基于 STM32 的散热风扇

学生姓名：　李宪忠　　组长　学号　2230506225

学生姓名：　李博宇　　组员　学号　2230506222

学生姓名：　张卓权　　组员　学号　2230506162

学生姓名：　杜骁健　　组员　学号　2230506210

专业班级：　**2022** 电子信息工程（**1**）（**2**）班

指导教师：　李宇

学院　医药信息工程学院

2025 年 1 月 10 日

目 录

1. 前言

一、设计目的 利用 FPGA 智能仪器 Eclypse All 开发平台设计一款风扇控制器。采用 PWM 实施基于温 度的风扇转速控制，达到良好的散热效果。 熟悉单片机、APP 交互开发、EDA 软件工具，掌握用 VERILOG 与 FPGA 器件进行 PWM 波形产生的基本方法和流程，提高工程设计实践能力。

2. 方案设计

2.1 设计思路

设计一款有界面控制的温控风扇系统。该平台能够监测风扇的转速、监测现场使用温 度的变化，控制风扇转速（风量），实施针对性散热。温度高了加快转速，温度低了降低 转速。温度恒定转速恒定。并在实验板上调试并实现所要求功能和技术指标，用示波器测 试转速以及各项指标，撰写实验报告，最后提交验收并答辩。 功能内容：

1. 掌握给定型号 AUB0912VH 的台达 4 线风扇的内部驱动控制原理，参数指标等。（10 分）

2. 人工测量并风扇最大以及最小转速时的电流电压；（10 分）

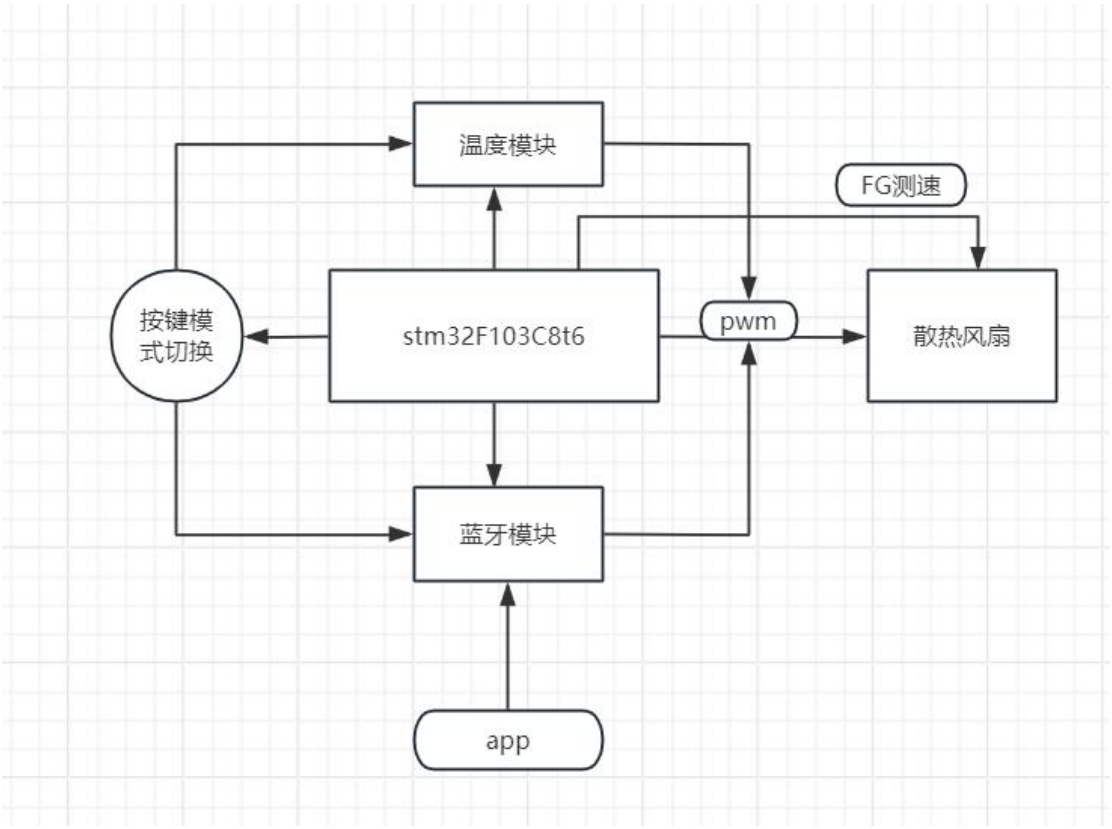
3. 采用 STM32 开发板设计 PWM 控制程序，用独立电源、开发板、手机搭建控制系统。APP 实施 PWM 控制交互。（40 分）

4. 设计测量程序，测量出风扇的转速范围。建立控制曲线，并与对应的温度传感器 建立控制策略，实施温控。（20 分）

5 将整个控制系统及其功能移植到 FPGA 开发平台。（20 分）

3. 硬件电路设计

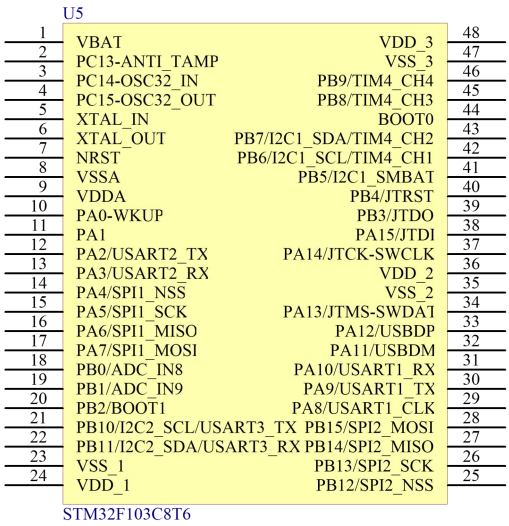
设计流程图



3.1 STM32 单片机最小系统电路设计

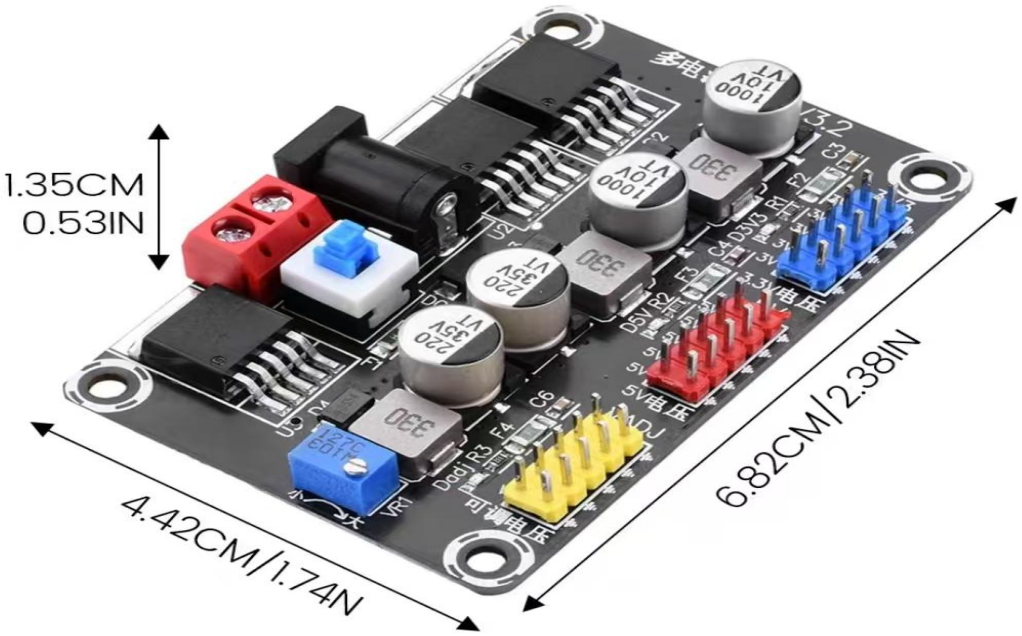
3.1.1 STM32 单片机简介

根据前面的方案设计可知，本设计使用一块 STM32F103C8T6 单片机作为主控制器。完成系统传感器的识别，以及风扇的控制。STM32F103C8T6 处理器的内核是 cortex-M3, 64KB 的 FLASH 内存容量以及 1 MB 的闪存，2 个 12 位，1us 模数转化器，可以映射到 16 个模数转化通道，3 个异步串行通信通道。STM32F103C8T6 单片机芯片原理图如下所示：



3.1.2 电源电路设计

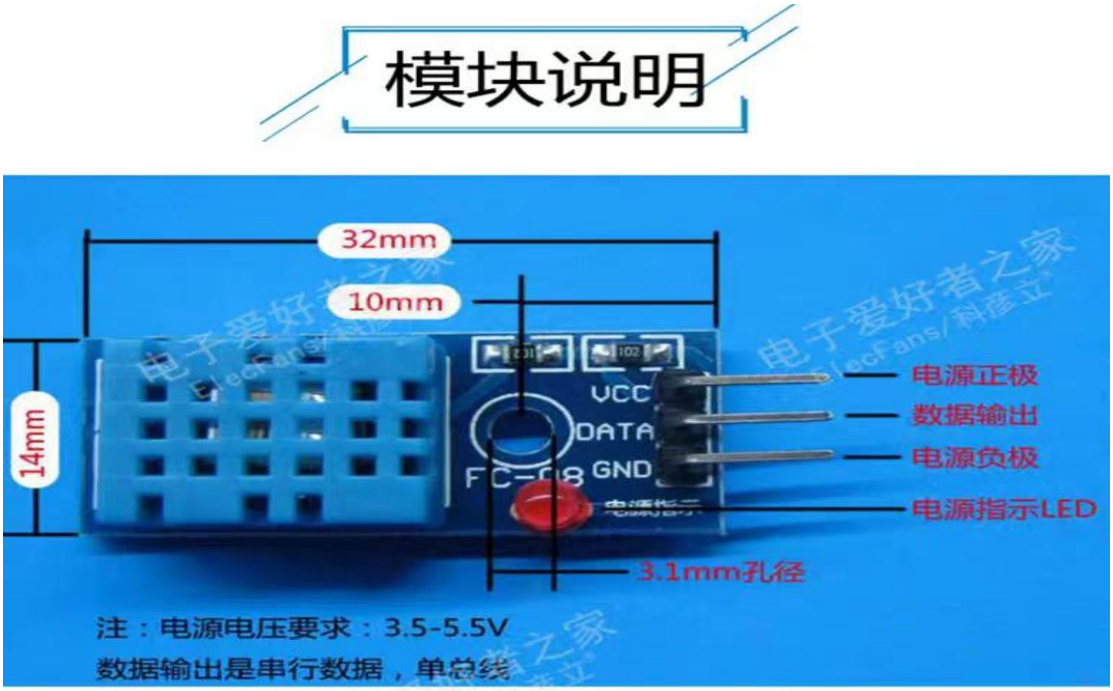
项目采用两个电源供应电路电源的方式，利用一块 12V 的锂电池把其中一端直接以 12V 直流电接到电源模块一端，供应给散热风扇，使其正常工作，分别从电源模块中引出 5V 和 3.3V 给蓝牙，温湿度模块和单片机供电。

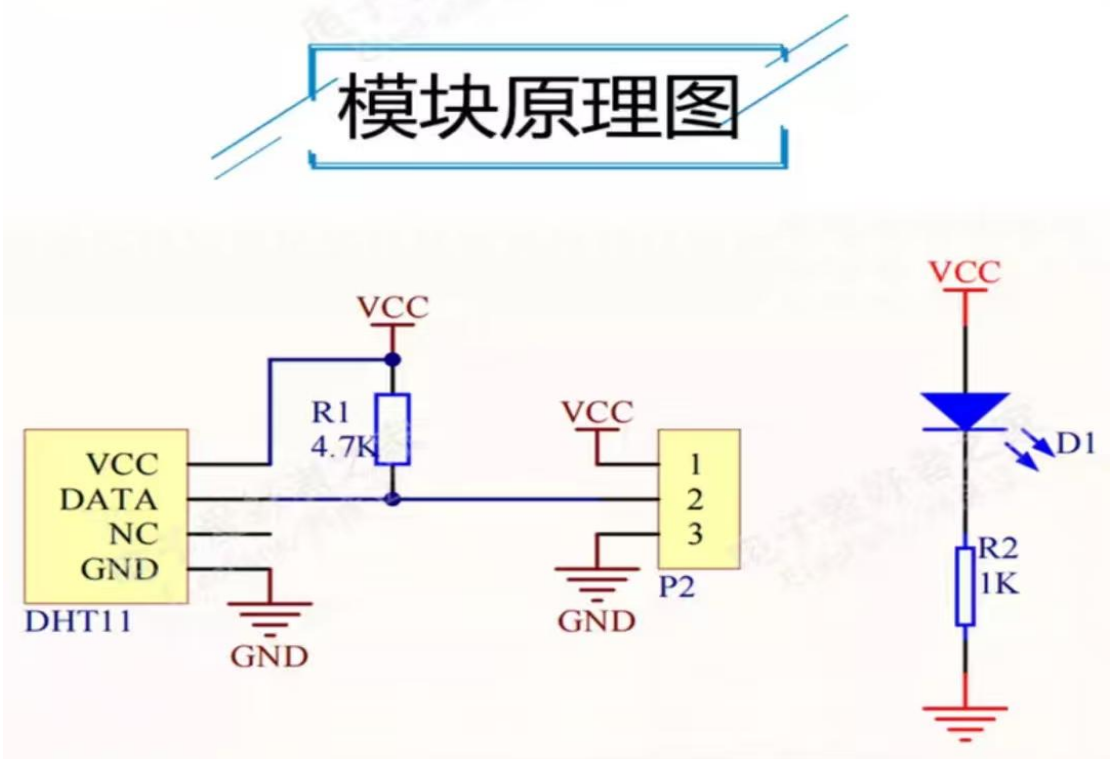


3.2 温湿度模块

通过温湿度模块读取外界温度变化，进而控制散热风扇随温度升高而提高转速，达到温控的效果，设计几个温度阈值，当处于某个阈值时散热风扇的对应风

力大小为设定的参数值。





温度模块相关代码

```
if(KeyNum==2)
{

   LED1_ON();
  if(humi<30)
  {
    LED1_OFF();
    TIM_SetCompare1(TIM2, 0);
    OLED_ShowNum(40,0,20,2,OLED_8X16);
    //OLED_ShowNum(40, 40, 000,3,OLED_8X16);
  }
  if(humi>30&&humi<40)
  {
    LED1_ON();
    TIM_SetCompare1(TIM2, 20);
    OLED_ShowNum(40,0,20,2,OLED_8X16);
    //OLED_ShowNum(40, 40, IC_GetFreq(),3,OLED_8X16);
  }
  if(humi>40&&humi<50)
   {
    LED1_OFF();
    TIM_SetCompare1(TIM2, 40);
     OLED_ShowNum(40,0,40,2,OLED_8X16);
     //OLED_ShowNum(40, 40, IC_GetFreq(),3,OLED_8X16);
   }
  if(humi>50&&humi<60)
   {
    LED1_ON();
   TIM_SetCompare1(TIM2, 60);
     OLED_ShowNum(40,0,60,2,OLED_8X16);
   //OLED_ShowNum(40, 40, IC_GetFreq(),3,OLED_8X16);
   }
  if(humi>60&&humi<70)
  {
    LED1_OFF();
    TIM_SetCompare1(TIM2, 80);
    OLED_ShowNum(40,0,80,2,OLED_8X16);
     //OLED_ShowNum(40, 40, IC_GetFreq(),3,OLED_8X16);
  }
}
```

3.3 散热风扇

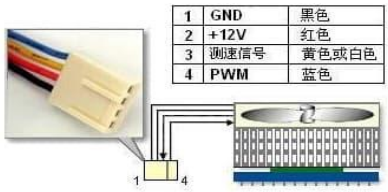
4 根线分别是 GND、VCC、FG、PWM。 位置可能不同。 FG 是转速信号，用于 CPU 侦测转 速。转速=频率*30（4 极风扇）。 PWM 通过方波占空比控制转速。PWM 接地--最低转速，PWM 不接最高转速。频率 25KHZ。一般 Duty 0%-20%转速是相同的。可以用函数信号 发生器提供信号测试 Duty--转速曲线。 MCU 可以控制 PWM 控制转速。可以通过 FG 做闭 环控制。 注意接口都是开集电极输出，需要上拉电阻。

黑色：地线

红色：正极（+12V）

黄色：转速信号

蓝色：调速（PWM）



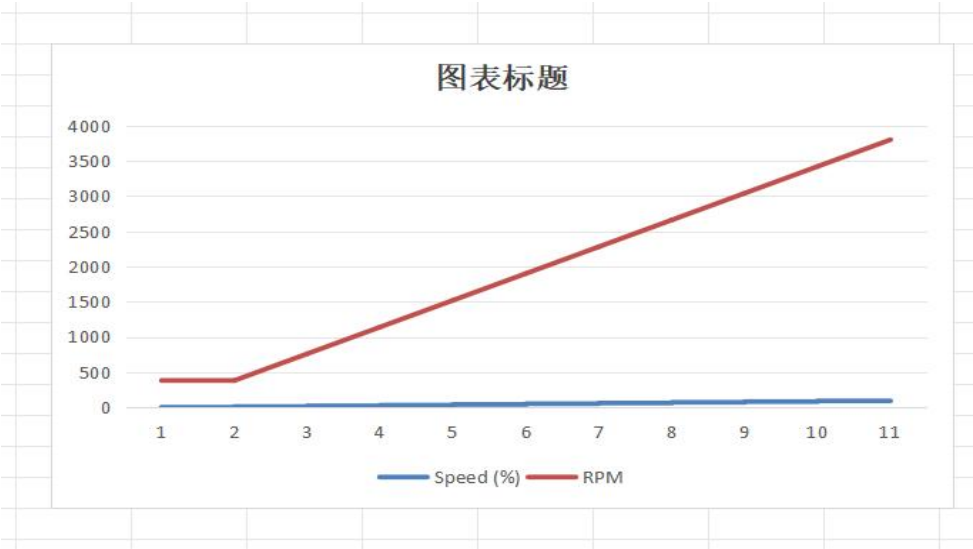
1	GND	黑色
2	+12V	红色
3	测速信号	黄色或白色
4	PWM	蓝色

CSDN @静思心远

PWM 初始化相关代码

```
#include "stm32f10x.h"                  // Device header

void PWM_Init(void)
{
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

  //RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);//
  //GPIO_PinRemapConfig(GPIO_PartialRemap1_TIM2,ENABLE);
  //GPIO_PinRemapConfig(GPIO_Remp_SWJ_JTAGDisable,ENABLE);

  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;//GPIO_Pin_15
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

   TIM_InternalClockConfig(TIM2);

  TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
  TIM_TimeBaseInitStructure.TIM_ClockDivision =TIM_CKD_DIV1;
  TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInitStructure.TIM_Period = 100 - 1;
  TIM_TimeBaseInitStructure.TIM_Prescaler =720 - 1;
  TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0;
  TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStructure);

   TIM_OCInitTypeDef TIM_OCInitStructure;
  TIM_OCStructInit(&TIM_OCInitStructure);
  TIM_OCInitStructure.TIM_OCMode =TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OCPolarity =TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState =TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse =0;
  TIM_OC1Init(TIM2,&TIM_OCInitStructure);
  TIM_OC2Init(TIM2,&TIM_OCInitStructure);
  TIM_Cmd(TIM2,ENABLE);
}
```

查找相关资料了解到该风扇的最高转速大概在 3800RPM，建立控制曲线

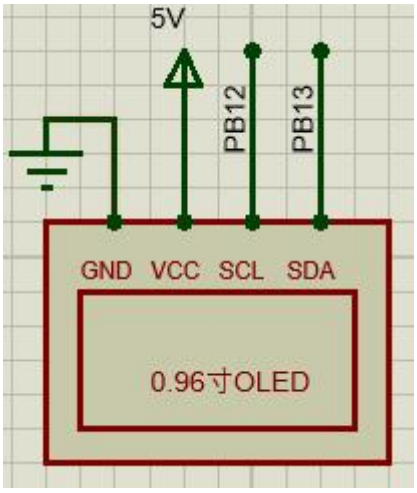


散热风扇最大风速和最小风速的电压和电流值：

最大风速，由于给定的电压是恒定的直流电压 12V，所以电流利用万用表测电流测得电流大概在 0.3A 附近，而风速最小时电流值在 0.08A 附近。

3.4 OLED 电路

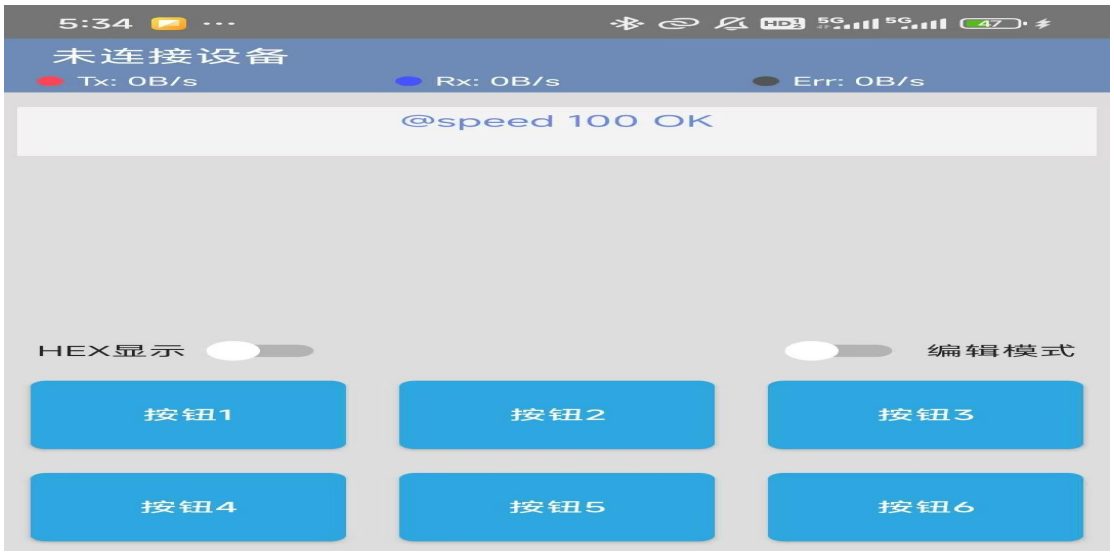
本次设计选用的是 I2C 通讯协议的 4 针 OLED 屏，通过 OLED 观察散热风扇 pwm 数值，温度数值，以及转速数值，电路图如下：

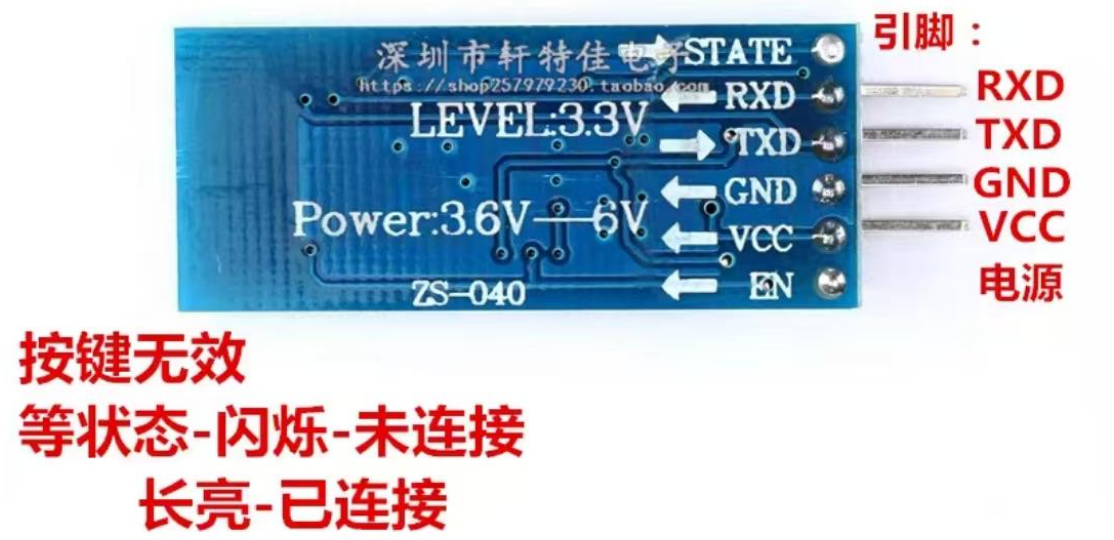


3.5 蓝牙电路

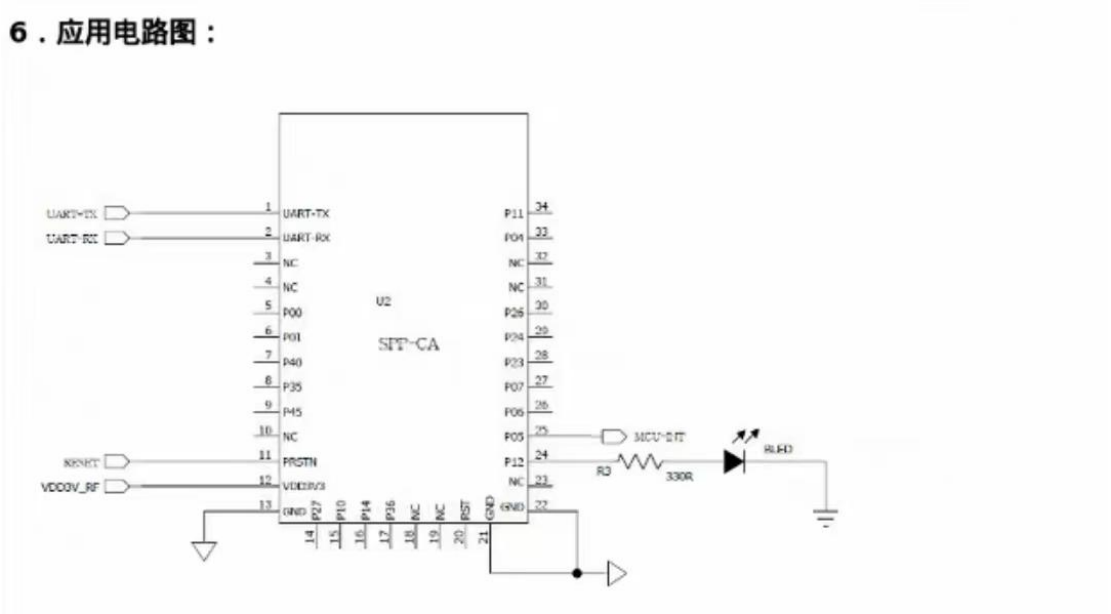
利用蓝牙模块实现手机与单片机信息交互，通过手机蓝牙

app 软件发送对应指令，单片机接收到指令对散热风扇的风速进行调控对手机 app 的界面设置占空比分别为 0，20，40，60，80，100，的六个数值指令，发送到单片上执行





6．应用电路图：



蓝牙相关代码

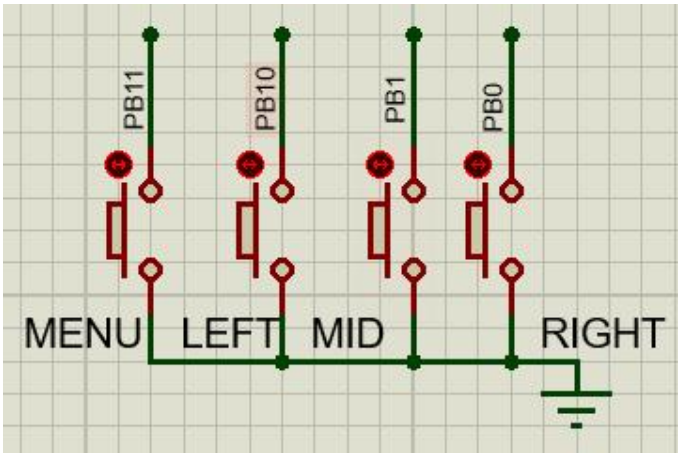
```
if(Serial_GetRxFlag()==1  && KeyNum!=2)
{
  if(strcmp(Serial_RxPacket,"speed20")==0)
  {
    LED1_ON();TIM_SetCompare1(TIM2, 20);
    OLED_ShowNum(40,0,20,2,OLED_8X16);
    Serial_SendString("@speed 20 OK\r\n");
  }
  else if(strcmp(Serial_RxPacket,"speed40")==0)
  {
  LED1_OFF();TIM_SetCompare1(TIM2, 40);
    OLED_ShowNum(40,0,40,2,OLED_8X16);
    Serial_SendString("@speed 40 OK\r\n");
  }
  else if(strcmp(Serial_RxPacket,"speed60")==0)
  {
  LED1_OFF();TIM_SetCompare1(TIM2, 60);
    OLED_ShowNum(40,0,60,2,OLED_8X16);
  //  OLED_ShowNum(40, 40, IC_GetFreq(),3,OLED_8X16);
    Serial_SendString("@speed 60 OK\r\n");
  }
  else if(strcmp(Serial_RxPacket,"speed80")==0)
  {
  LED1_OFF();TIM_SetCompare1(TIM2, 80);
    OLED_ShowNum(40,0,80,2,OLED_8X16);
  //  OLED_ShowNum(40, 40, IC_GetFreq(),3,OLED_8X16);
    Serial_SendString("@speed 80 OK\r\n");
  }
  else if(strcmp(Serial_RxPacket,"speed100")==0)
  {
  LED1_OFF();TIM_SetCompare1(TIM2, 100);
    OLED_ShowNum(40,0,99,2,OLED_8X16);
```

3.6 按键电路

通过按键切换蓝牙控制模式和温度控制模式

选取 PB0 和 PB1 为按键引脚，按键按下，GPIO 端口电平拉低，GPIO 检测输入电流。通过外部中断制标志位，定义全局变量 num 的值变为 2 为温控模式，为 1 并且接收到蓝牙 app 发送的信号时为蓝牙控制模式，以下是电路图：

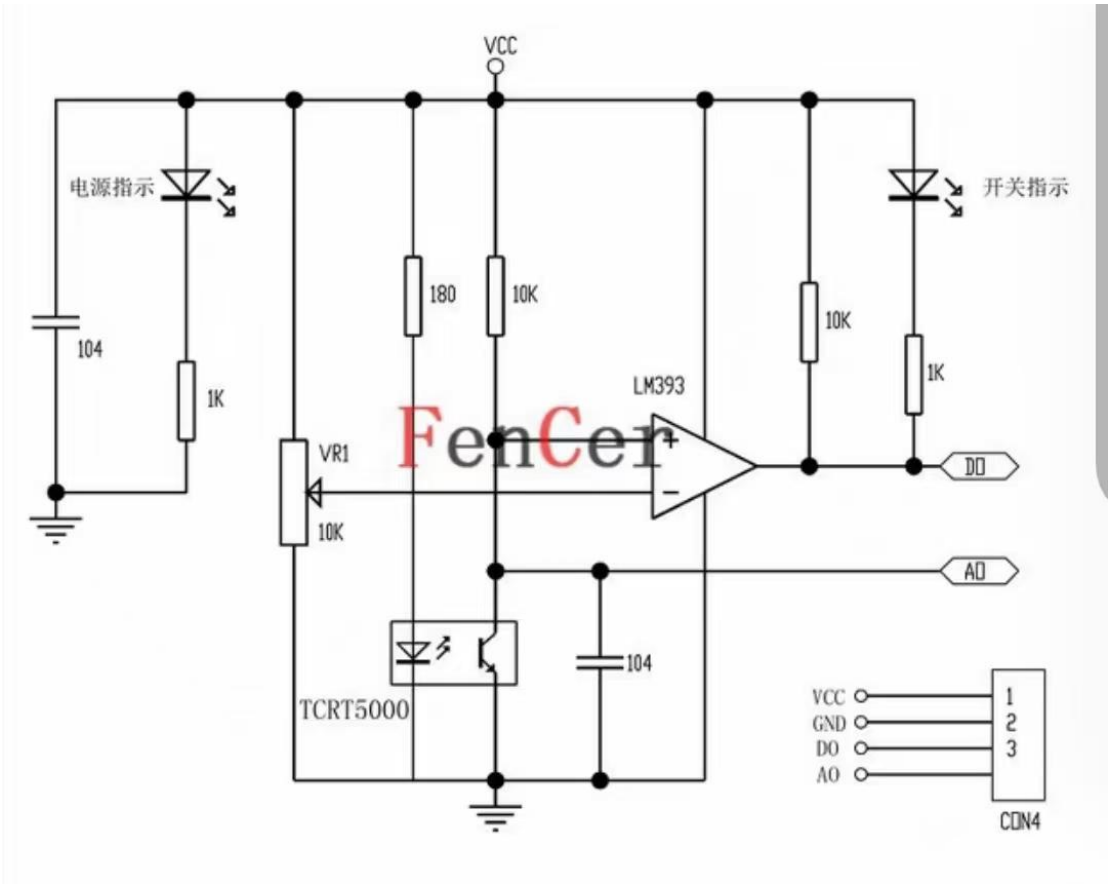
```
//     Num++;
//     TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
//   }
//}


void EXTI0_IRQHandler(void)
{

    if(EXTI_GetITStatus(EXTI_Line0)!=RESET)
    {

        KeyNum=1;
        EXTI_ClearITPendingBit(EXTI_Line0);


    }

}
void EXTI1_IRQHandler(void)
{


     if(EXTI_GetITStatus(EXTI_Line1)!=RESET)
    {

        KeyNum=2;
        EXTI_ClearITPendingBit(EXTI_Line1);


    }
}
```



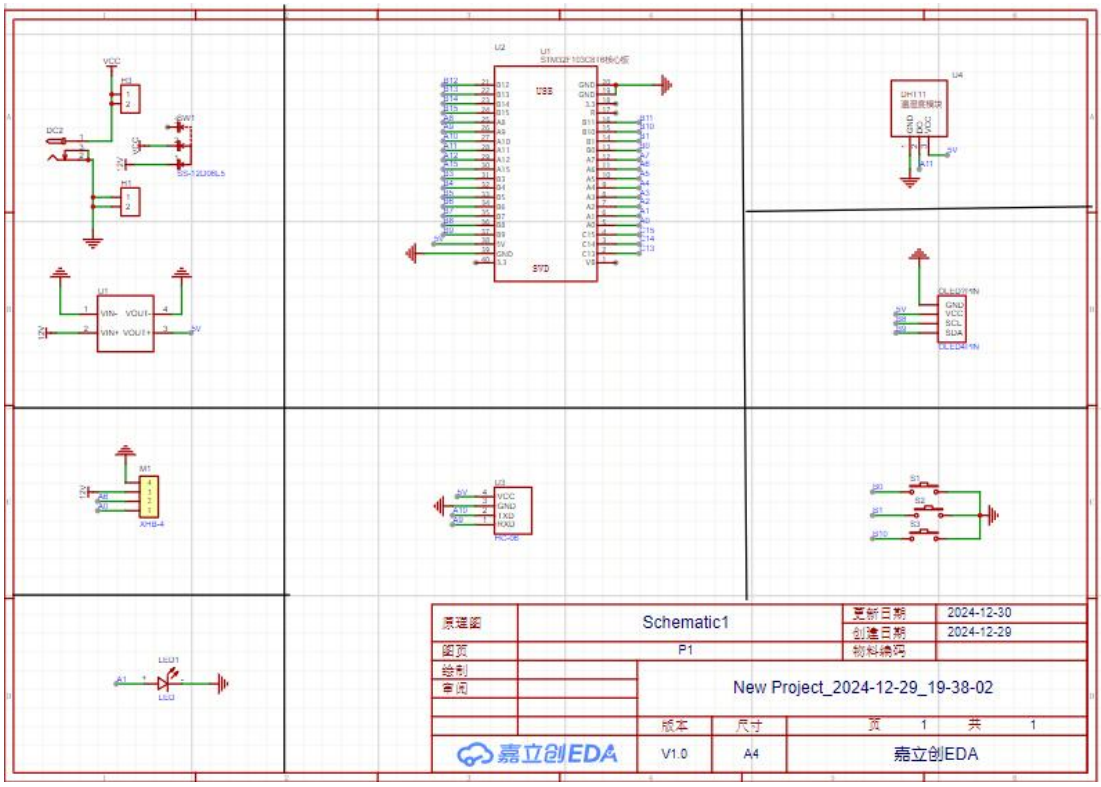
3.7 测速模块

测速可以采用散热风扇自带的 FG 线，但实测 FG 线在 12V 电压状态下产生的频率不稳定，通过输入捕获的方式难以测得实际的转速制，后面采用一款红外循迹模块作为测速使用，选取 PB14 为红外测速端口，

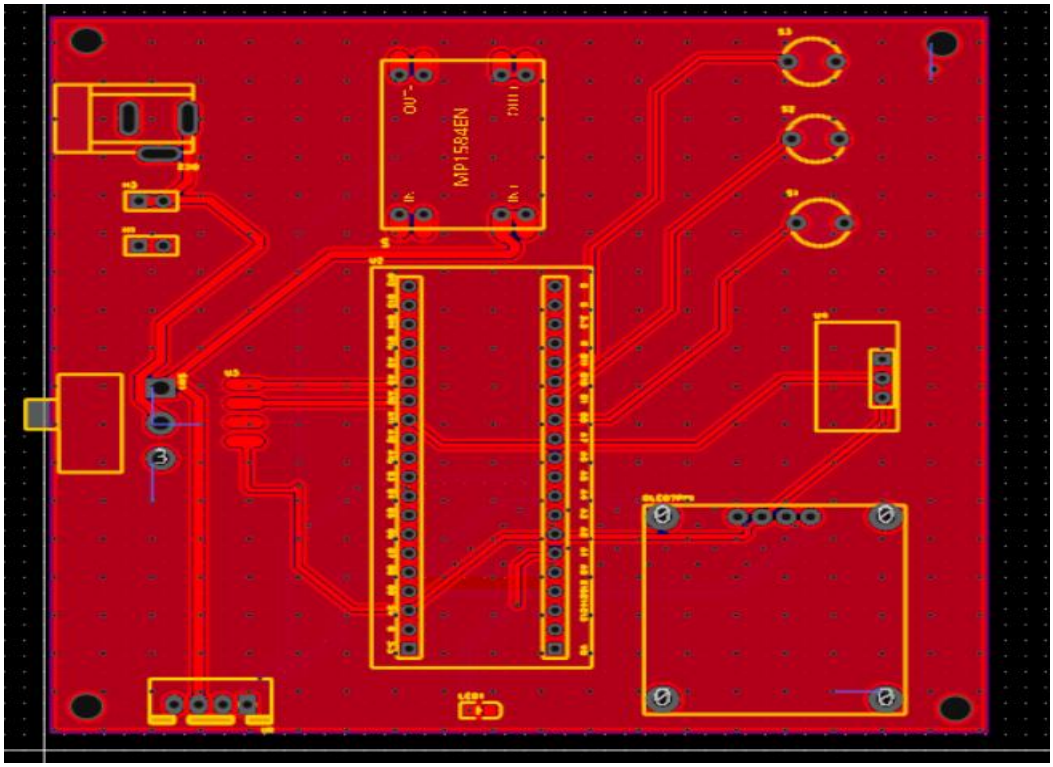
其基本原理通过检测黏贴在风扇表面的贴纸，风扇每自转一圈，红外检测模块检测到其信号，进行一次外部中断让计数值 cout 自增，再利用单片机设定一个定时器每 1s 进行一次定时器中断，在中断中让 OLED 显示出 cout 值也就是红外模块检测到的 cout 次数，即每秒的转速值，通过这个方法有效测出散热风扇的最大转速在每秒 63rad/s 左右,再乘以 60 得到风扇最高转速为 3800RPM 符合风扇给定的参数值。



3.8 总电路图


Schematic1
New Project_2024-12-29_19-38-02
2024-12-30
2024-12-29
V1.0
A4
嘉立创EDA

PCB 绘制


MP1584EN

3.9 FPGA 代码移植

由于时间和器件的原因 FPGA 部分进展缓慢，学习和参考网上学习资源，完成了 PWM,蓝牙模块和温度模块的相关代码的移植编译，但都是单独使用，没有整合到一起，无法测试代码的可行性，但也算是有所收获 FPGA 相关知识的学习经验。

FPGA 代码移植

PWM 代码

```
module pwm_ctrl(
    input sys_clk,           // 系统时钟
    input sys_rst_n,         // 复位信号，低有效
    input [6:0] duty_data,   // 输入占空比数据（范围0-100）

    output reg pwm_out       // 输出PWM信号
);
    wire [10:0] duty_count;    // 占空比对应的计数值，最大值为2000
    reg [10:0] pwm_counter;   // 用于生成25kHz的计数器，最大值为2000

assign duty_count = (duty_data * 2000) / 100;  // duty_data 变成占空比对应的计数值（最大2000）

// pwm_counter
always @(posedge sys_clk or negedge sys_rst_n)
        if (~sys_rst_n) begin
            pwm_counter <= 11'b0;  // 复位时，将PWM计数器清零
        end
        else   if (pwm_counter < 11'd1999) begin
            pwm_counter <= pwm_counter + 1;
        end
        else begin
            pwm_counter <= 11'b0;  // 计数器溢出，重新开始
        end

// pwm_out
always @(posedge sys_clk or negedge sys_rst_n)
        if (~sys_rst_n) begin
            pwm_out <= 1'b0;    // 复位时，PWM输出为低电平
        end
        else if (pwm_counter < duty_count)begin
            pwm_out <= 1'b1;  // 如果计数器小于 duty_count，输出高电平
        end
        else
            pwm_out <= 1'b0;  // 否则输出低电平
endmodule
```

温度传感器 FPGA 相关代码

```
reg          dq_out       ;    //输出总线数据，即FPGA给的总线数据值
reg          dq_en        ;    //输出总线数据使能信号

//温度转换,保留整数
assign temp_data = (data * 10'd625)/ 14'd1000;

//FPGA总线控制：当使能信号为1是总线的值为dq_out的值，为0时为高阻态
assign  dq  =   (dq_en ==1 ) ? dq_out : 1'bz;

//cnt:分频计数器
always@(posedge sys_clk or  negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        cnt <=  5'b0;
    else    if(cnt == 5'd24)
        cnt <=  5'b0;
    else
        cnt <=  cnt + 1'b1;

//clk_1us：产生单位时钟为1us的时钟
always@(posedge sys_clk or  negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        clk_1us <=  1'b0;
    else    if(cnt == 5'd24)
        clk_1us <=  ~clk_1us;
    else
        clk_1us <=  clk_1us;

//cnt_1us：1us时钟计数器，用于状态跳转
always@(posedge clk_1us or  negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        cnt_1us <=  20'b0;
    else    if(((state==S_WR_CMD || state==S_RD_CMD || state==S_RD_TEMP)
       && cnt_1us==20'd64) || ((state==S_INIT || state==S_INIT_AGAIN) &&
       cnt_1us==20'd999) || (state==S_WAIT && cnt_1us==S_WAIT_MAX))
        cnt_1us <=  20'b0;      // 不同状态对应不同的时间终值：
                                // S_INIT、S_INIT_AGAIN -> 1ns
```

蓝牙模块 FPGA 相关代码

```
        bit_cnt <= 4'b0;
    else    if((bit_cnt == 4'd8) && (bit_flag == 1'b1))
        bit_cnt <= 4'b0;
     else    if(bit_flag ==1'b1)
         bit_cnt <= bit_cnt + 1'b1;

//rx_data:输入数据进行移位
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        rx_data <= 16'b0;
    else    if((bit_cnt >= 4'd1)&&(bit_cnt <= 4'd8)&&(bit_flag == 1'b1)&&(byte_cnt == 1'b0))    // 第一个字节
        rx_data <= {8'b0, rx_reg3, rx_data[7:1]};
    else    if((bit_cnt >= 4'd1)&&(bit_cnt <= 4'd8)&&(bit_flag == 1'b1)&&(byte_cnt == 1'b1))    // 第二个字节
        rx_data <= {rx_reg3, rx_data[15:9], rx_data[7:0]};

//byte_cnt:两字节计数器，值为0,1
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        byte_cnt <= 1'b0;
    else    if((bit_cnt == 4'd8) && (bit_flag == 1'b1))
        byte_cnt <= ~byte_cnt;      // 取反

//output_flag:输出标志位
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        output_flag <= 1'b0;
    else    if((bit_cnt == 4'd8) && (bit_flag == 1'b1) && (byte_cnt == 1'b1))
        output_flag <= 1'b1;
    else
        output_flag <= 1'b0;

//unpack_data:输出完整的8位有效数据
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        unpack_data <= 13'b0;
    else    if(output_flag == 1'b1)
        unpack_data <= rx_data[12:0];       // 取完两个字节数据
```

6 总结

由于时间紧迫和知识贮备不足，本次设计目前部分功能以及电路连接方面还有许多不足，功能方面完善不够全面，还存在比较大的改善空间，FPGA 移植方面还是存在一些小问题，但相信通过后续的学习能解决这些小问题，通过这次课程设计巩固和提升了自己动手和实践能力，强化了自己的检查错误和查找问题并解决问题的能力，通过团队合作大家一起学习交流，收获很多。