# 计算机科学与技术学院

# 《单片机控制实验》

# 期末题目设计报告

## 班级：

## 教学班：周五下午 422

## 团队成员姓名及学号

### 队长: 周鸿哲 20211920

### 组员:

### 江鑫 21210721

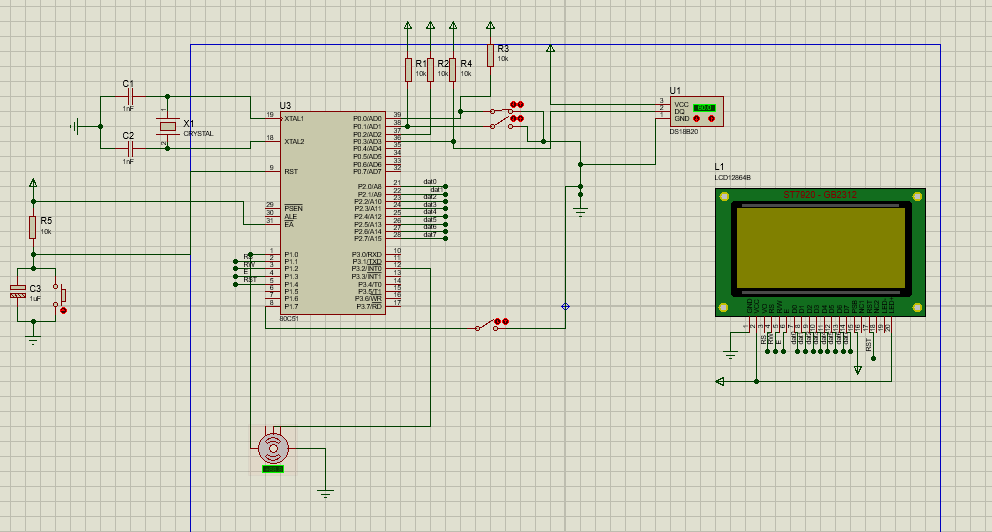
### 翟文岩 21210316

### 郭林铮 16210509

2025年 11月 26日

1. 设计题目 （每一个题目写一个报告)

**制冷系统**



1. 设计目标 （描述系统主要功能作用，包括扩展设计的内容）
2. 使用温度传感器 DS18B20 测量温度，每秒钟测量一次；
3. 按键来改变设定温度
4. 当测量温度高于设定温度时，自行设计一个函数决定电机的转速，差距越大，转 速越快；当测量温度不高于设定温度时，电机停止转动。
5. 直流电机使用 PWM 方式控制转速；
6. 液晶显示屏使用 STC7920 控制，有两种功能，功能 1：在屏幕上显示系统状态的 文字信息；功能 2：在适当位置显示若干秒以内的温度曲线（折线图）。功能切换 由某个按键开关控制。
7. 扩展部分:

ST7920通过并列模式，进行绘图,**同时会根据折线的上升下降趋势，显示对应的图案，如果不变则显示条状码**。绘制过程横向单位为区域(16像素)，且**只有绘制完区域左侧，才能绘制区域右侧**。**同一区域，一次字节写入（绘制8像素）会清空该字节对应的屏幕8像素**。因此设置秒为单位，均匀变化绘制折线。

先**计算每个像素点y坐标变化步长add**,与上次绘制最终点y坐标差值为row，

**如果row<1,则计算row变化超过1为止经过的位置，进行一次性绘制。**

**如果row>=1则需要一个像素点改变row次y坐标进行绘制。**

**这样保证了每次绘制至少变化一行，且在一个区域内每次绘制y坐标不会重复，防止了重置字节内已画区域的情况。**

* 绘制折线图根据温度变化梯度，进行8每秒以内的温度曲线图（初始温度未采样的点，初值为0，不会进行绘制。绘制折线图的同时，会在下方显示8秒以内的温度数值。为了保证绘制动态折线图和下方的静态温度数字显示正常不干扰。研究有关材料和多次实验调试，对于不同绘制指令设置延时操作（资料上0x01清空文字指令占1.6ms,其它指令大多占据72us,但实际仿真的时候，考虑到其它指令执行需要更改)，保证绘图操作完成。

绘图模式:

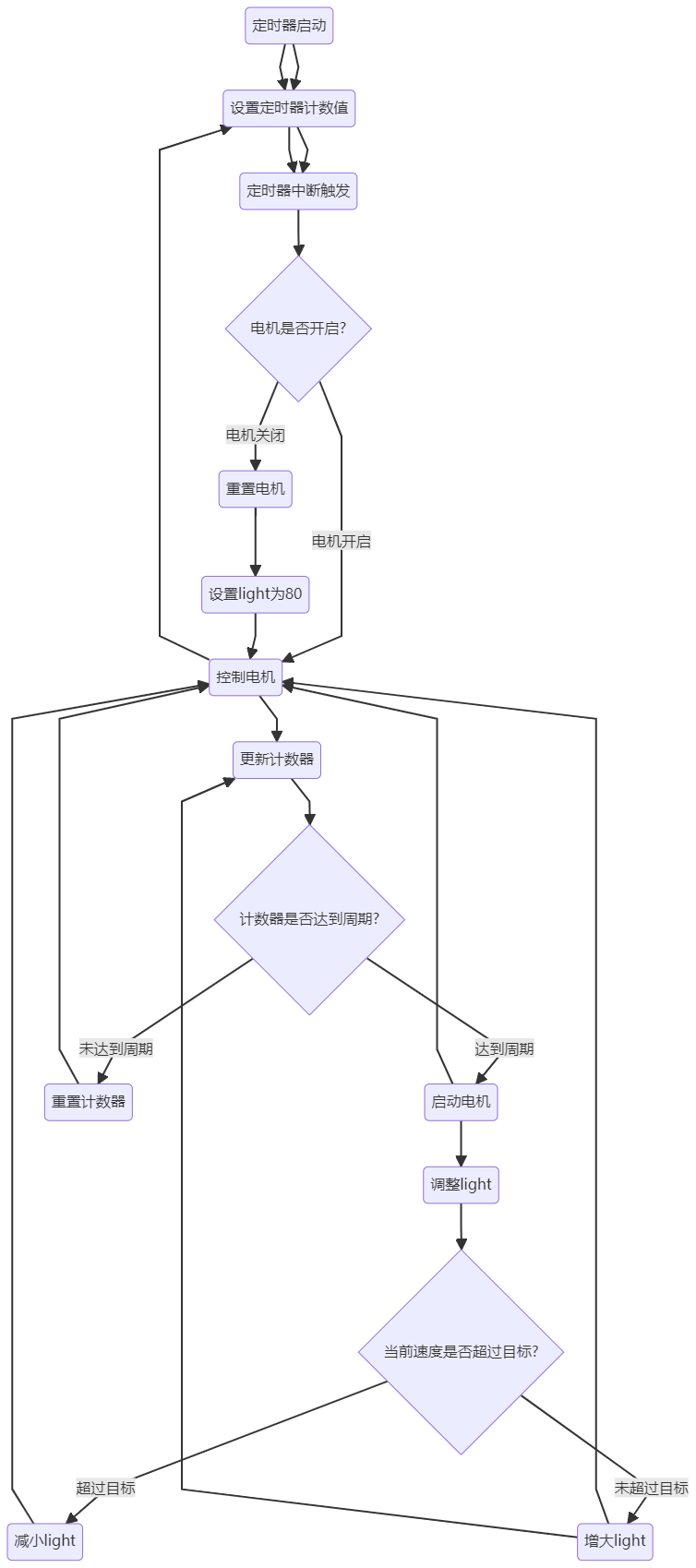
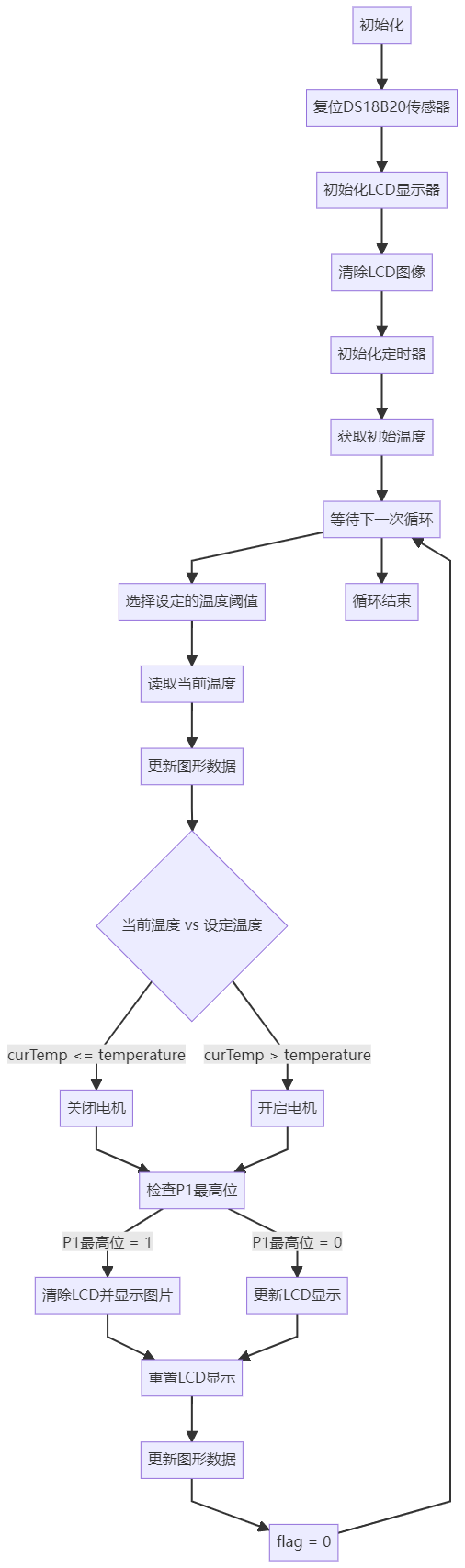
* 点与点之间各在一个区域(16像素)进行连线绘制。
* 对于起伏大的曲线，经过x坐标变化一个像素，y坐标会一个像素抬升多个像素格，均匀变化。
* 对于起伏小的曲线，x坐标经变化多个像素y坐标不一定变化，通过计算上一次绘制的行，一次性绘制之后相同y坐标的点直到变化为止。

1. 分工及成员贡献度 序号 学号 姓名 任务分工 工作量占比 1 2 3 4 说明：此表是老师为每位同学打分的重要参考，必须填写，由组长组织成员共同完 成。序号是经过组内认可的贡献排名；任务分工为学生实际在题目完成过程中的具体工 作内容，如：整体设计、模块实现、程序测试、报告撰写等；工作量占比是对题目完成 的贡献占比，原则上序号第一者占比最高，其他成员贡献权重由组内成员共同商议给出， 总和为 100%。

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 组内分工 | 工作量占百分比 |
| 周鸿哲 | 20211920 | 题目1：负数显示设计，程序测试，键盘扫描  题目2：ST7920显示（折线图,趋势图形），绘制机制设计，显示抗干扰调试，温度反馈调节直流电机 ,ds1820读取,电路设计。 | 50% |
| 郭林铮 | 16210509 | 题目1:加减乘除实现 题目2：清屏指令和显示指令延时测试 | 30% |
| 江鑫 | 21210721 | 程序测试 | 10% |
| 翟文岩 | 21210316 | 报告撰写 | 10% |

1. 设计与开发

总体设计：根据问题要求设计方案，说明涉及到的原理，并分析设计的局限 性。该项目是基于51单片机的温度监控系统，主要功能包括通过DS18B20温度传感器获取环境温度并显示在LCD屏上，同时实现温度控制，调整风扇或加热器的开关。系统在检测到当前温度高于设定的温度时会启动车辆电机，低于设定温度时关闭电机。

****

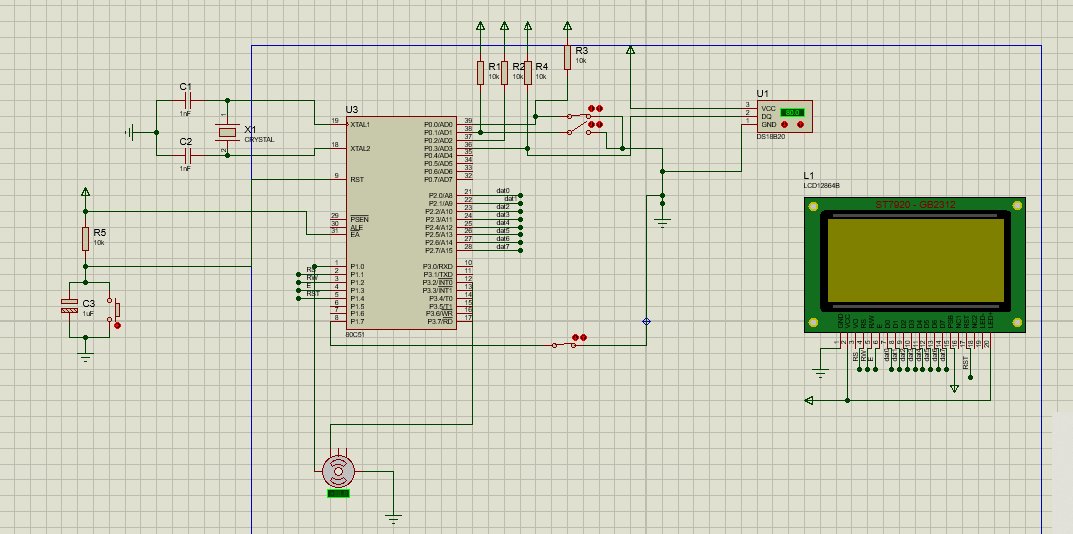
**原理分析：**

* **温度获取原理：** 该系统使用DS18B20温度传感器，基于1-Wire协议工作，通过单线接口与单片机通信，实现温度数据的读取。读取的数据经过转换得到实际温度值。
* **LCD显示原理：** LCD显示模块采用并行接口，通过控制寄存器完成字符显示，图形显示（例如温度波形图）通过控制LCD的内存地址进行逐点绘制。
* **定时器与中断：** 系统通过定时器定时获取温度并更新显示，同时通过外部中断实现某些触发条件（如按钮或外部传感器的输入）

**局限性：**

* 由于DS18B20的采样时间较长，更新频率较低，温度变化较快时可能存在一定的延迟，同时过于剧烈如1s温度变化70°C以上可能会出现噪点，因为为需要频繁设置坐标，再写像素点。
* LCD显示区域有限，无法同时显示更多的历史温度数据或波形。
* 系统需要在有限的硬件资源下工作，性能提升有限。

1. 硬件设计：说明设计思路，描述硬件组成，器件选型，接口方式等。给出具 体电路图。



P0.0-P0.2用作开关调节设定的温度。 p0.3用作ds1820数据线，

P1.0作为直流电机驱动，P1.1-P1.4分别为RS,RW,E,RST,接ST7920接口.

P1.7用作ST7920显示模式，显示系统信息或绘制曲线图切换开关

INT0外中断连接直流电机。

1. 软件设计：画出程序框图，对程序代码给出详细的分析和说明，并写出程序 调试的详细过程。
2. **温度读取**：通过DS18B20传感器读取温度数据，并根据设定的温度值控制风扇（DRV引脚）。
3. **LCD显示**：显示当前温度、设定温度、风速等信息。
4. **定时器控制**：使用定时器0定时更新数据并调整风扇的工作状态。
5. **温控逻辑**：当当前温度高于设定温度时，风扇启动，低于设定温度时，风扇停止。

### 关键代码分析：

**温度读取**：getTemper()函数通过DS18B20传感器获取温度。传感器在一个1-Wire协议下工作，reset\_ds18b20()和write\_byte()等函数负责与温度传感器进行通信并获取数据。

复位 DS1820 传感器

reset\_ds18b20();

调用 reset\_ds18b20() 函数复位传感器，确保传感器处于正确的工作状态。这是通过向总线发送复位脉冲来实现的，复位操作确保传感器能够正常响应之后的命令。

启动温度转换

WriteByte(0xcc);

WriteByte(0x44);

0xcc 是 跳过ROM命令，允许对总线上的所有设备发送命令，而无需选择特定设备。

0x44 是 温度转换命令，指示 DS1820 开始转换温度。温度转换需要一定的时间（大约 750 毫秒），在此期间传感器会计算当前的温度值。

等待温度转换完成 由于 DS1820 在进行温度转换时会占用总线，因此必须等待转换完成。可以通过某种方式等待固定时间或轮询来确保转换过程结束。

再次复位并读取温度数据

reset\_ds18b20();

WriteByte(0xbe);

再次复位 DS1820 传感器，并通过发送 0xbe 命令来告诉传感器准备返回温度数据。0xbe 是 读取温度命令，此命令触发传感器将温度值传输回主控。

读取温度低字节和高字节

tempL = ReadByte();

tempH = ReadByte();

通过调用 ReadByte()，分别读取温度的低字节（tempL）和高字节（tempH）。这些字节包含了传感器读取的原始温度数据。

处理温度数据

Temp = ((tempH<<8) + tempL) \* 0.0625;

将高字节和低字节合并成一个 16 位的整数。tempH << 8 将高字节左移 8 位，tempL 保持在低 8 位，从而组合成一个完整的 16 位温度数据。

乘以 0.0625 来将原始数据转换为实际的温度值，DS1820 的温度输出数据是基于 0.0625°C 的步进。

四舍五入处理

Temp = (int)(Temp + 0.5);

对浮动的温度值进行四舍五入，以减少显示时的误差，使温度值更加准确。

返回最终温度值

return Temp;

最后，返回处理过的温度值

LCD显示：LCD\_write\_data()和LCD\_write\_cmd()等函数用于LCD的显示控制。显示内容包括当前温度、设定温度、风速等。

数据展示 (display)

功能描述： 该函数的目的是将实时数据（如当前温度、目标温度、当前电机速度等）显示到 LCD 屏幕上，以便用户能够监控系统的状态。

详细分析：

该函数执行了以下几个步骤：

显示表头：

LCD\_ShowString(0, 0, "T4 T5 T6 T7");

这行代码会在 LCD 屏幕的第 0 行显示标题 "T4 T5 T6 T7"，这些可能是用来标识最近 4 个温度的空间（例如：T4 - 当前时间的温度，T5 - 上一时刻的温度，以此类推）。

graph[] 数组保存了最新的 4 个温度值，程序通过对每个温度值进行 除 10 和 取余 操作，拆分出高位和低位，并使用 LCD\_ShowChar() 显示到 LCD 上。

每个温度显示占用 LCD 屏幕的两个字符（一个高位，一个低位）。因此，通过循环显示，能够在屏幕上更新 4 个温度值。

显示目标温度和当前温度：

LCD\_ShowString(0, 3, "T:");    // 显示 "T:"

LCD\_ShowNum(2, 3, target, 3, 1); // 显示目标温度

LCD\_ShowString(8, 3, "Temp:"); // 显示 "Temp:"

LCD\_ShowNum(13, 3, curTemp, 3, 1); // 显示当前温度

LCD\_ShowString(0, 4, "Speed:"); // 显示 "Speed:"

LCD\_ShowNum(7, 4, speed, 3, 1); // 显示电机速度

通过 LCD\_ShowString() 和 LCD\_ShowNum() 函数，分别在屏幕的不同位置显示目标温度、当前温度和电机速度。

这些数据显示在屏幕的第三行和第四行，分别显示目标温度、当前温度和电机的运行速度。

总结： display 函数主要负责将温度和电机状态信息动态地更新到 LCD 屏幕上。通过逐个字符的显示，将数据清晰地展示给用户，让用户能够实时监控设备的运行状态。

**2. 定时器控制 (timer0)**

功能描述： timer0 函数通常作为定时器中断服务程序执行，用于周期性地更新设备的状态，例如更新电机的启动/停止状态、调整电机的速度等。

详细分析：

定时器溢出处理：

if (now\_time >= 1000) {

    now\_time = 0;

    update\_motor(); // 更新电机状态

}

now\_time 是一个计时器变量，在每次中断时都会增加。当 now\_time 达到 1000 时，意味着经过了 1 秒（根据定时器配置）。

达到 1 秒后，调用 update\_motor() 函数更新电机状态。此处可能涉及电机的启停操作，具体取决于控制逻辑。

电机启停控制：

if (motorOn == 1) {

    if (cnt >= CYCLE) {

        DRV = 1;   // 启动电机

    } else {

        DRV = 0;   // 关闭电机

    }

}

motorOn 标志决定电机是否应该运行。若 motorOn == 1，则根据计数器 cnt 和预设的 CYCLE（周期）值来控制电机的启停。

如果 cnt 达到或超过 CYCLE，则设置 DRV = 1 启动电机；否则，设置 DRV = 0 关闭电机。

电机速度调整：

if (speed > target) {

    LIGHT--;   // 降低电机速度

} else {

    LIGHT++;   // 提高电机速度

}

比较当前电机速度 speed 和目标速度 target。根据比较结果，增减 LIGHT 的值来调整电机的速度。若当前速度超过目标速度，则减少 LIGHT，反之则增加 LIGHT。

定时器更新时间和目标温度更新：

if (now\_time == 1000) {

    target = getTemper(); // 获取当前温度作为目标温度

    speed = calcSpeed(target); // 根据目标温度计算电机速度

}

每经过 1 秒钟，调用 getTemper() 获取当前温度作为新的目标温度 target。

根据新的目标温度，调用 calcSpeed() 函数计算新的电机速度，并更新电机的速度。

该函数主要负责在 LCD 屏幕上绘制图形，尤其是折线图，并动态地根据数据绘制每个数据点。下面详细分析该函数中的关键部分和逻辑：

**LCD\_dis\_pic() 函数分析**

**1. 清除屏幕和绘图区域**

EA = 0;  // 禁用中断

LCD\_set\_pos(2, 0); // 设置显示位置，可能是设置为第二行起始位置

for(i = 0; i < 4; i++) {

    j = graph[i] / 10;  // 将温度拆分为十位

    k = graph[i] % 10;  // 拆分为个位

    LCD\_write\_data(' ');  // 在 LCD 屏幕上写空格

    LCD\_write\_data(j + '0');  // 显示十位

    LCD\_write\_data(k + '0');  // 显示个位

}

LCD\_set\_pos(3, 0);  // 设置显示位置，可能是设置为第三行起始位置

for(i = 4; i < 8; i++) {

    j = graph[i] / 10;

    k = graph[i] % 10;

    LCD\_write\_data(' ');

    LCD\_write\_data(j + '0');

    LCD\_write\_data(k + '0');

}

EA = 0;：禁用全局中断，防止在 LCD 更新过程中产生中断干扰。

使用 LCD\_set\_pos() 设置光标位置，在屏幕的第 2 和第 3 行显示温度值。graph[] 数组中的每个元素代表一个温度值，分别通过除法和取余操作将其拆分为十位和个位。

显示格式为 "T4 T5 T6 T7"（分别为图表的最新数据），显示温度值。

2. 启用图形模式

LCD\_write\_cmd(0x34);  // 启用图形模式

LCD\_write\_cmd(0x34)：发送命令到 LCD 控制器，进入图形模式。这意味着后续的操作将用于绘制图形，而不是文本显示。

3. 绘制折线图的核心逻辑

for (i = 7; i >= 0 && graph[i] != 0; i--) {

    y = 31 - graph[i] / 3;  // 将温度值转换为 y 坐标

    beforeRow = y;

    if (i != 7) {

        add = (float)(j - y) / 16.0;  // 计算增量，用于平滑折线

        k = 0;

        while (k < 16) {

            lowBound = (int)(add \* k);

            bitPos = k;

            if (absN(add \* k - lowBound) > 0.5) {

                row = (int)absN((int)(add \* k + y) - beforeRow) + 1;

            } else {

                row = (int)absN((int)(add \* k + y) - beforeRow);

            }

            if (row == 0) {

                bitPos++;

                row = (int)absN((int)(add \* bitPos + y) - beforeRow);

                while (row == 0 && bitPos < 16) {

                    bitPos++;

                    row = (int)absN((int)(add \* bitPos + y) - beforeRow);

                }

                LCD\_write\_cmd(0x80 + beforeRow);  // 设置行位置

                LCD\_write\_cmd(0x80 + i);  // 设置列位置

                if (bitPos <= 8) {

                    for (n = k; n < bitPos; n++) {

                        byte1 |= (1 << (7 - n));

                    }

                    LCD\_write\_data(byte1);  // 显示图形数据

                    delay\_10us(10);

                } else if (k >= 8) {

                    for (n = k; n < bitPos; n++) {

                        byte2 |= (1 << (7 - n % 8));

                    }

                    LCD\_write\_data(0x00);  // 清除前半部分

                    LCD\_write\_data(byte2);  // 显示图形数据

                    delay\_10us(10);

                } else {

                    for (n = k; n < 8; n++) {

                        byte1 |= (1 << (7 - n));

                    }

                    for (n = 8; n < bitPos; n++) {

                        byte2 |= (1 << (7 - n % 8));

                    }

                    LCD\_write\_data(byte1);

                    LCD\_write\_data(byte2);

                    delay\_10us(10);

                }

                byte1 = 0;

                byte2 = 0;

                k = bitPos;

            } else {

                for (m = 1; m <= row; m++) {

                    if (add > 0 && beforeRow + m <= 30) {

                        LCD\_write\_cmd(0x80 + beforeRow + m);

                        LCD\_write\_cmd(0x80 + i);

                        if (k >= 8) LCD\_write\_data(0x00);

                        LCD\_write\_data(1 << (7 - k % 8));

                        delay\_10us(10);

                    } else if (add < 0 && beforeRow - m > 0) {

                        LCD\_write\_cmd(0x80 + beforeRow - m);

                        LCD\_write\_cmd(0x80 + i);

                        if (k >= 8) LCD\_write\_data(0x00);

                        LCD\_write\_data(1 << (7 - k % 8));

                        delay\_10us(10);

                    }

                }

                beforeRow = (add > 0) ? beforeRow + row : beforeRow - row;

                k++;

            }

        }

    }

    j = y;

}

循环解析：i 从 7 开始递减，表示绘制数据点的索引。graph[] 数组中的值代表温度，计算出对应的 y 坐标。

计算增量：根据前一个数据点的 y 值和当前数据点的 y 值之间的差距（add），通过这个增量来平滑折线的绘制，避免线条出现过于陡峭的跳跃。

绘制逻辑：

计算每个点的位置后，判断是否需要调整图形的 byte1 和 byte2，这两个字节控制 LCD 屏幕上具体的像素点。

bitPos 控制每一行需要绘制多少个像素点，byte1 和 byte2 对应于一行中的 8 个像素，依次填充。

如果当前的 y 值大于之前的 y 值（上升趋势），则通过调整 beforeRow 来逐步绘制折线图上的上升部分；如果 y 值较小（下降趋势），则相反，逐步绘制下降部分。

通过判断 row 的值来确定需要填充的像素行数，从而平滑地绘制折线图。

4. 恢复默认模式

LCD\_write\_cmd(0x36);  // 结束图形模式

LCD\_write\_cmd(0x30);  // 恢复默认模式

EA = 1;  // 启用中断

LCD\_write\_cmd(0x36)：发送命令结束图形模式。

LCD\_write\_cmd(0x30)：恢复默认的文本模式。

EA = 1;：重新启用全局中断，恢复系统的中断处理。

1. 分析和总结 按照总体任务和个人目标，对整个设计工作过程进行归纳和综合，对设计和 实现过程中所存在的问题和不足进行分析和总结，提出解决的路线建议。总结这 次设计取得的经验和教训，说明个人和团体的收获

|  |  |
| --- | --- |
| 周鸿哲 | **ST7920 屏幕花瓶和显示卡住的问题** 在使用 ST7920 LCD 屏幕进行温度显示时，我发现如果频繁调整温度变化的频率，屏幕会卡住，导致不再刷新显示。经分析，这可能是因为在频繁更新显示时，LCD 控制信号的时序或总线通信被中断，导致屏幕未能正确刷新。 **建议**：我可以尝试添加适当的延时来确保每次操作后屏幕有足够时间完成更新，避免同时操作过多。可能还需要优化硬件信号（如 E 信号）控制的时序。  **并行模式与延时问题** 在并行模式下操作 LCD 时，我需要添加一定的延时。因为在并行模式下，信号变化的时序较复杂，且 LCD 屏幕的刷新可能需要一定时间来稳定。如果没有适当的延时，可能会导致显示不正常。通过添加适当的延时（如 nop() 或 delay()）来确保每个操作后的信号稳定，并防止屏幕卡住。  nop**() 延时周期问题** 我查询了在 12 MHz 时钟频率下，nop() 的延时周期为一个时钟周期，约为 83.33 ns。  **DS18B20 温度传感器与 LCD 显示的集成** 通过 51 单片机使用 DS18B20 获取温度数据，并通过 LCD 屏幕显示当前温度、设定温度和电机状态。在此过程中，涉及到温度的采集、显示更新、定时器中断等操作。使用适当的延时和中断控制来确保温度数据的准确更新，并确保 LCD 显示的稳定。  **温度转换和电机控制** 根据传感器读取的温度，我控制电机的启停，并根据设定温度与当前温度的差异来判断电机状态。通过一个简单的比较逻辑，我控制了 motorOn 状态（电机是否启用）。  **定时器和中断控制**使用定时器 0 来周期性地更新温度并刷新显示。通过定时器中断来周期性地获取温度，并更新 LCD 显示。需要确保定时器的设置准确，并避免中断冲突或延迟，检查定时器中断的优先级和溢出时间，以确保系统响应及时。  **外部中断与计**需要确认外部中断触发条件和实际应用的需求，确保中断响应时间不会影响系统的整体性能。  **LCD 显示与数据结构** 使用 graph 数组来存储历史温度数据，并通过 LCD\_write\_data() 等函数显示在屏幕上，显示格式通过 display() 函数实现。 |