浙江大学

本科实验报告

课程名称：	电子电路系统综合实验
姓　　名：	胡康鑫　　施佳阳
学　　院：	信息与电子工程学院
系：	信息与电子工程学院
专　　业：	信息工程
学　　号：	3210105586　3210106341
指导教师：	楼东武 崔宁 綦成林

2023 年 7 月 15 日

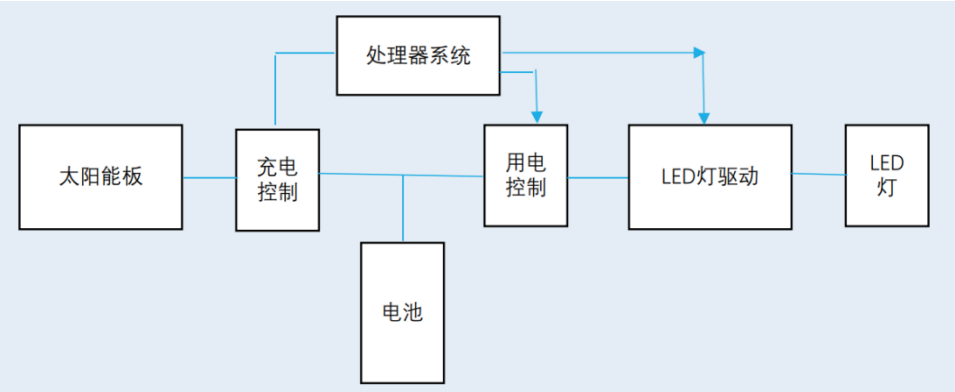
专业：信息工程
姓名：胡康鑫 施佳阳
学号：3210105586
3210106341
日期：2023.7.15
地点：东四-320

浙江大学实验报告

课程名称：电子电路系统综合实验 指导老师：楼东武 崔宁 綦成林 成绩：
实验名称：太阳能景观灯控制器设计 实验类型：设计型实验 同组学生姓名：胡康鑫 施佳阳

一、实验目的

1、设计并制作太阳能景观灯控制器，太阳能板开路电压 12V，最大电流 0.5A。
2、电池采用两个锂电子串接的电池组。
3、照明采用 LED 灯（1W LED 灯）。
原理示意图如下图所示：



二、实验任务与要求

1、基本要求：

- 实现对电池的充电控制，电池电压达到 8.2V 时停止充电，电池电压低于 7.8V 时继续充电。
- 实现对 LED 灯的用电控制，当电池电压低于 6V 时，停止供电，电池电压高于 6.4V 时继续供电。
- 对 LED 进行恒流驱动，电流 300mA，控制精度小于 15mA。

2、发挥部分：

- 系统工作状态有显示（OLED 屏）。
- 当电池电压达到 8.2V 时，采用 PWM 方式控制进行恒压限流充电，当电流低于 100mA 时停止充电。
- 通过设定实现 LED 的亮度调节，调节步进小于 50mA。
- 实现景观灯的光控功能，有白天灭，夜间点亮。
- 其它，如实现时空等。

三、实验原理

1、恒流充电法

恒流充电法是用调整充电装置输出电压或改变与蓄电池串联电阻的方法，保持充电电流强度不变的充电方法。控制方法简单，但由于电池的可接受电流能力是随着充电过程的进行而逐渐下降的，到充电后期，充电电流多用于电解水，产生气体，使出气过甚，因此，常选用阶段充电法。

2、恒压充电法

充电电源的电压在全部充电时间里保持恒定的数值，随着蓄电池端电压的逐渐升高，电流逐渐减少。与恒流充电法相比，其充电过程更接近于最佳充电曲线。用恒定电压快速充电，由于充电初期蓄电池电动势较低，充电电流很大，随着充电的进行，电流将逐渐减少，因此，只需简易控制系统。

3、阶段充电法

此方法包括二阶段充电法和三阶段充电法

①二阶段法采用恒电流和恒电压相结合的快速充电方法，首先，以恒电流充电至预定的电压值，然后，改为恒电压完成剩余的充电。一般两阶段之间的转换电压就是第二阶段的恒电压。

②三阶段充电法在充电开始和结束时采用恒电流充电，中间用恒电压充电。当电流衰减到预定值时，由第二阶段转换到第三阶段。这种方法可以将出气量减到最少，但作为一种快速充电方法使用，受到一定的限制。

4、快速充电法

①脉冲充电方式首先是用脉冲电流对电池充电，然后让电池停充一段时间，如此循环。能够减轻蓄电池的内压，使下一轮的恒流充电能够更加顺利地进行，使蓄电池可以吸收更多的电量。

②2REFLEXTM 快速充电法，它主要面对的充电对象是镍镉电池。由于它采用了新型的充电方法，解决了镍镉电池的记忆效应，因此，大大降低了蓄电池的快速充电的时间。

③变电流间歇充电法，这种充电方法建立在恒流充电和脉冲充电的基础上。其特点是将恒流充电段改为限压变电流间歇充电段。充电前期的各段采用变电流间歇充电的方法，保证加大充电电流，获得绝大部分充电量。充电后期采用定电压充电段，获得过充电量，将电池恢复至完全充电态。

④变电压间歇充电法，在变电流间歇充电法的基础上又有人提出了变电压间歇充电法。与变电流间歇充电方法不同之处在于第一阶段的不是间歇恒流，而是间歇恒压。在每个恒电压充电阶段，由于是恒压充电，充电电流自然按照指数规律下降，符合电池电流可接受率随着充电的进行逐渐下降的特点。

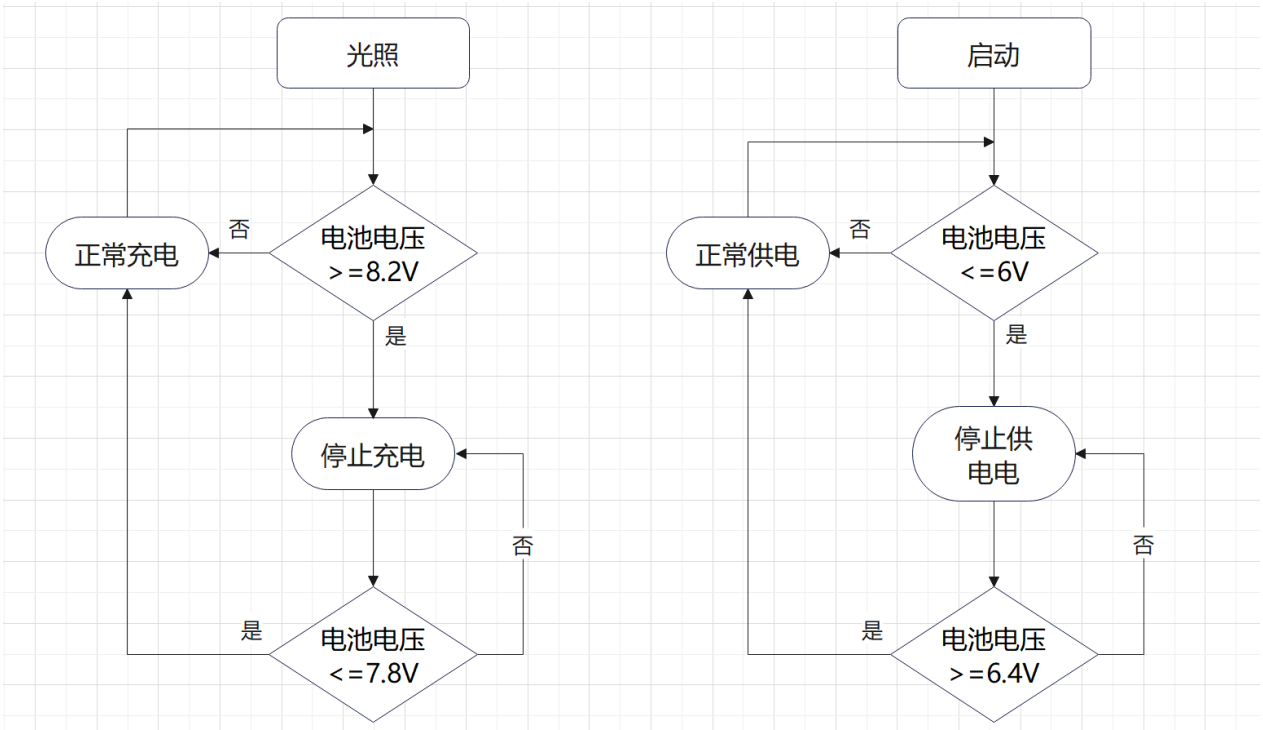
⑤变电压变电流波浪式间歇正负零脉冲快速充电法，合脉冲充电法、ReflexTM 快速充电法、变电流间歇充电法及变电压间歇充电法的优点，变电压变电流波浪式正负零脉冲间歇快速充电法得到发展应用。脉冲充电法充电电路的控制一般有两种：

1）脉冲电流的幅值可变；

2）脉冲电流幅值固定不变。

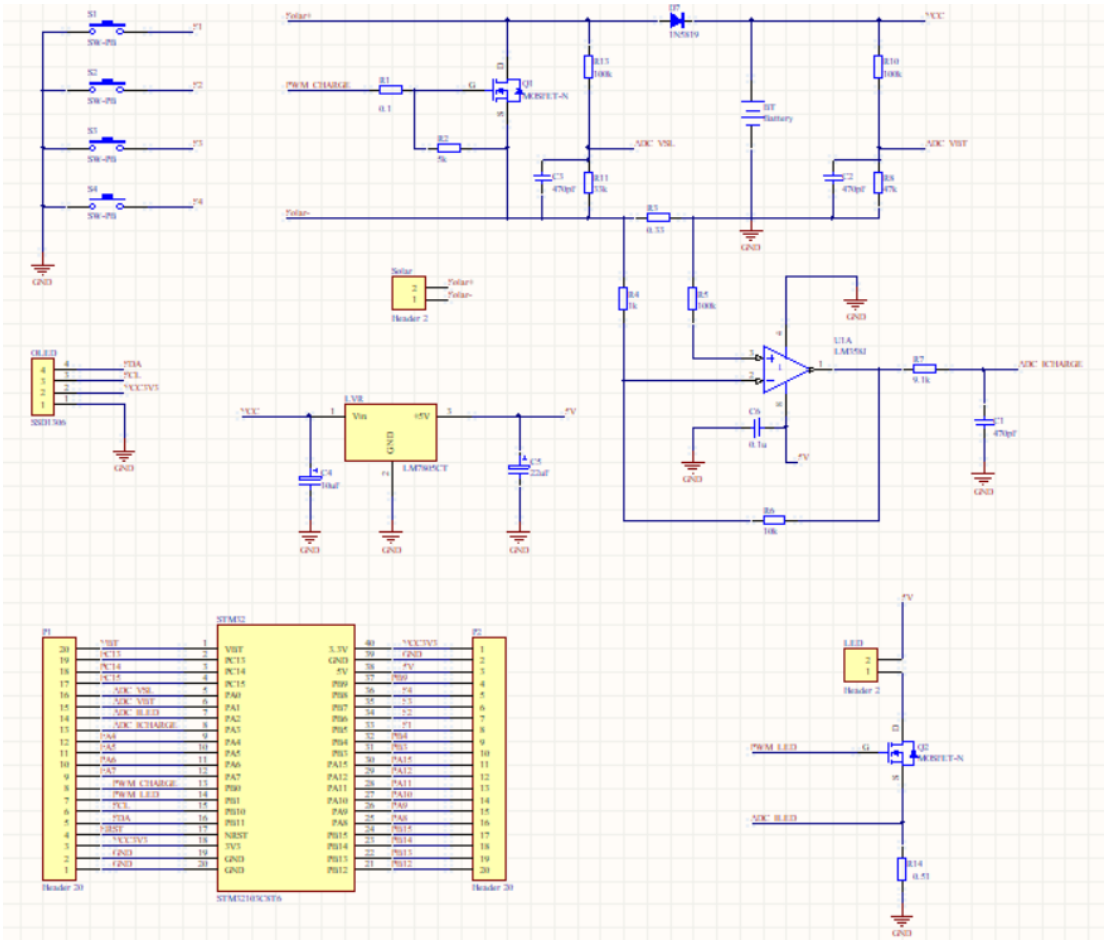
脉冲电流幅值和 PWM 信号的频率均固定，PWM 占空比可调，在此基础上加入间歇停充阶段，能够在较短的时间内充进更多的电量，提高蓄电池的充电接受能力。

根据实验要求，可画出基本的原理框图如下：

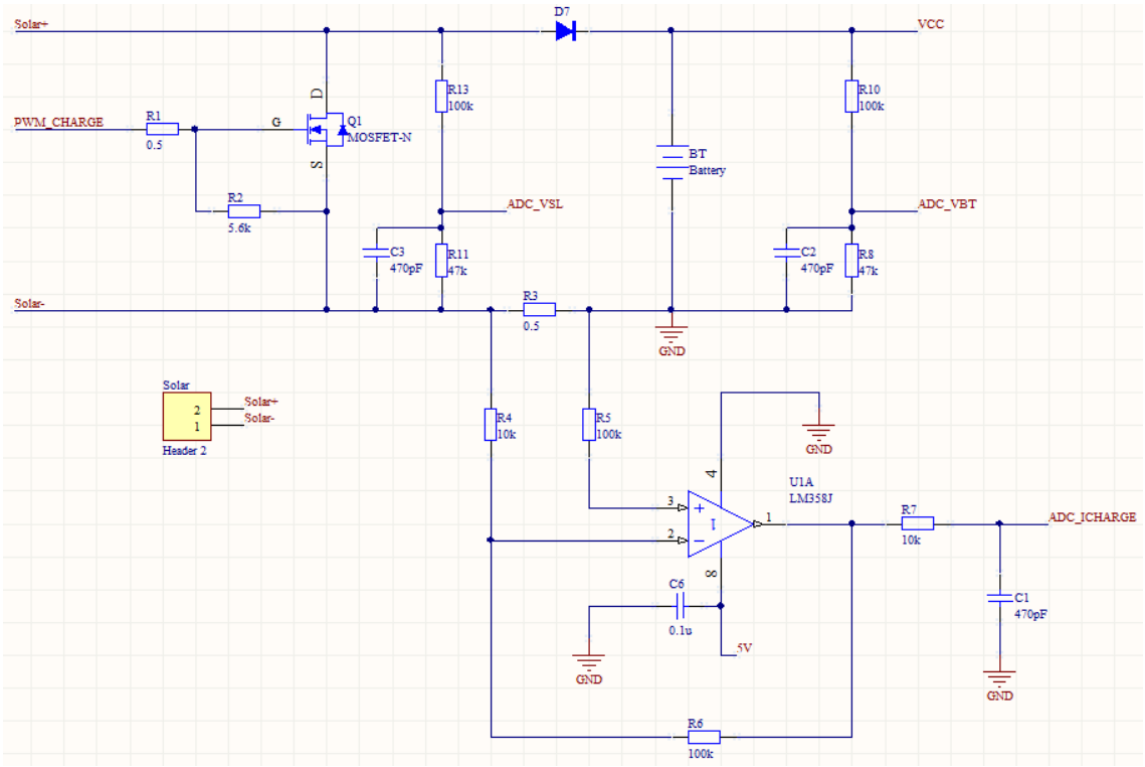


四、实验电路设计

电路总体设计：



1、充电电路模块

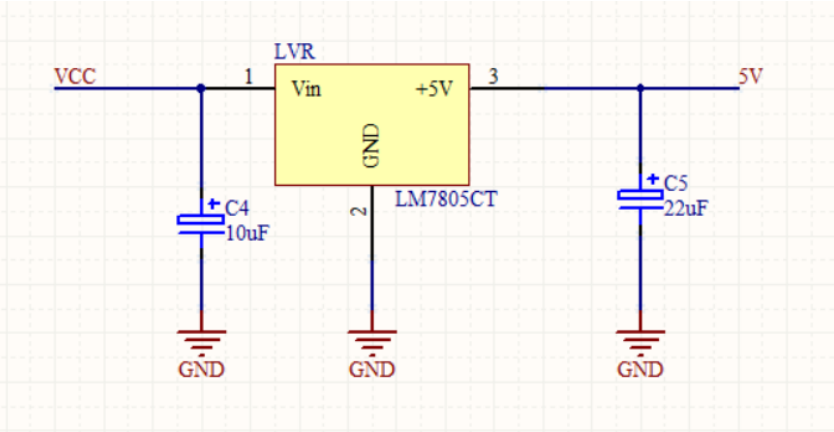


Solar+和 Solar-分别是太阳能板的阳极电压和阴极电压，通过电阻分压，引出 ADC_VSL 信号，ADC_VSL 的大小能够反映出太阳能板的电压，从而间接反映外部的光强，当光强较大时，代表外部为白天，代表无需光照，此时 ADC_VSL 较大，通过内部代码控制，使 LED 熄灭；当光强较小时，代表外部为黑夜，代表需要光照，此时 ADC_VSL 较小，通过内部代码控制，使 LED 亮起。从而实现景观灯的光控功能。

VCC 是蓄电池电压，通过电阻分压，引出 ADC_VBT 信号，（间接）检测蓄电池的电压，当检测到蓄电池电压≥7.8V 时，通过 PWM_CHARGE 对充电进行控制，此时减小充电时间所占的占空比，使得有效充电电流减小，进入限流充电环节；当蓄电池电压到达 8.2V 时，进入恒压充电环节，通过实时调节 PWM_CHARGE，从而调节有效充电电流，使蓄电池电压基本维持在 8.2V，实现蓄电池的恒压限流充电。充电电流流过 R3，使用运算放大器对 R3 两端的压降进行放大，然后引出 ADC_ICHARGE 检测充电电流，当充电电流低至 100mA 时，说明蓄电池基本充满，此时调节 PWM_CHARGE 为 100%，从而停止充电。

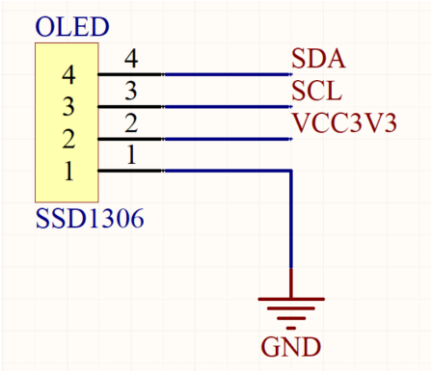
在此电路中，R1 和 R2 起到保护和分压的作用，因此 R1 选取小电阻 0.5Ω，而 R2 选取阻值相对较大的电阻 5.6kΩ。R8、R10、R11、R13 均是起到分压的作用，由于 VCC 正常情况下位于 7.4V~8.4V 之间，而 ADC 采样仅能对 3.3V 以下的正电压进行准确的采样，所以需要对 VCC 进行分压然后再采样，此处选取的分压比约为 47/147。R3 是采样电阻，其两端的电压分别为 0 和负值，故需要使用运放对其进行差分放大并转换为正电压再进行 ADC 采样，即 ADC_ICHARGE，此处使用 R4、R6、R7 进行放大，放大倍数为 110/10。二极管 D7 具有正向导通的性质，并且选取了能够承受千伏电压的器件型号，能够防止 DS 短路时（即不充电时）蓄电池发生短路引发危险，也能够避免反充的发生。电路中的电容均为保护作用。

2、稳压电路模块



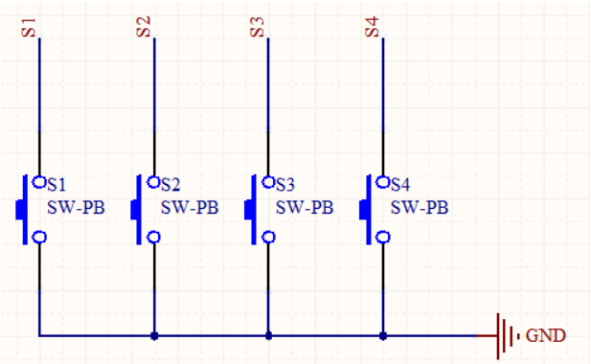
通过一个线性稳压芯片 LM7805 稳压电路对蓄电池电压 VCC 进行稳压，输出 5V 的电压，为运算放大器、STM32F103C8T6 最小系统板、LED 灯提供稳定电压。

3、OLED 显示模块



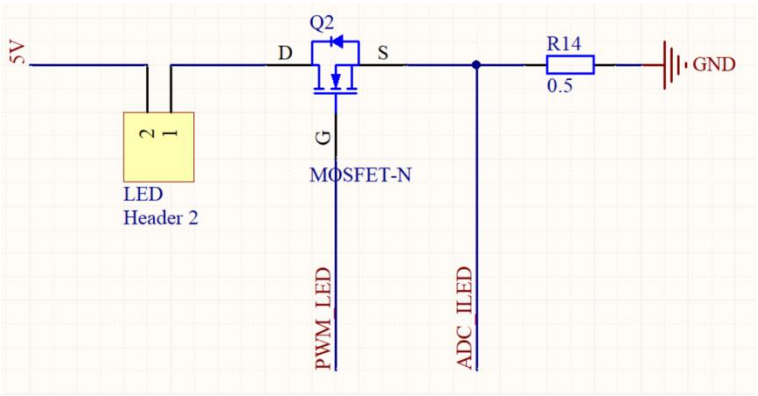
采用 SSD1306 进行 OLED 显示，由 STM32F103C8T6 最小系统板提供 3.3V 电压、GND 以及 SCL 和 SDA 信号。

4、按钮模块



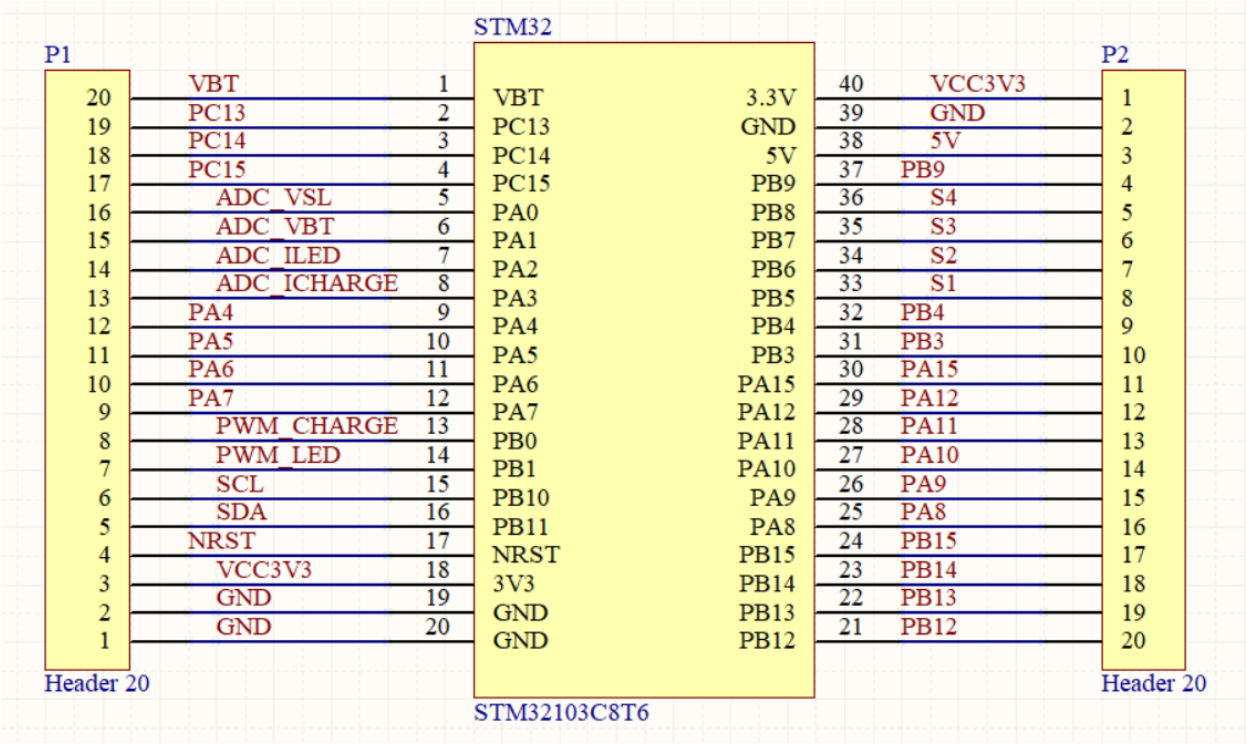
放置四个按钮，采用上拉式开关，可以实现模式切换、LED 上调亮度和下调亮度、切换待机等功能。

5、恒流 LED 模块



LED 模块使用 5V 进行供电，添加一个 0.5Ω 的小电阻作为采样电阻，引出 ADC_ILED 信号对采样电阻 R14 两端的压降进行采样，用 PWM_LED 控制导通时间的占空比，从而调节流过 LED 的有效电流并实时采样，直至采样出的电压稳定 0.15V（即对应 300mA 的电流值），从而实现 LED 的恒流控制。

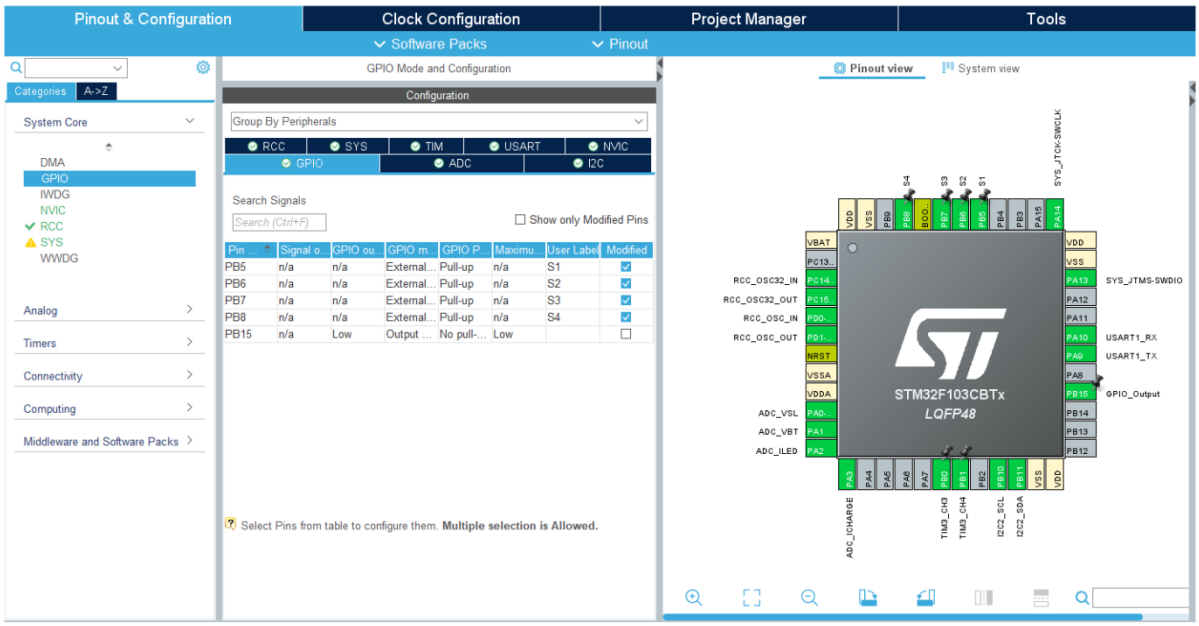
6、核心控制模块

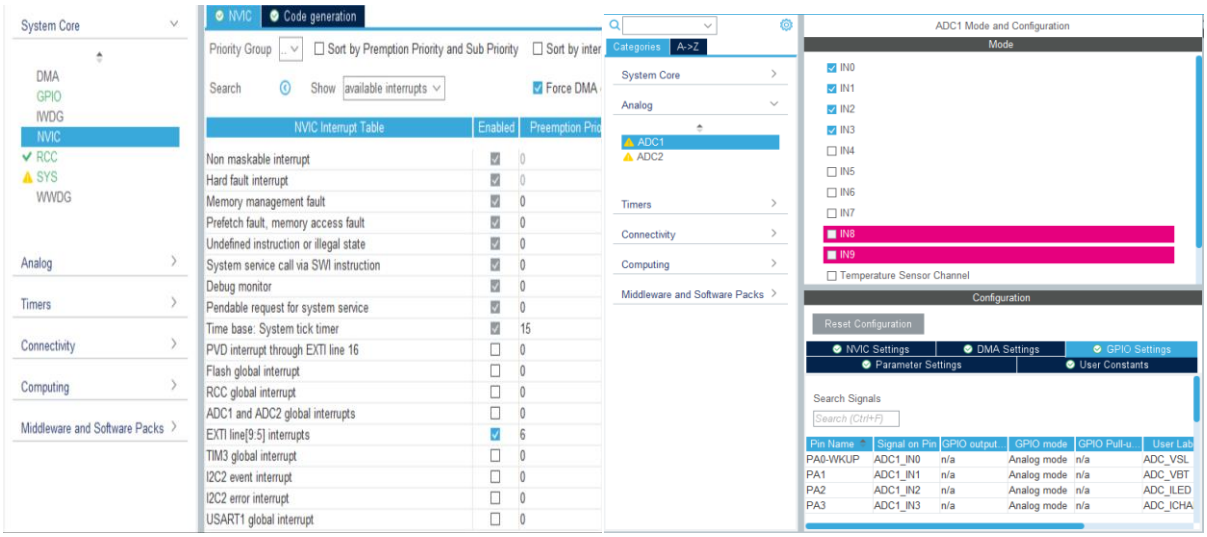


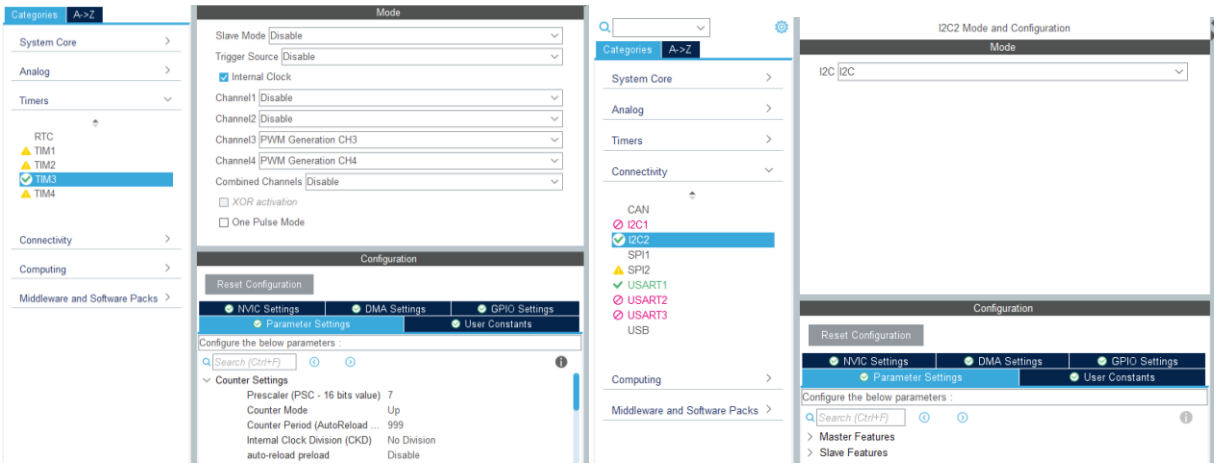
使用 STM32F103C8T6 最小系统板作为核心控制模块，其中各引脚分别连接到对应模块的对应引脚上，实现对应的引脚和模块功能。与此同时引出两排 20 脚的排针，方便后续进行调试和添加额外功能。

五、实验代码设计

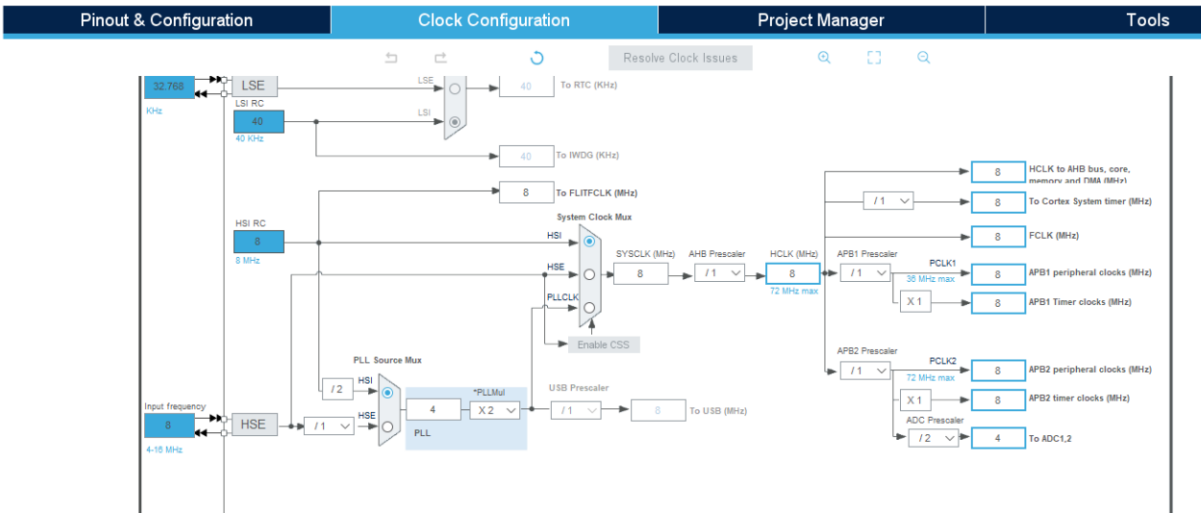
1、STM32CubeMX 引脚配置：





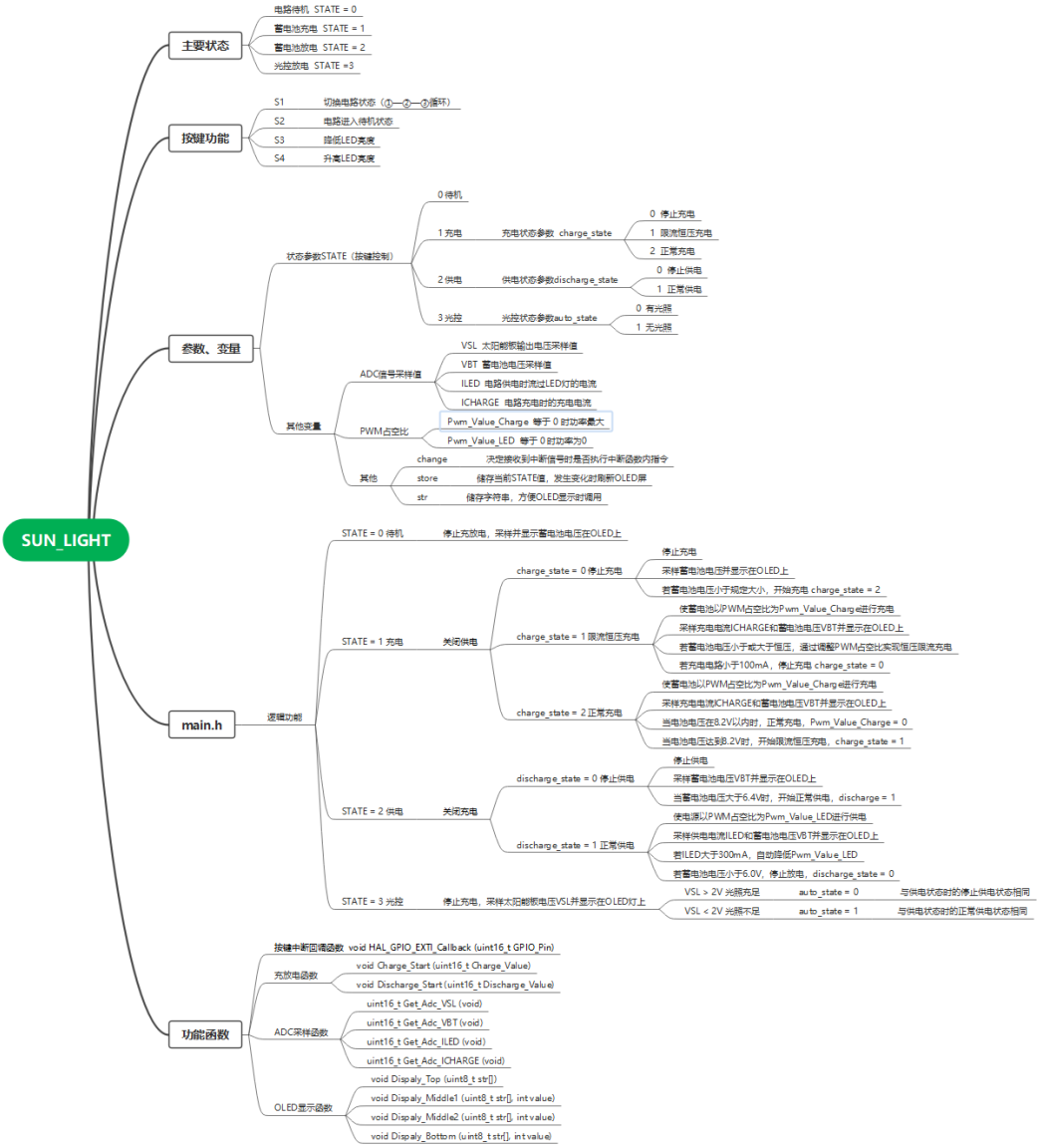


时钟配置（采用默认 8MHz）：

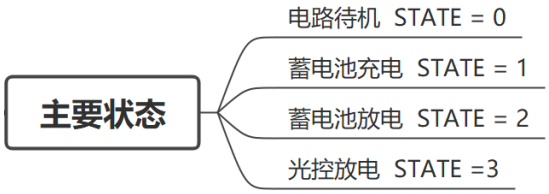


2、KEIL5 程序设计

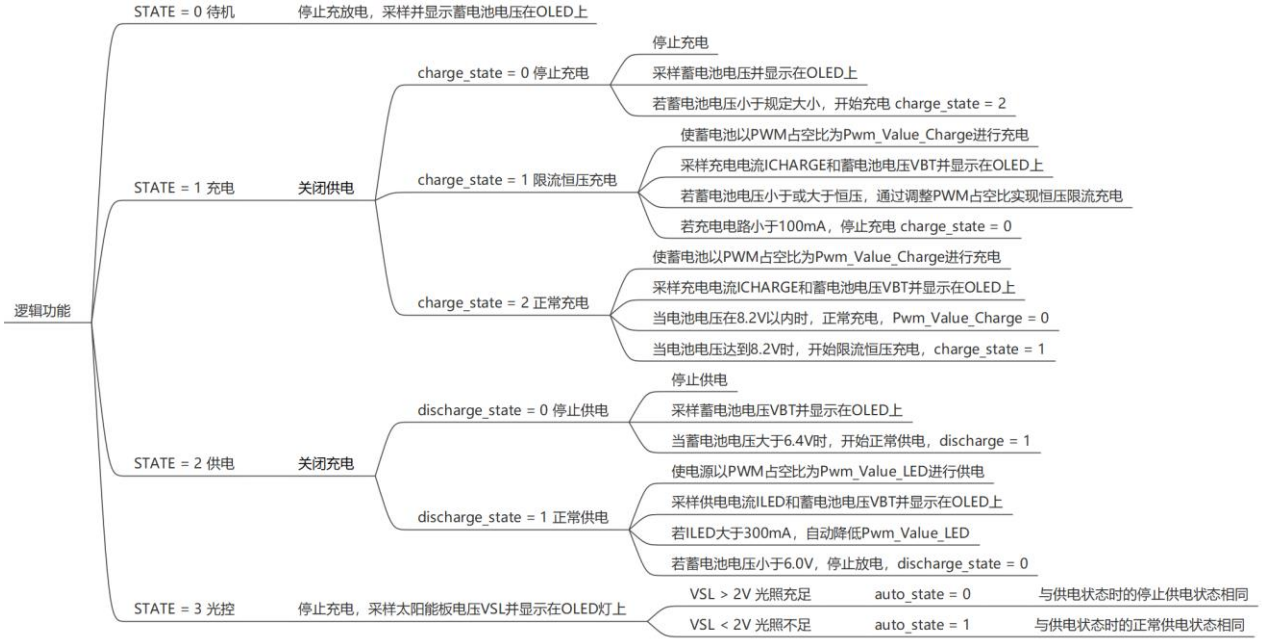
总体设计框架：



①设计思路



为了方便设计电路，我们将电路分为了四个状态，并通过按键来对电路的工作状态进行切换。



处于待机状态时（STATE＝0），电路会同时停止放电和充电，同时通过 ADC 采样蓄电池电压并显示在 OLED 显示屏上，等待下一次指令。

处于充电状态时（STATE＝1），对蓄电池进行充电并停止供电，程序会在每次循环时通过 ADC 采样监测蓄电池的电压 VBT 和充电电流 ICHARGE，根据采样的数据来控制充电的功率和方式。

处于供电状态时（STATE＝2），对景观灯进行供电并停止充电，程序会在每次循环时通过 ADC 采样监测流过景观灯的电流 ILED 和蓄电池电压 VBT，根据采样的数据来控制供电的功率和方式。

处于光控状态时，（STATE ＝ 3），程序会在每次循环时通过 ADC 采样太阳能板的电压 VSL 和蓄电池电压 VBT，根据采样的数据来判断是否要将景观灯亮起。

我们设计了四个按键来对电路进行控制。

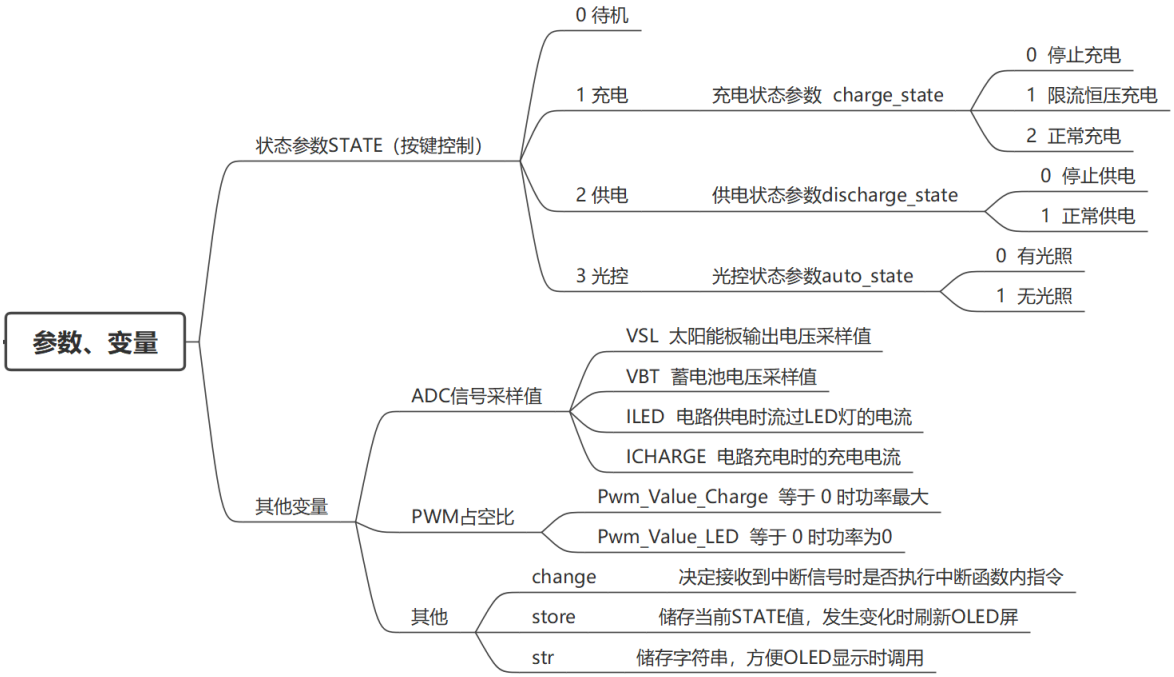
按键 S1 的功能是切换电路状态，当电路处于待机状态（STATE＝0）时，为了符合一般情况下的使用习惯，当我们按下 S1 按键时，会先切换为供电状态（STATE＝2），即 LED 灯亮起，之后再按下 S1 会使电路在三个状态（放电、供电、光控）之间按循环切换。

按键 S2 的功能是使电路进入待机状态。

按键 S3 和按键 S4 的功能是调节景观灯的亮度，即流过景观的的电流。

②具体代码

参数与变量设置

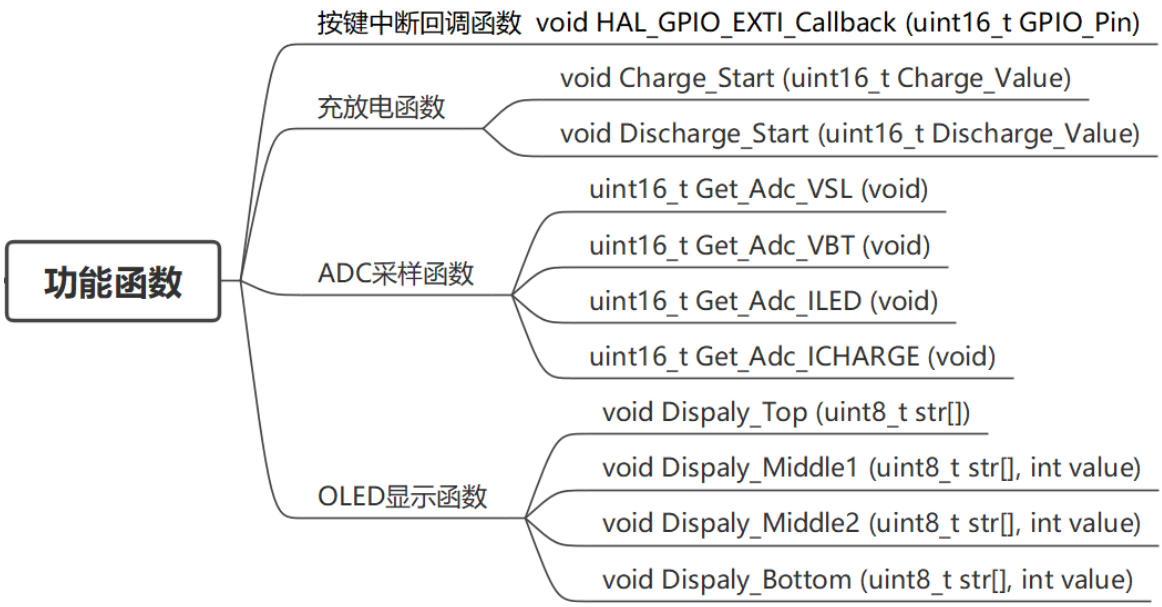



```
/* USER CODE BEGIN PTD */
uint8_t str01[] = {"VBT:"};
uint8_t str02[] = {"ILED:"};
uint8_t str03[] = {"ICHARGE:"};
uint8_t str04[] = {"VSL:"};
uint8_t str2[] = {"mV"};
uint8_t str3[] = {"mA"};
uint8_t str4[] = {"."};
uint8_t str10[] = {"STANDBY"};
uint8_t str11[] = {"CHARGING"};
uint8_t str12[] = {"DISCHARGING"};
uint8_t str13[] = {"STOP DISCHARGE"};
uint8_t str14[] = {"STOP CHARGE"};
uint8_t str15[] = {"XIAN LIU"};
uint8_t str16[] = {"HENG YA"};
uint8_t str18[] = {"LIGHT"};
uint8_t str19[] = {"NO LIGHT"};

/* USER CODE END PTD */
```

```
/* USER CODE BEGIN PV */
uint8_t STATE = 0 ;
int VSL;
int VBT;
int ILED;
int ICHARGE;
static uint16_t Pwm_Value_Charge = 1000;
static uint16_t Pwm_Value_LED = 0;
uint8_t key;
uint8_t discharge_state = 0;     //放电状态参数
uint8_t charge_state = 0;        //充电状态参数
uint8_t auto_state = 0;          //光控状态参数
int change = 1;
int store = 0;
/* USER CODE END PV */
```

功能函数



为了简化代码，我们设计了四种功能函数。

按键中断回调函数

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(change == 1){
      if(GPIO_Pin & S1_Pin){
          if(STATE == 2) STATE = 3;
          else if(STATE == 3)STATE = 1;
          else STATE = 2;
        }
      if(GPIO_Pin & S2_Pin){
          STATE = 0;
        }
      if(GPIO_Pin & S3_Pin){
          if(Pwm_Value_LED>=50) Pwm_Value_LED -=20;
              else Pwm_Value_LED = 0;
        }
      if(GPIO_Pin & S4_Pin){
          if(Pwm_Value_LED<=950) Pwm_Value_LED +=20;
              else Pwm_Value_LED = 1000;
        }
    change = 0;
  }
}
```

通过按键回调函数 HAL_GPIO_EXTI_Callback 实现四个按键控制功能。同时，为了解决下降沿抖动带来的问题，我们设置了参数 change，每次执行完中断回调函数内的指令后，将 change 置为 0，只有当 change = 1 时才进行指令。

```
    if(change == 0){
       HAL_Delay(20);
        change = 1;
    }
    if(store != STATE){
       OLED_Clear();
        store = STATE;
    }
```

（这段代码放在每次循环的开头）

充放电函数

```
void Charge_Start(uint16_t Charge_Value) //对蓄电池按照给定的PWM占空比充电
{
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, Charge_Value);
}

void Discharge_Start(uint16_t Discharge_Value)  //对景观灯按照给定的PWM占空比供电
{
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, Discharge_Value);
}
```

通过__HAL_TIM_SET_COMPARE 函数发送 PWM 信号的占空比。根据原理图，当充电时的 PWM 占空比越大时，充电的功率越低。而供电时的 PWM 占空比越大，供电电流就越大。

ADC 采样函数

```
uint16_t Get_Adc_VSL(void){
    ADC_ChannelConfTypeDef sConfig;
    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
      if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
    VSL = Get_Adc_Average(sConfig.Channel,10);
    VSL = VSL*147*3300/4095/47;
    return VSL;
}
```

```
uint16_t Get_Adc_VBT(void){
    ADC_ChannelConfTypeDef sConfig;
    sConfig.Channel = ADC_CHANNEL_1;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
      if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
    VBT = Get_Adc_Average(sConfig.Channel,10);
    VBT = VBT*3300*147/4095/47 + 200;
    return VBT;
}
```

```
uint16_t Get_Adc_ILED(void){
    ADC_ChannelConfTypeDef sConfig;
    sConfig.Channel = ADC_CHANNEL_2;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
      if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
    ILED = Get_Adc(sConfig.Channel);
    ILED = ILED*3300*2/4095;
    ILED = ILED*Pwm_Value_LED/1000;
    return ILED;
}
```

```
uint16_t Get_Adc_ICHARGE(void){
    ADC_ChannelConfTypeDef sConfig;
    sConfig.Channel = ADC_CHANNEL_3;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
      if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
    ICHARGE = Get_Adc(sConfig.Channel);
    ICHARGE = ICHARGE*3300*2/4095/11 + 20;
    ICHARGE = ICHARGE*(1000-Pwm_Value_Charge)/1000;
    return ICHARGE;
}
```

由于我们使用到了多路 ADC 采样，所以在采样时不能直接使用 Adc_Get 函数，而要先配置好每次采样时的通道。

采样获取的数据并不能直接作为判断和显示的标准，我们需要先转换为相应的电压值或电流值，再根据实际测量矫正误差。

ADC 采样时获得的是瞬时值，所以在监测充电电流 ICHARGE 和景观灯电流 ILED 时，我们需要将采样值乘 PWM 占空比得到的才是有效值。

OLED 显示函数

```
void Dispaly_Top(uint8_t str[]){
     OLED_ShowString(4,0,str,16);
}
void Dispaly_Middle1(uint8_t str[], int value){
     OLED_ShowString(4,10,str,16);
     OLED_ShowNum(66,10,value/1000,1,16);
     OLED_ShowNum(75,10,(value-value/1000*1000)/100,1,16);
     OLED_ShowNum(84,10,value%100/10,1,16);
     OLED_ShowNum(93,10,value%1000,1,16);
     OLED_ShowString(105,10,str2,16);
}

void Dispaly_Bottom(uint8_t str[], int value){
     OLED_ShowString(4,14,str,16);
     OLED_ShowNum(66,14,value/1000,1,16);
     OLED_ShowNum(75,14,(value-value/1000*1000)/100,1,16);
     OLED_ShowNum(84,14,value%100/10,1,16);
     OLED_ShowNum(93,14,value%1000,1,16);
     OLED_ShowString(105,14,str2,16);
}
```

```
void Dispaly_Middle2(uint8_t str[], int value){
     OLED_ShowString(4,12,str,16);
     OLED_ShowNum(66,12,value/1000,1,16);
     OLED_ShowNum(75,12,(value-value/1000*1000)/100,1,16);
     OLED_ShowNum(84,12,value%100/10,1,16);
     OLED_ShowNum(93,12,value%1000,1,16);
     if(value == VSL) OLED_ShowString(105,12,str2,16);
     else OLED_ShowString(105,12,str3,16);
}
```

根据输入的字符串和数据在 OLED 显示屏上显示。

main 函数

通过 switch 函数来进行状态的判断

各状态对应的代码如下：

待机状态

```
switch (STATE)
  {
      case 0:                          //进入待机状态，停止充电和放电
           Charge_Start(1000);
           Discharge_Start(0);
           Dispaly_Top(str10);         //停止充放电，并显示状态

           Get_Adc_VBT();
           Dispaly_Bottom(str01, VBT);
                                       //采样蓄电池电压，显示
      break;
```

充电状态

```
case 1:
     Discharge_Start(0);       //进入充电状态，停止充电

     Charge_Start(0);
     HAL_Delay(100);
     Get_Adc_VSL();
     Charge_Start(1000);
     Dispaly_Middle1(str04, VSL);
                               //短暂开启充电，采样太阳能板输出电压，并显示
```

停止充电状态

```
if(charge_state == 0){
   Charge_Start(1000);
   Dispaly_Top(str14);  //停止充电，显示状态
   Get_Adc_VBT();
   Dispaly_Bottom(str01, VBT); //采样蓄电池电压，显示
   if(VBT <= 7800){
      charge_state = 2;
      OLED_Clear();
   }                     //若检测到蓄电池电压小于7.8V,重新开始充电
}
```

恒压限流充电状态

```
if(charge_state == 1){
   Charge_Start(1000);
   Dispaly_Top(str16);
   Get_Adc_VBT();
   Dispaly_Bottom(str01, VBT);
                  //进入限流充电状态，采样并显示蓄电池电压和电路状态
      if(VBT <= 8200) Pwm_Value_Charge -= 10;
      else Pwm_Value_Charge += 10;
                  //若蓄电池电压大于设定值(8.2V),降低充电效率,反之升高
   Charge_Start(Pwm_Value_Charge);
   Get_Adc_ICHARGE();
   Dispaly_Middle2(str03, ICHARGE);
                  //以当前的PWM占空比进行充电，并采样充电电流
   if((ICHARGE <= 100)||(Pwm_Value_Charge == 1000)) {
       charge_state = 0;
       OLED_Clear();
   }              //若充电电流小于100mA,停止充电
}
```

正常充电状态

```
if(charge_state == 2){
   Charge_Start(0);
   HAL_Delay(20);
   Get_Adc_VSL();   //短暂进行充电，检测太阳能板的充电电压
   Charge_Start(Pwm_Value_Charge);
   Dispaly_Top(str11);
   Get_Adc_ICHARGE();
                  //以给定的PWM占空比进行充电，显示电路状态并采样充电电流
   if(ICHARGE != 0||Pwm_Value_Charge == 1000||VSL <6000){
       Dispaly_Middle2(str03, ICHARGE);
   }              //显示充电电流

   Get_Adc_VBT();
   if(VBT >= 8000)Pwm_Value_Charge = 500;
   else Pwm_Value_Charge = 0;
                  //若蓄电池电压大于8V，降低充电效率
   Dispaly_Bottom(str01, VBT);
   if(VBT >= 8200){
      charge_state = 1;
      OLED_Clear();
   }              //若蓄电池电压大于8.2V，停止充电
}
```

供电状态

```
case 2:
     Charge_Start(1000);    //进入放电状态，停止充电
```

停止充电状态

```
if(discharge_state == 0){
     Discharge_Start(0);
     Dispaly_Top(str13);
     Get_Adc_VBT();
     Dispaly_Bottom(str01, VBT); //采样蓄电池电压并显示
     if(VBT >= 6400){
         discharge_state = 1;
         OLED_Clear();
     }              //若蓄电池电压大于6.4V，返回供电状态
}
```

正常充电状态

```
if(discharge_state == 1){
    Discharge_Start(Pwm_Value_LED);
    Dispaly_Top(str12);
    Get_Adc_ILED();
    Get_Adc_VBT();
                  //以给定的PWM进行放电，并采样放电电流
    if(ILED >= 300){
       Pwm_Value_LED -= 50;
    }             //保护景观灯，若电流大于300mA，自动升高占空比

    if(ILED != 0||Pwm_Value_LED == 0){
       Dispaly_Middle2(str02, ILED);
    }
    Dispaly_Bottom(str01, VBT);
                  //显示蓄电池电压和放电电流
    if(VBT <= 6000){
        discharge_state = 0;
        OLED_Clear();
    }             //若蓄电池电压小于6V，停止充电
}
```

光控状态

```
case 3 :
      Charge_Start(0);
      HAL_Delay(100);
      Get_Adc_VSL();
      Charge_Start(1000);
      Dispaly_Middle2(str04, VSL);
                  //短暂开启充电，采样太阳能板输出电压，并显示

      if(VSL >= 2000) {
          auto_state = 0;
          Dispaly_Top(str18);
      }else{
          auto_state = 1;
          Dispaly_Top(str19);
      }             //若蓄电池电压大于2V，说明光照充足，反之说明光照不足
         Get_Adc_VBT();
         Dispaly_Bottom(str01, VBT);
                  //采样蓄电池电压并显示
```

有光照

```
  if(auto_state == 0){
     Discharge_Start(0);
 }               //若有光照，景观灯熄灭
```

无光照

```
if(auto_state == 1){
    Discharge_Start(Pwm_Value_LED);
              //若无光照，景观灯以给定的PWM占空比发光
    Get_Adc_ILED();
    if(ILED >= 300){
       Pwm_Value_LED -= 10;
    }else{
       Pwm_Value_LED += 10;
    }          //稳流，使景观灯电流稳定在300mA
    if(VBT <= 6000){
        STATE = 0;
        OLED_Clear();
    }           //若蓄电池电压小于6V，回到待机状态
}
```

六、主要仪器设备

1、Altium Designer 09 电子设计软件，STM32CubeMX 软件，Keil uVision5 软件

2、面包板，杜邦线，电阻，电容，开关，接线端口

3、STM32F103RCT6 正点原子开发板，STM32F103C8T6 最小系统板，OLED 显示模块 SSD1306

4、NMOS 场效应管 D4144，二极管 IN4007，线性稳压器 LM7805CT，运算放大器 LM258

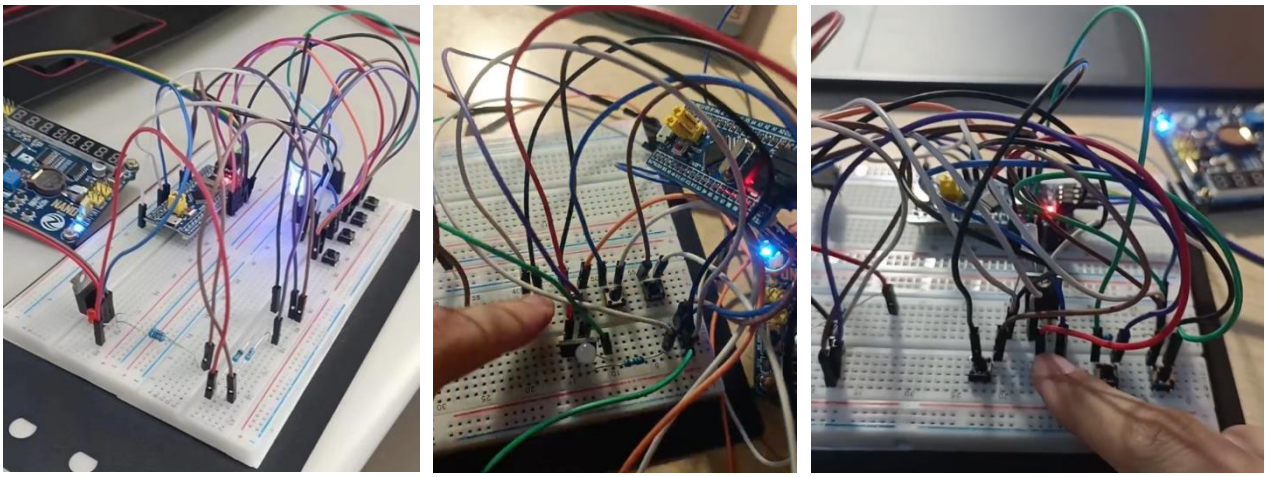
5、蓄电池，太阳能板，LED

6、PCB 电路板，电烙铁，焊锡丝，热风枪，松香

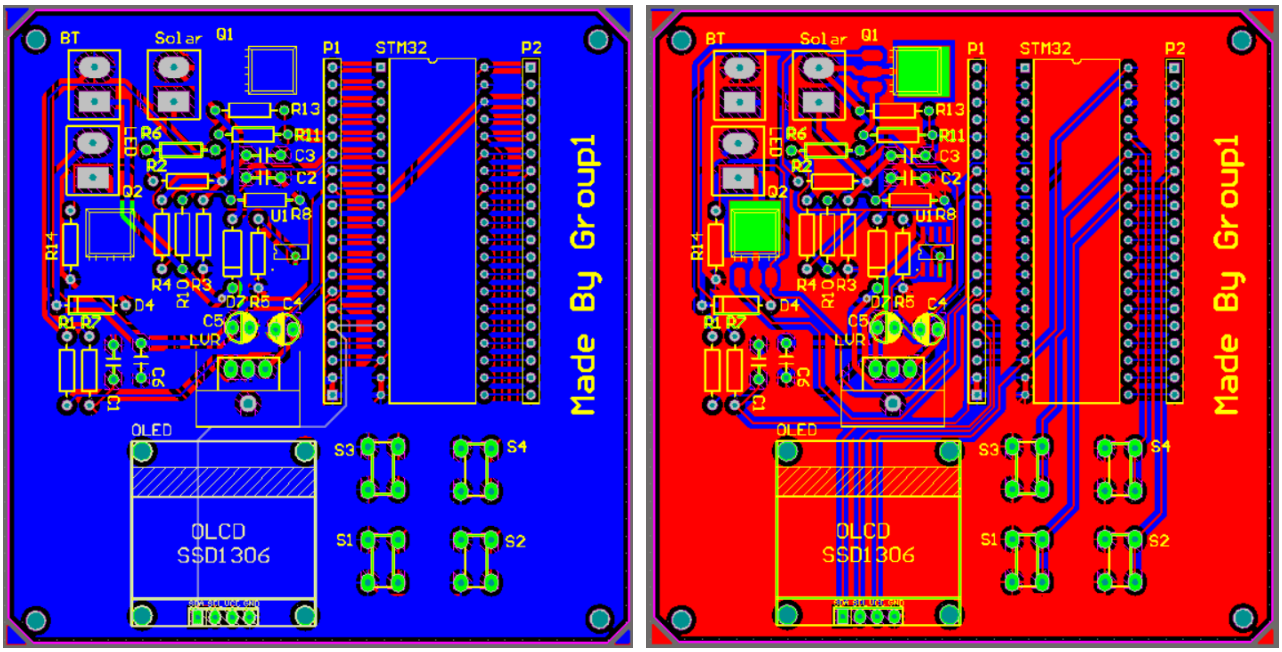
7、万用表，示波器，直流电源

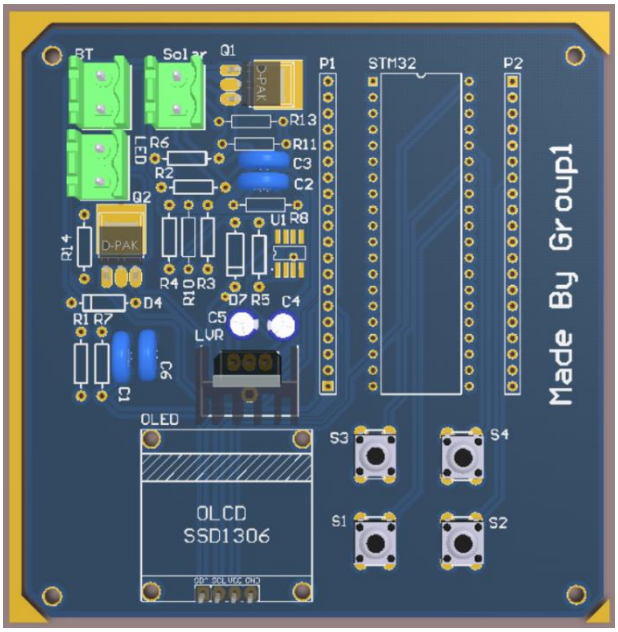
七、实验过程与数据处理及记录

1、搭建面包板

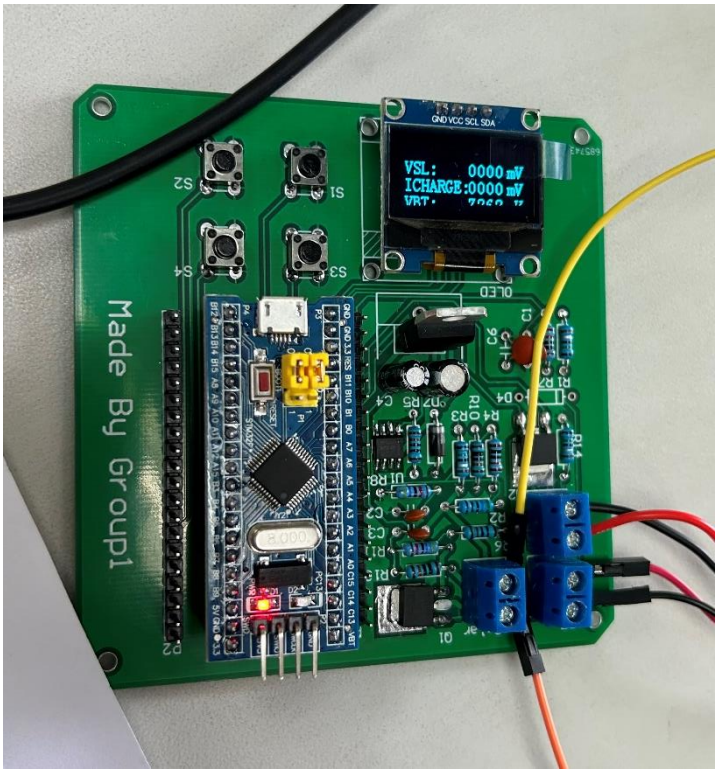


2、PCB 板设计





3、PCB 焊接与调试



七、分工

胡康鑫：负责原理图设计，PCB 设计与焊接，实验报告公共部分和硬件部分，以及共同参与所有调试。

施佳阳：负责代码设计及编写，实验报告代码设计部分，以及共同参与所有调试。

八、心得体会

1、胡康鑫

本次实验我们小组挑选的是太阳能景观灯控制器设计任务，其实在开题之前就有所听说，这个任务难度相对比较大，但同时老师也比较推荐这个任务，我们小组想要挑战一下，所以就选了这个。

我主要负责的部分是硬件部分也就是设计原理图和 PCB 板，以及后续在面包板上的电路调试和 PCB 板焊接。在画原理图初期确实是十分折磨，尽管我完整地学过模拟电路的知识，但是理论知识和工程设计之间有着不小的差别，再加上老师给的参考文档逻辑混乱且错误频出，而且任务要求也不是特别清晰，所以在设计原理图开始时，我几乎完全不知道从哪里下手。好在有互联网可以运用，我在 CSDN 等开发者网站上搜索到了一些有用资料，再加上 ChatGPT 等工具，我大致了解了设计思路和方法。然后我参考了一些其他课题的参考资料，并且也请教了一些其他同学，掌握了一些有用的设计方法，此时我对原理图的设计已经初步完成。之后金向东老师又在学在浙大上上传了一些指导视频，我参考老师提到的要点和重点，对自己的原理图进行了进一步的完善。此时的原理图在逻辑上已经能够实现基本的功能。

随后还需要和队友进行讨论，由于队友负责了代码的主要部分，所以为了方便后续的代码设计和调试，我们还需要共同讨论，对原理图进行优化，形成更高效、更合理的设计方案。然后要对所用到的器件选择正确的封装和合适的参数，比如采用运放进行电压放大时，要对电阻值进行初步的计算和选取，但是这些都是初步的粗浅计算，毕竟具体的参数选择还要在实践中进一步确定。封装则是要十分仔细才行，比如老师给的库里，SSD1306 的引脚封装和实际相反，需要手动进行调整，如果直接用，那么在后面焊接时会有很多麻烦。

接下来就是画 PCB，在电设中，我初步学习过画 PCB 的相关知识，但是这次的电路要远远比电设中的电路更加复杂，需要考虑的因素也更多。比如在电设的 PCB 设计中，由于元件很少，可以采用美观至上的原则进行布局，也可以采用自动布线进行连接；但是此次的电路不仅元件数量很多，而且有很多功率器件，这就要求我们按照模块进行布局，我把充电模块统一放在左上角，然后 LED 模块紧邻充电模块，并且使用粗线进行连接；而对于一些按钮和 OLED 等人机交互模块，则要简洁地放在另一个区域，方便进行操作和观察。在最后，我还进行了之前未曾进行过的操作，也就是覆铜，这个步骤是为了将所有的 GND 信号进行强制的共地。

由于我们班的 PCB 下发相对较晚，所以我们不得不先用面包板进行调试，由于没有之后真正要用的运放和 MOS 管，所以这个步骤的调试也基本上是定性层面的调试。在按钮环节我们进行了相当长时间的调试，一些是代码的问题，一些是面包板和杜邦线接触不良的问题。而放电环节的调试则相对顺利，也确定了具体采用的电阻值。

随后 PCB 板下发，我们先将调试完成的模块焊接并进行调试，很顺利地就完成了。对于未曾在面包板上进行过调试的充电模块，则是边调试边焊接，根据实验室所提供的元件确定相应的电压放大倍数，其中贴片器件的焊接是具有一定挑战的环节，需要仔细操作。

尽管代码部分主要由队友完成，但是由于需要一起 debug 和调试、修改，所以我也需要从代码中学到了很多，也了解了 STM32 是如何运行起来的。在调试过程中，由于我们需要对四个不同的信号进行 ADC 采样，但是例程只教了不同 ADC 的采样，我们的电路则是设计为一个 ADC 的四个不同通道进行采样，在请教老师和查阅互联网之后，我们掌握了多通道 ADC 采样的方法。此外，我们的按键总是存在抖动，导致无法稳定实现对应功能，也是进行了多种尝试，最终解决了这个问题。

调试时为了接近实际，我们顶着太阳去室外采集太阳能，遇到多云阴天的天气还要等很久才能给蓄电池充上电，后来发现蓄电池充电实在是太慢了，所以还是进实验室用电源模拟太阳能板了，但是最终的效果基本上是一致的，功能也都成功实现了，验收总体上也是顺利的。验收之后由于还有不少时间，我们组从寝室拿了一个风扇进行拆除，加到了我们的项目里，也就是检测到光强较大时意味着太阳毒辣，此时开启风扇，而光强较弱时意味着阴天，不需要开启风扇，最终也成功实现了这个额外的小功能。

这次的实验是一次完整的从设计到调试到验收的工程综合实现，具有不小的挑战性，尽管过程有点痛苦，但是我们也从中学到不少东西，得到了提高。

2、施佳阳

在整个项目中，我负责的主要是软件编程设计部分。

在开始设计之前，我先认真将 STM32 讲义的实验都做了一遍，最开始的两个实验我还只是按着实验

步骤一步步照抄，慢慢就搞懂了每个步骤的意义和整个工程是怎么实现的，这个过程让我后来进行程序设计时少走了很多弯路。

在做完推荐实验之后我才开始构思整个电路功能的实现，程序架构的构思并没有花费太多时间。在本学期的数字系统设计实验课程中我学习了用 verilog 语言对硬件功能的设计，一开始我以为两者差不多，但很块我发现二者有一个很明显的区别，verilog 每个模块是可以同时工作的，而 c 语言的函数只有在调动时才会发挥作用，因此在设计逻辑时有很大的区别。

最开始的时候，我将四个状态的指令写在了四个函数的循环里，直接使用 KEY_scan 函数在每次循环结束监测按键是否按下，然而这样的电路性能很差，如果循环中有延迟函数，会导致有时按下按键时信号不会被接收。意识到这一点之后，我先采用的方法是用定时扫描的方法，定时监测按键是否按下，但是效果并不理想，于是又采用了按键中断回调函数来实现。但是在使用按键中断回调函数的时候同样遇到了一个很严重的问题，由于下降沿响应时存在抖动，这导致了按一次按键会执行多次按键中断。最开始时，我直接在按键中断函数中加入了延迟函数，结果按下按键后电路就中断停止工作了。在询问过老师后才明白是二者的中断优先级发生了冲突，因此我换了一种办法，加入变量 change 来防抖动，结果非常的成功。

除此之外，由于我们需要监测的外部电压、电流一共有四个，因此要用到至少四个 ADC 采样通道，而实验讲义中并没有讲到多路 ADC 采样的方法，后来在询问了老师之后，才学会了如何在 ADC 采样时更改对应通道。同时我们还发现 ADC 采样只能显示瞬时值，并且与实际值存在误差，采样的 ADC 电压要比实际的电压小，需要我们进行手动校准。最终我们成功把采样误差控制在了十毫伏和五毫安以内。

在进行 OLED 显示的控制时，最开始我是让每次循环结束后都刷新显示屏，但是实际使用起来对读取数据很不友好，于是我加入了 store 变量来存储当前的状态 STATE，当 STATE 发生变化时，才进行显示屏的刷新。

在调试过程中多亏了老师的指导，有很多问题是我们之前根本没有考虑到的，在刚遇到困难时完全不知道要如何解决，只能不断打扰老师询问细节。在代码编写、调试的过程中，我意识到最重要的一件事就是软件的编程不能脱离硬件，在 PCB 还没分发下来时，我就已经把初版代码完成了，检查了几遍都没有发现什么问题，但是在面包板上调试时却处处碰壁，碰到了很多问题。而重新在 PCB 板上实验时，也碰到了很多意想不到的小 bug，所幸都成功解决了。因此虽然一开始我没有一起画原理图和 PCB，但是在结束之后我仍旧对电路设计有了更深刻的理解，也有自信若再次面对电路设计的课题，能更加游刃有余地完成任务。

九、致谢

团队成员合照：

指导教师：