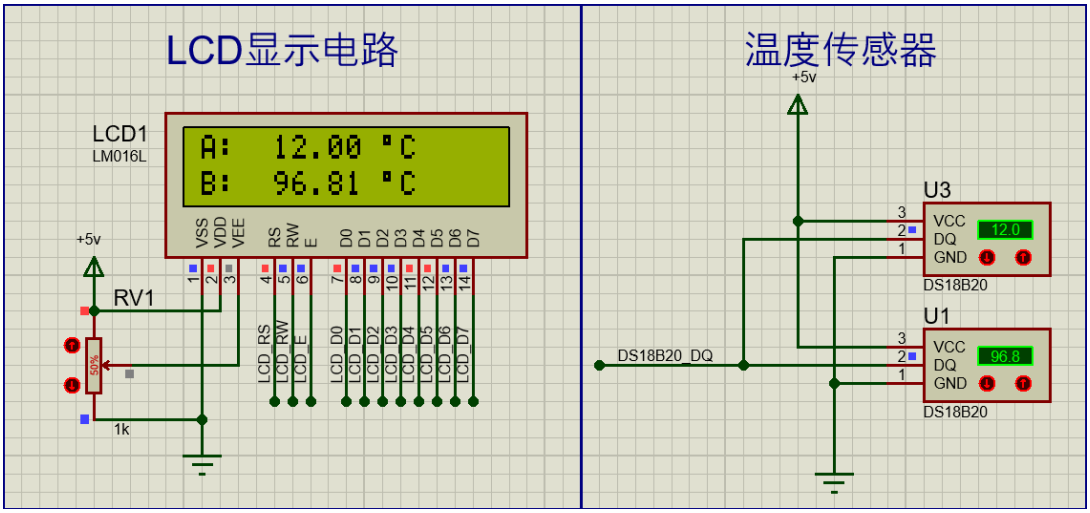


图 4.1 上电仿真结果

当按下“SET”按钮时进入设置模式。内置预设上限为 110° C，下限为-5° C。效果如图 4.2 所示。

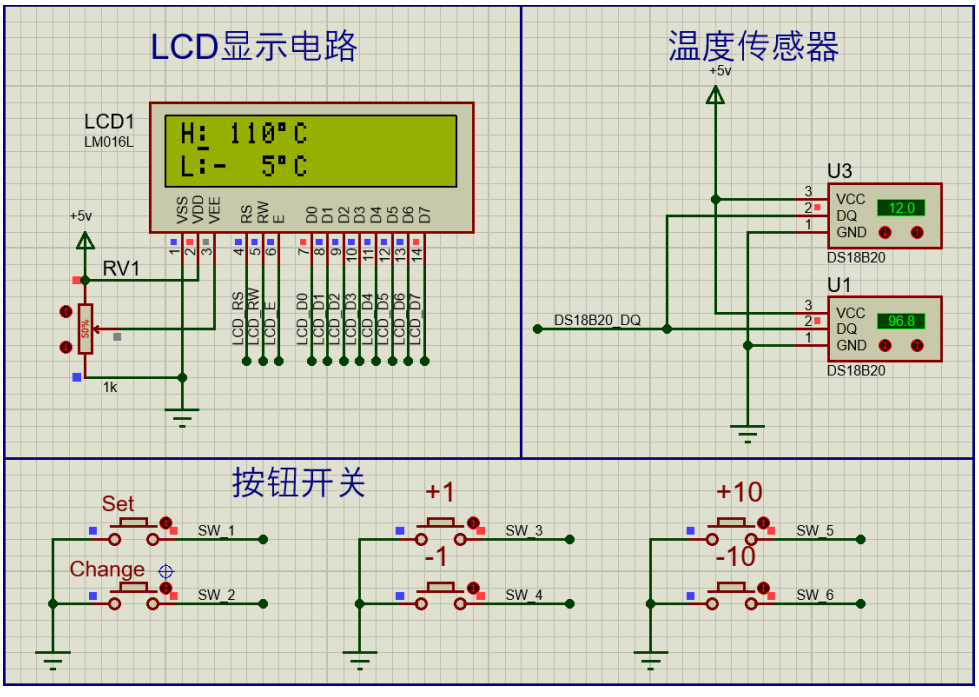


图 4.2 进入设置模式

用户可以通过“+1”按钮、“-1”按钮、“+10”按钮、“-10”按钮设置上下限温度的加减，通过“Change”按钮来选择时改变上限还是下限，当用户选择上限时，光标将会在上限处闪烁，设置下限时同理，用于提示用户当前设置的是上限还是下限。

这里通过按钮将上限设置为 100° C，下限设置为 0° C。效果如图 4.3 所示。

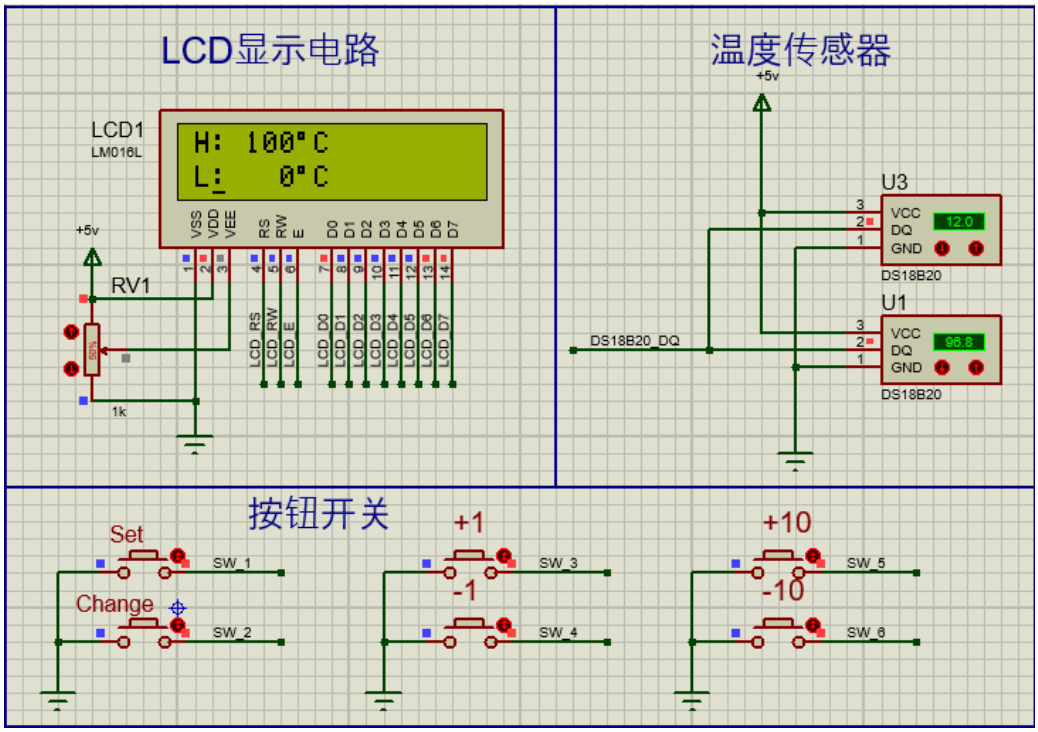


图 4.3　设定报警上下限

当温度任何一个传感器的温度超出上下限时报警电路就会报警，效果如图 4.4 所示。

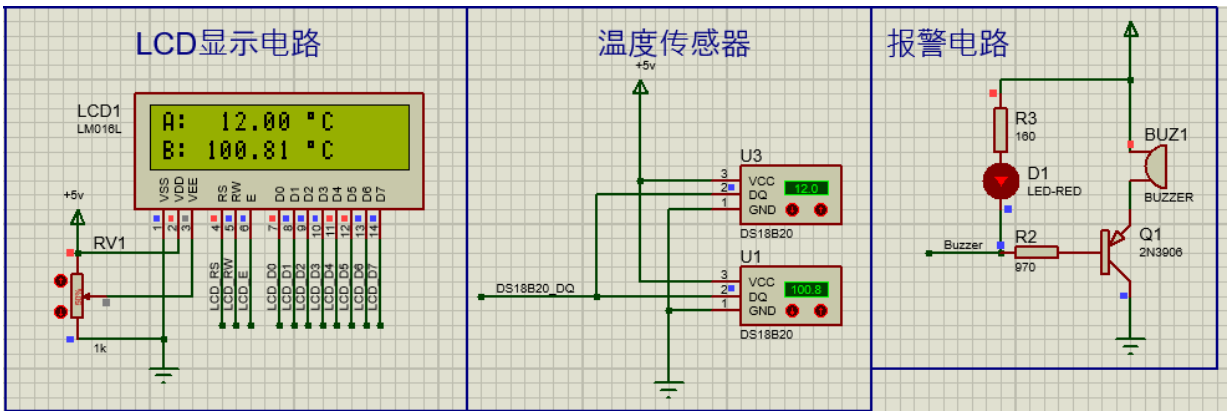


图 4.4　报警

第五章 总结

非常感谢学校和老师能给我们提供这一次课程设计的机会，通过这次课程设计，我学到了许多我在课本上学不到的知识。这次课程设计让我们更加直接的了解到使用的电气知识。虽然只是仿真设计，但能最终做出来也不是一件容易事。

在课程设计当中，我也遇到过许多困难，不过最终都花时间和精力去查阅参考文献和请教老师同学解决了。通过这次课程设计，第一次根据芯片的数据手册编写程序，第一次自己去完整的阅读一份英文的资料，对电气方面的了解又更深入了一个层次。一开始也是不会如何显示多点温度，对一根总线实现检测多点温度还是一头雾水，但是不断的查阅相关资料，反复阅读和参考，终于解决了问题。不过有一个小遗憾就是由于时间较紧，没能做出一个矩阵键盘，用户使用时还不够人性化，如果以后有时间我将会对其进一步的完善，也可以增加一些别的功能。

通过本次课程设计，我也发现了我在知识上存在许多的漏洞，还有很多知识掌握的不够牢固。在课程设计的过程中，通过看书和上网等途径进行知识的漏洞修补。由于我们的知识还不够完善，本次实验的设计还存在很多不完善的地方，需要老师给予指正，同时我们也将继续加强有关知识的学习。

参 考 文 献

[1] 李锐.基于 AT89C51 单片机的大棚温控系统设计.电子制作,2020(2):8-10.

[2] 徐雷,崔子晨,刘俊俊,王浩然,李文娟.基于 DS18B20 温度监测报警系统的设计与实现.安庆师范大学学报(自然科学版),2020,26(1): 89-92.

[3] 朱定华，戴淑萍.单片机微机原理与应用[M].北京：清华大学出版社，2003.

[4] 陈芳，江和.温度传感器 DS18B20 在 Proteus 中的仿真[J].单片机与嵌入式系统应用，2010(7)：17-20.

[5] 李光飞.单片机课程设计实例指导[M].北京：北京航空航天大学出版社，2004.

[6] 金建设. 单片机单片机系统及应用[M].2 版.北京：北京邮电大学出版社，2013.

[7] 张毅刚，杨智明，付宁. 基于 Protues 的单片机课程的基础实验和课程设计[M]. 北京人民邮电出版社，2013.

[8] 周国运.单片机原理与接口技术（C 语言版）.北京:清华大学出版社,2014.

[9] 岂兴明，唐杰.51 单片机编程基础与开发实例详解[M].北京：人民邮电出版社，2008.

[10] 谢自美．1-wire 单总线的基本原理[OL].[2010-03]www.symcukf.com.

[11]谢自美．电子线路设计·实验·测试 [M]．华中理工大学出版社，2001.

[12]翟政凯.基于单片机的单总线多点温度测控系统 [J] .电子测试,2017, (1)

[13] 胡汗才.单片机原理与接口技术[M].北京：清华大学出版社，2004.

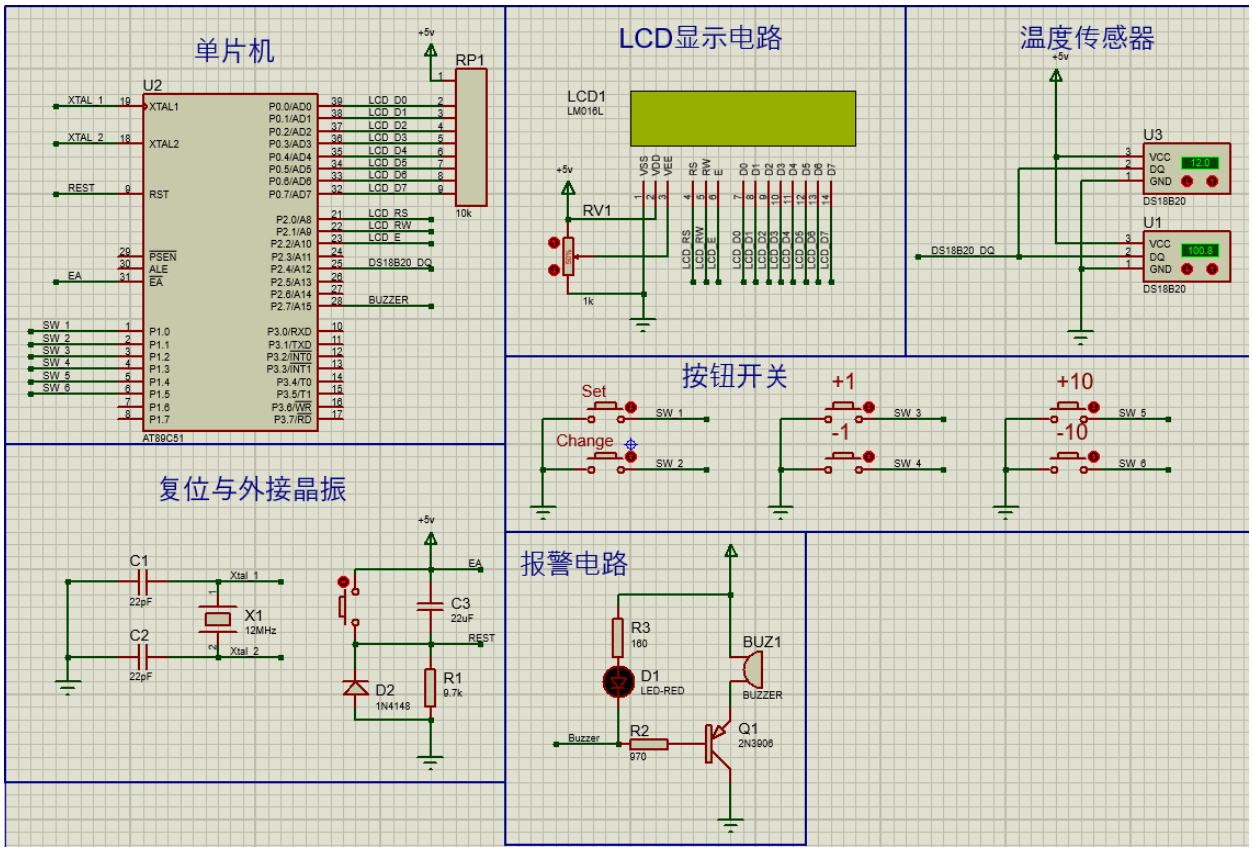
[14] 郑三婷.温度传感器 DS18B20 在温度计设计中的应用[J].电子制作.2019(12).

[15] 张汝铅,王玉暖,杜军,赵曰峰.基于单片机的温度检测系统的设计[J].张汝铅,王玉暖,杜军,赵曰峰.山东师范大学学报(自然科学版).2016(02).

[16] 张仲明,郭东伟,吕巍,张立明.基于 DS18B20 温度传感器的温度测量系统设计[J].实验技术与管理.2018(05).

附　　录

附录 1：完整电路图


单片机
U2
AT89C51
RP1
10k
LCD显示电路
LCD1
LM016L
RV1
1k
温度传感器
U3
U1
DS18B20
DS18B20_DQ
按钮开关
Set
Change
+1
-1
+10
-10
SW 1
SW 2
SW 3
SW 4
SW 5
SW 6
复位与外接晶振
C1
22pF
C2
22pF
X1
12MHz
C3
22uF
R1
9.7k
D2
1N4148
报警电路
R3
160
D1
LED-RED
R2
970
Q1
2N3906
BUZ1
BUZZER
Buzzer

附录 2：完整程序

Main.c

```
#include"ds18b20.h"
#include"keypad.h"
#include"alarm.h"

void main(void)
{
    lcd_init();//LCD 初始化显示
    while(1)
    {
        keypadscan();
        if(dismod_change==1)
        {
            lcd_dis_mod(1);
            while(dismod_change)  //在此状态重复读取温度值并显示
            {
                read_temp(1);//读取温度
                buz_led_alarm();
                lcd_disp_18b20(1);//显示
                read_temp(2);//读取温度
                buz_led_alarm();
                lcd_disp_18b20(2);//显示
                keypadscan();
            }
        }
        if(dismod_change==0)   //在此状态循环检测键盘设置上下限
        {
            lcd_dis_mod(2);
            while(!dismod_change)
            {
                keypadscan();
                if(temp_change)//判断当前设置是上限还是下限用于提醒用户
                {
                    lcd_write_com(0x81); //让光标闪烁
                }
                else
                    lcd_write_com(0xc1);
            }
        }
    }
}
```

userdef.h

```
#ifndef _userdef_H_
#define _userdef_H_

typedef unsigned int uint;
typedef unsigned char uchar;

#endif
```

Lcd1602.c

```
#include <reg51.h>
```

```
#include"userdef.h"
#include"lcd1602.h"
#include"ds18b20.h"

uchar data disdata[4];     //用于存放显示数字数组

static void delay1ms(uint x)  /*延时 1 毫秒函数*/
{
    uint i,j;
    for(i=0;i<x;i++)
       for(j=0;j<100;j++);
}

extern void lcd_write_com(uchar com)    /*LCD 写命令函数*/
{
     delay1ms(1);
     LCD_RS = 0;
     LCD_RW = 0;
     LCD_E = 0;
     P0 = com;
     delay1ms(1);
     LCD_E = 1;
     delay1ms(1);
     LCD_E = 0;
}

static void lcd_write_dat(uchar dat)    /*LCD 写数据函数*/
{
     delay1ms(1);;
     LCD_RS = 1;
     LCD_RW = 0;
     LCD_E = 0;
     P0 = dat;
     delay1ms(1);
     LCD_E = 1;
     delay1ms(1);
     LCD_E = 0;
}

extern void lcd_init()          /*LCD 初始化设置*/
{
    lcd_write_com(0x01);    //清屏
    delay1ms(1);
    lcd_write_com(0x38);    //设置显示两行、每个字符为 5*7 点阵、数据线为
8 根
delay1ms(1);
lcd_write_com(0x06);    //输入后地址指针右移
delay1ms(1);
}

extern void lcd_dis_mod(uchar x)      /*显示方式设置，1 为实时显示，2 为
设置模式*/
{
if(x==1)
{
   lcd_write_com(0x0c);    //显示打开、光标关闭
   delay1ms(1);
   lcd_write_com(0x80);
```

```
    lcd_write_dat('A');
    lcd_write_dat(':');
    lcd_write_com(0x8A);
    lcd_write_dat(0xdf);     //显示摄氏温度字符°
    lcd_write_dat('C');
    lcd_write_com(0xc0);
    lcd_write_dat('B');
    lcd_write_dat(':');
    lcd_write_com(0xcA);
    lcd_write_dat(0xdf);         //显示摄氏温度字符°
    lcd_write_dat('C');
}
if(x==2)
{
    lcd_write_com(0x0f);    //显示打开、光标打开并闪烁
    delay1ms(1);
    lcd_write_com(0x80);
    lcd_write_dat('H');
    lcd_write_dat(':');
    lcd_write_com(0x86);
    lcd_write_dat(0xdf);         //显示摄氏温度字符°
    lcd_write_dat('C');
    lcd_write_com(0xc0);
    lcd_write_dat('L');
    lcd_write_dat(':');
    lcd_write_com(0xc6);
    lcd_write_dat(0xdf);         //显示摄氏温度字符°
    lcd_write_dat('C');
}
}

extern void lcd_disp_alarm(char xx)    /*报警设置显示函数*/
{
uchar flagdat;
if(xx>=0)
{
    flagdat=0x20;//正温度不显示符号
}
else
{
    flagdat=0x2d;//负温度显示负号:-
    xx = xx * -1;
}

disdata[0]=xx/100+0x30;//百位数
disdata[1]=xx/10%10+0x30;//十位数
disdata[2]=xx%10+0x30;//个位数

if(disdata[0]==0x30)
{
    disdata[0]=0x20;//如果百位为0，不显示
        if(disdata[1]==0x30)
    {
        disdata[1]=0x20;//如果百位为0，十位为0也不显示
    }
}
    lcd_write_dat(flagdat);//显示符号位
    lcd_write_dat(disdata[0]);//显示百位
```

```
    lcd_write_dat(disdata[1]);//显示十位
    lcd_write_dat(disdata[2]);//显示个位
}

extern void lcd_disp_18b20(uchar x)    /*温度值显示函数*/
{
uchar flagdat;
disdata[0]=tvalue/10000+0x30;//百位数
disdata[1]=tvalue%10000/1000+0x30;//十位数
disdata[2]=tvalue%1000/100+0x30;//个位数
disdata[3]=tvalue%100/10+0x30;//小数点后一位
disdata[4]=tvalue%10+0x30;//小数点后两位

if(tflag==0)
   flagdat=0x20;//正温度不显示符号
else
   flagdat=0x2d;//负温度显示负号:-

if(disdata[0]==0x30)
{
   disdata[0]=0x20;//如果百位为0，不显示
       if(disdata[1]==0x30)
   {
      disdata[1]=0x20;//如果百位为0，十位为0也不显示
   }
}

    if(x==1)
    {
   lcd_write_com(0x82); //显示地址位置
}
if(x==2)
{
   lcd_write_com(0xc2);
}
    lcd_write_dat(flagdat);//显示符号位
    lcd_write_dat(disdata[0]);//显示百位
    lcd_write_dat(disdata[1]);//显示十位
    lcd_write_dat(disdata[2]);//显示个位
    lcd_write_dat(0x2e);//显示小数点
lcd_write_dat(disdata[3]);//显示小数点后一位
lcd_write_dat(disdata[4]);//显示小数点后两位
}
```

Lcd1602.h

```
#ifndef _lcd1602_H_
#define _lcd1602_H_

sbit LCD_E = P2^2;
sbit LCD_RW = P2^1;
sbit LCD_RS = P2^0;

extern void lcd_init();
extern void lcd_write_com(uchar com);
extern void lcd_dis_mod(uchar x);
extern void lcd_disp_18b20(uchar);
extern void lcd_disp_alarm(char xx);
```

```
#endif
```

DS18b20.c

```
#include <reg51.h>

#include"userdef.h"
#include"lcd1602.h"
#include"ds18b20.h"

uchar code str1[] = {0x28,0x30,0xC5,0xB8,0x00,0x00,0x00,0x8E}; //ROM1
uchar code str2[] = {0x28,0x31,0xC5,0xB8,0x00,0x00,0x00,0xB9}; //ROM2
uint tvalue;//温度值
bit tflag;//温度正负标志

static void delay5us(uint i)  /*延时 5 微秒函数*/
{
     while(i--);
}

static void ds18b20reset()     /*ds1820 复位函数*/
{
    uchar x=0;
    LCE_DQ = 1;        //DQ 复位
    delay5us(4);    //延时
    LCE_DQ = 0;        //DQ 拉低
    delay5us(100);      //精确延时大于 480us
    LCE_DQ = 1;        //DQ 拉高
    delay5us(40);
}

static uchar ds18b20_read()   /*读一个字节数据*/
{
uchar i = 0;
    uchar dat = 0;
    for (i=0;i<8;i++)
{
   LCE_DQ = 0; //给脉冲信号
   dat>>=1;
   LCE_DQ = 1; //给脉冲信号
   if(LCE_DQ)
   dat|=0x80;
   delay5us(10);
}
return(dat);
}

static void ds18b20_write(uchar dat)  /*写一个字节数据*/
{
uchar i=0;
    for (i=0; i<8; i++)
{
   LCE_DQ = 0;
        LCE_DQ = dat&0x01;
        delay5us(10);
        LCE_DQ = 1;
        dat>>=1;
    }
```

```
}

static void b20_Matchrom(uchar x)  /*匹配 ROM 函数*/
{
    char j;
    ds18b20_write(0x55);    //发送匹配 ROM 命令
    if (x==1)
    {
        for(j=0;j<8;j++)
        ds18b20_write(str1[j]); //发送 18B20 的序列号，先发送低字节
    }
    if(x==2)
    {
    for(j=0;j<8;j++)
        ds18b20_write(str2[j]); //发送 18B20 的序列号，先发送低字节
    }
}

extern uchar read_temp(uchar z)   /*读取温度值并转换函数*/
{
    uchar th,tl;
    float tt;
/*************************************************************************
*/
    ds18b20reset();        //初始化
//  ds18b20_write(0xCC);      //发送 Skip Rom 命令，跳过匹配直接转换温度
if(z==1)
    {
        b20_Matchrom(1);  //匹配 ROM1
    }
    if(z==2)
{
        b20_Matchrom(2);  //匹配 ROM2
}
ds18b20_write(0x44);       //发送 Convert T 命令，开始转换温度
/*************************************************************************
/
    ds18b20reset();        //初始化
    if(z==1)
    {
        b20_Matchrom(1);  //匹配 ROM1
    }
    if(z==2)
{
        b20_Matchrom(2);  //匹配 ROM2
}
ds18b20_write(0x4E);       //发送 write scratchpad 命令,向 18B20 写三个字
节，分别为报警上限、下限以及转换分辨率设置
ds18b20_write(0xFF);       //由于不用 18B20 内部的报警，所以将上下限设置
为最高与最低
ds18b20_write(0x00);
ds18b20_write(0x7F);       //设置转换精度——1F：9 位————3F：10 位——
————5F：11 位————7F：12 位——
/*************************************************************************
****/
    ds18b20reset();       //初始化
    if(z==1)
    {
```

```
        b20_Matchrom(1);  //匹配 ROM1
    }
    if(z==2)
{
        b20_Matchrom(2);  //匹配 ROM2
}
ds18b20_write(0xbe);    //读取温度
tl = ds18b20_read();    //低 8 位
th = ds18b20_read();    //高 8 位
tvalue = th;
tvalue<<=8;
tvalue=tvalue|tl;
if(tvalue<0x0fff)
       tflag=0;
    else
    {
    tvalue=~tvalue+1;  //补码取反加一
    tflag=1;
    }
    tt=tvalue*0.0625;   //乘最小位的权值转换为 10 进制数值
    tvalue=tt*100;          //要显示两位小数，乘 100 用于取模提取数字
    return(tvalue);
}
```

DS18b20.h

```
#ifndef _ds18b20_H_
#define _ds18b20_H_

sbit LCE_DQ=P2^4;

extern uint tvalue;//温度值
extern bit tflag;//温度正负标志

extern uchar read_temp(uchar);

#endif
```

keypad.c

```
#include <reg52.h>
#include"userdef.h"
#include"lcd1602.h"
#include"keypad.h"

bit dismod_change = 1;       //显示模式设置，1 为实时显示模式，0 为设置报
警模式
bit temp_change = 1;    //温度设置标志位，1 为设置上限，0 为设置下限
char higtemp = 110;      //分别存放报警上下限
char lowtemp = -5;

static void delay1ms(uint x)  /*延时 1 毫秒函数*/
{
    uint i,j;
    for(i=0;i<x;i++)
       for(j=0;j<100;j++);
}

extern void keypadscan(void)     /*键盘读取函数*/
```

```
{
if(SW_1==0)              //显示模式转化开关
{
    delay1ms(10);    //延时 10 毫秒后再次扫描消抖
    if(SW_1==0)
    {
        lcd_init();    //LCD 初始化显示
        dismod_change = !dismod_change;
        lcd_write_com(0x82);
        lcd_disp_alarm(higtemp);
        lcd_write_com(0xC2);
        lcd_disp_alarm(lowtemp);
        while(!SW_1);    //直到松开按钮，防止误触
    }
}

if(SW_2==0)              //上下限转换开关
{
    delay1ms(10);
    if(SW_2==0)
    {
        temp_change = !temp_change;
        while(!SW_2);
    }
}

if(SW_3==0)            //＋1 开关
{
    delay1ms(10);
    if(SW_3==0)
    {
        if(!dismod_change)
        {
            if(temp_change)
                higtemp = higtemp + 1;
            if(!temp_change)
            {
                lowtemp = lowtemp + 1;
            }
            lcd_write_com(0x82);
            lcd_disp_alarm(higtemp);
            lcd_write_com(0xC2);
            lcd_disp_alarm(lowtemp);
            lcd_dis_mod(2);
        }
        while(!SW_3);
    }
}

if(SW_4==0)           //-1 开关
{
    delay1ms(10);
    if(SW_4==0)
    {
        if(!dismod_change)
        {
            if(temp_change)
                higtemp = higtemp - 1;
```

```
            if(!temp_change)
            {
                lowtemp = lowtemp - 1;
            }
            lcd_write_com(0x82);
            lcd_disp_alarm(higtemp);
            lcd_write_com(0xC2);
            lcd_disp_alarm(lowtemp);
            lcd_dis_mod(2);
        }
        while(!SW_4);
    }
}

if(SW_5==0)             //+10 开关
{
    delay1ms(10);
    if(SW_5==0)
    {
        if(!dismod_change)
        {
            if(temp_change)
                higtemp = higtemp + 10;
            if(!temp_change)
            {
                lowtemp = lowtemp + 10;
            }
            lcd_write_com(0x82);
            lcd_disp_alarm(higtemp);
            lcd_write_com(0xC2);
            lcd_disp_alarm(lowtemp);
            lcd_dis_mod(2);
        }
        while(!SW_5);
    }
}

if(SW_6==0)             //-10 开关
{
    delay1ms(10);
    if(SW_6==0)
    {
        if(!dismod_change)
        {
            if(temp_change)
                higtemp = higtemp - 10;
            if(!temp_change)
            {
                lowtemp = lowtemp - 10;
            }
            lcd_write_com(0x82);
            lcd_disp_alarm(higtemp);
            lcd_write_com(0xC2);
            lcd_disp_alarm(lowtemp);
            lcd_dis_mod(2);
        }
        while(!SW_6);
    }
```

```
}
}
```

keypad.h

```
#ifndef _keypad_H_
#define _keypad_H_

sbit SW_1 = P1^0;
sbit SW_2 = P1^1;
sbit SW_3 = P1^2;
sbit SW_4 = P1^3;
sbit SW_5 = P1^4;
sbit SW_6 = P1^5;
extern bit dismod_change;    //显示模式设置，1 为时时显示模式，0 为设置
报警模式
extern bit temp_change;     //温度设置标志位，1 为设置上限，0 为设置下限
extern char higtemp;      //存放温度报警上限
extern char lowtemp;      //存放温度报警下限
extern void keypadscan(void);
#endif
```

alarm.c

```
#include <reg52.h>
#include"userdef.h"
#include"alarm.h"
#include"ds18b20.h"
#include"keypad.h"
extern void buz_led_alarm(void)
{
int temp_compare_hig;
int temp_compare_low;
int re_value = tvalue;
temp_compare_hig = higtemp * 100;
temp_compare_low = lowtemp * 100;

if(tflag==1)      //如果是负数
   re_value = re_value * -1;
if(re_value>temp_compare_hig)
{
   BUZ = 0;
}
else
{
   if(re_value<temp_compare_low)
      BUZ = 0;
   else
      BUZ = 1;
}
}
```

alarm.h

```
#ifndef _alarm_H_
#define _alarm_H_
sbit BUZ = P2^7;

extern void buz_led_alarm(void);
#endif
```