卷帘大将

说明文档

OpticalMoe

2020/08/23

前 言

该项目旨在开发一款电动窗帘设备。

设备使用 STM32 单片机做主控，通过控制编码器电机正反转实现窗帘的电动开合，并通过 EMW3080 接入阿里云物联网平台，实现本地/云端控制。

电源：采用 LP6498 芯片，设计输出 5V@1A。LDO 采用 HX9193，设计输出 3.3V。

WIFI：采用 EMW3080（阿里飞燕固件），可接入阿里云生活物联网平台。

电机驱动：配有 AB 相编码器接口，可实现电机的位置环、速度环 PID 控制；驱动芯片采用 RZ7899，支持 25V@3A（最大 5A）；另配置了一限位开关接口，防止窗帘超限位运行损坏。

外设：配有无源蜂鸣器，通过 TIM 定时器控制，可奏乐；两个 LED 指示灯用于指示 WIFI 联网状态和设备运行状态；光敏电阻和 NTC 热敏电阻，可反馈光照和温度；两枚按键，单键可手动控制窗帘的上下行，双键可进入配网模式。

使用方法

一、配网

1. 上电后任意时刻，同时按下两按键两秒（先后按下两按键也可识别）。

2. 设备奏乐“Bi~~~Pu”,开始配网流程。蓝色、绿色指示灯快闪。

3. 手机打开“云智慧”，扫码后，按提示开始配网。（扫码可下载 APP）。

4. 等待 5s 后设备奏乐“嘀嘀嘀”三声，设备开启 AP 热点。

5. 手机端输入设备将要连接的家庭路由器名称和密码，点击“开始连接”，开始配网。（低版本安卓系统可直接接入 adh_xxx 热点，高版本安卓需手动连接该热点，热点连接成功后会断开，返回 APP 界面即可）。

6. 耐心等待设备上云。若过程中设备奏乐“滴~~~~~”表示出错，可以断电重启后从第 2 步重试。

7. 当设备配网成功后，奏乐“1234567”，蓝色、绿色指示灯慢闪。

注：在配网过程中任意时刻意外断电导致设备配网失败，下次上电自动进入配网模式。从第 2 步开始依次执行配网操作直至配网完成。

二、联网控制

设备配网后可实现联网控制。

设备联网成功后奏乐“123”，蓝色指示灯开始慢闪。

按下上键或下键，窗帘上行或下行，松手后，APP 端自动刷新窗帘位置。

APP 端滑动窗帘位置滑块，窗帘运行到指定位置停止。

点击 APP 端“快捷操作”，可直接控制窗帘到达预设位置。

点击 APP 端“状态信息”，可查看设备温度、电压、光强等数据。

配网二维码：（未安装 APP 可扫码下载）

目 录

一 器件选型

本次活动要求设计一款物联网设备。为了控制成本，器件选型尽可能地选择性价比高的器件。

1. 电源

电源输入插座采用 DC005 插座，设计可承受 30V@3A。

Datasheet:DC005-30A

商城编号：C111573

封装：DC005-T25

输入：30V（最大）

电流：3A（最大）

注意选用 A 级插座，并注意可承受的电压电流是否满足。

DCDC 采用 LP6498AB6F 芯片，设计输出 5.12V@1A。

Datasheet:LP6498

商城编号：C387722

封装：SOT23-6

输入：4.5～30V

输出：4.8～12V

电流：1200mA（最大）

该芯片耐压高，输入、输出电压范围宽，电流大。体积小，外围电路简单，输出电压可调。便宜皮实，性价比高。

LDO 采用 HX9193-33GB，设计输出 3.3V@600mA。

Datasheet:HX9193-33GB

商城编号：C296123

封装：SOT-23-5

输入：6V（最大）

输出：3.3V（固定）

电流：600mA（最大）

压降：480mV（最大）

该芯片电流大，压降小。体积小，外围电路简单。便宜皮实，性价比高。

2. MCU

MCU 采用 STM32F030K6 单片机。

Datasheet:STM32F030K6T6

商城编号：C88446

封装：LQFP32

选用这款单片机的主要原因是 ~~便宜~~ 性价比高。同时 STM32 芯片可以使用 ST-Link 连接 Keil 在线 DEBUG，也可以使用 STM32CubeMonitor 软件打印内部变量变化曲线，方便 PID 调试。

这款单片拥有 32KB FLASH，4KB RAM[1]，48MHz 的主频，LQFP-32 的封装，一个串口，5 个定时器，一个 10 通道 12bit AD，26 个 IO。可谓是小巧精悍，实力不凡。

[1] 原本计划上 RTOS 的，但是 4KB 的 RAM 跑 OS 有点勉强，稍稍加点东西就超，最后没跑上。

3. WIFI

WIFI 选用的是 EMW3080V2（阿里云飞燕固件）。WIFI 选型没有经验，全跟课程走。

4. 电机驱动

电机驱动部分采用 RZ7899 驱动芯片。

Datasheet:RZ7899

商城编号：C92373

封装：SOP-8_150mil

输入：3 ~ 25V

电流：3A

峰值电流：5A

内建刹车功能、内置过温保护、内置短路保护、内置过流保护。

电流传感器采用 CC6900SO-5A 芯片。

Datasheet:CC6900SO-5A

商城编号：C350864

封装：SOP-8

增益：400mV/A

电流：5A

5. 外设

外设部分设计有：一个无源蜂鸣器，两个按键，两个 LED 指示灯。编码器接口，限位开关接口。

二 原理图设计

根据个人的设计习惯，原理图按功能划分，设计在 5 张 A4 图纸上。下面依次介绍。

1. 电源

电源部分主要分为四块。分别是：电源输入插座、DCDC 降压、LDO 降压、测试点。

电源输入插座正极先通过 SS54 二极管，再接入设备。

DCDC 和 LDO 部分按照官方手册绘制就可。

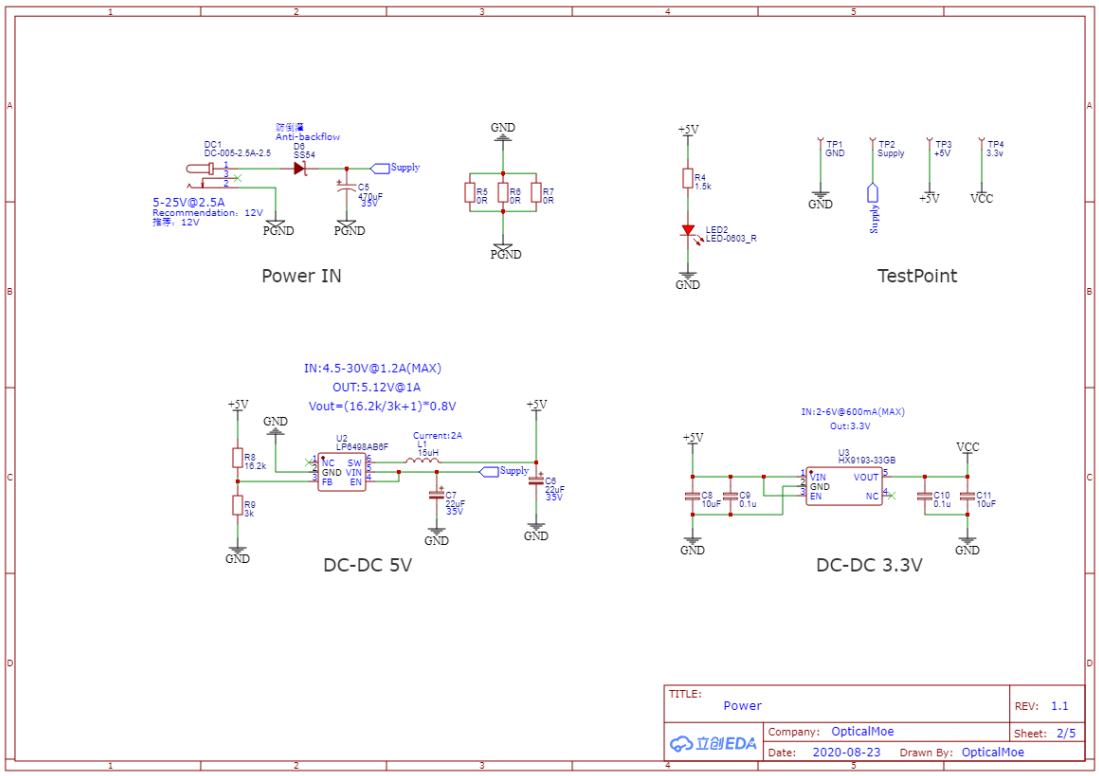

图 2.1 原理图-电源

2. MCU

MCU 部分主要设计晶振电路，复位电路，SWD 下载接口。

晶振采用 SMD-3225 封装的 8MHz 无源晶振，该封装对烙铁焊接不友好。晶振电路主要由晶振和两个 22pF 无极性陶瓷电容构成。

复位电路由 10k 上拉电阻和 0.1uF 电容构成，主要完成上电复位功能。

SWD 接口用于调试和下载程序，引出了 SWCLK、SWDIO、NRST，采用 XH2.5-4P 端子接口。引出 NRST 引脚，即使程序中未使能 SWD 调试接口仍能下载、调试程序。

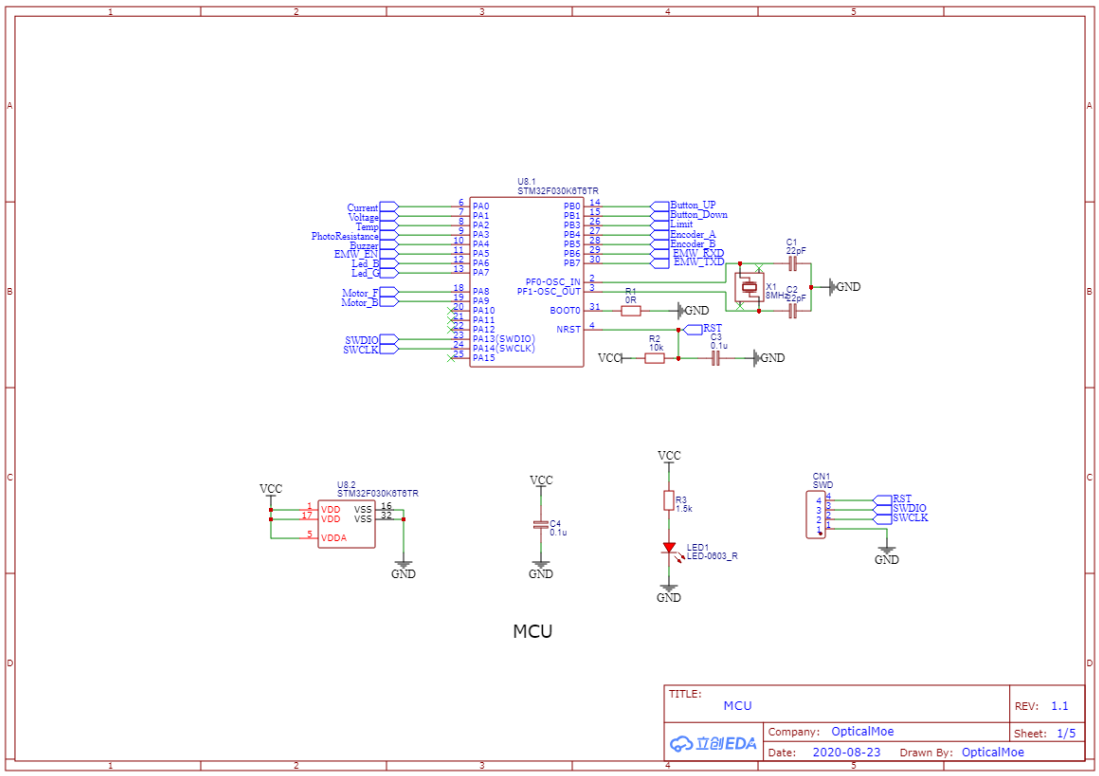

图 2.2 原理图-MCU

3. EMW3080

EMW3080 电路主要包括电源滤波、串口、BOOT、测试点。

电源滤波采用 0.1uF 和 10uF 组合的形式；串口通过 0R 电阻交叉连接到 MCU 串口；GPIO23 根据手册要求通过 10k 上拉；BOOT 引脚预留 0R 电阻接地，但不焊接；EN 引脚通过 0R 电阻连接到 MCU 和按键，主要完成 WIFI 的复位工作。

根据手册要求，电源采用 3.3V。

测试点包括串口的 TXD 和 RXD 接口。调试时连接串口，可监视 MCU 和 WIFI 间交换的所有数据[2]。

[2] 连接外部串口监视数据时，MCU 串口需设置为开漏+上拉模式，否则会导致 MCU 与 WIFI 间数据乱码。

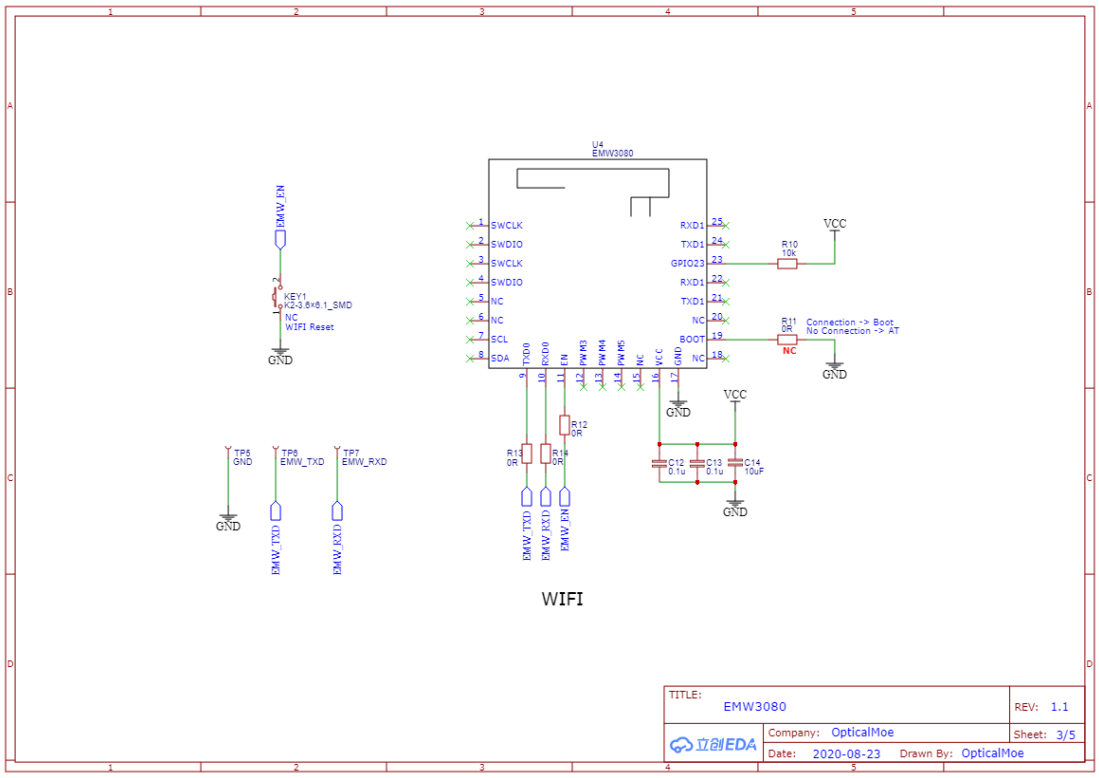

图 2.3 原理图-EMW3080

4. 电机驱动

电机驱动部分主要完成电流传感器电流采样、电源电压采样、电机驱动、测试点。

电源电压采样采用分压电阻结构，通过 100k 和 10k 电阻获得低的采样电压送入 MCU-ADC 接口。

电流采样按照 CC6900SO-5A 官方手册绘制即可。

电机驱动芯片按照官方手册绘制，注意输入和输出接口走线宽度。同时在电机接口上设计四个二极管钳位。电机采用 5.0-2P 接口，方便拆装。

测试点主要有电流采样点、电压采样点、电机驱动正反转信号点，便于调试时确定状态。

注意功率地与信号地分开，并连接

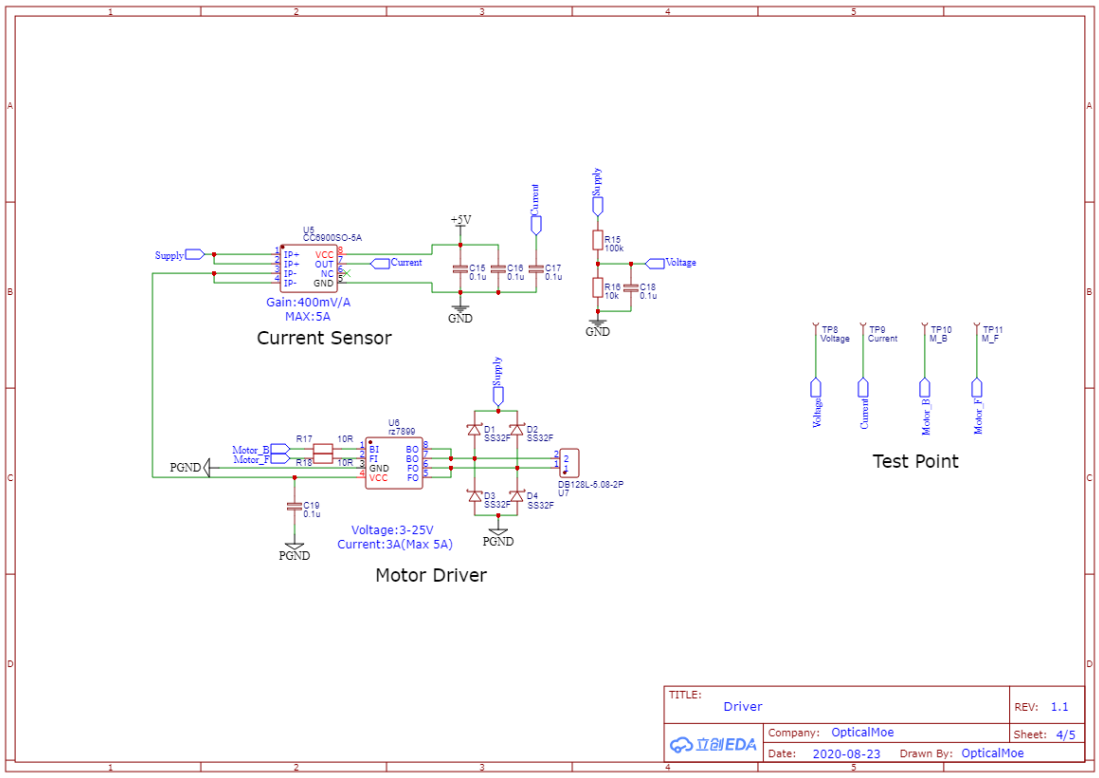

图 2.4 原理图-电机驱动

5. 外设

外设主要设计无源蜂鸣器、光敏电阻、热敏电阻、编码器接口、按键、LED 指示灯、测试点、机械孔。

无源蜂鸣器需连接到 TIM-PWM 输出引脚，可以通过调整 TIM 装载值和比较值控制蜂鸣器音调和音量。

热敏电阻和光敏电阻需要串联一个已知阻值的电阻接入电路，通过 MCU-ADC 测量中间点电压反向推算出外部温度和光强[3]。

[3] 热敏电阻和光敏电阻测量精度非常有限，即使程序中加入修正，采样值仍可能和实际值偏差较大。对温度和光强精度要求高的场所慎用。

编码器接口是为了电机的位置环、速度环 PID 设计，可连接 AB 相编码器。接口内已设计上拉电阻和硬件消抖电路，编码器电源通过两个 0R 电阻在 5V 和 3.3V 间选择，注意不可同时连接 5V 和 3.3V 电阻。

按键用于控制窗帘的上拉、下拉动作，同时在必要时刻充当配网开关。

蓝色 LED 指示灯用于指示 WIFI 连接状态，绿色 LED 用于指示设备运行状态。

测试点可测量光敏电阻和热敏电阻输出电压。

机械孔是四个 M3 螺丝孔，方便设备通过螺丝安装在需要的地方。

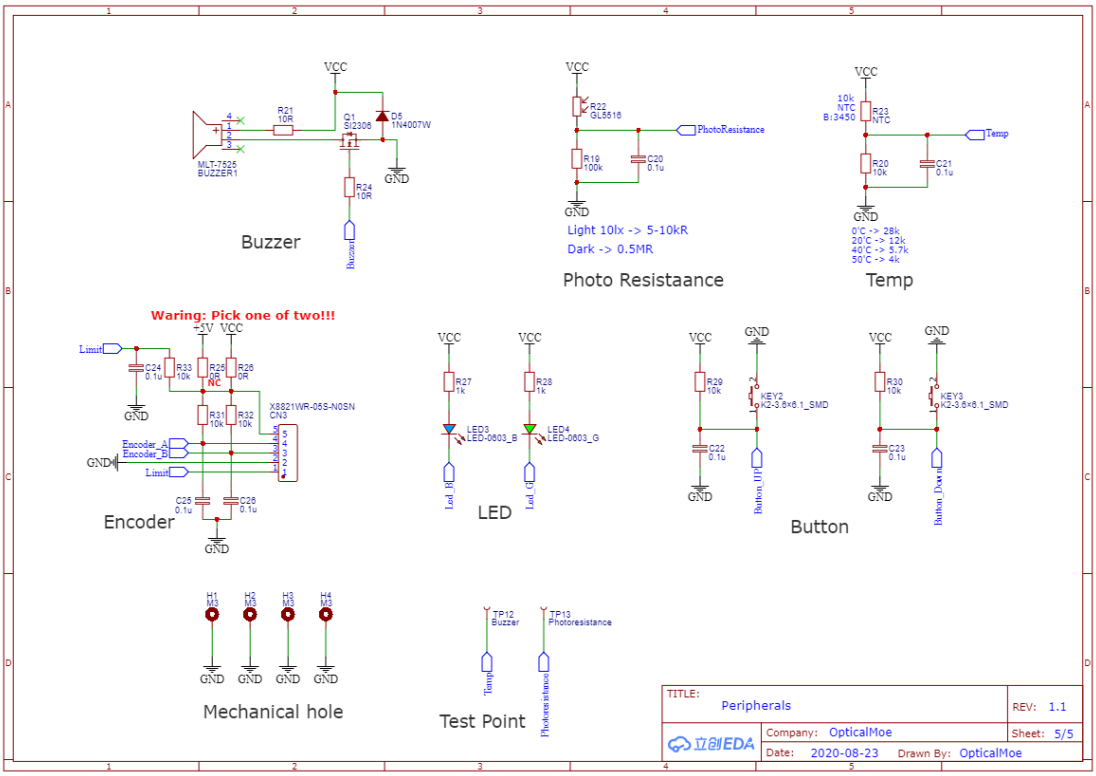
Buzzer
Photo Resistaance
Light 10lx -> 5-10kR
Dark -> 0.5MR
Temp
0'C -> 28k
20'C -> 12k
40'C -> 5.7k
50'C -> 4k
Waring: Pick one of two!!!
Encoder
LED
Button
Mechanical hole
Test Point
TITLE: Peripherals
REV: 1.1
Company: OpticalMoe
Sheet: 5/5
Date: 2020-08-23
Drawn By: OpticalMoe

图 2.5 原理图-外设

三 PCB 设计

PCB 设计经验不足，在此抛砖引玉。如有错误之处，还望大佬不惜赐教。

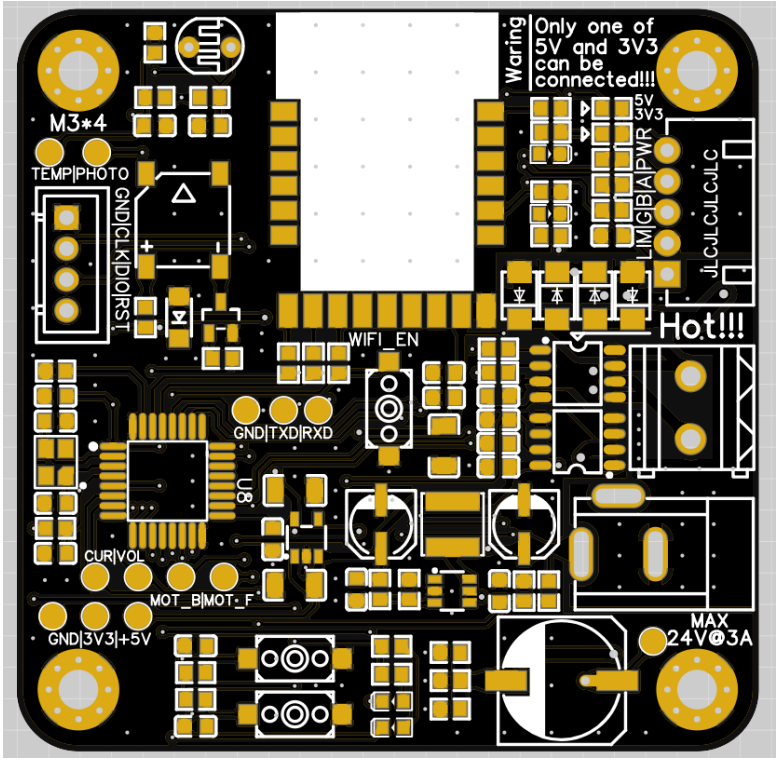

图 3.1 PCB 效果图

1. 电源&电机驱动

电源和电机驱动主要注意走线宽度、功率地和信号地分开、端子下面挖空防止接地短路等。

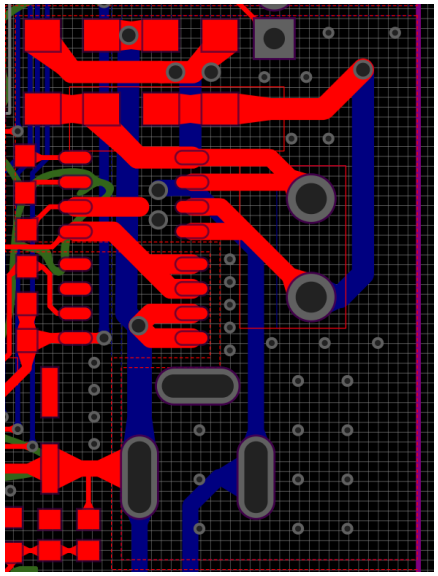

图 3.2 PCB-电源&电机驱动

左侧为 LDO，右侧为 DCDC。注意电感离 DCDC 芯片近一些，电感下面不要走信号线。（此图为错误示范）

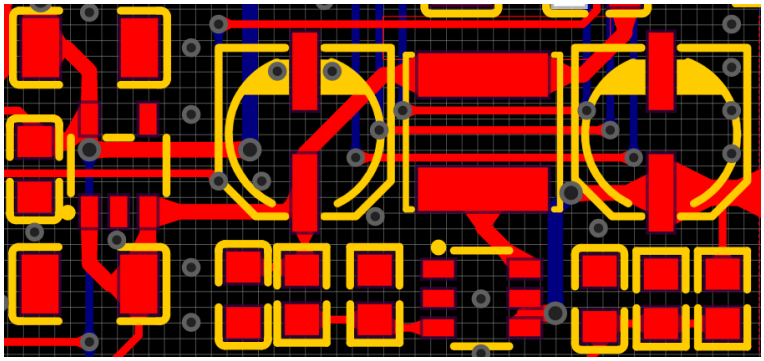

图 3.3 PCB-DCDC&LDO

2. MCU

MCU 主要注意晶振连线短一些，滤波电容靠近 MCU 电源引脚。

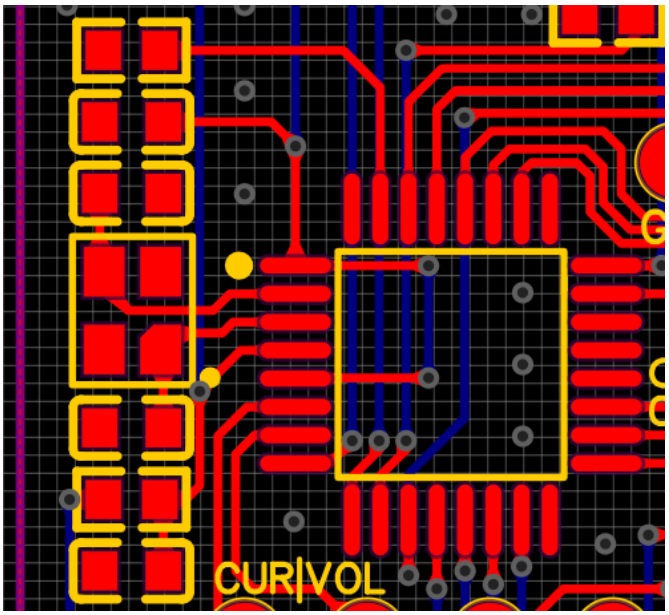

图 3.4 PCB-MCU

3. EMW3080

EMW3080 按照官方手册要求，1、2、24、25 脚不接，天线前方、左右留 16mm 净空区。搜索 EDA 中所有的封装都不完全满足官方手册要求，我自己画了一个。

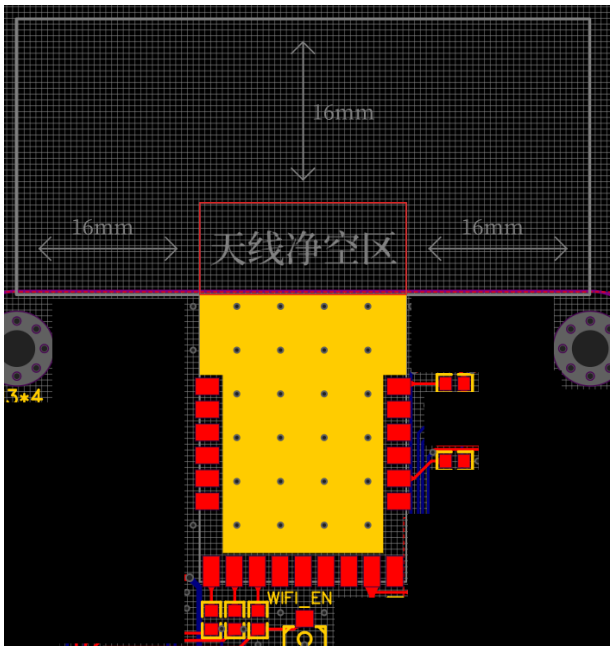

图 3.5 PCB-EMW3080

4. 外设

热敏避开发热区域；光敏避开 LED 区域；编码器接口放在电机端子旁边，方便连接；按键和 LED 指示灯放在板子下方，方便操作。

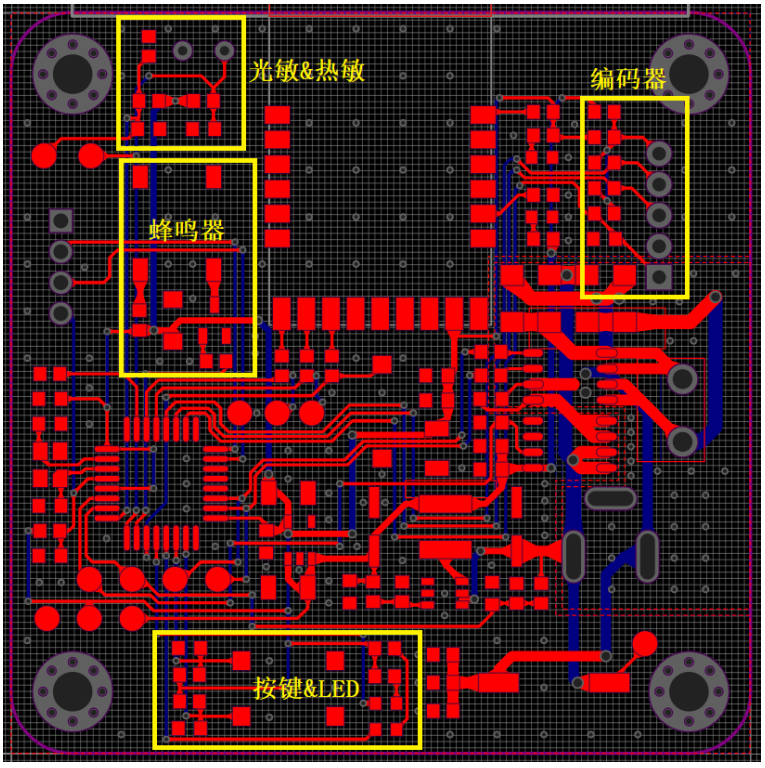

图 3.6 PCB-外设

四 焊接

拿到 PCB，准备焊接工具，开始焊接。

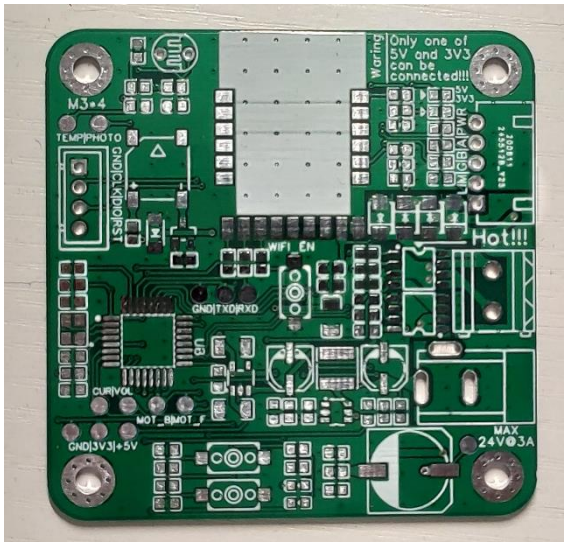

图 4.1 PCB

因为部分 PCB 中有一些封装对烙铁十分不友好。所以，上风枪。

图 4.2 工具

下面简述下焊接步骤和注意事项。

首先，准备 0.5mm 左右的焊锡丝、焊锡膏、助焊剂。清理烙铁头，烙铁温度 350℃。尖嘴镊子。提前释放身上静电。

第一步，焊接 DCDC 芯片及外围电路。电感封装问题，只能用风枪和焊锡膏焊。焊接完成后，焊接 DC005 接口。接入 12V 电源，使用万用表电压档测量 5V 测试点电压是否在 5.12V 左右。若电压不正确，需核对反馈电阻阻值是否正确。

第二步，焊接 LDO 电路。焊完后上电测试输出电压是否在 3.3V 左右。

第三步，电源没有问题后，焊接其他元件。顺序没有要求，一般由高度低的元件开始焊接。

单片机可以使用针管挤焊锡膏在焊盘上，摆好单片机，烙铁走一遍就能焊好，不连锡，贼好用。

电容焊盘也较短，需要焊锡膏和烙铁配合焊接。

PCB 注意有几个元件不能焊接。分别是：编码器电源 5V 处 0R 电阻，EMW3080 的 BOOT 接地电阻。

焊接完成后，效果如下。

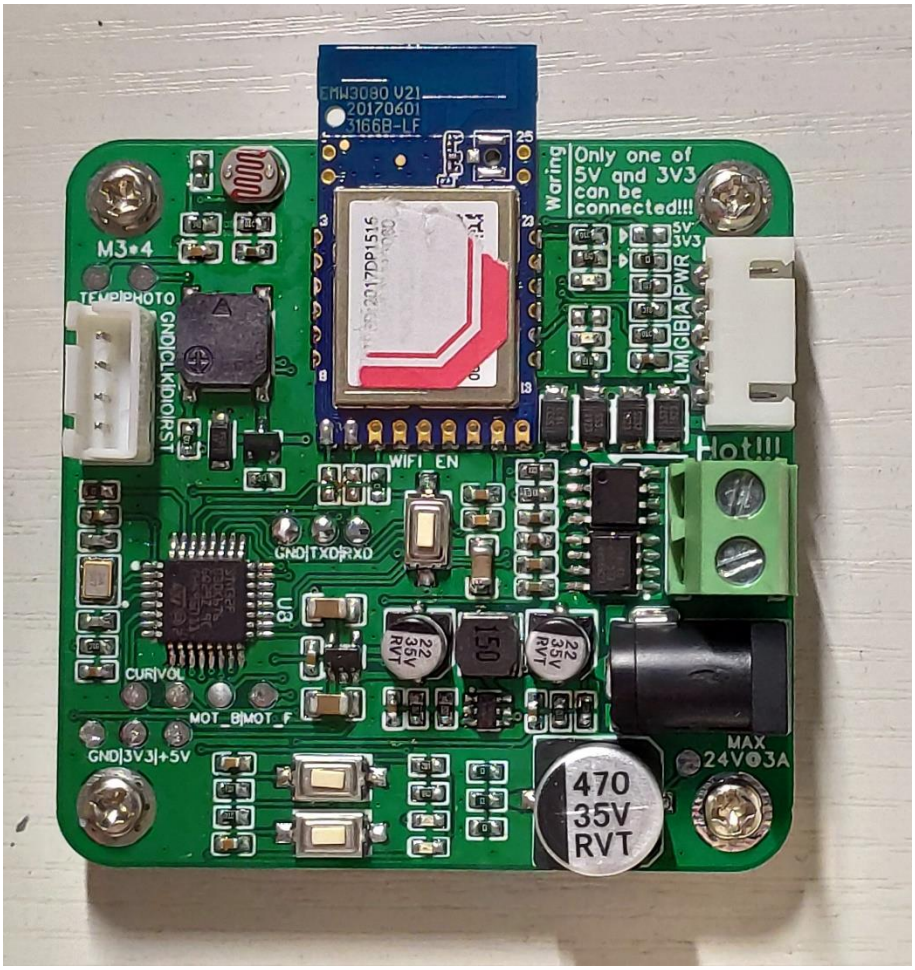

图 4.3 实物图

五 APP 设计

APP 设计采用阿里云物联网平台。具体过程可参考 B 站课程回放

一些属性参数如下：（部分功能未使用）

类型	名称	标识符	数据类型	数据定义	操作
属性	窗帘打开位置	curtainPosition	int32(整数型)	取值范围：0 ~ 100	编辑 删除

自定义功能 添加功能

类型	名称	标识符	数据类型	数据定义	操作
属性	快捷操作	QuickOperation	enum(枚举型)	枚举值: 0 - 全关 1 - 开一小半 2 - 开一半 3 - 开一大半 4 - 全开 5 - 自定义	编辑 删除
属性	光强	Lluminance	float(单精度浮点型)	取值范围：0 ~ 10000	编辑 删除
属性	电压	Voltage	float(单精度浮点型)	取值范围：8 ~ 30	编辑 删除
属性	电机模式	MotorMode	enum(枚举型)	枚举值: 0 - 开环 1 - 电流环 2 - 速度环 3 - 位置环	编辑 删除
属性	温度	Temp	float(单精度浮点型)	取值范围：-20 ~ 70	编辑 删除
属性	模式 推荐	Mode	enum(枚举型)	枚举值: 0 - 手动 1 - 自动 2 - 定时	编辑 删除
属性	限位状态	LimitStatus	enum(枚举型)	枚举值: 0 - 上限位 1 - 未限位 2 - 下限位	编辑 删除

图 5.1 属性参数

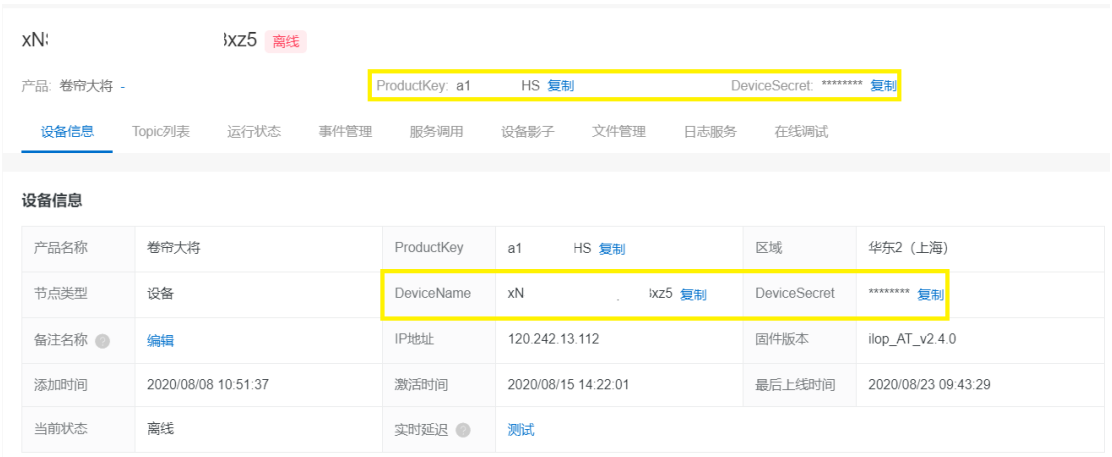

图 5.2 三元组信息

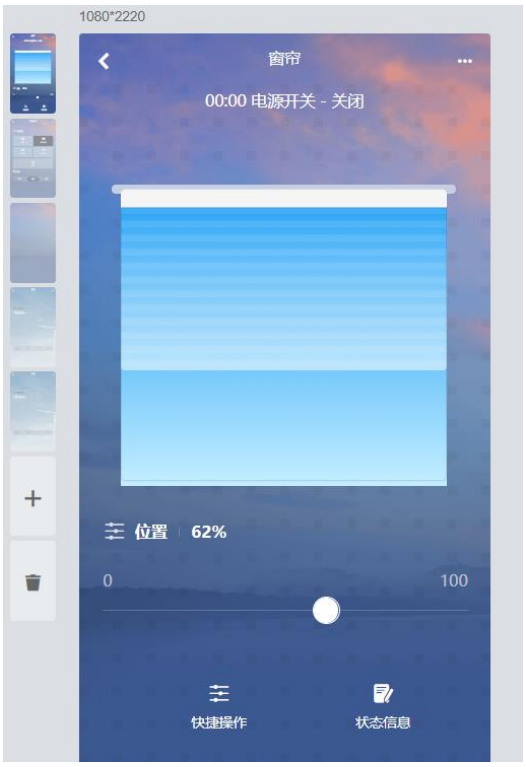

图 5.3 APP 主界面

六 程序调试

程序调试主要按功能块调试。调试日志按以下顺序依次进行：代码移植、上传数据、蜂鸣器驱动、ADC&DMA 采样、PID&参数整定、按键控制、任务调度器改写、配网模式。

1. 代码移植&上传数据

宋工给的例程是基于 51 单片机。51 程序和 STM32 不兼容，需要移植一些底层代码[1]。代码平台 CubeMX&HAL 库，MDK-ARM 5.27。

[1] 移植需要一定的软件操作基础，过程不完整。

移植前可通过串口先让 WIFI 上云，减小移植难度。

首先打开 CubeMX 软件，选择 STM32F030K6T6，使能外部晶振，使能 SWD 接口。勾选必要的外设。设置时钟 48MHz。填写工程名称，保存位置。选择使用的 IDE 为 MDK-ARM 5.27，勾选“为每一个外设生成.c/.h 文件”。点击“生成代码”。

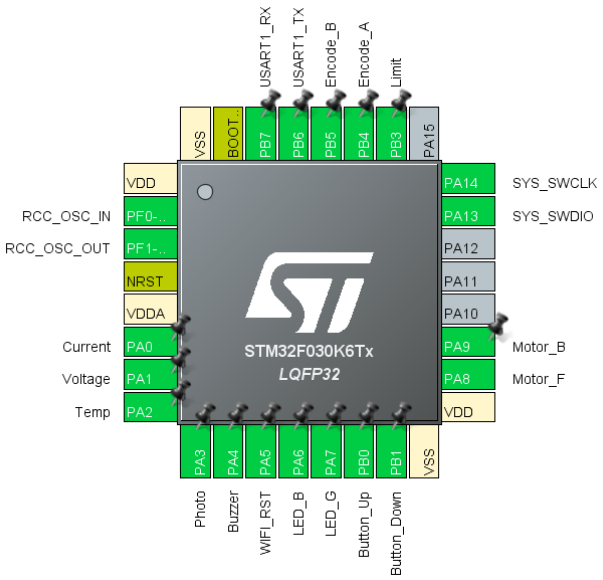

图 6.1 引脚配置

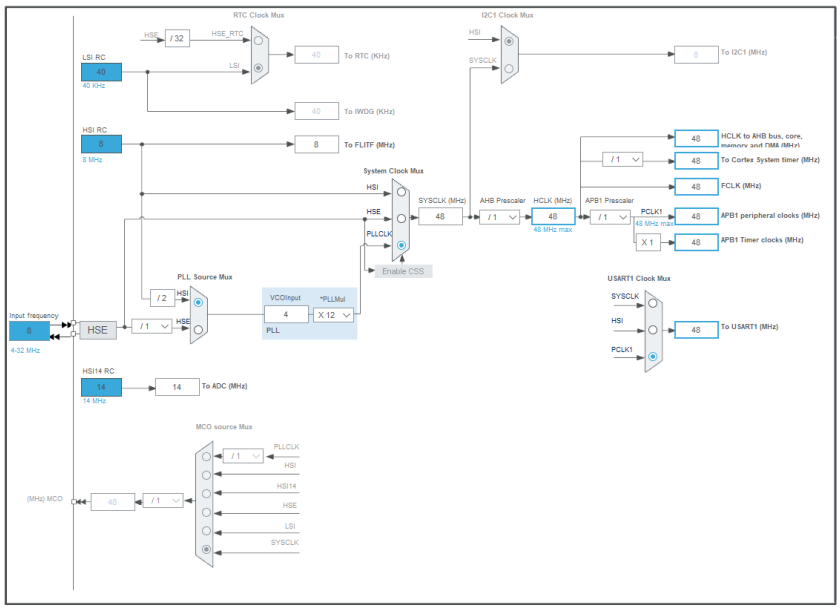

图 6.2 时钟树

代码生成后，点击“打开工程”，自动调用 Keil 软件。

首先，在 keil 左侧 Project 中添加一个文件夹，用来存放我们的.c 文件。双击该文件夹，加入 Code_User 文件夹下所有文件。

然后，点击魔术棒，点击“C/C++”选项卡，点击“Include Paths”后面三点，添加“Code_User”文件夹路径。

编译文件，不出意外，您会收获 error(s)。

下面开始解决这些错误。

1.大多数错误是由于#include <STC15.h>造成的，这是 51 的头文件，32 不使用，删除所有该语句。

2.Button.c 文件实现按键的短按、长按识别。该设备只需识别单按键和双按键，修改其中的 Button_Loop 函数。

3.DeviceName.c 文件操作三元组数据。我们把三元组数据移植到 usart.c 文件下，删除该文件。

4.DS18B20.c 文件用于读取温度传感器数据。我们温度采用热敏电阻配合 ADC，不涉及该文件，删除。

5. IAP_EEPROM.c 文件操作 EEPROM。32 没有，删除文件。

6.main.c 文件是设备主逻辑实现。复制内容到 CubeMX 创建的 main.c 文件中，删除该文件。

7.Mode.c 文件操作 LED 和继电器。我们没有继电器，LED 灯移植到 WIFI.c 文件下，删除该文件。

8.Relay.c 文件操作继电器，删除。

9.Timer0.c 文件实现任务调度器，需要修改定时器底层。删除定时器初始化结尾前代码，并添加“HAL_TIM_Base_Start_IT(&htim17);”启动定时器。

10. Uart_1.c 文件用于转发串口 2 数据到电脑。STM32F030K6 只有一个串口，删除该文件。

11.Uart2.c 文件主要和 WIFI 交换数据。修改底层代码，使用 STM32 的 DMA+空闲中断接收不定长数据。

12.WDT.c 文件实现看门狗。未使用，删除。

其他错误可双击编译结果跳转至指定位置。具体修改自行完成，不赘述。

首次移植可只删除和移植串口代码，其他无关紧要的稍后移植。

如果一切顺利，编译没有错误，可下载到 MCU。WIFI 能够上云。

```
122     //OS
123     Timer_0_Add_Fun(10,Button_Loop);            //按键检测底层业务
124     Timer_0_Add_Fun(5,Uart2_CheckMessageLoop);  //帧处理函数
125     Timer_0_Add_Fun(250,WIFI_LED_Loop);         //网络状态指示灯
126
127 //  Timer_0_Add_Fun(1010,Read_DS18B20_T1_Loop); //定时读取DS18B20
128 //  Timer_0_Add_Fun(1000,Mode_3LED_Loop);
129
130     Button_SetFun(0,'C',Main_EnterFactory);     //长按按键后调用 配网
131 //  Button_SetFun(0,'D',Mode_DianDong);         //点动按键后调用
132
133   /* USER CODE END 2 */
134
135   /* Infinite loop */
136   /* USER CODE BEGIN WHILE */
137   while (1)
138   {
139         Timer0_SYS_APP_LOOP();
140         Timer0_SYS_APP_LOOP_Message();
141         Timer0_SYS_APP_LOOP_Once();
142     /* USER CODE END WHILE */
143
144     /* USER CODE BEGIN 3 */
145   }
```

图 6.3 主程序部分代码

上传数据基于宋工代码结构。把所有上传项目独立，可配置不同项目不同上传频率。

```
148     Timer_0_Add_Fun(60 * 1000, WIFI_SubTemp);       //上报一次 温度 信息
149     Timer_0_Add_Fun(61 * 1000, WIFI_SubLlluminance);//上报一次 光强 信息
150     Timer_0_Add_Fun(62 * 1000, WIFI_SubVoltage);    //上报一次 电压 信息
151
152 //      Timer_0_Add_Fun(32 * 1000, WIFI_SubMotorMode);  //上报一次 电机运行模式 信息
153 //      Timer_0_Add_Fun(33 * 1000, WIFI_SubLimitStatus);//上报一次 限位状态 信息
154 //      Timer_0_Add_Fun(34 * 1000, WIFI_SubAction);     //上报一次 电机动作 信息
155 //      Timer_0_Add_Fun(35 * 1000, WIFI_SubMode);       //上报一次 窗帘模式 信息
156 //      Timer_0_Add_Fun(36 * 1000, WIFI_SubPosition);   //上报一次 窗帘位置 信息
157
```

图 6.4 定时上传数据

```
220  //*********************************
221  void WIFI_SubTemp(void)
222  {
223      WIFI_SubStation(0);
224  }
225
226  void WIFI_SubLlluminance(void)
227  {
228      WIFI_SubStation(1);
229  }
230
```

图 6.5 上传数据实现

```
155     //++++要上传的数据
156     switch(function)
157     {
158         case 0:
159             Len = JSON_Join_Key(JsonStr,"Temp", AdcGetTemp());                    //温度
160             break;
161
162         case 1:
163             Len = JSON_Join_Key(JsonStr,"Llluminance", AdcGetLlluminance());              //光强
164             break;
165
166         case 2:
167 //              Len = JSON_Join_Key(JsonStr,"MotorMode", MotorGetMode());   //电机运行模式
168             break;
169
170         case 3:
171             Len = JSON_Join_Key(JsonStr,"LimitStatus", 1);              //限位信息
172             break;
173
174         //4 删除
175
176         case 5:
177 //              Len = JSON_Join_Key(JsonStr,"Mode", MotorGetMode());    //模式
178             break;
179
180         case 6:
181             Len = JSON_Join_Key(JsonStr,"curtainPosition", MotorGetPosition());    //位置
182             break;
183
184         case 7:
185             Len = JSON_Join_Key(JsonStr,"Voltage", AdcGetVoltage());    //电压
186             break;
187
188         default:
189             return;
190     }
```

图 6.6 上传数据 Json 部分

2. ADC&DMA 采样

ADC 采用 DMA 多通道不连续采集。使用二维数组缓存数据，每次获取 ADC 测量值时均采样 10 次求平均后上传。

```
12  void AdcInit(void)
13  {
14      //校准ADC
15      HAL_ADCEx_Calibration_Start(&hadc);
16      //开DMA
17      AdcValuePosition = 0;
18  //  HAL_ADC_Start_DMA(&hadc, (uint32_t *)AdcValue[AdcValuePosition], 5);
19  }
20
21  void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
22  {
23      HAL_ADC_Stop_DMA(&hadc);
24      if(++AdcValuePosition >= 10)
25          AdcValuePosition = 0;
26      HAL_ADC_Start_DMA(&hadc, (uint32_t *)AdcValue[AdcValuePosition], 5);
27  }
28
```

图 6.7 ADC 初始化&回调函数

温度和光强公式根据元件手册给出的温度-阻值、光强-阻值曲线拟合而

成，辅以修正因子修正。

```
        //电流，中点2.5v，增益100mV/A
        AdcActualValue[i][0] = ( 2.5 - ( AdcValue[i][0] * PowerVoltage / 4096 )) / 0.1;
        //电压，1/11
        AdcActualValue[i][1] = ( AdcValue[i][1] * PowerVoltage / 4096 ) * 11;
        //温度
        //-10~50℃:  y = -33.186 * x + 80.268  (R^2 = 0.998)  //修正：-5
        AdcActualValue[i][2] = -33.186 * ((float)AdcValue[i][2] * PowerVoltage / 4096) + 80.268 - 5;
        //光强：[<400lx] y = 30.24 * x^(-3.54) (R^2 = 0.9653); [>400lx] y = -16691 * x + 8262.9 (R^2 = 0.9764);
        if(AdcValue[i][3] > (0.47 * 4096 / PowerVoltage))
            AdcActualValue[i][3] = 30.24 * pow(( (float)AdcValue[i][3] * PowerVoltage / 4096 ), -3.54);
        else
            AdcActualValue[i][3] = -16691 * ((float)AdcValue[i][3] * PowerVoltage / 4096 ) + 8262.9;
```

图 6.8 电压&温度&光强换算公式

3. 蜂鸣器驱动

蜂鸣器通过 TIM14-1 通道驱动。内置 25 个环形缓存区数组实现蜂鸣器音调、音量、延时功能。

```
void BuzzerInterrupt(void)
{
    if(BuzzerTime > 0)  //延时中...
    {
        BuzzerTime -= BuzzerBeat;
    }
    else                //切换
    {
        if(BuzzerCount == 0)    //OVER
        {
            BuzzerWorking = 0;
            HAL_TIM_PWM_Stop(&htim14, TIM_CHANNEL_1);
            HAL_TIM_Base_Stop_IT(&htim14);
            HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, GPIO_PIN_RESET);
        }
        else            //NEXT
        {
            BuzzerReload = 1000000.0 / BuzzerParameter[BuzzerPositionOut][0];
            __HAL_TIM_SET_AUTORELOAD(&htim14, (uint16_t)BuzzerReload - 1);
            __HAL_TIM_SET_COMPARE(&htim14, TIM_CHANNEL_1,
                (uint16_t)(BuzzerReload * BuzzerParameter[BuzzerPositionOut][1] * 0.01 * 0.9));

            BuzzerBeat = BuzzerReload / 1000;       //ms
            BuzzerTime = BuzzerParameter[BuzzerPositionOut][2];
            HAL_TIM_PWM_Start(&htim14, TIM_CHANNEL_1);
            HAL_TIM_Base_Start_IT(&htim14);

            BuzzerPositionOut = (BuzzerPositionOut + 1) % BuzzerParameterMax;
            BuzzerWorking = 1;
            BuzzerCount--;
        }
    }
}
```

图 6.9 TIM 中断实现

```
uint8_t BuzzerSetParameter(uint16_t frequncy, uint8_t voltage, uint16_t time)
{
    BuzzerParameter[BuzzerPositionIn][0] = frequncy;
    BuzzerParameter[BuzzerPositionIn][1] = voltage;
    BuzzerParameter[BuzzerPositionIn][2] = time;

    BuzzerPositionIn = (BuzzerPositionIn + 1) % BuzzerParameterMax;

    if((++BuzzerCount == 1) && (!BuzzerWorking))      //从停止状态启动
    {
        BuzzerTime = -1.0;
        BuzzerInterrupt();
    }

    if(BuzzerCount == BuzzerParameterMax)
        return 0;
    else
        return 1;
}
```

图 6.10 蜂鸣器参数设置

4. PID 及参数整定

PID 使用增量式和位置式，分别用于速度环和位置环。

PID 参数因电机而异，需要自行耐心调整。

```
void MotorIncrementPID(struct PID *pid, int16_t pidInput)
{
    pid->PidInput = pidInput;
    //Pid
    pid->PidEt = pid->PidSetPoint - pid->PidInput;
    pid->PidOutput += pid->PidKp * (pid->PidEt - pid->PidLastErr) \
                    + pid->PidKi * pid->PidEt \
                    + pid->PidKd * (pid->PidEt - 2 * pid->PidLastErr + pid->PidLastTwoErr);
    //Pid限幅
    pid->PidOutput = pid->PidOutput > pid->PidLimitUp ? pid->PidLimitUp : pid->PidOutput;
    pid->PidOutput = pid->PidOutput < pid->PidLimitDown ? pid->PidLimitDown : pid->PidOutput;
    //覆写
    pid->PidLastTwoErr = pid->PidLastErr;
    pid->PidLastErr = pid->PidEt;
}
```

图 6.11 增量式 PID 实现过程

```
void MotorPositionPID(struct PID *pid, int32_t pidInput)
{
    pid->PidInput = pidInput;
    //Pid
    pid->PidEt = pid->PidSetPoint - pid->PidInput;
    pid->PidEtSum += pid->PidEt;
    pid->PidOutput = pid->PidKp * pid->PidEt + pid->PidKi * pid->PidEtSum
                   + pid->PidKd * (pid->PidEt - pid->PidLastErr);
    //Pid限幅
    pid->PidOutput = pid->PidOutput > pid->PidLimitUp ? pid->PidLimitUp : pid->PidOutput;
    pid->PidOutput = pid->PidOutput < pid->PidLimitDown ? pid->PidLimitDown : pid->PidOutput;
    //覆写
    pid->PidLastErr = pid->PidEt;
}
```

图 6.12 位置式 PID 实现过程

因为使用的编码器电机阻尼大，大约 5V 电压才能启动，为了避免电机从停止状态退出过程时间过长，在 PID 输出和电机间添加 MotorCurve 函数。保证低速时呈对数变化，高速时线性变化。该函数已在我的遥控车项目验证，效果非常棒。

```
float MotorCurve(float inPut)
{
    float outPut = 0;
    outPut = (inPut < 0 ? -1 : 1);
    inPut = inPut > 265 ? 265 : inPut;
    inPut = inPut < -265 ? -265 : inPut;
    //输出曲线。类似对数曲线。
    //分界点 15:266.2; 0.002425; -0.155; use
    //可快速从驻车状态退出，且低速时刹车距离变短。
    //f(x) = 256.7[227.6, 285.8] * exp(0.002453[0.001927, 0.002979] * x)
    //             + (-256.7[-297.8, -215.6]) * exp(-0.09653[-0.1635, -0.02957] * x);
    outPut *= 266.2 * (exp(0.002425 * (inPut < 0 ? - inPut : inPut))
                                    - exp(-0.155 * (inPut < 0 ? - inPut : inPut))) + 0.5;
    return outPut;
}
```

图 6.13 电机曲线函数实现

```
//************ Speed Loop ******************************//
void MotorSpeedInit(void)
{
    //PID: 5,  0.6,  1
    MotorInit(&PidSpeed);
    MotorSetPidParameter(&PidSpeed, 5, 0.6, 1);
    PidSpeed.PidLimitUp = 150;
    PidSpeed.PidLimitDown = -150;
}
```

图 6.14 速度环参数

从速度环曲线可以看出，调整时间大约 0.3s。过程存在一点超调，正常现象，可以保证更快速的调整。

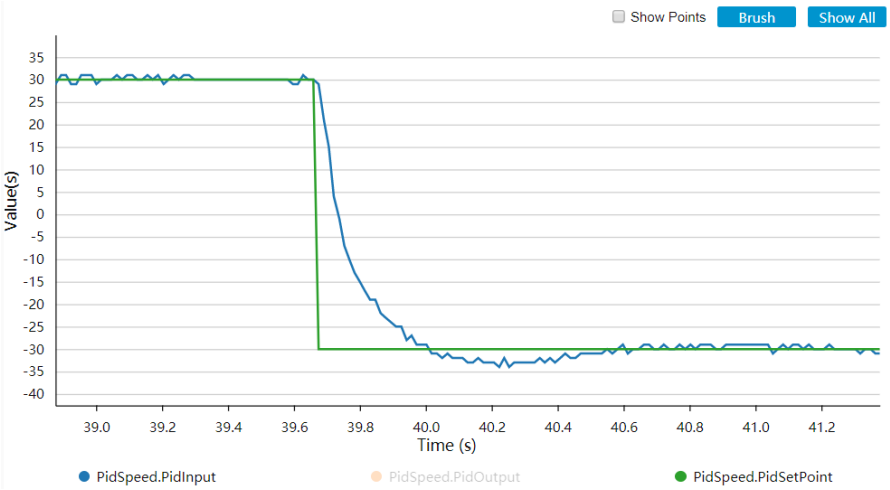

图 6.15 速度环调试曲线

```
183  //*************** Position Loop ************************//
184  void MotorPositionInit(void)
185  {
186      //0.6, 0, 13
187      MotorInit(&PidPosition);
188      MotorSetPidParameter(&PidPosition, 0.6, 0, 13);      //微调该参数
189      PidPosition.PidLimitUp = 200;
190      PidPosition.PidLimitDown = -200;
191  }
```

图 6.16 位置环参数

位置环中间线性段受限于电机输出限幅。后段波动是施加扰动和释放扰动造成。

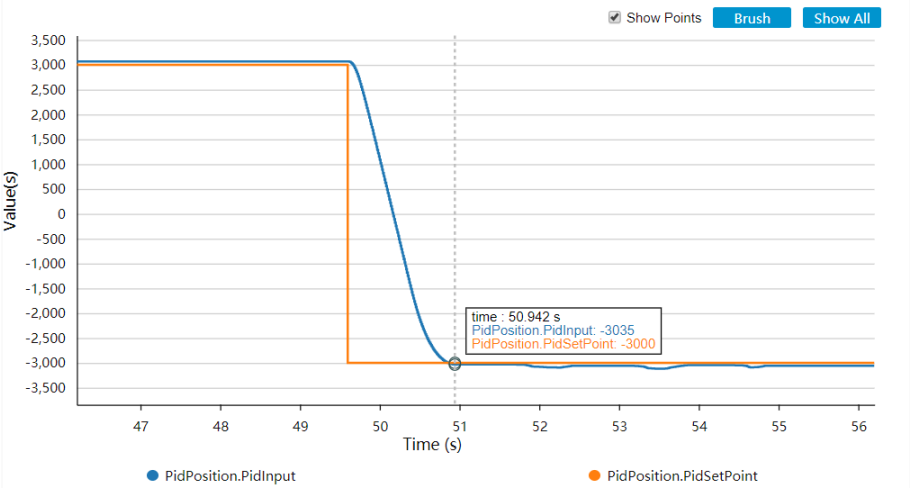

图 6.17 位置环曲线

5. 按键控制

按键部分删除了原来的 Button_Loop 实现函数。添加了 User_Button 函数，用于识别单按键按下，双按键按下，按键松开等动作。并向系统发布消息。

```c
void UserButton(void)
{
    if( ((Button_ReadIO(0)) && (Button_ReadIO(1))) \
        && (ButtonTwoTime || Button_Timer[0] || Button_Timer[1]) )
    {
        Timer0_SendMessage('S');          //松手
    }

    if((!Button_ReadIO(0)) && (!Button_ReadIO(1)))      //同时按下
    {
        ButtonTwoTime++;
        Button_Timer[0] = 0;
        Button_Timer[1] = 0;
    }
    else
    {
        ButtonTwoTime = 0;

        if(Button_ReadIO(0) == 0)
            Button_Timer[0]++;
        else
            Button_Timer[0] = 0;

        if(Button_ReadIO(1) == 0)
            Button_Timer[1]++;
        else
            Button_Timer[1] = 0;
    }

    if(ButtonTwoTime > 65500)
        ButtonTwoTime = 65500;
    if(Button_Timer[0] > 65500)
        Button_Timer[0] = 65500;
    if(Button_Timer[1] > 65500)

    if(ButtonTwoTime > Button_L_Time)
    {
        Timer0_SendMessage('F');          //恢复出场设置
        return;
    }
    if(ButtonTwoTime == 0)              //按键未同时按下
    {
        if((Button_Timer[0] > Button_G_Time) && (Button_Timer[0] < Button_L_Time))
        {
            Timer0_SendMessage('U');          //上
            Button_Timer[0] = 65500;
            return;
        }
        if((Button_Timer[1] > Button_G_Time) && (Button_Timer[1] < Button_L_Time))
        {
            Timer0_SendMessage('D');          //下
            Button_Timer[1] = 65500;
            return;
        }
    }
}
```

图 6.18 按键部分实现代码

为了实现一个消息可以对应多个功能函数，我们修改了 Message 循环部分代码。修改后的代码，可以实现：按下上键，电机速度环模式上行，同时蜂鸣器“滴”提示音。下键同理。

```c
//******** Message ***************//
    Timer0_Add_MessageFun('A', DistributionNetwork);    //上次AP配网不成功，开机会自动进“配网模式”
    Timer0_Add_MessageFun('F', DistributionNetwork);    //上下按键同时长按2S  配网
    Timer0_Add_MessageFun('U', MotorUp);                //上键  上行
    Timer0_Add_MessageFun('D', MotorDown);              //下键  下行
    Timer0_Add_MessageFun('S', LetGo);                  //松手检测
    Timer0_Add_MessageFun('S', WIFI_SubPosition);       //回传位置

//******** Buzzer ***************//
    Timer0_Add_MessageFun('C', Buzzer_DJI);             //连接网络
    Timer0_Add_MessageFun('U', Buzzer_Di);              //上键  上行
    Timer0_Add_MessageFun('D', Buzzer_Di);              //下键  下行

//********  ***************//
    Timer0_Add_MessageFun('C', ReStartUpload); //开机 连接网络成功
```

图 6.19 按键发送消息-系统接收部分

6. 任务/消息调度器改写

为了实现一些功能，我们在宋工的任务调度器基础上进行了修改。

对 Flag 部分重新规划。

```
struct
{
    unsigned char Flag[Timer_0_List_Count];             //0:空；1:运行；2:暂停；10:暂停所有
    void (*Fun_Point_List[Timer_0_List_Count])(void);
    unsigned long Counter[Timer_0_List_Count];
    unsigned long Timer[Timer_0_List_Count];
}Timer0_Struct;
```

图 6.20 任务-结构体

修改后的任务调度器添加了：删除、暂停、恢复、暂停所有、恢复所有功能。

```
//任务
unsigned char Timer_0_Add_Fun(unsigned long Time,void (*Fun)(void));          //添加 一个
unsigned char Timer_0_Add_Fun_Once(unsigned long Time,void (*Fun)(void));     //添加 一个 一次
unsigned char Timer_0_Del_Fun(void (*Fun)(void));          //删除 一个
unsigned char Timer_0_Pause_Fun(void (*Fun)(void));        //暂停 一个
unsigned char Timer_0_ReStart_Fun(void (*Fun)(void));      //恢复 一个
unsigned char Timer_0_Pause_All(void);                     //暂停 所有
unsigned char Timer_0_ReStart_All(void);                   //恢复 通过"暂停所有"功能暂停的功能
```

图 6.21 任务调度器添加函数部分

为了实现一些功能，我们添加了 Flag，并对其重新规划。

```
struct
{
    unsigned char Flag[Timer_0_List_Count];             //0:空；1:运行；2:暂停；10:暂停所有
    unsigned char MessageQueue[Timer_0_List_Count];
    unsigned char MessageList[Timer_0_List_Count];
    void (*MessageFun_Point_List[Timer_0_List_Count])(void);
}Timer0_Message_Struct;
```

图 6.22 消息-结构体

修改后的消息调度器添加了：删除、暂停、恢复、暂停所有、恢复所有功能。

```
//消息
unsigned char Timer0_Add_MessageFun(unsigned char Message,void (*Fun)(void));    //添加
unsigned char Timer0_Del_MessageFun(void (*Fun)(void));          //删除
unsigned char Timer0_Pause_MessageFun(void (*Fun)(void));        //暂停
unsigned char Timer0_ReStart_MessageFun(void (*Fun)(void));      //恢复
unsigned char Timer0_Pause_MessageAll(void);                     //暂停 所有
unsigned char Timer0_ReStart_MessageAll(void);                   //恢复 所有
```

图 6.23 消息调度器添加函数部分

7. 配网模式

同时长按上下两按键 2S，进入 AP 热点配网模式。

```
375 //配网模式
376 void DistributionNetwork(void)
377 {
378     MotorPause();
379     Timer_0_Pause_All();          //暂停所有任务
380     Timer0_Pause_MessageAll();    //暂停所有消息
381     Buzzer_BiPu();                //奏乐
382 
383     //开始配网流程
384     Timer_0_Add_Fun_Once(1000, WIFI_CloseRTE);    //关闭回显
385     Timer_0_Add_Fun_Once(1500, WIFI_ResetAuthor);//解除绑定关系,
386     Timer_0_Add_Fun_Once(2000, WIFI_SendAT);      //AT
387     Timer_0_Add_Fun_Once(2500, WIFI_SetILOP);     //设置三元组
388     Timer_0_Add_Fun_Once(3000, WIFI_StartILOP);   //开启ILOP
389 //    Timer_0_Add_Fun_Once(5000, WIFI_StartAWS);    //路由器配网
390     Timer_0_Add_Fun_Once(5000, WIFI_StartAP);     //热点配网
391 
392     //其他操作
393     Timer_0_Add_Fun(100, WIFI_LED_Loop);             //WIFI指示灯快闪 5Hz
394     Timer_0_Add_Fun(100, Mode_LED_Loop);             //快闪 5Hz
395     Timer_0_ReStart_Fun(Uart2_CheckMessageLoop);     //帧处理函数
396 
397     Timer0_Add_MessageFun('C', NetworkConnected);    //连接成功
398     Timer0_Add_MessageFun('E', Buzzer_DiLong);       //Error
399     Timer0_Add_MessageFun('A', Buzzer_DiDiDi);       //AP已开启
400 }
```

图 6.24 配网模式实现代码

配网成功，则执行 NetworkConnected 函数。删除配网过程创建的任务和消息，启用电机，恢复配网前所有任务和消息。最后奏乐，返回原来任务。无需重启设备，节省了重启&联网时间。

这点和宋工代码有较大区别。宋工实现过程是：进入配网过程，重新初始化所有任务、消息。创建配网需要的任务和消息，配网成功后调用软件重启指令，重新进入正常工作模式。这样会多一次重启 WIFI 上云过程。

```
358 //配网成功 已连接
359 void NetworkConnected(void)
360 {
361     //删除 配网 过程创建的所有任务
362     Timer_0_Del_Fun(WIFI_LED_Loop);              //WIFI指示灯快闪 5Hz
363     Timer_0_Del_Fun(Mode_LED_Loop);              //快闪 5Hz
364     Timer0_Del_MessageFun(NetworkConnected);     //连接成功
365     Timer0_Del_MessageFun(Buzzer_DiLong);        //Error
366     Timer0_Del_MessageFun(Buzzer_DiDiDi);        //AP已开启
367 
368     MotorReStart();
369 
370     Timer_0_ReStart_All();                       //恢复 配网前所有任务
371     Timer0_ReStart_MessageAll();                 //恢复 配网前所有消息
372     Buzzer_DoToXi();                             //奏乐
373 }
```

图 6.25 配网成功

结论

该项目参考宋工代码框架，删除了一些未使用的功能，修改了一些底层功能，增加了一些定制化功能。

设备基本实现了电动窗帘的本地/云端控制功能，能够按一定频率上传设备运行状态。当设备发生故障时，具有一定的处理能力。有着较为友好的交互方式。成本控制得当。

程序设计采用任务/消息调度器模式，可方便地添加、删减功能而不影响其他功能运行。

由于时间、精力、能力有限，设备还存在着诸多问题待修复，已知问题罗列如下，如您有时间、有精力、有能力，可尝试修复。

1. 软硬限位未实现，窗帘存在超限位运行损坏风险。窗帘超限位后 APP 无法显示实际位置；

2. PID 参数设定需要手动完成。

3. 当外接电源为开关电源时，电机反转会触发开关电源过压保护。加入 SS54 二极管防倒灌可解决。但电机反转时设备电压会被拉高，当使用 24V 电源时可能会导致电机驱动芯片过压保护。

4. 光敏电阻和热敏电阻测量数值不准确。

致 谢

感谢立创 EDA 开设活动，提供一个 ~~白嫖~~ 学习的机会。

感谢宋工源码，学习了任务/消息调度器、环形缓存区、JSON 字符串比较等知识，受益匪浅。

感谢客编：481978A 大佬上传的文件，解决了令人头秃的配网流程。