合肥工业大学大学计算机与信息学院

《单片机课程设计》报告

（2023 -2024 学年第二学期）

课程题目：　电风扇模拟控制系统设计

姓　　名：

学　　号：

班　　级：

指导老师：

时　　间：　　2024 年 3 月 10 日

成　　绩：

目录

1、设计目的

设计一个电风扇，可以通过单片机按键控制不同的模式和不同的挡位，有过热报警的保护功能，并且将相关的信息在液晶显示屏上显示出来，可以通过红外遥控器进行远程控制。

2、设计任务

题目要求：

（1）通过矩阵键盘控制风扇模式和挡位，风扇的模式有三个：“自然风”、“睡眠风”、“常风”，每一个模式下都有“1234”四个挡位。

（2）通过独立按键控制温度的阈值，可以通过按键控制温度阈值的升高和降低。

（3）通过 128B20 测电机的温度，温度值保留两位小数。

（4）通过 LCD12864 对电风扇模式、挡位、温度阈值和电机的温度显示出来。模式等一些文字需要使用 LCD12864 中的中文进行显示。

（5）电机过热保护，当测得的电机温度大于温度阈值，蜂鸣器报警，哔哔的声音，同时电机停运 10 秒钟。、

拓展功能：

（1）通过 AT24C02 对温度阈值进行存储，断电之后设置的温度仍然可以保存。

（2）通过红外遥控器控制电风扇的模式和挡位，指标和矩阵键盘一样。

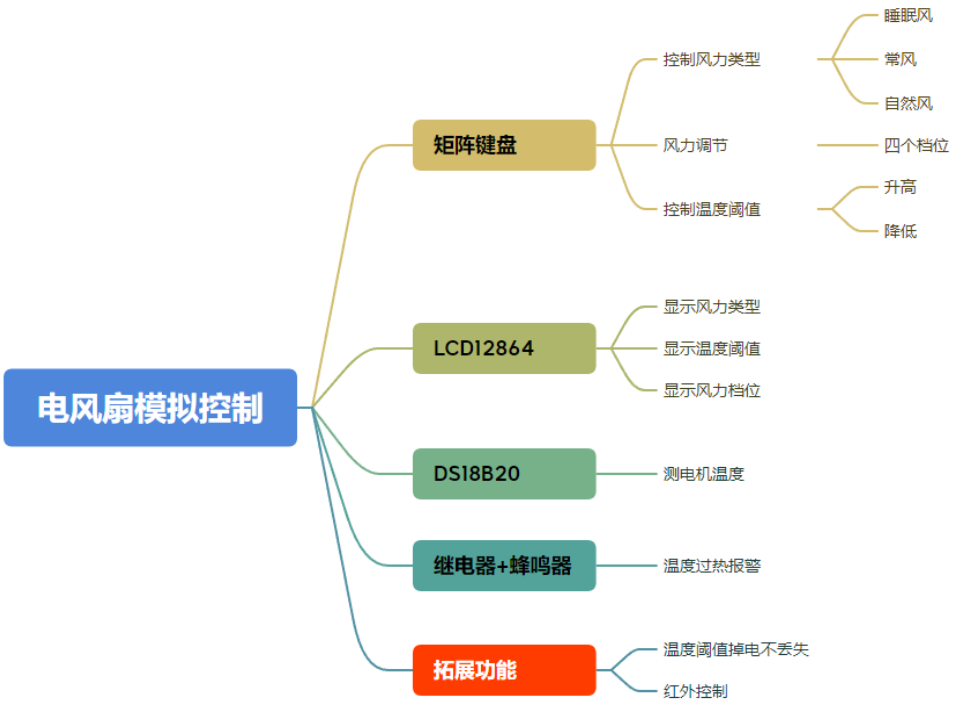


图 2-1 实现功能总结

3、原理框图及说明

根据设计任务，设计如下框图：

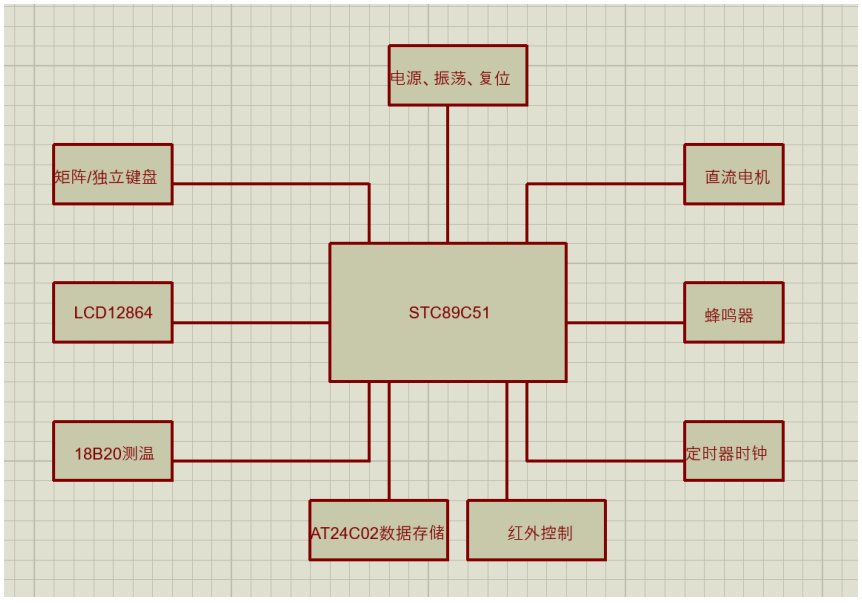


图 3-1 原理框图

电源部分，为单片机提供 5V 稳压电源；矩阵键盘和独立按键给用户提供了设置电风扇模式挡位和设置温度阈值的方式，风扇的模式有三个：“自然风”、

“睡眠风”、“常风”，每一个模式下都有“1234”四个挡位；LCD12864 用来向用户显示当前的模式、挡位、温度阈值和电机的温度；18B20 是测风扇电机温度使用的；AT24C02 对温度阈值进行存储，断电之后设置的温度仍然可以保存；直流电机是风扇转动需要的模块；蜂鸣器是在当风扇过热时报警使用的；定时器时钟是风扇过热保护停转 10 秒计时使用的。

4、主要电路说明、元件选择及参数计算

4.1 复位电路

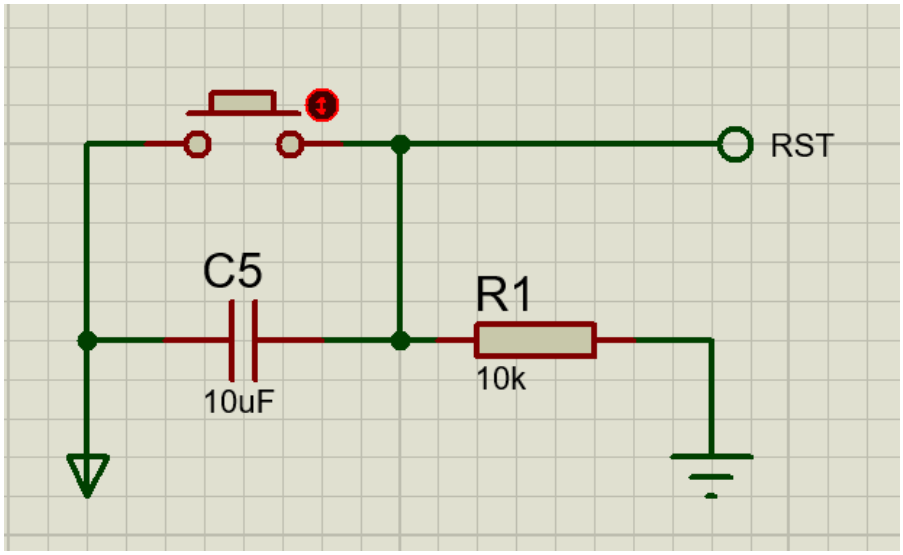


图 4-1 复位与振荡电路

（1）开机复位

在电路图中，电容的的大小是 10uf，电阻的大小是 10k。

在 5V 正常工作的 51 单片机中小于 1.5V 的电压信号为低电平信号，而大于 1.5V 的电压信号为高电平信号。可以算出电容充电到电源电压的 0.7 倍，即电容两端电压为 3.5V、电阻两端电压为 1.5V 时，需要的时间约为 T=RC=10K*10UF=0.1S。

也就是说在单片机上电启动的 0.1S 内，电容两端的电压从 0-3.5V 不断增加，这个时候 10K 电阻两端的电压为从 5-1.5V 不断减少（串联电路各处电压之和为总电压），所以 RST 引脚所接收到的电压是 5V-1.5V 的过程，也就是高电平到低电平的过程。

单片机 RST 引脚是高电平有效，即复位；低电平无效，即单片机正常工作。所以在开机 0.1S 内，单片机系统 RST 引脚接收到了时间为 0.1S 左右的高电平信号，所以实现了自动复位。

（2）按键复位

在单片机启动 0.1S 后，电容 C 两端的电压持续充电为 5V，这是时候 10K 电阻两端的电压接近于 0V，RST 处于低电平所以系统正常工作。当按键按下的时候，开关导通，这个时候电容两端形成了一个回路，电容被短路，所以在按键按下的这个过程中，电容开始释放之前充的电量。随着时间的推移，电容的电压在 0.1S 内，从 5V 释放到变为了 1.5V，甚至更小。根据串联电路电压为各处之和，这个时候 10K 电阻两端的电压为 3.5V，甚至更大，所以 RST 引脚又接收到高电平。单片机系统自动复位。

4.2 矩阵键盘电路

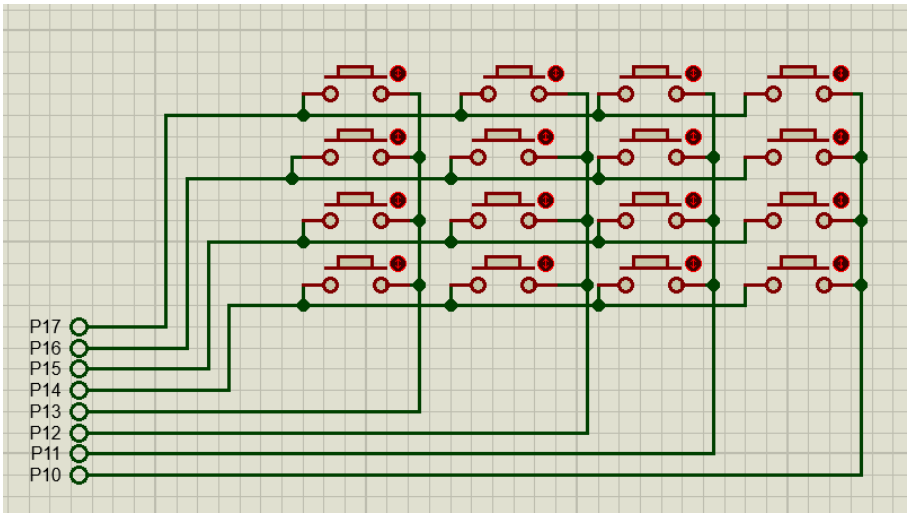


图 4-2 矩阵键盘电路

矩阵按键就是独立按键的结合体，矩阵按键的优点是提高了 I/O 端口的利用率，用 M+N 个 I/O 端口就可以控制 M×N 个独立按键。而一个独立按键就需要一个 I/O 端口来控制，I/O 端口的利用率很低。如上图所示，每列按键引出一条线由一个 I/O 端口控制，每行按键引出一条线由一个 I/O 端口控制。采用行列扫描法控制：以上图矩阵按键电路图，端口高四位是行，低四位是列。

确认行，首先给端口高四位为高电平，低四位为低电平，即 11110000，然

后检测高四位接收到的值是否全为 1。若不全为 1，则证明有键被按下。以按键 S1 为例，当按键 S1 被按下时，P17 端口与 P13 端口通路，此时 P17 端口为低电平。如果检测到 P1 端口数据为 01110000，则第一行有按键被按下，其他行同理。

确认列，与确认行原理相似，给 P1 端口低四位为高电平，高四位为低电平，即 00001111。检测低四位是否全为 1，若低四位为 0111，则第一列有按键被按下。根据确认行和确认列就可以判断是哪个坐标的按键被按下。

4.3 LCD12864 显示电路

LCD12864 模块主要用来显示所要的界面信息或数据，所以要求能显示汉字，字符和数字，而 LCD12864 满足系统要求的显示功能。LCD12864 在显示字母和数字时，是 4*16 的显示字符模块，即可以显示 4 行，每行可以显示 16 个字母或数字；在显示汉字时，是 4*8 的汉字显示模块，即可以显示 4 行，每行可以显示 8 个汉字。下面进行介绍的是并行通信的显示方式。

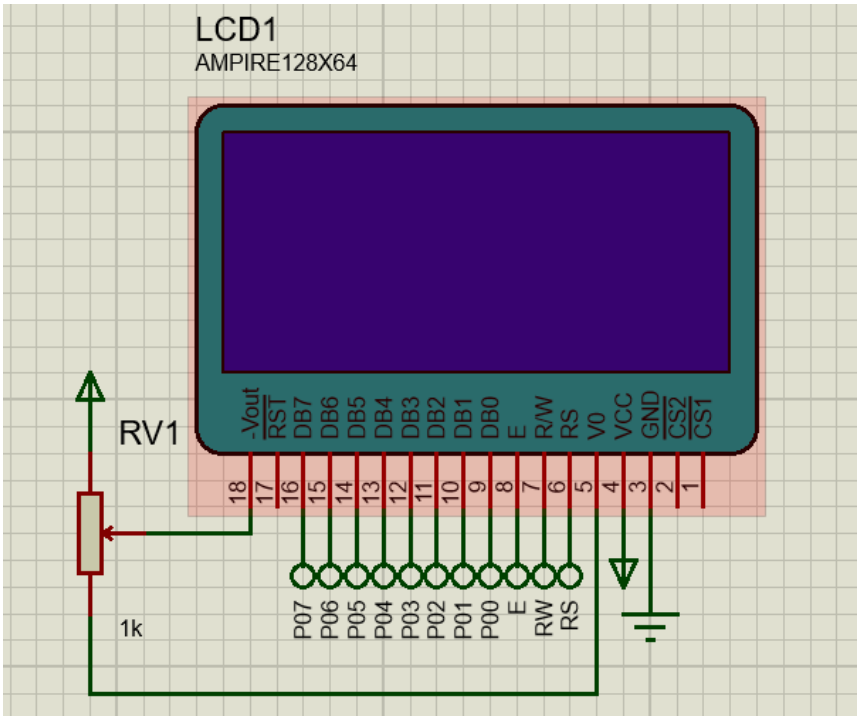


图 4-3 LCD12864 电路

	X 坐标							
Line1	80H	81H	82H	83H	84H	85H	86H	87H
Line2	90H	91H	92H	93H	94H	95H	96H	97H
Line3	88H	89H	8AH	8BH	8CH	8DH	8EH	8FH
Line4	98H	99H	9AH	9BH	9CH	9DH	9EH	9FH

图 4-4 LCD12864 坐标

按照电路原理图跟单片机最小系统进行连线，如图所示。LCD12864 共有 20 个引脚，其引脚具体功能如表 1 所示，由表可得 LCD12864 引脚组成为 8 位数据传输端口（DB0-DB7）；两个电源引脚（VCC，GND）；两个电源背光引脚（BLK，BLA），控制 LCD 的背景亮度；一个 VO 引脚，外接一个上拉电阻（控制 LCD12864 的字符对比度，让字符更加的清晰可见）；RST 复位引脚，低电平有效，此处直接接高电平；第 16、17 位空引脚，不用管；剩下的 RS，RW，EN 和 PSB 四个引脚则跟 LCD12864 的写入息息相关，通过 PSB 可以控制 LCD12864 跟单片机的通信方式，输入高电平，则 LCD12864 跟单片机的通信模式为并行通信，低电平则为串行通信。单片机对 RS，RW，EN 端口的写入控制，则可以控制 LCD12864 的数据传输模式，决定单片机写入 LCD12864 数据端口 DB 的是命令还是数据。写入命令可以控制 LCD 的模式和工作状态，然后写入数据，让 LCD 显示需要的界面。

4.4 电机驱动电路

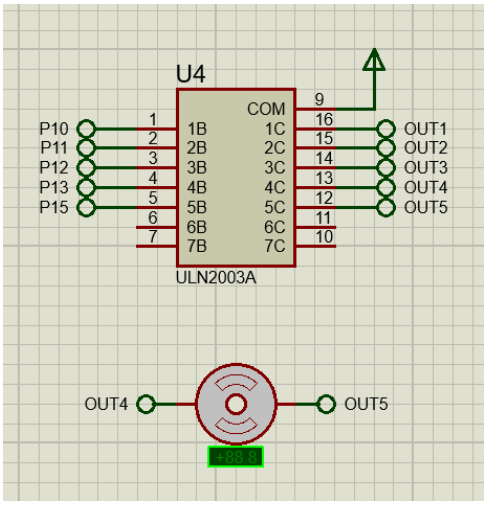


图 4-5 直流电机驱动

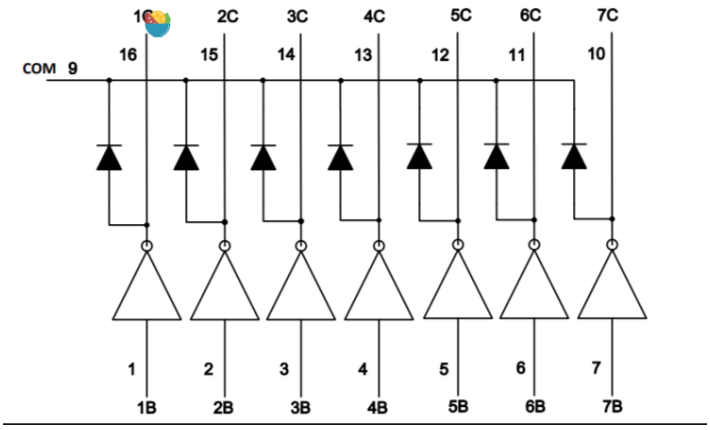


图 4-6 ULN2002D 芯片逻辑图

ULN2002D 是一种高压高电流达林顿晶体管阵列，每个阵列包含七个开路集电极达林顿对，共用发射极。每个通道的额定电流为 500 毫安，峰值电流可达 600 毫安。该芯片内置了抑制二极管，用于驱动感性负载。ULN2002D 可以用来驱动直流电机。选择将电机的一端接到 out4 另一端接到 out5,通过程序控制 ULN2002D 芯片的 IO 口变化高低电平，实现 PWM 控制电机转速。

4.5 测温电路

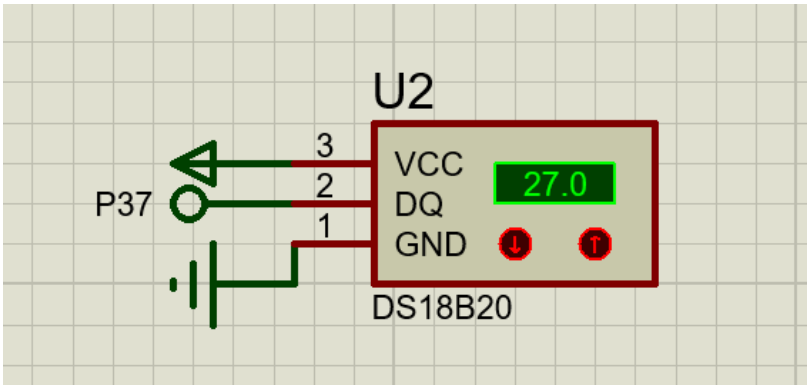


图 4-7 测温电路

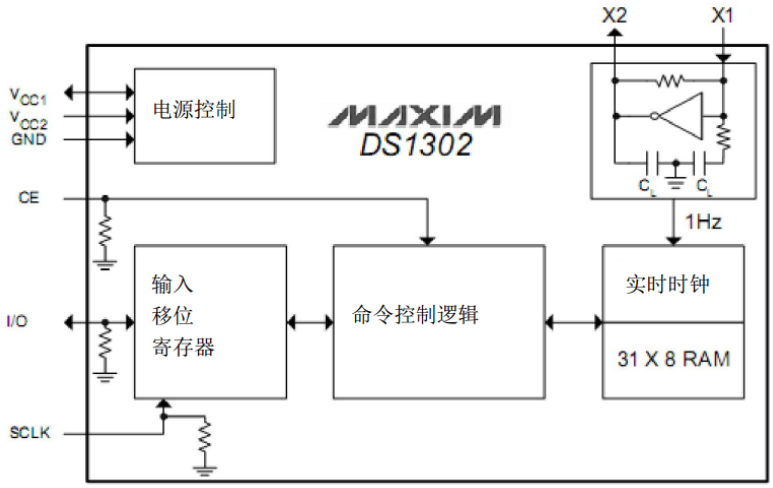


图 4-8 DS18B20 内部原理图

DS18B20 是一种单总线数字温度传感器，可以直接读取被测物体的温度，测量范围为-55～+125℃，分辨率为 9～12 位 1。DS18B20 可以采用外部供电或寄生电源的方式工作。管脚 1 为 GND，管脚 2 为数据 DQ，管脚 3 为 VDD。如果把传感器插反，那么电源将短路，传感器就会发烫，很容易损坏，所以一定要注意传感器方向。

DS18B20 的元件选择和参数计算主要涉及到上拉电阻 R1 和去耦电容 C1。R1 的大小取决于数据线的长度、负载数量和供电方式。一般情况下，R1 可以选用 4.7kΩ～10kΩ之间的值 3。C1 的大小取决于数据线和地线之间的寄生电容以及供电稳定性。一般情况下，C1 可以选用 0.01μF～0.47μF 之间的值 3。

4.6 蜂鸣器电路

P1.0 是单片机的输出口，BZ1 是蜂鸣器，R1 和 R2 是限流电阻，其阻值和功率需要根据蜂鸣器和单片机的特性选取。

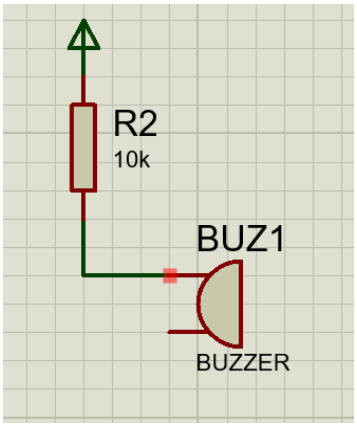


图 4-9 蜂鸣器电路

元件选择和参数计算

(1) 蜂鸣器

蜂鸣器一般分为有源蜂鸣器和无源蜂鸣器两种类型。有源蜂鸣器是一种带有振荡电路的声音发生器，只需接入电源即可发声。无源蜂鸣器则需要外接振荡电路才能发声。本电路采用有源蜂鸣器。根据需要产生的声音频率和音量大小，可以选择不同类型的有源蜂鸣器。一般而言，有源蜂鸣器的工作电压为 5V 左右，工作电流一般在 20mA 以下，音量大小与工作电流和驱动频率有关。具体的选型需要根据具体的应用场景进行选择。

(2) 限流电阻

限流电阻的主要作用是限制电流大小，保护单片机输出口和蜂鸣器不受损坏。限流电阻的阻值需要根据单片机的驱动能力、蜂鸣器的工作电流和驱动电压等参数来确定。通常可以选择 100 欧姆至 1 千欧姆的电阻。

(3) 单片机输出口

单片机的输出口需要具备驱动蜂鸣器的能力，通常情况下单片机的 IO 输出电流为几毫安

4.7 AT24C02 断电存储电路

AT24C02 是一种常用的 I2C 串行 EEPROM 芯片，适用于储存小量数据。下面是一种基于 51 单片机控制 AT24C02 的电路设计和元件选择：其中，AT24C02 是 I2C 串行 EEPROM 芯片，SCL 和 SDA 是 I2C 总线的时钟线和数据线，R1 是 I2C 总线上的上拉电阻，其阻值需要根据具体的应用场景来选择。

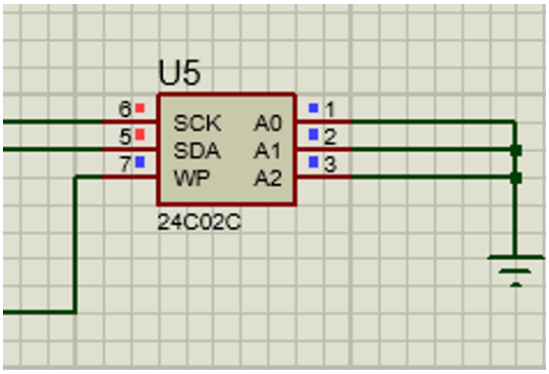


图 4-10 电源电路

元件选择和参数计算

(1) AT24C02

AT24C02 是一种常用的 I2C 串行 EEPROM 芯片，容量为 2Kbit，采用 8 位地址位和 8 位数据位。具体的选型需要根据需要存储的数据量和工作电压等参数来选择。

(2) 上拉电阻

上拉电阻的作用是将 I2C 总线上的 SCL 和 SDA 线拉高至高电平，以便于单片机和 AT24C02 之间进行通信。上拉电阻的阻值需要根据具体的 I2C 设备来选择。一般情况下，I2C 设备的最大上拉电阻一般在 4.7kΩ左右。

(3) 时钟频率

I2C 总线的时钟频率需要在 50kHz 到 400kHz 之间，具体的选取需要根据 I2C 设备的要求来确定。在 AT24C02 的数据手册中，建议使用时钟频率在 100kHz 左右。

(4) 电源电压

AT24C02 的工作电源电压范围为 1.8V 到 5.5V，需要根据具体的应用场景来选择。

4.8 红外控制电路

51 单片机红外控制电路是一种常见的控制电路，可以通过红外遥控器实现对电器的控制。电路采用红外接收头接收红外信号，并通过 51 单片机对红外信号进行解码和控制。51 单片机红外控制电路需要一个红外接收头和一些上拉电阻和限流电阻即可实现对电器的控制。

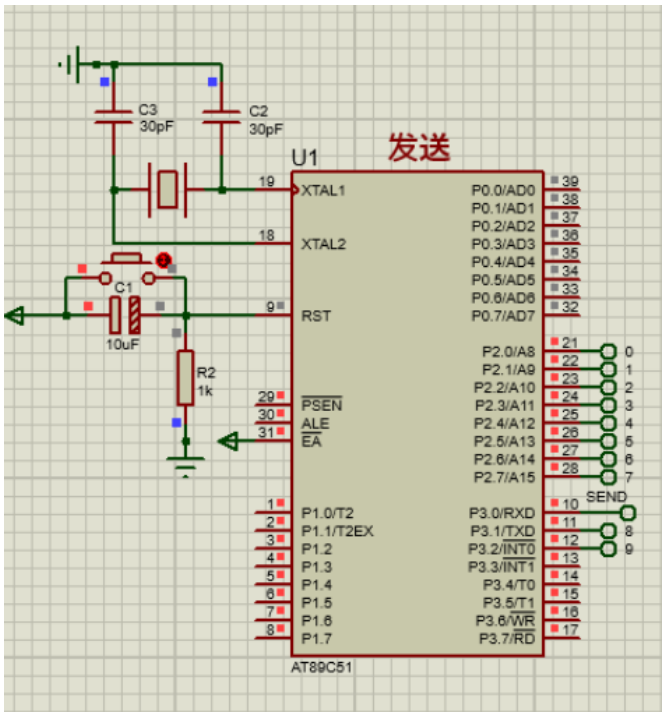


图 4-11 红外控制电路

元件选择和参数计算

(1) IR LED

红外发射管是通过发射红外信号来实现对电器的控制。常见的红外发射管有红外光电二极管等型号，需要根据具体的应用场景和要控制的设备来选择。

(2) 上拉电阻

红外接收头的上拉电阻的作用是将红外接收头的输出信号拉高至高电平。一般情况下，红外接收头的上拉电阻的阻值为 10kΩ左右。

(3) 限流电阻

限流电阻的作用是限制红外接收头的输出电流，以保护 51 单片机不被过电流烧毁。一般情况下，限流电阻的阻值在 100Ω左右。

(4) 程序设计

除了硬件电路设计外，还需要编写相应的程序代码来实现对红外信号的解码和控制。具体的程序代码可以根据具体的应用场景和要控制的设备来进行编写。

5、总电路设计

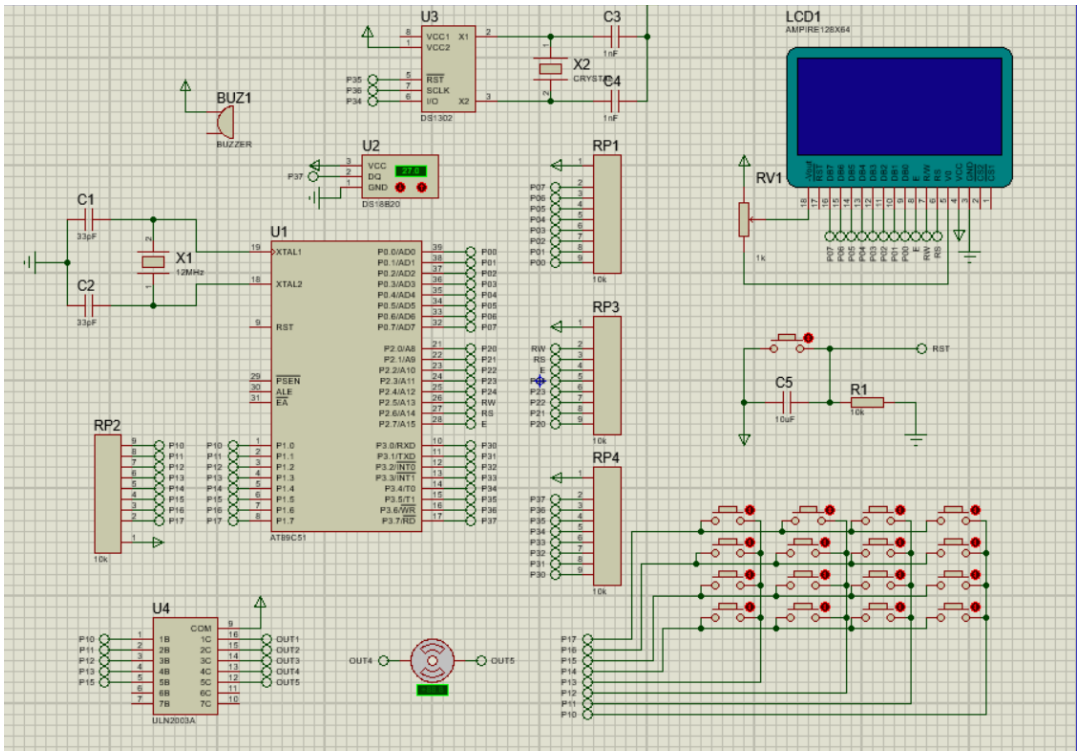


图 5-1 总电路连接图

整个系统的核心是一块 AT89S52 单片机，电路实现的功能所需要的模块包括 LDC12864，矩阵键盘，蜂鸣器，电机驱动及直流电机，红外控制模块。LDC12864 有 18 个引脚，包括电源供电，命令控制和数据三种，电源由 5V 供电。控制 LCD12864 命令的引脚 LCD_EN，LCD_RS，LCD_RW 使用单片机的 P27，P26，P25 口控制，LCD12864 的数据口由单片机的 P0 的八位控制。矩阵键盘需要八位进行控制行和列的选择，采用单片机的 P11~P13 控制列的数据响应，采用 P14~P17 控制行的数据响应。蜂鸣器需要单片机提供 5V 的电源进行驱动，IO 口连接了 P23。红外控制电路需要 5V 电源供电，使用的 P32 接受数据。直流电机驱动模块使用 P3 口控制直流电机的旋转和停止。

6、软件设计思路及框图

6.1 软件设计思路及主流程图

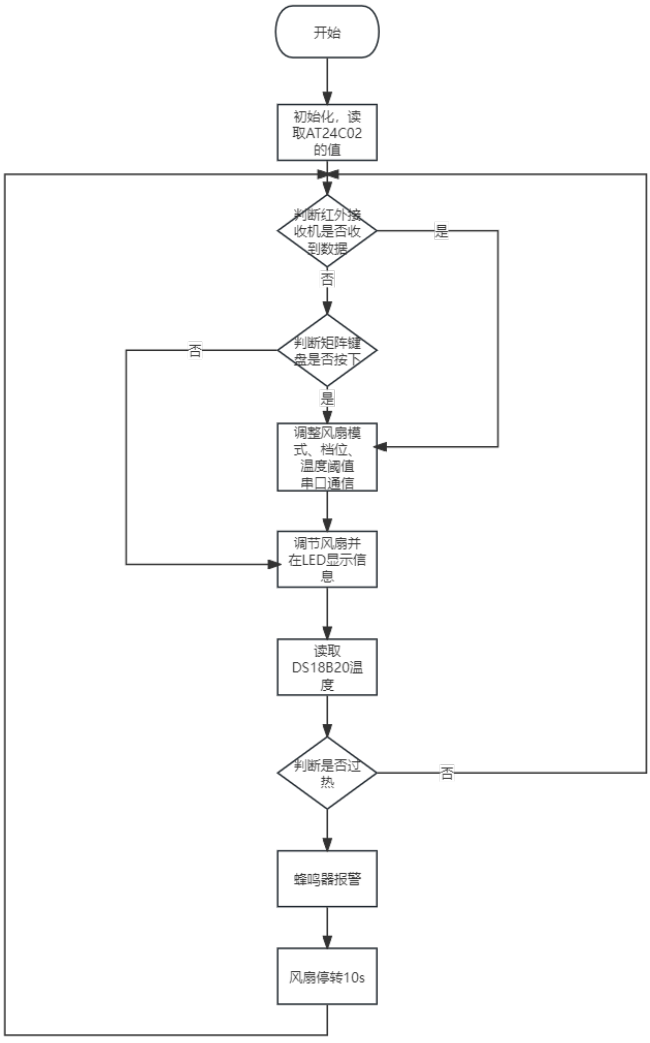


图 6-1 主程序框图

根据设计要求，本单片机课程设计旨在设计一个可控制电风扇的系统。通过矩阵键盘可以选择风扇的模式和挡位，包括自然风、睡眠风和常风三种模式，每种模式下有四个挡位可供选择。独立按键可以控制温度阈值的升高和降低，并通过 AT24C02 进行存储，保证设置的温度阈值可以在断电后继续使用。通过 128B20 测量电机的温度，温度值保留两位小数。通过 LCD12864 可以显示电风扇的模式、挡位、温度阈值和电机的温度，包括中文文字的显示。在电机过热时，系统会自动保护，蜂鸣器会发出哔哔的声音，同时电机停运 10 秒钟。此外，本系统还可

以通过红外遥控器控制电风扇的模式和挡位，功能与矩阵键盘相同。

设计全局变量：

(1)fantype 点风扇的模式，包括自然风、睡眠风和常风三种模式(2)Speed 设计的 PWM 的具体值，调节电风扇的直流电机速度 (3)type 设置 PWM 的起始值，可以控制占空比调节电机速度(4)gear 表示电风扇当前的速度挡位，可以和 type 共同组成 Speed 的值，计算得到当前风扇的 PWM 占空比的具体值

(5)Command 红外接收器接受到的数据信息，根据不同的按键响应不同的命令

(6)KeyNum 矩阵键盘的响应值，可以控制风扇的挡位，模式和调节温度阈值

(7)Thigh 设置的高温报警阈值温度(8)T 代表当前 DS18B20 获得的温度值

设计函数：

（1）主函数 main()

入口：无，出口：无。 一个大的循环，程序用不结束。

（2）void lcd12864_init();

功能：初始化 LCD12864

入口：无

出口：无

（3）void lcd12864_show_string(u8 x,u8 y,u8 *str);

功能：在特定（x,y）的位置显示 str 值

入口：横坐标 x,纵坐标 y,需要显示的内容 str

出口：无

（4）void lcd12864_clear(void);

功能：清空 LCD12864 屏幕上的所有东西

入口：无

出口：无

（5）IR_Init();

功能：红外控制功能初始化

入口：无

出口：无

（6）IR_GetDataFlag();

功能：红外遥控获取收到数据帧标志位

入口：无

出口：无

（7）IR_GetCommand();

功能：红外遥控获取收到的命令数据

入口：无

出口：无

（8）void Motor_SetSpeed(unsigned char Speed);

功能：电机设置 PWM 的速度值，控制电机旋转的速度

入口：设置的速度值，范围是[0,100]

出口：无

（9）void Motor_Init(void);

功能：电机控制初始化

入口：无

出口：无

（10）void Motor_Delay();

功能：设置在电机高温时停转 10 秒

入口：无

出口：无

（11）unsigned char MatrixKey();

功能：获得矩阵键盘的按键参数

入口：无

出口：返回矩阵键盘的按键值

（12）void Delay(unsigned int xms);

功能：延迟 x 毫秒的时间

入口：无

出口：无

（13）void AT24C02_WriteByte(unsigned char WordAddress,Data);

功能：向 AT24C02 芯片写入数据

入口：写入的数据地址和数据值

出口：无

（14）unsigned char AT24C02_ReadByte(unsigned char WordAddress);

功能：从 AT24C02 芯片读取数据

入口：数据地址

出口：数据值

（15）void DS18B20_ConvertT(void);

功能：DS18B20 开始温度变换

入口：无

出口：无

（16）float DS18B20_ReadT(void);

功能：DS18B20 读取温度

入口：无

出口：返回读取到的温度值

6.2 其他模块软件设计及流程图

6.2.1 矩阵键盘设计及流程图

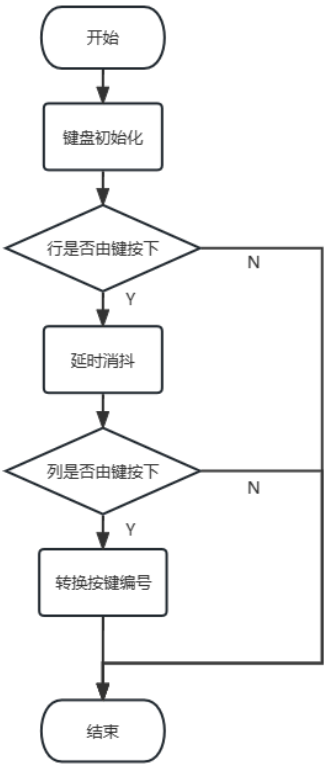


图 6-2 矩阵键盘设计及流程图

单片机矩阵键盘设计流程包括以下步骤：首先初始化所需的 I/O 口，将其设置为输入口并上拉电阻；然后通过扫描矩阵键盘的每一行和每一列的状态，判断是否有按键按下；接着进行去抖动处理，防止误触发，并等待按键松手；然后返回按键的键值，用于后续处理；最后重复执行上述流程，以实现连续扫描和处理矩阵键盘的按键状态。具体的编程实现可以使用循环语句和延时函数等方式来实现。

6.2.2 LCD12864 设计及流程图

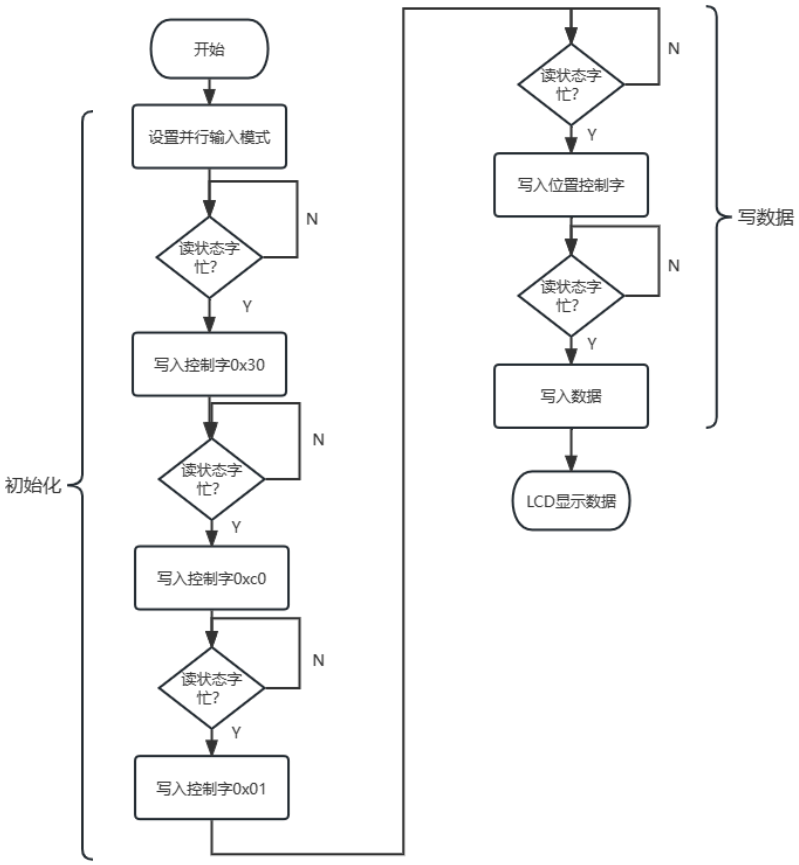


图 6-3 LCD12864 设计及流程图

设计 LCD12864 的流程包括以下步骤：首先确定所使用的 LCD12864 类型和控制方式，然后进行初始化操作，将需要显示的数据显示到指定的位置。初始化操作是根据 LCD12864 数据手册设置初始化参数，第一步设置选择并行输入；第二步检测 LCD 是否忙，如果不忙写入命令字 0x30，选择基本指令操作；第三步检测 LCD 是否忙，如果不忙写入命令字 0xc0，显示开，关光标；第四步检测 LCD 是否忙，如果不忙写入命令字 0x01，清屏，LCD12864 初始化完成。接着就可以调用相应的显示函数，写入命令字设置数据端口位置，写入数据字，用于显示所需的文字内容。根据需要更新显示内容，例如定时更新温度、电机状态等。在程序结束时，关闭 LCD12864，释放所占用的 I/O 口。

6.2.3 直流电机设计及流程图

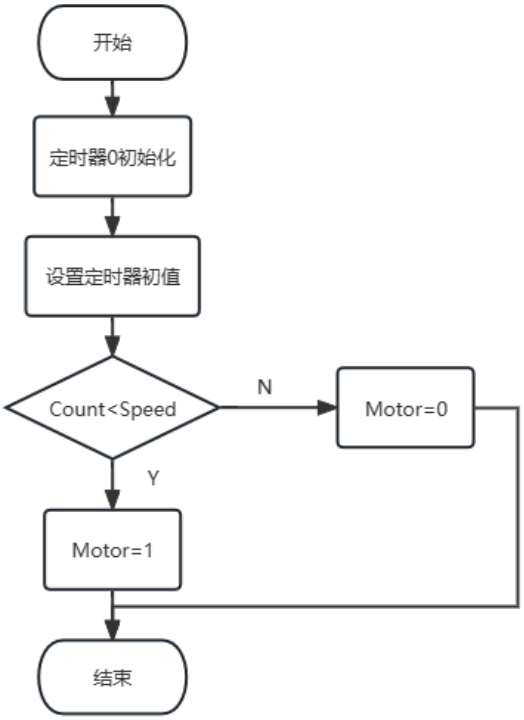


图 6-4 PWM 设计流程图

设计控制直流电机的控制是通过 PWM 进行控制的：PWM 通过设置每一个周期的占空比实现对直流电机速度的控制。采用定时器 1 的中断进行过控制占空比，首先对定时器 1 进行初始化，设置定时器的模式 8 位，设置定时器的初值（TL1=0x9C;TH1=0xFF），每间隔 100us 执行一次，在中断中设置计数器参数 Counter，没执行一次中断计数值递增，同时限定计数的范围是 0-100。在定时器计数达到一定阈值（通过电机挡位控制设置）时，调整 PWM 的占空比，从而实现了 PWM 控制电机速度。

6.2.4 DS18B20 测温设计及流程图

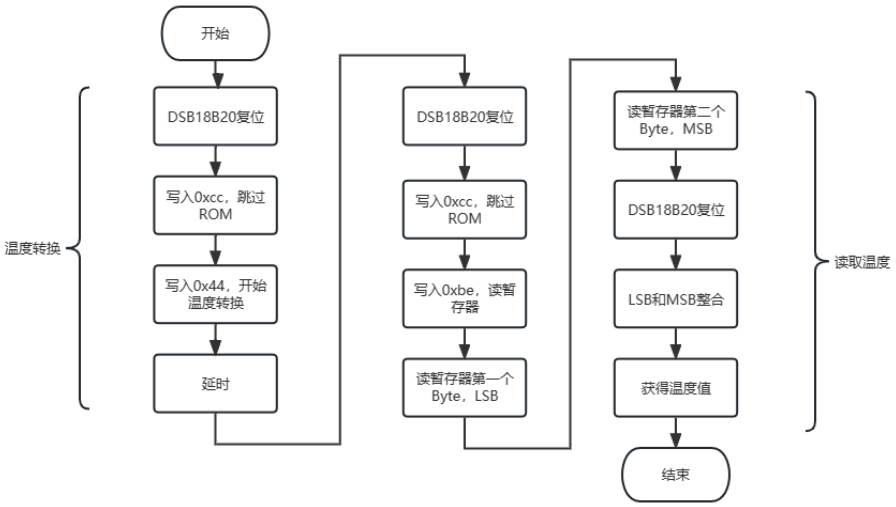


图 6-5 DS18B20 测温流程图

首先需要复位 DS18B20，然后发送跳过 ROM 指令以跳过设备地址，接着发送开始温度转换的指令。等待 700~900 毫秒，等待温度转换完成。然后再次复位 DS18B20，发送跳过 ROM 指令以跳过设备地址，然后发送读取高速暂存器的指令。接着读取暂存器的第 0 字节和第 1 字节，分别为温度数据的 LSB 和 MSB。然后将 LSB 和 MSB 整合成一个 16 位数据。最后，判断读取结果的符号，进行正负温度的数据处理。需要注意的是，DS18B20 的温度转换时间取决于设定的分辨率，不同的分辨率会有不同的转换时间。而在读取温度值时，需要先判断读取结果的符号，如果为负数则需要进行符号扩展处理。

6.2.5 蜂鸣器设计及流程图

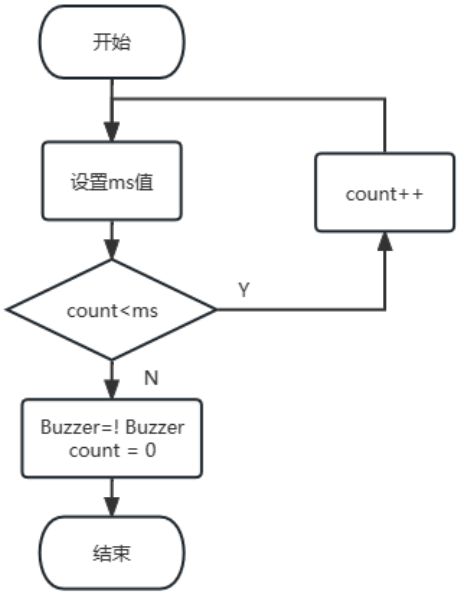
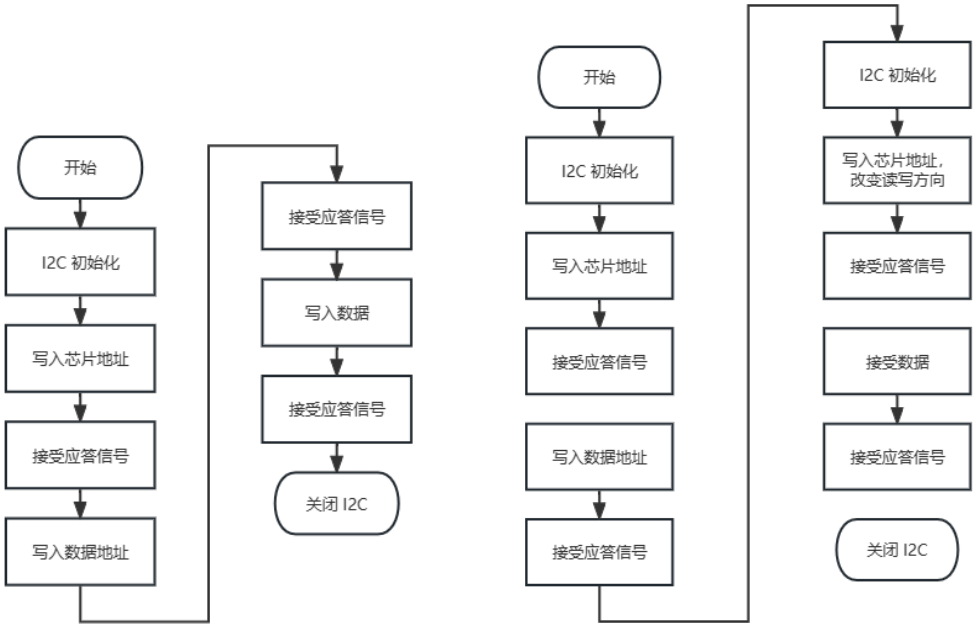


图 6-6 蜂鸣器流程图

51 单片机上使用的是无源蜂鸣器，通过用 2K~5K 的方波去驱动它发声，实验流程是将控制蜂鸣器的 IO 口输出电平变化，控制蜂鸣器响铃。执行蜂鸣器响铃的函数，其中通过延时函产生一定频率的方波，延时函数延时一定的时间，然后控制蜂鸣器的 IO 口取反，达到一定的频率驱动蜂鸣器的目的。

6.2.6 AT24C02 断电存储设计及流程图



(a) (b)

图 6-7 AT24C02 芯片写入(a)和读取(b)执行流程

在 51 单片机与 AT24C02 进行 I2C 通信时，需要按照以下流程进行操作：首先进行 I2C 总线的初始化，然后发送起始信号，接着发送设备地址和读写位以及要读写的内存单元地址。如果进行的是写操作，则发送要写入的数据；如果进行的是读操作，则等待 AT24C02 发送数据。最后发送停止信号。在进行 I2C 通信时，需要严格按照 I2C 协议的规范进行操作，以确保数据的正确读写。此外，由于 AT24C02 的读写速度比较慢，还需要在读写数据时添加适当的延时，以确保数据的正确性和可靠性。

6.2.7 红外控制设计及流程图

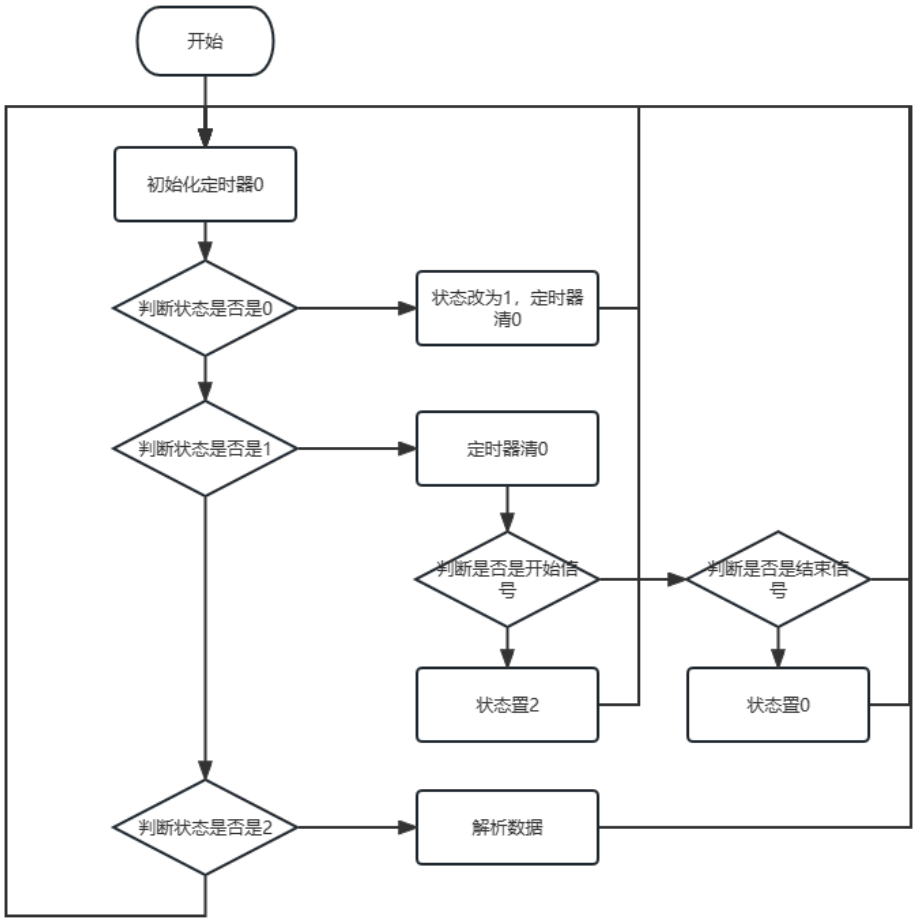


图 6-8 红外控制电路流程图

采用计时器中断加状态机的方式进行红外控制电路的执行逻辑，有三个状态，判断是状态 0 的时候执行将计数器置零，改变状态为 1，当状态是 1 的时候获得计数器的计数器，判断计数值是开始标志，如果是开始标志，设置状态为 2，同时计数器置零。如果不是开始标志，是结束标志，设置状态为 0，同时计数器置零。当状态是 2 的时候，获得计数器的计数值，判断计数值是二进制的 0 还是 1，当接受的数据达到了 32 位数据，结束本次数据的接受，将状态置 0，计数器置零。通过循环执行上面中断中的状态机，就可实现解析红外发射器发送来的数据，判断红外遥控器的按键。

7、结果及测试分析

7.1 硬件电路

含硬件电路实物图及说明：

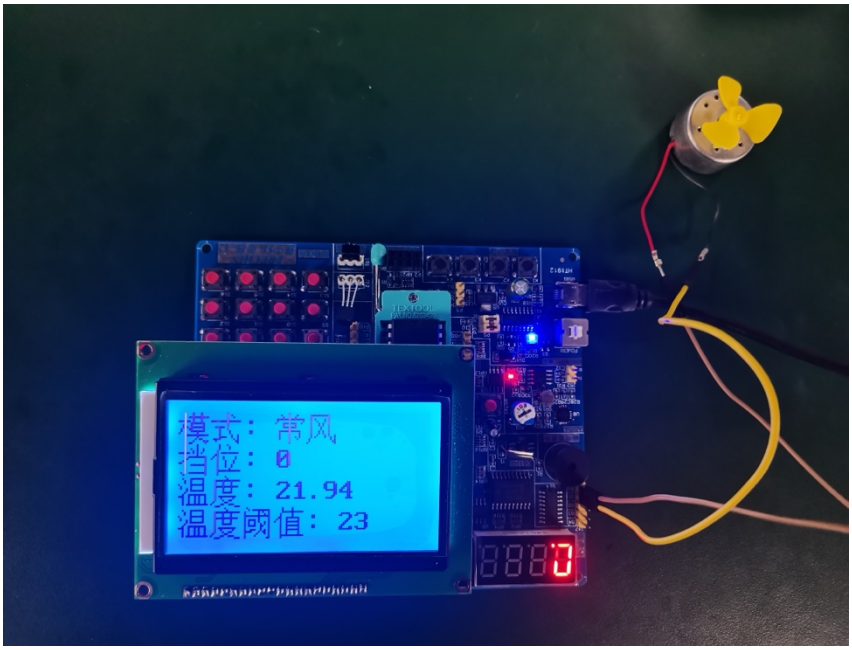


图 7-1 硬件电路实物图

通过矩阵键盘控制调节风扇模式“自然风”、“睡眠风”、“常风”；控制每一个模式下都有“1234”四个挡位；控制调节温度阈值。检测温度并且保留两位小数，显示在 LDC12864 上。当温度超过时蜂鸣器报警，并且风扇停转十秒。掉电重启，温度阈值保存不会丢失。通过红外遥控器控制电风扇的模式和挡位。

7.2 测试记录及结果分析

测试功能：

制作完成后，进行如下测试，结果合格。（如果有测试数据对数据进行分析。）

功能要求	功能描述	测试结果
（1）通过矩阵键盘控制风扇模式和挡位，风扇的模式有三个：“自然风”、“睡眠风”、“常风”，每一个模式下都有“1234”四个挡位。	S5，S6 ，S7 控制三个模式 S1，S2 ，S3，S4 控制挡位	S5 按下显示自然风 S6 按下显示睡眠风 S7 按下显示睡眠风 S1 按下显示 1 档 S2 按下显示 2 档 S3 按下显示 3 档

		S4 按下显示 4 档
（2）通过独立按键控制温度的阈值，可以通过按键控制温度阈值的升高和降低。	S9，S10 控制调节温度阈值	S9 按下温度阈值提高 S10 温度阈值降低
（3）通过 AT24C02 对温度阈值进行存储，断电之后设置的温度仍然可以保存。	关闭电源之后重新开启电源，上次的温度阈值没有变化，断电保存数据成果	断电之后重新上电，温度阈值没有变化。
（4）通过 128B20 测电机的温度，温度值保留两位小数。	LCD12864 显示温度，并且时两位小数	通过
（5）通过 LCD12864 对电风扇模式、挡位、温度阈值和电机的温度显示出来。模式等一些文字需要使用 LCD12864 中的中文进行显示。	LCD12864 屏幕显示中文的模式，挡位，温度阈值，温度等汉字	通过
（6）电机过热保护，当测得的电机温度大于温度阈值，蜂鸣器报警，哔哔的声音，同时电机停运 10 秒钟。	温度高于温度阈值时，蜂鸣器报警并且电机停转 10 秒	通过
（7）通过红外遥控器控制电风扇的模式和挡位，指标和矩阵键盘一样。	红外遥控器控制风扇	通过

测试图片：

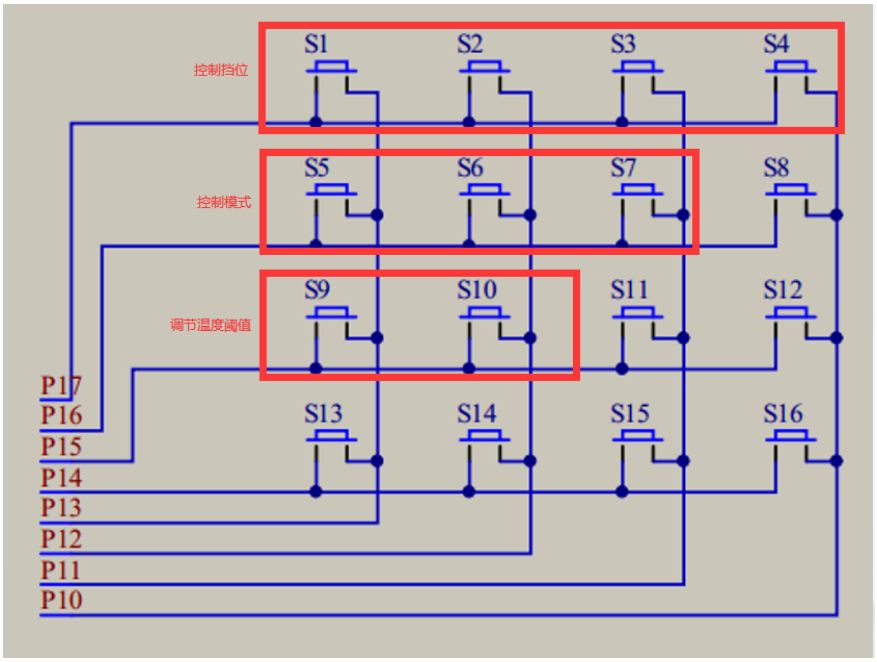


图 7-2 矩阵键盘控制介绍

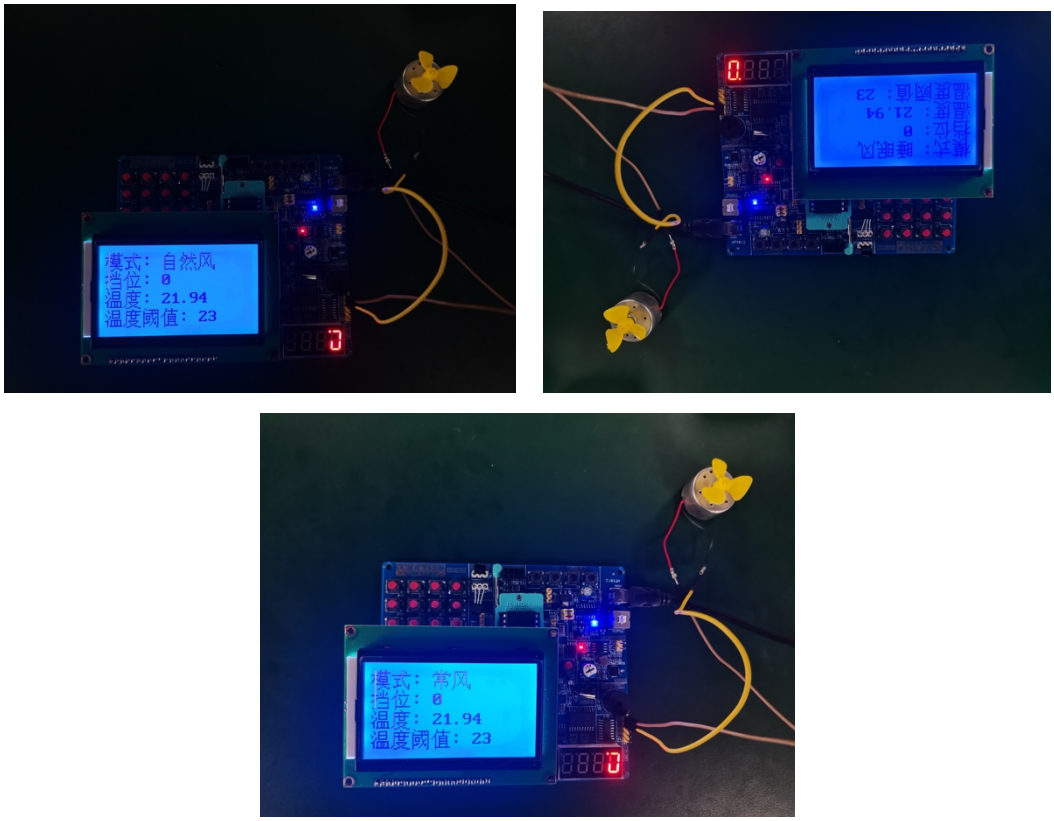


图 7-3 控制设置三种模式

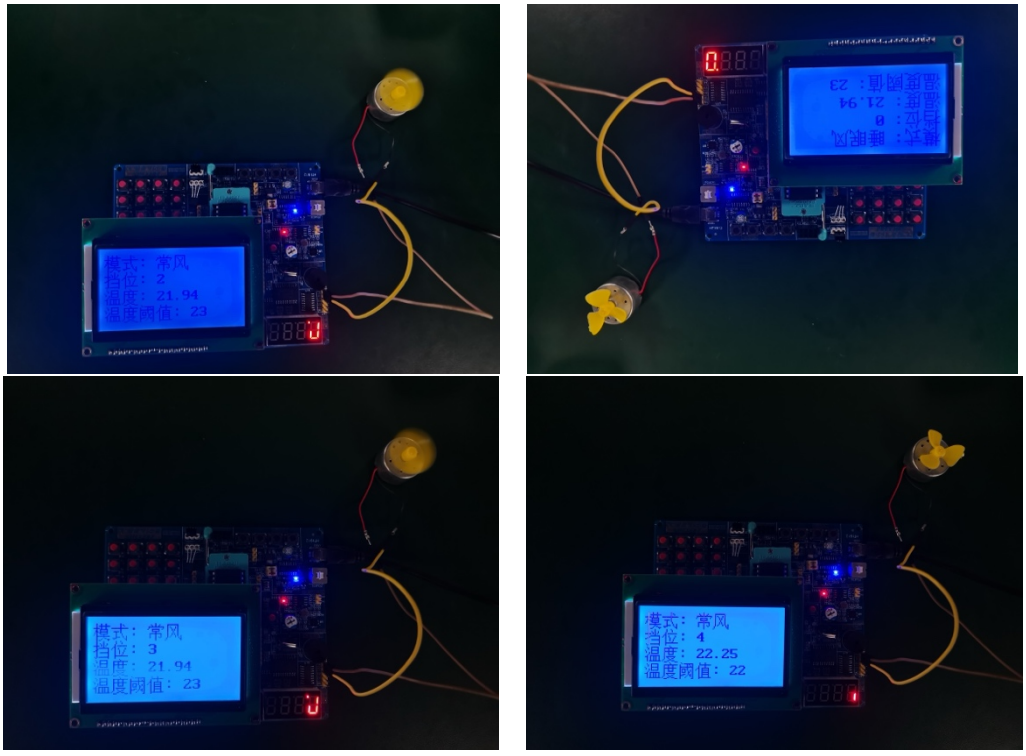


图 7-4 控制风扇的挡位

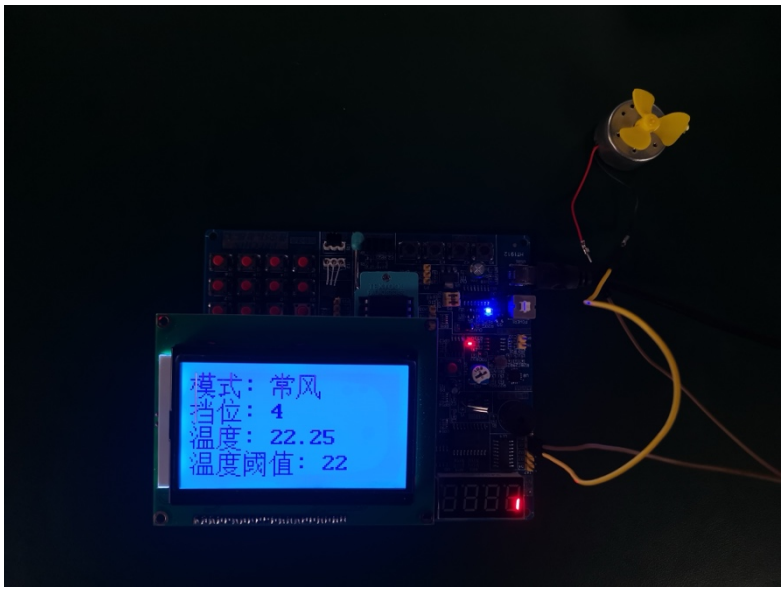


图 7-5 控制调节风扇的阈值

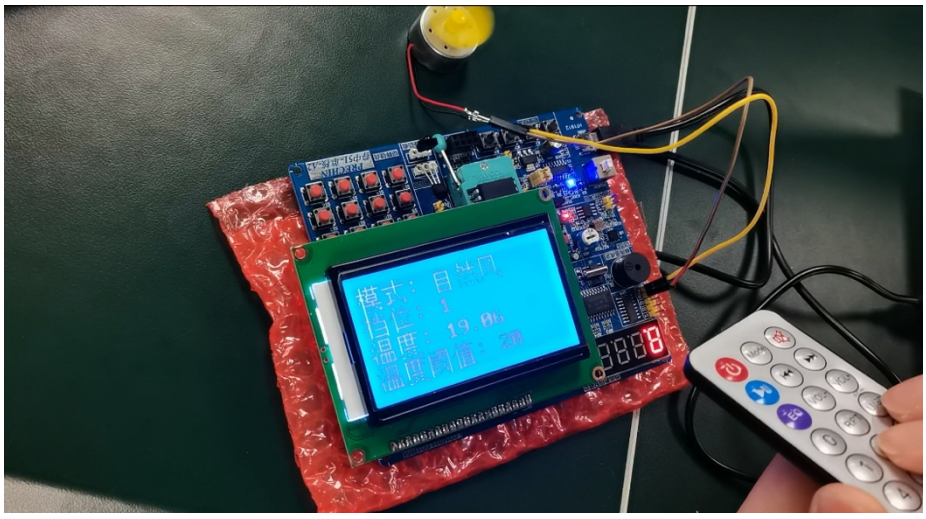


图 7-6 红外控制

8、体会与感想

在本次单片机课程设计中，我使用 51 单片机实现了模拟电风扇控制系统，该模拟系统有三种风力类型，分别为常风、自然风、睡眠风，每种风力类型都有四个档位，可以通过矩阵键盘调节风力类型与档位，此外，该模拟系统可以实现过热保护，当温度超过设置的温度阈值，蜂鸣器报警，并且风扇停转 10s。除了这些基本功能外，我还利用 AT24C02 进行温度阈值的存储，使其掉电不丢失，并且利用红外遥控实现了上述所有功能。

在整个设计和实现过程中，我深入学习并应用了多个组件，包括但不限于矩阵键盘、AT24C02、128B20 和 LCD12864。这些组件的集成不仅是对我之前所学编程技术和电路原理知识的实践，也极大地拓展了我的技术视野和解决问题的能力。

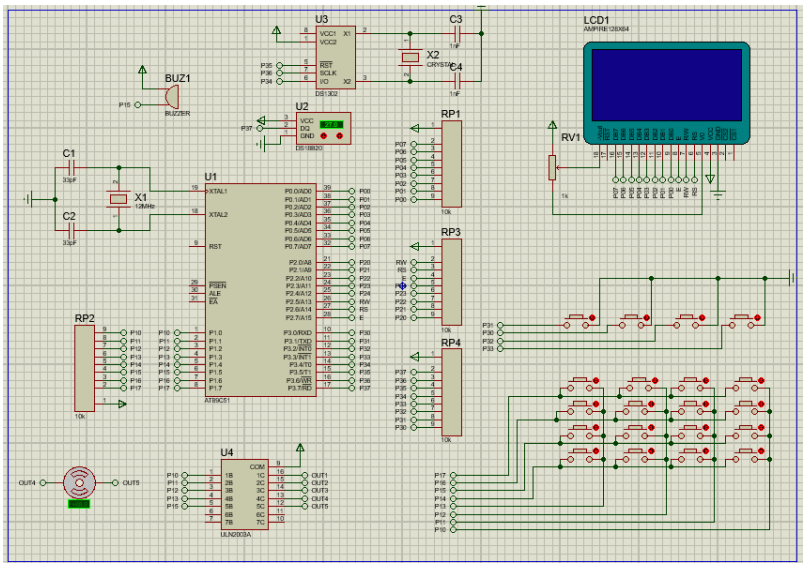
通过本项目，我不仅加深了对 51 单片机及相关组件的理解和应用能力，也对整个电子设计过程从概念到实现的每一个环节有了更加全面的把握。这个经历无疑为我未来在电子工程领域的深入研究和实践打下了坚实的基础。

在完成课程设计的过程中，我也遇到了很多问题，比如 LCD12864 使用中写入中文出现乱码的情况、直流电机高温保护使系统混乱，通过查阅资料、询问老师、与同学交流等方式，最终这些问题都得到了解决。在这个过程中，我解决问题的能力也得到了提高，在此，也想向帮助我过我的几位老师表示感谢。

《单片机系统设计》评分表

序号	课程目标	指标点	评价观察点	评价方式		得分（百分制）
				依据/评价人	权 重 w_j	
1	(CO1) 能通过查找文献及相关参考资料，对单片机系统设计任务进行需求分析和关键问题分析，确定解决方案和技术路线。	GR3.1	过程中检查能否通过查找文献及相关参考资料，对设计任务进行需求分析、关键问题分析、确定解决方案和技术路线。	实际操作/教师	0.2	
2	(CO2) 能综合经济、健康、安全、法律、文化、环境等因素，选择合适的软硬件开发平台，设计、整合相关程序，能对现有方案、方法进行改进，并能通过调试、运行，验证方案的有效性、先进性。	GR3.3	能否利用相关语言进行单片机系统的编程和设计实现；设计过程中态度及完成质量；验收过程中演示是否成功，能否正确回答老师的提问，并准确描述发现问题的过程和解决问题的办法。	验收/教师	0.3	
			设计报告的格式规范程度，是否图文并茂，语言规范及流畅程度；实验数据是否充分、结果是否正确，分析、对比是否充分，结论是否正确；是否提出了自己的独到见解。	设计报告/教师	0.3	
3	(CO3) 能够在系统开发及设计的过程中，认识到自主学习的必要性，发展终身学习的意识。	GR12.1	是否能够通过查找资料自主学习，并且在完成设计要求的基础上对设计内容进行拓展，是否具有自主学习、终身学习的意识。	验收/教师	0.1	
				设计报告/教师	0.1	
评分方法：评价人根据观察点要求对各项课程目标的完成情况进行评分（百分制）。 评分标准：1）观察点任务完成、能力达成，且表现突出：评为优秀，得分范围为 85~100； 2）观察点任务完成、能力达成，表现良好：评为良，得分范围为 75~84.9； 3）观察点任务完成、能力达成一般，表现较好：评为中，得分范围为 66~74.9； 4）观察点任务基本完成、能力基本达成，表现一般：评为及格，得分范围为 60~65.9； 5）观察点任务未完成、能力未达成，表现差：评为不及格，得分范围为 0-59；						
总分：						

附 1、原理图



附 2、使用说明书

将单片机上电后，屏幕会显示当前的模式，挡位，温度和温度阈值的信息，通过矩阵键盘 S5，S6 和 S7，可以控制风扇的模式，模式包括三个“自然风”、“睡眠风”、“常风”，第一排的 S1，S2，S3 和 S4 的四个按键，可以控制风扇的挡位，分为 1234，四个挡位。第三排的 S9 和 S10 可以控制温度阈值的增加和降低，当检测的温度超过温度阈值，蜂鸣器会报警，并且风扇会停转 10 秒。通过单片机配套的红外遥控器可以远程控制风扇的模式和挡位，遥控器的 7，8，9 对应着“自然风”、“睡眠风”、“常风”三个模式；1，2，3，4 对应着风扇的四个挡位。

附 3、主要程序清单

主函数：

```
1.#include "reg52.h"
2.#include "Delay.h"
3.#include "lcd12864.h"
4.#include "MatrixKey.h"
5.#include "Key.h"
6.#include <stdio.h>
```

```
#include "DS18B20.h"
#include "OneWire.h"
#include "AT24C02.h"
#include "Buzzer.h"
#include "Motor.h"
#include "IR.h"
#include <INTRINS.H>
#include "UART.h"


unsigned char fantype;
unsigned char Speed, type;   // 电机速度和风类型
int gear;                   // 风力档位
char gearstr[2];            // 用于转换风力档位为字符串
unsigned char Command;
unsigned char KeyNum, Tkeynum;  // 矩阵键盘和单独按键的返回值
unsigned char THigh;       // 温度阈值，默认值为 21 摄氏度
int THighint;
char THighstr[2];              // 用于转换温度阈值为字符串
float T;                       // 温度值
char Tstr[4];                  // 用于转换温度值为字符串
int flag_first = 1;
int time;


void main()
{
        THigh = 25;
        fantype=0;
        time=0;

    // LCD12864 初始化
    lcd12864_init();
    lcd12864_show_string(1, 0, "模式:");       // 第 1 行字符串显示
    lcd12864_show_string(2, 0, "挡位:");       // 第 2 行字符串显示
    lcd12864_show_string(3, 0, "温度:");       // 第 3 行字符串显示
    lcd12864_show_string(4, 0, "温度阈值:");   // 第 4 行字符串显示

        // 电机和红外
    Motor_Init();
    IR_Init();
        //UART_Init();

    //测温
```

```
    DS18B20_ConvertT();        // 上电先转换一次温度，防止第一次读数据错误
    Delay(1000);               // 等待转换完成
    THigh = AT24C02_ReadByte(0);    // 读取温度阈值数据

    if (THigh > 60)
    {
        THigh = 25;            // 如果阈值非法，则设为默认值
    }

    while (1)
    {
      time=0;
            if(IR_GetDataFlag()) //如果收到数据帧
                {
                    Command=IR_GetCommand();      //获取遥控器命令码
                    if(Command==IR_7)
                    {
                        type = 20;
                        fantype = 1;
                        gear=0;
                    }
                    if(Command==IR_8)
                    {
                        type = 40;
                        fantype = 2;
                        gear=0;
                    }
                    if(Command==IR_9)
                    {
                        type = 59;
                        fantype = 3;
                        gear=0;
                    }
                    if(Command==IR_1)
                    {
                        gear = 1;
                    }
                    if(Command==IR_2)
                    {
                        gear = 2;
                    }
                    if(Command==IR_3)
                    {
                        gear = 3;
```

```
95.                     }
96.                     if(Command==IR_4)
97.                     {
98.                         gear = 4;
99.                     }
100.                        if(Command==IR_6)
101.                        {
102.                            time = 2000;
103.                            P2= 0X7E;
104.                        }
105.                        if(Command==IR_5){
106.                            P2= 0X3E;
107.                            UART_Init();
108.                            if(gear == 1){
109.                            UART_SendByte(0x01);
110.                                }
111.                            if(gear == 2){
112.                            UART_SendByte(0x02);
113.                                }
114.                            if(gear == 3){
115.                            UART_SendByte(0x03);
116.                                }
117.                            if(gear == 4){
118.                            UART_SendByte(0x04);
119.                                }
120.                            Timer1_Init();
121.                        }
122.
123.                        if(Command==IR_VOL_ADD)
124.                        {
125.                            THigh++;
126.                            AT24C02_WriteByte(0,THigh);
127.                            Delay(5);
128.                        }
129.                        if(Command==IR_VOL_MINUS)
130.                        {
131.                            THigh--;
132.                            AT24C02_WriteByte(0,THigh);
133.                            Delay(5);
134.                        }
135.
136.                        Delay(100);
137.
138.                        lcd12864_clear();
```

```
                    lcd12864_show_string(1, 0, "模式:");       // 第1行字符串显示
                    lcd12864_show_string(2, 0, "挡位:");       // 第2行字符串显示
                    lcd12864_show_string(3, 0, "温度:");       // 第3行字符串显示
                    lcd12864_show_string(4, 0, "温度阈值:");   // 第4行字符串显示
                }

                KeyNum = MatrixKey();    // 矩阵键盘的返回值
                if(KeyNum)
                {
                    if(KeyNum==8){
                        THigh++;
                        AT24C02_WriteByte(0,THigh);
                        Delay(5);
                    }
                    if(KeyNum==7){
                        THigh--;
                        AT24C02_WriteByte(0,THigh);
                        Delay(5);
                    }
                    if(KeyNum==6){
                        time = 2000;
                        Delay(5);
                    }
                    if(KeyNum==12){
                        type = 20;
                        fantype = 1;
                        gear=0;
                        Motor_SetSpeed(0);
                    }
                    if(KeyNum==11){
                        type = 40;
                        fantype = 2;
                        gear=0;
                        Motor_SetSpeed(0);
                    }
                    if(KeyNum==10){
                        type = 59;
                        fantype = 3;
                        gear=0;
                        Motor_SetSpeed(0);
```

```
                        }
                        if(KeyNum==1){
                            UART_Init();
                            if(gear == 1){
                            UART_SendByte(0x01);
                                }
                            if(gear == 2){
                            UART_SendByte(0x02);
                                }
                            if(gear == 3){
                            UART_SendByte(0x03);
                                }
                            if(gear == 4){
                            UART_SendByte(0x04);
                                }
                            Timer1_Init();
                        }

                        if (KeyNum > 12)
            {
              gear = KeyNum - 12;
            }

                    }
                    if (gear)
          {
              Speed = type + gear * 10;
              sprintf(gearstr, "%d", gear);
              lcd12864_show_string(2, 3, gearstr);
              Motor_SetSpeed(Speed);

                                    if(time!=0)
                                {
                                    Delay(1000);
                                    Speed = 0;
                                    gear = 0;
                                    sprintf(gearstr, "%d", gear);
                                    lcd12864_show_string(2, 3,
   gearstr);
                                    Motor_SetSpeed(Speed);
                                    time = 0;
                                }
          }
                    else{
```

```
                        Motor_SetSpeed(0);
                        lcd12864_show_string(2, 3, "0");
                    }


                    if(fantype){
                        if(fantype==1){
                            lcd12864_show_string(1, 3, "自然风");
                        }
                        if(fantype==2){
                            lcd12864_show_string(1, 3, "睡眠风");
                        }
                        if(fantype==3){
                            lcd12864_show_string(1, 3, "常风 ");
                        }
                    }

                    /*温度读取及显示*/
          DS18B20_ConvertT();    // 转换温度
          T = DS18B20_ReadT();    // 读取温度
          sprintf(Tstr, "%.2f", T);
          lcd12864_show_string(3, 3, Tstr);
          THighint = THigh;
          sprintf(THighstr, "%d", THighint);
          lcd12864_show_string(4, 5, THighstr);    // 显示阈值数据


               if (T > THigh && flag_first==1)
          {
                        Motor_Delay();
                        flag_first = 0;
          }

      }
  }
```

矩阵键盘控制：

```
#include <REGX52.H>
#include "Delay.h"
unsigned char MatrixKey()
{
    unsigned char KeyNumber=0;

```

```
    P1=0xFF;
    P1_3=0;
    if(P1_7==0){Delay(20);while(P1_7==0);Delay(20);KeyNumber=1;}
     if(P1_6==0){Delay(20);while(P1_6==0);Delay(20);KeyNumber=5;}
     if(P1_5==0){Delay(20);while(P1_5==0);Delay(20);KeyNumber=9;}
     if(P1_4==0){Delay(20);while(P1_4==0);Delay(20);KeyNumber=13;}

     P1=0xFF;
     P1_2=0;
     if(P1_7==0){Delay(20);while(P1_7==0);Delay(20);KeyNumber=2;}
     if(P1_6==0){Delay(20);while(P1_6==0);Delay(20);KeyNumber=6;}
     if(P1_5==0){Delay(20);while(P1_5==0);Delay(20);KeyNumber=10;}
     if(P1_4==0){Delay(20);while(P1_4==0);Delay(20);KeyNumber=14;}

     P1=0xFF;
     P1_1=0;
     if(P1_7==0){Delay(20);while(P1_7==0);Delay(20);KeyNumber=3;}
     if(P1_6==0){Delay(20);while(P1_6==0);Delay(20);KeyNumber=7;}
     if(P1_5==0){Delay(20);while(P1_5==0);Delay(20);KeyNumber=11;}
     if(P1_4==0){Delay(20);while(P1_4==0);Delay(20);KeyNumber=15;}

     P1=0xFF;
     P1_0=0;
     if(P1_7==0){Delay(20);while(P1_7==0);Delay(20);KeyNumber=4;}
     if(P1_6==0){Delay(20);while(P1_6==0);Delay(20);KeyNumber=8;}
     if(P1_5==0){Delay(20);while(P1_5==0);Delay(20);KeyNumber=12;}
     if(P1_4==0){Delay(20);while(P1_4==0);Delay(20);KeyNumber=16;}
     return KeyNumber;
}
```

PWM 电机驱动：

```
#include <REGX52.H>
#include "Timer1.h"
sbit Motor=P1^0;
unsigned char Counter,Compare;
unsigned char isdelay;
unsigned long delaycounter;

void Motor_Init(void)
{
    isdelay=0;
    Timer1_Init();
}

void Motor_SetSpeed(unsigned char Speed)
```

```
{
    Compare=Speed;
}
// 设置是否延时 10s
void Motor_Delay(){
    isdelay=1;
}
//定时器 1 中断函数
void Timer1_Routine() interrupt 3
{
    TL1 = 0x9C;      //设置定时初值
    TH1 = 0xFF;      //设置定时初值
    Counter++;
    Counter%=100;    //计数值变化范围限制在 0~99
    if(isdelay==0){
        if(Counter<Compare)  //计数值小于比较值
        {
            Motor=1;         //输出 1
        }
        else                 //计数值大于比较值
        {
            Motor=0;         //输出 0
        }
    }else{
        delaycounter++;
        if(delaycounter>100000){
            isdelay=0;
            delaycounter=0;
        }
    }

}
```

DS18B20 温控芯片使用：

```
#include <REGX52.H>
#include "OneWire.h"
//DS18B20 指令
#define DS18B20_SKIP_ROM            0xCC
#define DS18B20_CONVERT_T           0x44
#define DS18B20_READ_SCRATCHPAD     0xBE
void DS18B20_ConvertT(void)
{
    OneWire_Init();
    OneWire_SendByte(DS18B20_SKIP_ROM);
    OneWire_SendByte(DS18B20_CONVERT_T);
```

```
}
float DS18B20_ReadT(void)
{
    unsigned char TLSB,TMSB;
    int Temp;
    float T;
    OneWire_Init();
    OneWire_SendByte(DS18B20_SKIP_ROM);
    OneWire_SendByte(DS18B20_READ_SCRATCHPAD);
    TLSB=OneWire_ReceiveByte();
    TMSB=OneWire_ReceiveByte();
    Temp=(TMSB<<8)|TLSB;
    T=Temp/16.0;
    return T;
}
```

LCD12864 显示：

```
#include "reg52.h"
#include "public.h"
sbit RS=P2^6;//数据命令选择
sbit RW=P2^5;//读写选择
sbit EN=P2^7;//使能信号
#define LCD12864_DATAPORT   P0  //LCD12864 数据端口定义
sbit LCD12864_PSB=P3^2;//8 位或 4 并口/串口选择
void lcd12864_write_cmd(u8 cmd)
{
    RS=0;//选择命令
    RW=0;//选择写
    EN=0;
    LCD12864_DATAPORT=cmd;//准备命令
    delay_ms(1);
    EN=1;//使能脚 E 先上升沿写入
    delay_ms(1);
    EN=0;//使能脚 E 后负跳变完成写入
}

void lcd12864_write_data(u8 dat)
{
    RS=1;//选择数据
    RW=0;//选择写
    EN=0;
    LCD12864_DATAPORT=dat;//准备数据
    delay_ms(1);
    EN=1;//使能脚 E 先上升沿写入
    delay_ms(1);
```

```
    EN=0;//使能脚 E 后负跳变完成写入
}

void lcd12864_init(void)
{
    LCD12864_PSB=1;//选择 8 位或 4 位并口方式
    lcd12864_write_cmd(0x30);//数据总线 8 位，基本指令操作
    lcd12864_write_cmd(0x0c);//整体显示关，游标显示关，游标正常显示
    lcd12864_write_cmd(0x06);//写入新数据后光标右移，显示屏不移动
    lcd12864_write_cmd(0x01);//清屏
}

void lcd12864_clear(void)
{
    lcd12864_write_cmd(0x01);
}

void lcd12864_show_string(u8 x,u8 y,u8 *str)
{
    unsigned char postion;
    switch(x)
    {
        case 1: x=0x80;break;//第 1 行地址+x 的偏移
        case 2: x=0x90;break;//第 2 行地址+x 的偏移
        case 3: x=0x88;break;//第 3 行地址+x 的偏移
        case 4: x=0x98;break;//第 4 行地址+x 的偏移
    }
    postion=x+y;
    lcd12864_write_cmd(postion);
    while(*str!='\0')
    {
        lcd12864_write_data(*str);
        str++;
    }

}
```

AT24C02 芯片使用:

```
#include <REGX52.H>
#include "I2C.h"
#define AT24C02_ADDRESS     0xA0
void AT24C02_WriteByte(unsigned char WordAddress,Data)
{
    I2C_Start();
    I2C_SendByte(AT24C02_ADDRESS); //芯片地址寻址
```

```
    I2C_ReceiveAck();
    I2C_SendByte(WordAddress);
     I2C_ReceiveAck();
     I2C_SendByte(Data);
     I2C_ReceiveAck();
     I2C_Stop();
}

unsigned char AT24C02_ReadByte(unsigned char WordAddress)
{
     unsigned char Data;
     I2C_Start();
     I2C_SendByte(AT24C02_ADDRESS);
     I2C_ReceiveAck();
     I2C_SendByte(WordAddress);
     I2C_ReceiveAck();
     I2C_Start();
     I2C_SendByte(AT24C02_ADDRESS|0x01); // 0xa1 芯片地址寻址，改变读写的方向
     I2C_ReceiveAck();
     Data=I2C_ReceiveByte();
     I2C_SendAck(1);
     I2C_Stop();
     return Data;
}
```

蜂鸣器使用：

```
#include <REGX52.H>
#include <INTRINS.H>

//蜂鸣器端口：
sbit Buzzer=P1^5;
void Buzzer_Delay500us()        //@12.000MHz
{
    unsigned char i;

     _nop_();
     i = 247;
     while (--i);
}

void Buzzer_Time(unsigned int ms)
{
     unsigned int i;
     for(i=0;i<ms*2;i++)
```

```
    {
        Buzzer=!Buzzer;
        Buzzer_Delay500us();
    }
}
```