CW32 电压电流模块

主要功能：

1、对 5-99V 的电压与 0.1-3A 的电流进行测量与显示

2、通过蓝牙发送测得的数据

3、作为一款 CW32+数码管的迷你开发板

设计要点：

1、使用 CW32F003E4P7 设计，使用其内置电压跟随器的功能简化外围电路

2、使用了和市面电压电流表的同款接口（XH2.54+CH3.96），方便通用

3、最大 40V 的表头供电电压，覆盖大部分常用电压范围

4、板载低成本蓝牙通信电路，使用单芯片+晶振便可实现 BLE 通信

5、模块使用的所有 0603 器件使用了更加方便手工焊接的 0603L 封装

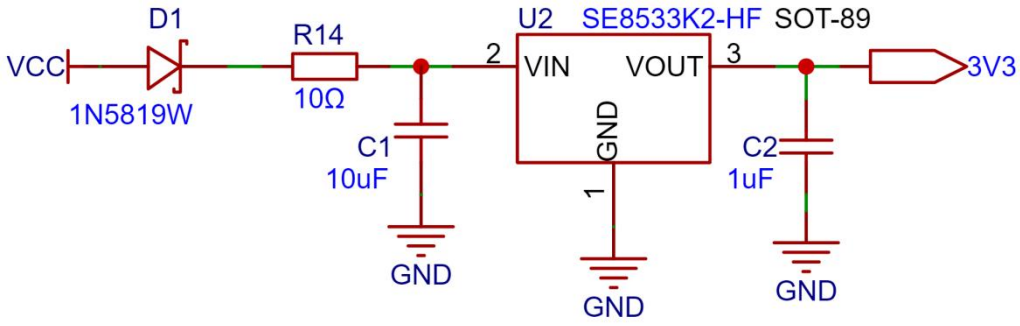
电路设计与电路原理：

本项目电路图采用模块化绘制，在电路图中对不同模块使用线条进行了分区以便于读图，下面将分模块对电路图进行分析

1、供电电路

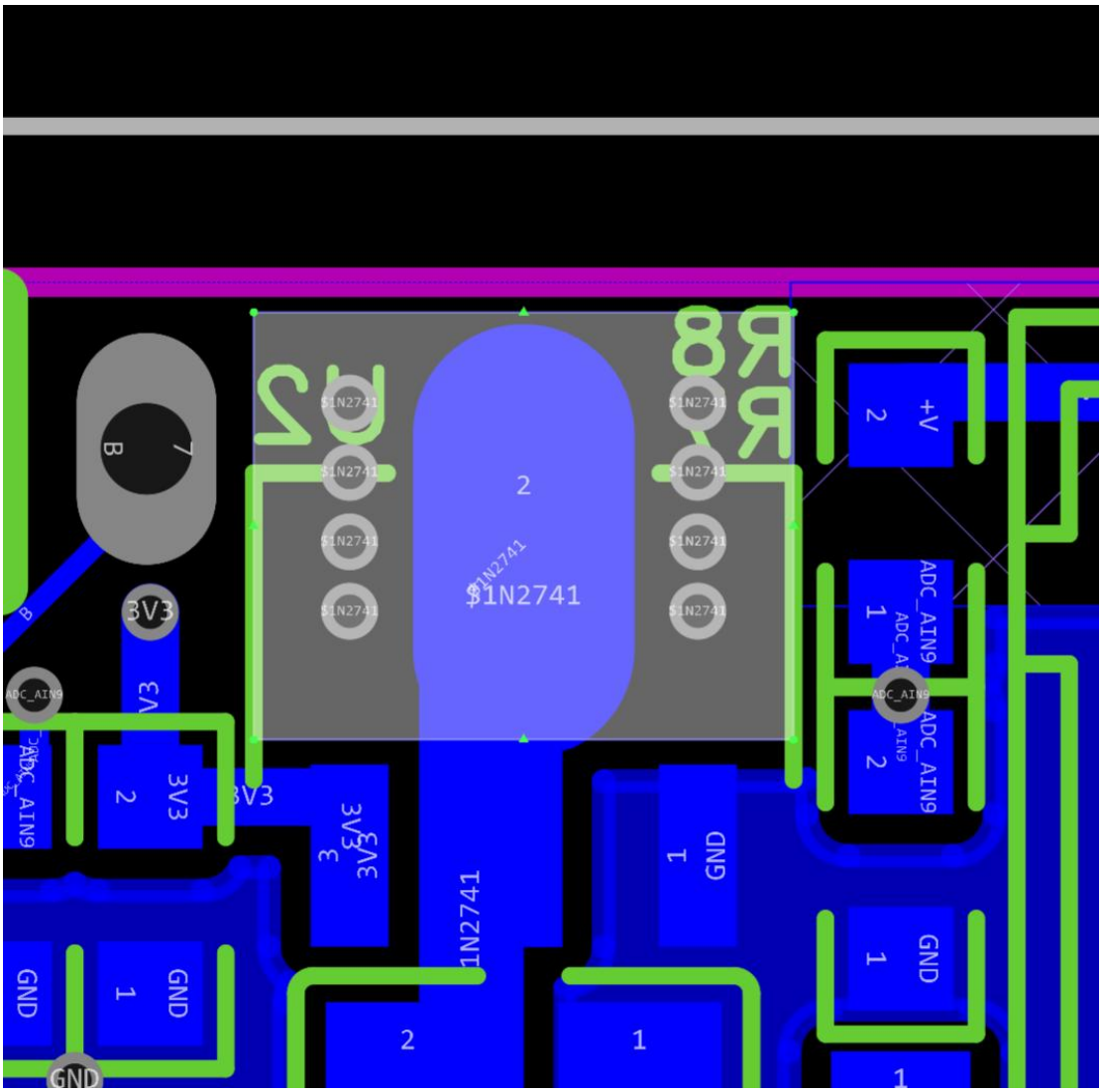
本项目使用 LDO 作为电源，考虑到电压表头可能在 24V 或 36V 供电的工业场景中使用，本项目选择了最高输入电压高达 40V 的 SE8533K2 作为电源。本项目没有使用 DCDC 降压电路来应对大压差的主要原因为减少 PCB 面积占用，次要原因为降低表头成本

考虑到高电压反接将会给模块带来不可逆的损坏，电压表头供电电路采用了串联二极管的方案进行防反接



注：本项目使用串联二极管进行防反接考虑到了本设备供电电压通常高于 5V 的使用场景，二极管的 0.7V 压降将不会供电造成影响。在常规的电路设计中不推荐该方案，而是建议使用反向并联二极管+串接保险丝的方案

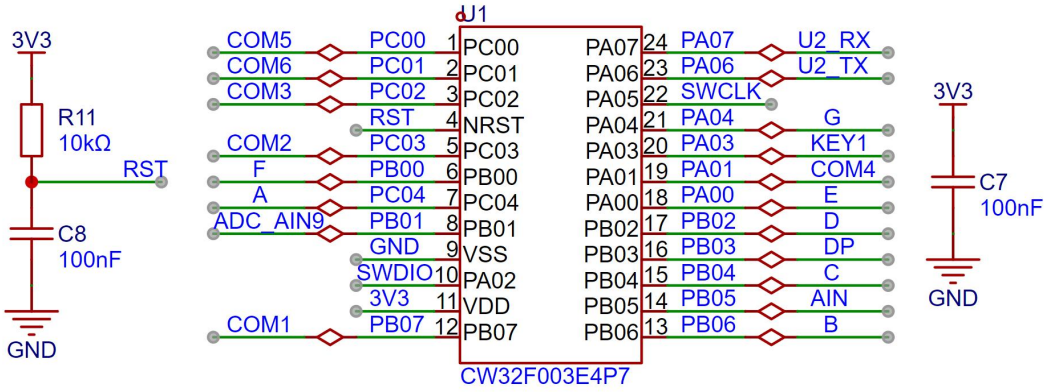
本项目额外使用了串联小电阻（10Ω）来进行分压操作，从而减少在高电压情况下 LDO 由于较大的压差导致发热严重的问题，如果实际使用场景电源电压小于 12V，可以将电阻替换为 0Ω电阻来提升电源效率



SOT-89 封装 LDO 的 2 号焊盘为散热焊盘，由于 LDO 会因为较大的压差导致发热严重，因此需要扩大与散热焊盘连接的铜箔的面积，表头在 2 号焊盘下设置了单独的铜箔，即上图中灰色半透明区域（正面也有设置独立的散热铜箔区域），同时增加了过孔，以便于将热量通过铜箔散发出去

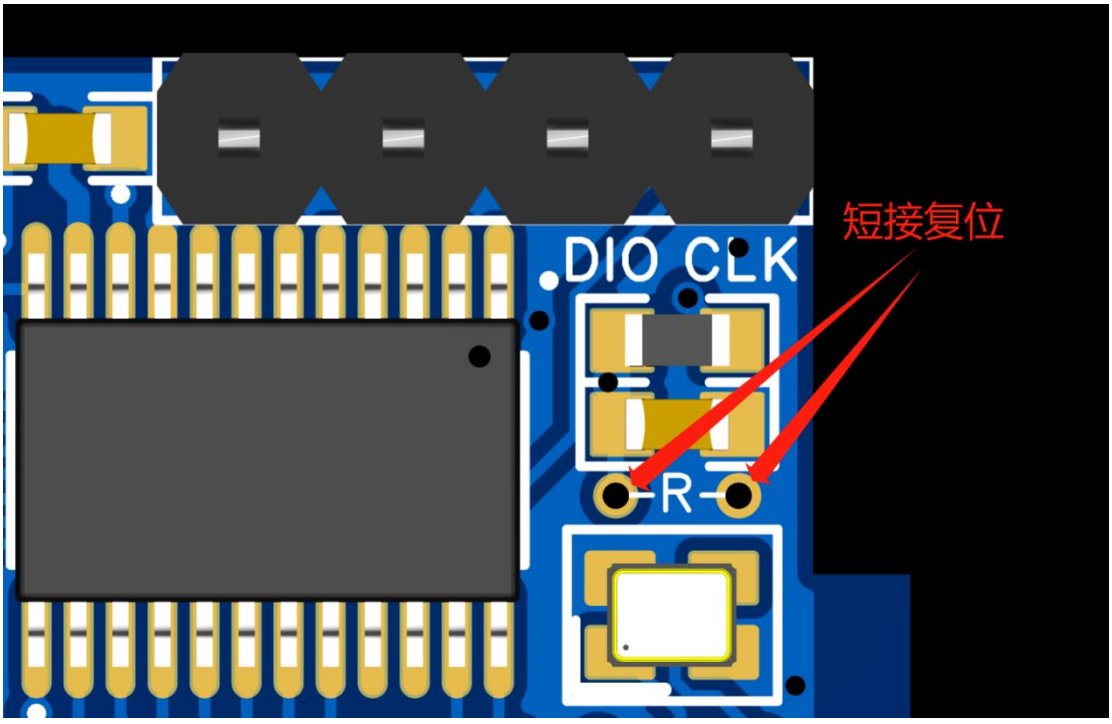
2、主控芯片

本项目使用 CW32F003E4P7 作为主控芯片



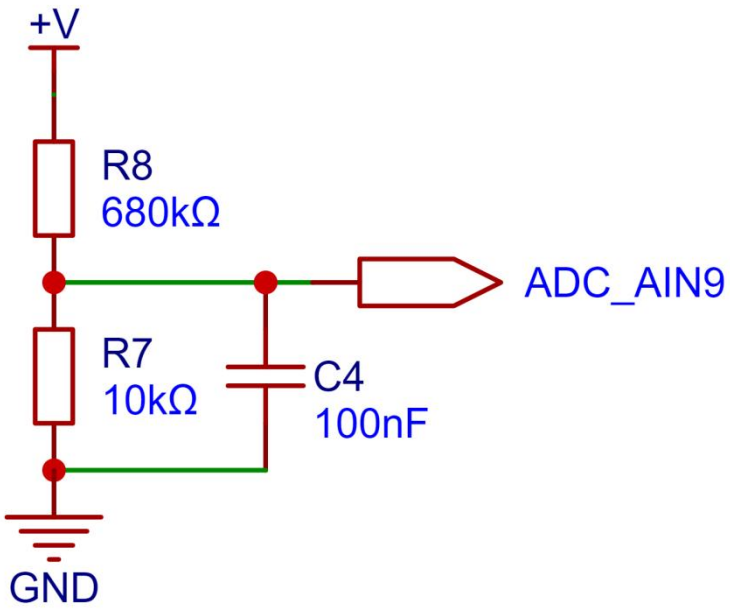
本项目使用了 CW32F003 的最小系统，既主控芯片+复位电路，而不需要晶振等其余外围电路，其中芯片的 PA05 和 PA02 分别为 SWD 接口的 CLK 和 DIO 引脚，表头模块通过 2.54 标准间距的排针引出了相关引脚

考虑到模块的尺寸问题，本模块并没有设置复位按键，而是在 PCB 上设置了一组短接触点，可以使用镊子等工具短接该组触点实现 CW32 芯片的复位



3、电压采集电路

本项目采用分压电路实现高电压采集



本项目设计分压电阻为 680K+10K，因此分压比例为 69:1（约等于 0.0145）

分压电阻选型主要需要参考以下几个方面：

1、设计测量电压的最大值，本项目中为 100V（实际最大显示 99.9V）；

2、ADC 参考电压，本项目中为 1.5V，该参考电压可以通过程序进行配置；

3、功耗，为了降低采样电路的功耗，通常根据经验值将低侧电阻选择为 10K；

随后便可以通过以上参数计算出分压电阻的高侧电阻：

1、计算所需的分压比例：即 ADC 参考电压:设计输入电压，通过已知参数可以计算出 1.5V/100V≈0.015

2、计算高侧电阻：即低侧电阻/分压比例，通过已知参数可以计算出 10K/0.015≈666.666K

3、选择标准电阻：选择一颗等于或略高于计算值的电阻，计算值约为 666K，通常我们选择 E24 系列电阻，因此本项目中选择大于 666K 且最接近的 680K

如果在实际使用中，需要测量的电压低于 2/3 的模块设计电压 66V，则可以根据实际情况更换分压电阻并修改程序从而提升测量的精度，下面将进行案例说明：

1、假设被测电压不高于 24V，其他参数不变

2、通过计算可以得到 1.5V/24V=0.0625，10K/0.0625=160K，160K 为标准 E24 电阻可以直接选用，或适当留出冗余量选择更高阻值的 180K

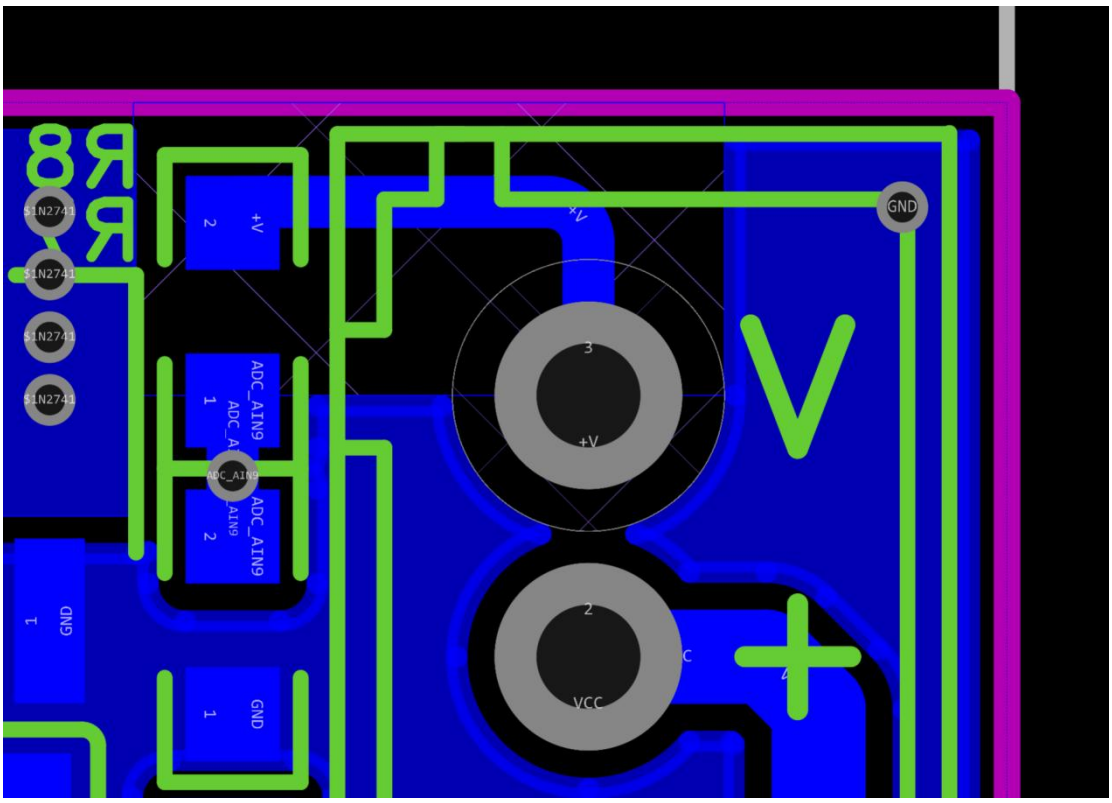
如果在实际使用中，需要测量的电压或高于模块 99V 的设计电压，可以选择更换分压

电阻或通过修改基准电压来实现更大量程的电压测量范围，下面将进行案例说明：

1、假设被测电压为 160V，选择提升电压基准的方案扩大量程

2、已知选用电阻的分压比例为 0.0145，通过公式反推，我们可以计算出 160V*0.0145=2.32V，因此我们可以选择 2.5V 的电压基准来实现量程的提升（扩大量程将会降低精度）

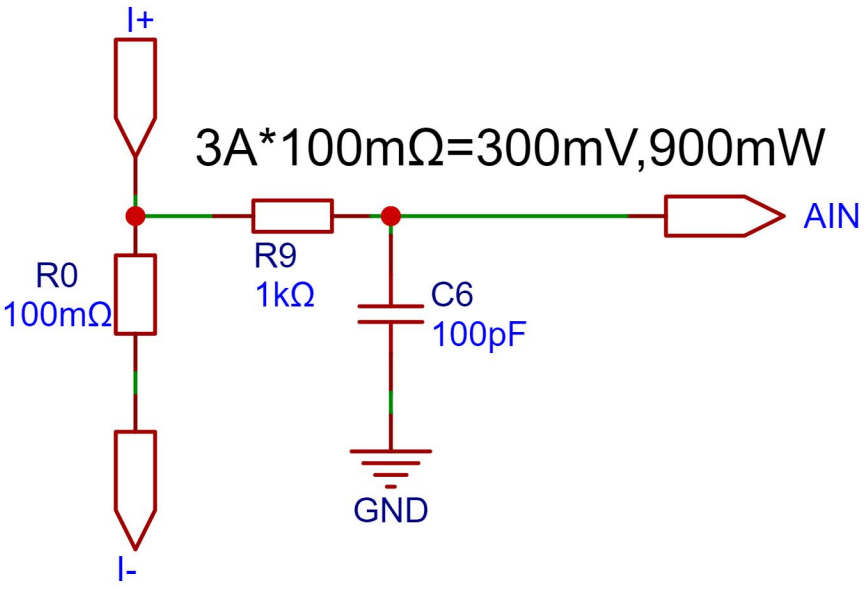
考虑到被测电源可能存在波动，在电路设计时，在低侧分压电阻上并联了 0.1uF 的滤波电容提高测量稳定性



在 PCB 进行 Layout 需要特别注意，由于需要采样的电压可能较高，因此需要在线路与铺铜之间设置更大的间距已保证安全性，在上图中，我使用了“铺铜禁止区域”来避免铺铜靠近网络的线路，另外也可以使用“约束区域”对需要注意的部分设置独立的铺铜规则来增加间距

4、电流采集电路

本项目采用低侧电流采样电路进行电流检测，采样电路的低侧与表头供地



本项目设计的采样电流为 3A，选择的采样电阻为 100mΩ

采样选型主要需要参考以下几个方面：

1、设计测量电流的最大值，本项目中为 3A

2、检流电阻带来的压差，一般不建议超过 0.5V

3、检流电阻的功耗，应当根据该参数选择合适的封装，本项目考虑到 PCB 尺寸，选择了 2512 封装

4、检流电阻上电压的放大倍数：本项目中没有使用放大电路，因此倍率为 1

随后便可以通过以上参数计算出检流的阻值选择：

1、由于本项目没有使用放大电路，因此需要选择更大的采样电阻获得更高的被测电压以便于进行测量

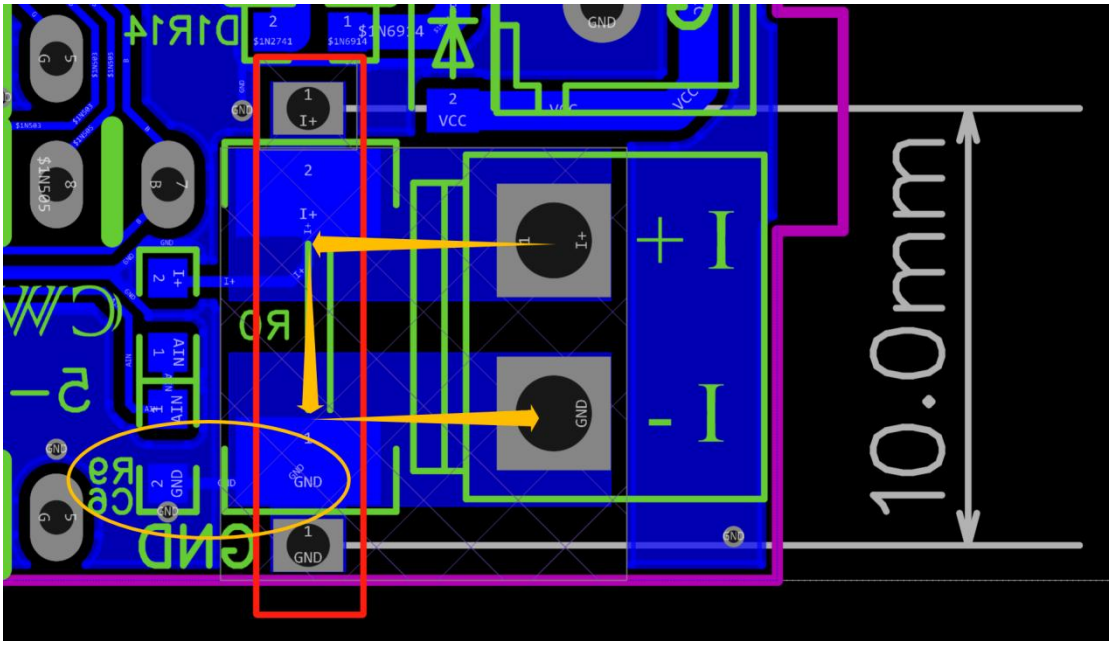
2、考虑到更大的电阻会带来更大的压差、更高的功耗，因此也不能无限制的选择更大的电阻

3、本项目选用了 2512 封装的电阻，对应的温升功率为 1W

综合以上数据，本项目选择了 100mΩ的检流电阻，根据公式可以计算出 3A*100mΩ=300mV,900mW

表头在设计时考虑到了贴片采样电阻不能够应对不同的使用环境，尤其是电流较大的场景，因此预留了 10mm 间距的康铜丝直插焊盘，可以更具实际使用场景，使用康铜丝替换贴片采样电阻

下图中红色方框框选出的即是康铜丝焊接焊盘

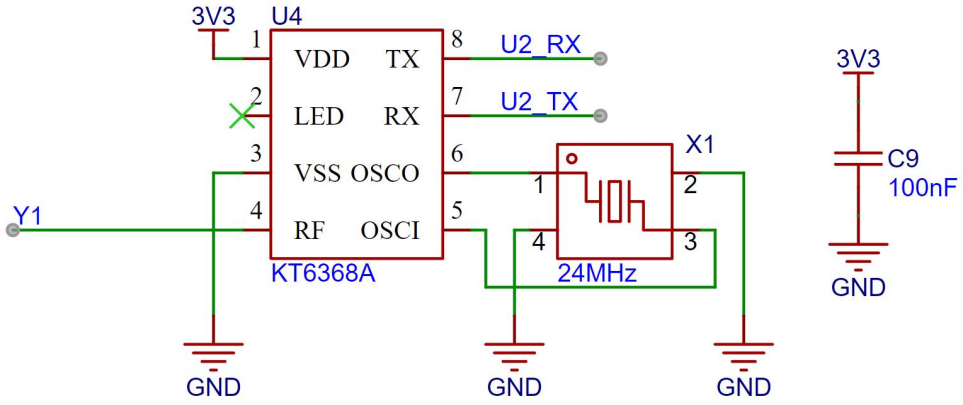


在 PCB 进行 Layout 也需要特别注意，虽然 I-网络与 GND 网络在电气上为同一网络，但是需要注意的是 I-会有大电流通过，属于“功率地”，即使该点已经接地也会因为电流的波动造成网络电平变化，因此我们可以将该网络视为一个“干扰源”；而 GND 网络为表头电源负极，即“信号地”，同时，由于单片机的 AGND 与表头 GND 并未进行隔离，那此时可以将表头 GND 视为“敏感地”，因此需要避免被干扰，因此在 Layout 时选择在 I-网络附近设置了铺铜禁止区，再使用导线将 I-网络与 GND 网络相连接，并且连接点紧靠 RC 滤波网络的电容负极，进一步减少干扰对 GND 网络的影响

在上图中，黄色箭头标注的即为大电流流通路径，通过接口的 I+流入、流经采样电阻、通过接口的 I-流出，因此从相对远离大电流路径的左下角（黄色圆圈处）引线将 I-网络与 GND 网络进行电气连接，该点也紧靠采样电路的 RC 滤波网络的 C6 电容负极

5、蓝牙通信电路

本项目使用 KT6368A 作为蓝牙主控芯片

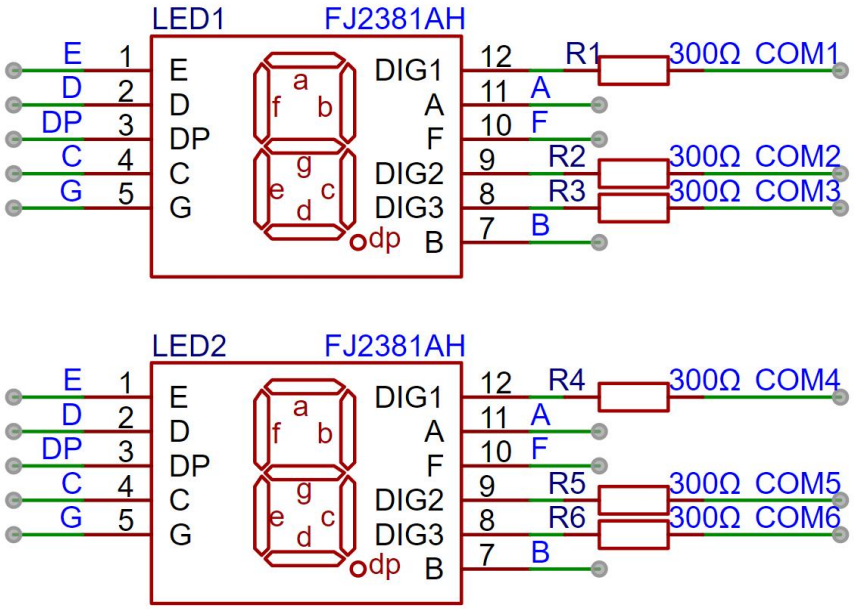


本项目只需要通过蓝牙进行数据透传，也就是通过蓝牙把数据发送出去，便于用户通过手机或电脑对被测电压电流进行无线监控，不需要其他复杂功能，因此本项目中选择了外围电路极其简单的 KT6368A，只需要使用单芯片+晶振便可实现 BLE 通信，同时该芯片为双模芯片，还可以支持 SPP 通信

为了降低项目成本，模块采用了 PCB 板载天线替代外接天线或陶瓷天线，在室内环境依旧可以保持良好的通信效果，若实际使用场景对通信距离有要求，可根据实际情况改为不同的天线类型

6、数码管

本项目采用了数码管作为显示单元

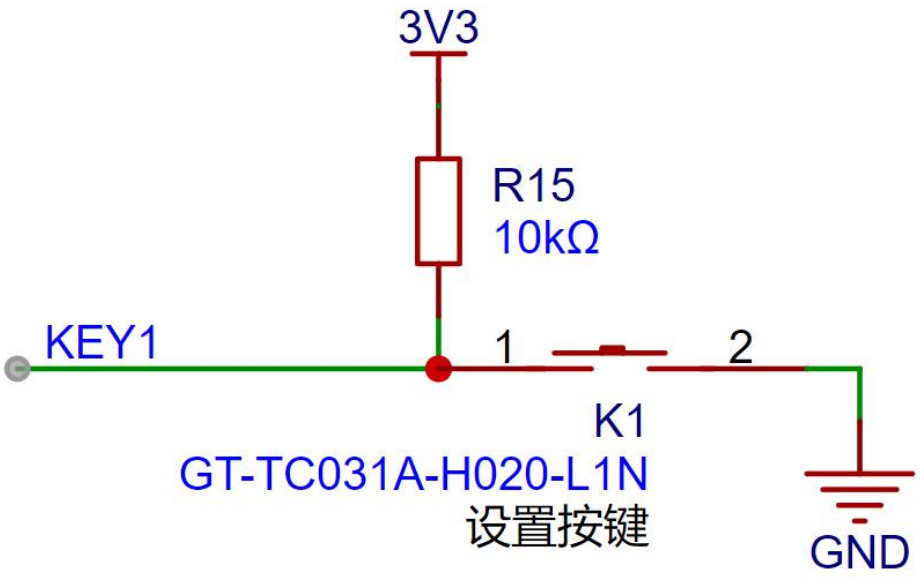


在本项目中使用了两颗 0.28 寸的三位共阴数码管作为显示器件，相较于显示屏，数码管在复杂环境中拥有更好的识别度，可以根据实际使用环境的需求，改为更小的限流电阻实现更高的数码管亮度；在另一方面，数码管拥有较好的机械性能，不会像显示屏一样容易被外力损坏

在本项目中，经过实际测试，数码管的限流电阻被配置为 300Ω，对应的亮度无论是红色还是蓝色数码管，均具有较好的识别度，且亮度柔和不刺眼

7、按键

本项目预留有一颗按键与配套电路

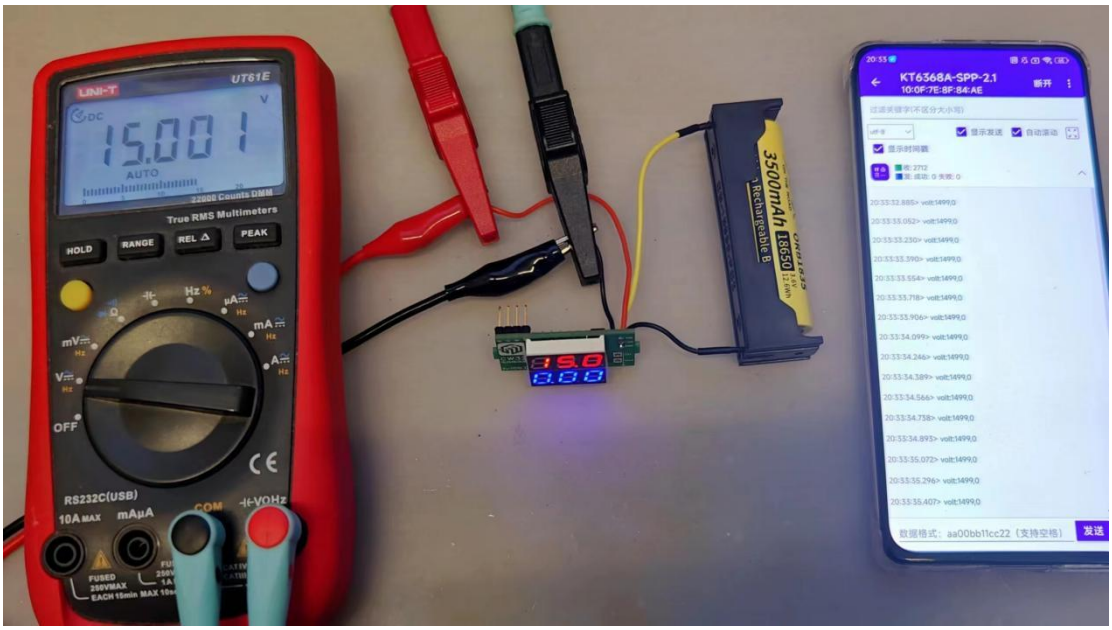


考虑到用户可能需要对表头进行二次开发，本项目预留有一颗按键，按键 io 默认上拉，按下后则拉低，用户可以根据需求修改程序代码，使用按键实现不同的功能

表头精度测试：

以下测试环境为本人工作室，非专业计量机构，测试数据仅供参考

1、电压测试：



电压测试使用数控线性电源作为被测对象，锂电池作为表头供电，万用表参考数据读取自 22000 字四位半万用表，同时记录表头模块发出的蓝牙数据作为辅助参考

由于数控电源限制，电压测试范围为 5-24.5V，测量数据步进为 0.5V

表头测量精度为小数点后两位，即 xx.xx，在显示的数据≥10 时，由于表头是三位的，会通过四舍五入舍弃最后一位，显示为 xx.x，但串口（蓝牙）会继续发送原始数据

电压设置	表头示数	蓝牙数据	万用表读数	数据偏差
5.0	5.01	501	5.00	+0.01
5.5	5.49	549	5.50	-0.01
6.0	5.97	597	6.00	-0.03
6.5	6.50	650	6.50	0
7.0	6.99	699	6.99	0
7.5	7.49	749	7.49	0
8.0	7.97	797	7.99	-0.02
8.5	8.51	851	8.49	+0.02
9.0	8.99	899	9.00	-0.01
9.5	9.47	947	9.50	-0.03

点击图片可查看完整电子表格

注：设万用表度数为基准，数据偏差为 “基准数据” - “表头示数”

2、电流测试

电流测试使用数控线性电源作为被测对象，使用限流功能将电源作为恒流源使用，锂电池作为表头供电，万用表参考数据读取自 22000 字四位半万用表，同时记录表头模块发出的蓝牙数据作为辅助参考

由于表头量程限制，电流测试范围为 0.3-3.0V，测量数据步进为 0.1A

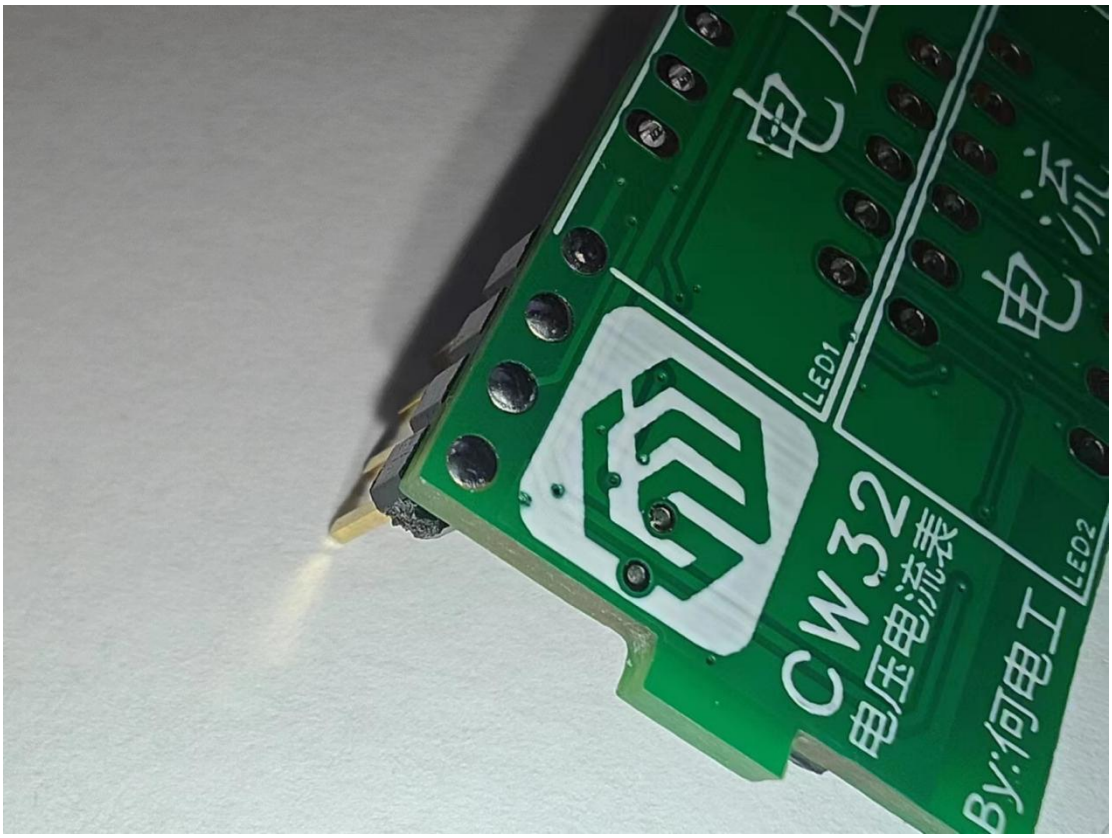
电流设置	表头示数	蓝牙数据	万用表读数	数据偏差
0.1	0.10	10	0.10	0
0.2	0.19	19	0.20	-0.01
0.3	0.29	29	0.30	-0.01
0.4	0.39	39	0.40	-0.01
0.5	0.50	50	0.50	0
0.6	0.59	59	0.60	-0.01
0.7	0.70	70	0.70	0
0.8	0.80	80	0.80	0
0.9	0.91	90	0.90	+0.01
1.0	1.00	100	1.00	0

点击图片可查看完整电子表格

注：设万用表度数为基准，数据偏差为 “基准数据” - “表头示数”

复刻注意事项：

1、烧录排针焊接



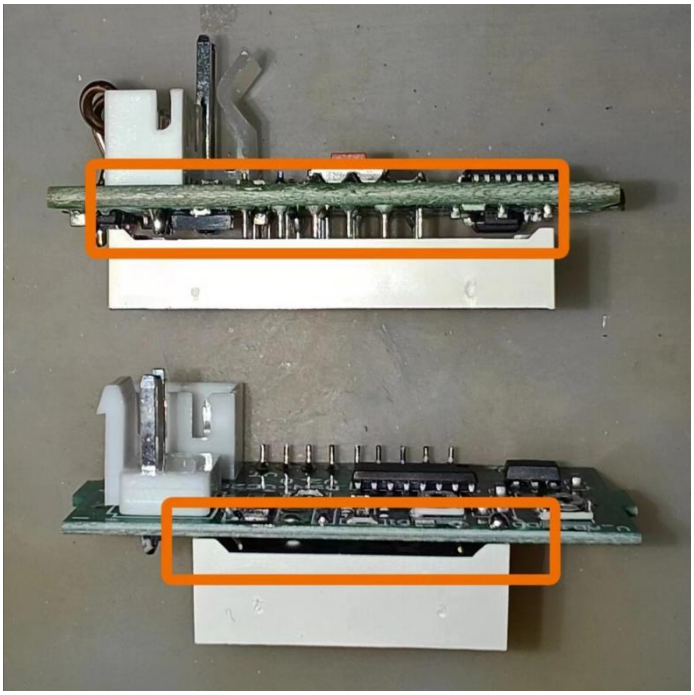
如果需要焊接 SWD 接口排针，推荐只焊接 SWDCLK、SWDIO、GND 三根排针，表头可以使用独立供电，只需要供地就可以进行烧录以及调试了

如果需要焊接四根排针，需要先焊接排针再焊接数码管，且需要像上图一样确保排针底部不要凸出 PCB，否则会与数码管发生干涉

2、外壳安装

如果您计划使用淘宝或其他渠道购买的公版外壳，在焊接数码管前请先仔细阅读下方内容

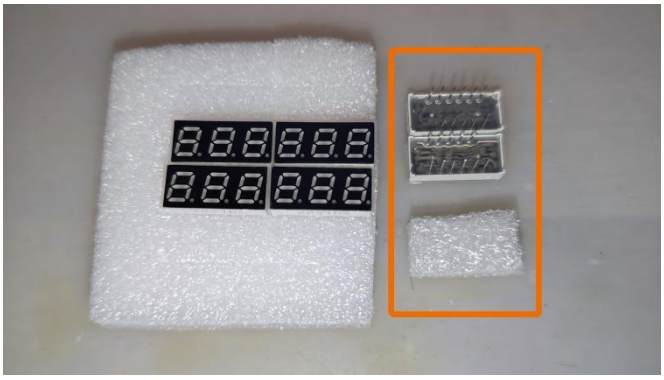
注：建议先焊接数码管，再焊接连接器



上图为两种型号的电压电流表头，均购自淘宝，我们可以看到因为数码管高度的不同，上方的表头的数码管在焊接时需要使数码管距离 PCB 一定高度（参考上图），具体高度依据采购的数码管尺寸而定

如果您购买的是本文推荐的数码管链接中的数码管器件，则数码管相对较薄，在焊接时也需要使数码管距离 PCB 一定高度才能使得表头在外壳中稳定安装

以下为推荐的焊接技巧：

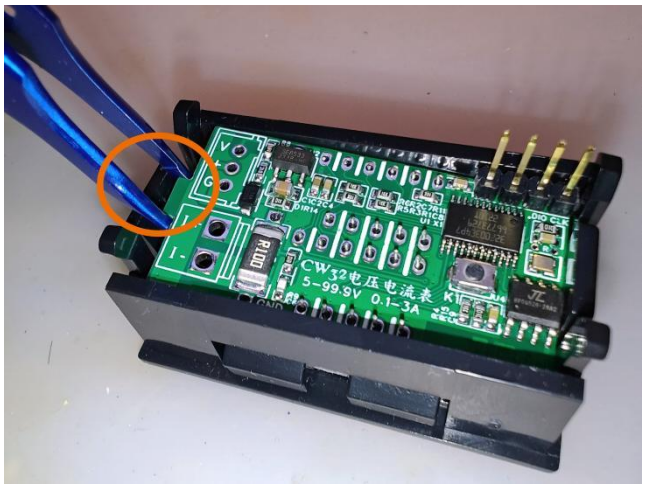


在购买数码管时，卖家为了避免在运输中避免数码管管脚在快递运输过程中弯折损坏，通常会使用泡沫塑料保护数码管管脚，而这个泡沫塑料还有另外一个妙用，就是帮助我们焊接数码管

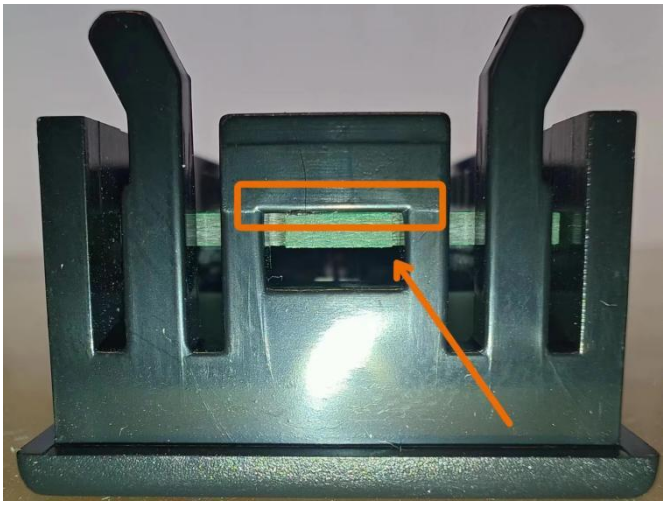
将两颗数码管上下并列排列，裁剪出一块和两颗数码管面积总和差不多的泡沫（参考上图），建议适当减小一些长宽，再从侧面切开（参考下图）从而减少塑料泡沫的厚度使其略薄于数码管的厚度

将数码管先插入泡沫，再将“数码管+泡沫组合体”插入 PB，此时仅需要将数码管插入 PCB，不需要进行焊接

随后即可将以上“组合体”安装入外壳（参考上图）



在安装时，先将一侧 PCB 的“小耳朵”插入外壳侧面孔洞，再在另外一侧使用镊子插入外壳与 PCB 的缝隙中制造出一个斜面，将 PCB 压入外壳（参考上图），确保 PCB 两侧的“小耳朵”可以将表头模组固定在外壳中



安装后，需要确保 PCB 的“小耳朵”能够通过泡沫的弹力自己顶住外壳开口的上沿（参考上图），晃动外壳确保模块不会在外壳内晃动，此时便可对数码管进行焊接

焊接完数码管后，再使用镊子反向操作取出表头模块，焊接连接器

如何后续发现其他注意事项再补充

采购链接：

CW32F003E4P7 芯片：CW32F003E4P7_(CW(芯源))CW32F003E4P7 中文资料_价格_PDF 手册-立创电子商城

数码管（0.28 英寸，共阴，推荐使用红色+蓝色）：item.taobao.com

KT6368A 蓝牙芯片：item.taobao.com

表头外壳：0.56 寸电压表外壳配滤光片（因为运费会远高于外壳的价格，而不同地区的运费又不一样，大家根据运费自己挑一家买就行）

其他：

如果对项目有任何疑问或建议，欢迎留言讨论

如果您对此项目感兴趣，欢迎加入何电工的交流群 1016193632（作者的个人群）

了解更多 CW32 信息欢迎加入生态社区交流群：2 群：652777214；3 群：657586457

软件部分

0.源码资料

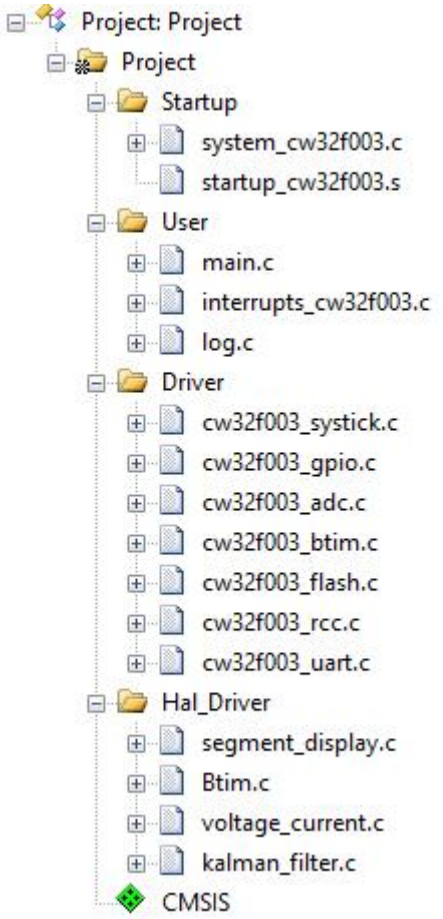
1.代码结构

Startup --- 系统启动文件

User --- 用户 app 代码

Driver --- 底层驱动代码

Hal --- 抽象层驱动代码



2.初始化部分

初始化部分分为：时钟初始化，GPIIO 初始化，ADC 初始化，定时器初始化。

2.1.时钟初始化

CW32F003 的最大主频为 48MHZ,选择的是内部晶振。

```
C
RCC_HSI_Enable(RCC_HSIOSC_DIV1); //设置频率为 48M
```

设置主频为 48MHZ 需要注意，Flah 的访问周期需要更改。

```
C
FLASH_SetLatency(FLASH_Latency_2);
```

2.2.定时器初始化

2.2.1.滴答定时器初始化

滴答定时器用于使用延时函数。初始化如下：

```
C
InitTick(8000000); //SYSTICK 的工作频率为 8MHz，每 ms 中断一次
```

2.2.2.BTIM 定时器初始化

BTIM 定时器用作定时 ADC 采样使用，定时周期为 1ms。

```
C
void btim_init(void)
{
    BTIM_TimeBaseInitTypeDef BTIM_TimeBaseInitStruct;
    __RCC_BTIM_CLK_ENABLE();

    /* NVIC Configuration */
    NVIC_EnableIRQ(BTIM1_IRQn);

    BTIM_TimeBaseInitStruct.BTIM_Mode = BTIM_Mode_TIMER;
    BTIM_TimeBaseInitStruct.BTIM_Period = 6000-1;//1ms 采集 1 次
    BTIM_TimeBaseInitStruct.BTIM_Prescaler =
BTIM_PRS_DIV8;//48/8=6MHZ
    BTIM_TimeBaseInitStruct.BTIM_OPMode = BTIM_OPMode_Repetitive;

    BTIM_TimeBaseInit(CW_BTIM1, &BTIM_TimeBaseInitStruct);
    BTIM_ITConfig(CW_BTIM1, BTIM_IT_OV, ENABLE);
    BTIM_Cmd(CW_BTIM1, ENABLE);
}
```

2.3.ADC 初始化

- ADC 需要使用两个通道，分别采集电压电路的电压值和电流电路的电压值。
- 多通道 ADC 使用的是序列连续采样模式。
- 由于电流采集采集电路的电压范围很小，因此选择最低的 ADC 参考电压（内部 1.5v 基准电压）。
- 采样周期选择 10 个周期，因为采样周期约大，采集到的 adc 值越稳定。由于电压信号是慢系统，选择周期越大越好。

```
C
/**
```

```
 * @brief adc 相关的初始化
 *
 */
void voltage_adc_init(void)
{
    ADC_InitTypeDef     ADC_InitStructure;
    ADC_SerialChTypeDef ADC_SerialChStructure;
    GPIO_InitTypeDef    GPIO_Init_Struct;
    __RCC_GPIOB_CLK_ENABLE();
    // 打开 ADC 时钟
    __RCC_ADC_CLK_ENABLE();
    GPIO_Init_Struct.IT   = GPIO_IT_NONE;
    GPIO_Init_Struct.Mode = GPIO_MODE_ANALOG;
    GPIO_Init_Struct.Pins = GPIO_PIN_1 | GPIO_PIN_5;
    GPIO_Init(CW_GPIOB, &GPIO_Init_Struct);
    // set PB01 as AIN9 INPUT
    PB01_ANALOG_ENABLE();
    // set PB05 as AIN11 INPUT
    PB05_DIGTAL_ENABLE();
    // ADC 默认值初始化
    ADC_StructInit(&ADC_InitStructure);
    // ADC 工作时钟配置
    ADC_InitStructure.ADC_ClkDiv     = ADC_Clk_Div32; // PCLK/32 =
48/32 = 1.5Mhz
    ADC_InitStructure.ADC_VrefSel    = ADC_Vref_BGR1p5;
    ADC_InitStructure.ADC_SampleTime = ADC_SampTime10Clk;

    ADC_SerialChStructure.ADC_Sqr0Chmux  = ADC_SqrCh9;
    ADC_SerialChStructure.ADC_Sqr1Chmux  = ADC_SqrCh11;
    ADC_SerialChStructure.ADC_SqrEns     = ADC_SqrEns01;
    ADC_SerialChStructure.ADC_InitStruct = ADC_InitStructure;

    ADC_SerialChContinuousModeCfg(&ADC_SerialChStructure);

    ADC_ClearITPendingAll();
    ADC_BufEnSerialCh(ADC_SqrVref1P2); // 开启跟随器
    // ADC 使能
    ADC_Enable();
    ADC_SoftwareStartConvCmd(ENABLE);
}
```

3.数码管驱动

3.1 重映射数码管引脚

将数码管的段码脚和公共脚重映射。如下：

```
C
#define SEG_A_GPIO  CW_GPIOC
#define SEG_A_PIN   GPIO_PIN_4
#define SEG_B_GPIO  CW_GPIOB
#define SEG_B_PIN   GPIO_PIN_6
#define SEG_C_GPIO  CW_GPIOB
#define SEG_C_PIN   GPIO_PIN_4
#define SEG_D_GPIO  CW_GPIOB
#define SEG_D_PIN   GPIO_PIN_2
#define SEG_E_GPIO  CW_GPIOA
#define SEG_E_PIN   GPIO_PIN_0
#define SEG_F_GPIO  CW_GPIOB
#define SEG_F_PIN   GPIO_PIN_0
#define SEG_G_GPIO  CW_GPIOA
#define SEG_G_PIN   GPIO_PIN_4
#define SEG_DP_GPIO CW_GPIOB
#define SEG_DP_PIN  GPIO_PIN_3

#define SEG_COM1_GPIO CW_GPIOB
#define SEG_COM1_PIN  GPIO_PIN_7
#define SEG_COM2_GPIO CW_GPIOC
#define SEG_COM2_PIN  GPIO_PIN_3
#define SEG_COM3_GPIO CW_GPIOC
#define SEG_COM3_PIN  GPIO_PIN_2
#define SEG_COM4_GPIO CW_GPIOA
#define SEG_COM4_PIN  GPIO_PIN_1
#define SEG_COM5_GPIO CW_GPIOC
#define SEG_COM5_PIN  GPIO_PIN_0
#define SEG_COM6_GPIO CW_GPIOC
#define SEG_COM6_PIN  GPIO_PIN_1
```

3.2 数码管段码表

使用的是共阴数码管，先把段码位和 bit 对应，做出显示 0-9 的段码表。

显示数\段码引脚	A	B	C	D	E	F	G	D
0	亮	亮	亮	亮	亮	亮	灭	灭
1	灭	亮	亮	灭	灭	灭	灭	灭
2	亮	亮	灭	亮	亮	灭	亮	灭
3	亮	亮	亮	亮	灭	灭	亮	灭
4	灭	亮	亮	灭	灭	亮	亮	灭
5	亮	灭	亮	亮	灭	亮	亮	灭
6	亮	灭	亮	亮	亮	亮	亮	灭
7	亮	亮	亮	灭	灭	灭	灭	灭
8	亮	亮	亮	亮	亮	亮	亮	灭
9	亮	亮	亮	亮	亮	亮	亮	灭

点击图片可查看完整电子表格

共阴数码管段码引脚设置高电平时才可以点亮,因此亮用 1 表示,灭用 0 表示,A 段用 bit0 表示,B 段用 bit1 表示...DP 段用 bit7 表示。将上表翻译为二进制和十六进制数，如下：

显示数	二进制	十六进制
0	00111111	0x3F
1	00000110	0x06
2	01011011	0x5B
3	01001111	0x4F
4	01100110	0x66
5	01101101	0x6C
6	01111101	0x9B
7	00000111	0x07
8	01111111	0x7F
9	01101111	0x6F

点击图片可查看完整电子表格

```
C
/*  共阴数码管编码表：
 0x3f   0x06   0x5b   0x4f  0x66  0x6d
   0      1      2      3     4     5
 0x7d  0x07   0x7f    0x6f  0x77  0x7c
   6     7      8      9     A     B
 0x39  0x5e   0x79    0x71
   C     D      E      F      */
uint8_t seg_table[16] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07,
                         0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79,
```

```
0x71};
```

3.3 单段数码管驱动

使用 for 依次取出每个 bit 对应的值，然后输出到 A~DP 引脚。

(dis_value >> i) & 0x01，i 表示的是第几 bit，移位后在与 0x01 就可以取出对应 bit 的值。

取出 bit 值，在调用设置函数 SET_SEG_X（X：A~DP），输出对应电平。

```
C
static void set_one_seg(uint8_t dis_value)
{
    int i;
    for ( i = 0; i < 8; i++)
    {
        /* code */
        //最低 bit 显示的是 a 段,依次递增,最高 bit 显示 dp
        switch (i)
        {
        case 0:
            SET_SEG_A((dis_value >> i) & 0x01);
            break;
        case 1:
            SET_SEG_B((dis_value >> i) & 0x01);
            break;
        case 2:
            SET_SEG_C((dis_value >> i) & 0x01);
            break;
        case 3:
            SET_SEG_D((dis_value >> i) & 0x01);
            break;
        case 4:
            SET_SEG_E((dis_value >> i) & 0x01);
            break;
        case 5:
            SET_SEG_F((dis_value >> i) & 0x01);
            break;
        case 6:
            SET_SEG_G((dis_value >> i) & 0x01);
            break;
        case 7:
            SET_SEG_DP((dis_value >> i) & 0x01);
```

```
            break;
        default:
            break;
        }
    }
}
```

3.4 公共端切换

数码管需要使用轮询扫描的方式显示，每次只能开启一个公共端引脚。因为是共阴数码管，开启就需要设置低电平。

```C
/**
 * @brief 公共端选择
 *        共阴数码管,公共端低电平点亮
 *
 * @param index
 */
static void select_comx(uint8_t index)
{
    switch(index)
    {
        case 0:
            SET_SEG_COM1(GPIO_Pin_RESET);
            break;
        case 1:
            SET_SEG_COM2(GPIO_Pin_RESET);
            break;
        case 2:
            SET_SEG_COM3(GPIO_Pin_RESET);
            break;
        case 3:
            SET_SEG_COM4(GPIO_Pin_RESET);
            break;
        case 4:
            SET_SEG_COM5(GPIO_Pin_RESET);
            break;
        case 5:
            SET_SEG_COM6(GPIO_Pin_RESET);
            break;
        default:
            break;
```

```
    }
}
```

3.5 数码管扫描驱动

每次显示一个数码管，需要先关闭公共端,再取出段码值，最后选择开启的公共端，最后将段码值输出。依次操作 6 个数码管，即可完成 6 个数码管的显示。需要注意数码管的扫描间隔需要小于人眼视觉滞留时间，否则无法看到全部数码一起显示。

```
C
/**
 * @brief 数码管扫描显示函数,定时器周期性调用
 *
 */
uint8_t display_pro(void)
{
    static uint8_t num = 0;
    uint8_t        dis_value;

    dis_value = *((uint8_t *)seg_display + num); // 取出段码值
    close_com();//先关闭公共端,防止重影
    set_one_seg(dis_value);
    select_comx(num);
    num++;
    if(num > 5)
    {
        num = 0;
    }
    return num;
}
```

3.6 显示电压,电流值

声明一个二维数组,用来存储电压和电流的显示段码值。数组为 uint8_t seg_display[2][3]。

将电压值分解为四位数，先判断第四为数是否有值，有值就显示前三位数，没有值就显示后三位数。

需要显示小数点就点亮 DP，只需要段码或上第 8 位，就是 0x80 即可。

```
C
```

```
/**
 * @brief 将电压,电流拆解为段码值,存储到 seg_display
 *
 * @param value 电压或者电流值
 * @param type 0 -- 表示电压 , 1-- 表示电流
 */
void set_voltage_current(uint32_t value, uint8_t type)
{
    uint8_t thousands;
    uint8_t hundreds;
    uint8_t tens;
    uint8_t Units; // 个位数
    // 计算位数
    thousands = value / 1000;
    if(thousands > 0)
    {
        Units     = value % 10;
        value     = Units > 5 ? (value + 10) : value; // 根据后一位四舍五入
        thousands = value / 1000 % 10;
        hundreds  = value / 100 % 10;
        tens      = value / 10 % 10;
        // 显示 xx.x 伏
        seg_display[type][0] = seg_table[thousands];
        seg_display[type][1] = seg_table[hundreds] | 0x80; // 加 dp 显示
        seg_display[type][2] = seg_table[tens];
    }
    else
    {
        hundreds = value / 100 % 10;
        tens     = value / 10 % 10;
        Units    = value % 10;
        // 显示 x.xx 伏
        seg_display[type][0] = seg_table[hundreds] | 0x80; // 加 dp 显示
        seg_display[type][1] = seg_table[tens];
        seg_display[type][2] = seg_table[Units];
    }
}
```

4.电压值计算

- 将 AD 采集的数据做平均算法,得到 AD 均值。
- AD 均值 * 参考电压 >> ADC 位数 = 实际采样电压值。
- 使用电压分压公式计算实际电压值
- 四舍五入
- 卡尔曼滤波算法，使测量值更稳定。
- 消除零点误差

```C
/**
 * @brief 获取电压值
 *
 * @return uint32_t 单位 10mv
 */
uint32_t get_voltage_value(void)
{
    uint32_t r_mv;
    uint32_t voltage;
    uint32_t sum = 0;

    sum     = mean_value_filter(vol_buf, ADC_SAMPLE_SIZE);
    r_mv    = sum * ADC_REF_VALUE >> 12;
    voltage = r_mv * (R2 + R1) / R1;
    // 四舍五入
    if(voltage % 10 >= 5)
    {
        voltage = voltage / 10 + 1;
    }
    else
    {
        voltage = voltage / 10;
    }
    voltage = KalmanFilter(&kfpVol, voltage);
    return voltage > vol_zero ? (voltage - vol_zero) : 0;
}
```

4.1 均值滤波

将采样点全部累加,去掉最大最小值,再做平均值。

```C
/**
```

```
 * @brief 均值滤波
 *
 * @return uint32_t
 */
uint32_t mean_value_filter(uint16_t *value, uint32_t size)
{
    uint32_t sum = 0;
    uint16_t max = 0;
    uint16_t min = 0xffff;
    int      i;

    for(i = 0; i < size; i++)
    {
        sum += value[i];
        if(value[i] > max)
        {
            max = value[i];
        }
        if(value[i] < min)
        {
            min = value[i];
        }
    }
    sum -= max + min;
    sum  = sum / (size - 2);
    return sum;
}
```

4.2 卡尔曼滤波算法

根据卡尔曼化简公式编写，详细了解卡尔曼滤波可以在网上查阅相关资料。

```
C
/**

**********************************************************************
************
 * @brief  卡尔曼滤波器 函数
 * @param  *kfp     - 卡尔曼结构体参数
 * @param  input    - 需要滤波的参数的测量值（即传感器的采集值）
 * @return 卡尔曼滤波器输出值（最优值）
 * @note
```

```
**********************************************************************
************
 */
int32_t KalmanFilter(KFPTypeS *kfpVar,int32_t input)
{
    KFPTypeS *kfp = kfpVar;
    // 估算协方差方程：当前 估算协方差 = 上次更新 协方差 + 过程噪声协方
差
    kfp->P = kfp->P + kfp->Q;
    // printf("kfp->P:%d,kfp->G :%d\n",kfp->P,kfp->G);
    // 卡尔曼增益方程：当前 卡尔曼增益 = 当前 估算协方差 / （当前 估算
协方差 + 测量噪声协方差）
    kfp->G = kfp->P*1000 / (kfp->P + kfp->R);

    // 更新最优值方程：当前 最优值 = 当前 估算值 + 卡尔曼增益 * （当前
测量值 - 当前 估算值）
    kfp->Output = kfp->Output + kfp->G * (input - kfp-
>Output)/1000; // 当前 估算值 = 上次 最优值

    // 更新 协方差 = （1 - 卡尔曼增益） * 当前 估算协方差。
    kfp->P = (1000 - kfp->G) * kfp->P;
    // printf("kfp->P:%d,kfp->G :%d\n",kfp->P,kfp->G);
    return kfp->Output;
}
```

5.电流值计算

- 将 AD 采集的数据做平均算法,得到 AD 均值。
- AD 均值 * 参考电压 >> ADC 位数 = 实际采样电压值。
- 使用欧姆定律计算电流值（电流 = 电压 / 电阻）。
- 卡尔曼滤波算法，使测量值更稳定。
- 曲线模拟公式校准,
- 消除零点误差

```
C
/**
 * @brief 获取电流值
 *
 * @return uint32_t 单位 10ma
 */
```

```
uint32_t get_current_value(void)
{
    uint32_t r_mv;
    uint32_t r_ma;
    uint32_t sum = 0;

    sum   = mean_value_filter(cur_buf, ADC_SAMPLE_SIZE);
    r_mv  = sum * ADC_REF_VALUE >> 12; //
    r_mv  = KalmanFilter(&kfpCur, r_mv);
    r_ma  = 99 * r_mv + 3; //曲线模拟公式校准 单位 10ma, mv * 100
/100m 欧
    r_ma /= 100;
    return r_ma > 1 ? (r_ma - 1) : 0;
}
```

6.校准

校准功能是在零电压,零电流的情况下,按下按键,将此刻的电压,电流值作为零点参考值。

6.1 按键初始化

```
C
void key_init(void)
{
    GPIO_InitTypeDef GPIO_Init_Struct;
    __RCC_GPIOA_CLK_ENABLE();

    GPIO_Init_Struct.IT   = GPIO_IT_NONE;
    GPIO_Init_Struct.Mode = GPIO_MODE_INPUT;
    GPIO_Init_Struct.Pins = GPIO_PIN_3;
    GPIO_Init(CW_GPIOA, &GPIO_Init_Struct);
}
```

6.2 按键检测

按键采用的是松手检测。

```
C
void key_scan(void)
{
    if(GPIO_ReadPin(CW_GPIOA,GPIO_PIN_3) == GPIO_Pin_RESET)
```

```
    {
        key_flag = 1;
    }
    if(key_flag)
    {
        if(GPIO_ReadPin(CW_GPIOA,GPIO_PIN_3) == GPIO_Pin_SET)
        {
            key_flag = 0;
            save_vol_cur_calibration();
        }
    }
}
```

6.3 保存校准值

ADC 采集会在一定范围内波动，得到的校准值可以会出现零点时显示为 0.01，由于很小可以忽略不记。

```
C
/**
 * @brief
 *
 */
void save_vol_cur_calibration(void)
{
    uint16_t calibration_value[2];

    calibration_value[0] = __get_vol_value();
    calibration_value[1] = __get_cur_value();
    vol_zero             = calibration_value[0];
    cur_zero             = calibration_value[1];
    flash_erase();
    flash_write(calibration_value, 2);
}
```

7.应用层逻辑

- 初始化
- 定时器间隔 3ms 扫描一次数码管，并显示。(6 个数码管显示时间为 18ms,看起来就不会闪)
- 间隔 300ms，计算一次电压，电流值,并更新数码管码值。

```C
int32_t main(void)
{
    FLASH_SetLatency(FLASH_Latency_2); // 设置主频为 48MHZ 需要注意，Flah 的访问周期需要更改为 FLASH_Latency_2。
    RCC_HSI_Enable(RCC_HSIOSC_DIV1);   // 设置频率为 48M
    FLASH_SetLatency(FLASH_Latency_2); // 设置主频为 48MHZ 需要注意，Flah 的访问周期需要更改为 FLASH_Latency_2。
    InitTick(48000000);                // SYSTICK 的工作频率为 8MHz，每 ms 中断一次
    btim_init();                       // 定时器初始化
    LogInit();                         // 串口初始化
    seg_gpio_init();                   // 初始化数码管
    voltage_adc_init();                // 电压,电流采集 adc 初始化
    read_vol_cur_calibration();        // 读取校准值
    key_init();                        // 按键初始化
    while(1)
    {
        if(GetTick() >= (ble_time + 1000))
        {
            printf("volt:%u,%u\n", volt, curr); // 1000ms 上传一次数据给蓝牙
            ble_time = GetTick();
        }
        if(GetTick() >= (led_dis_time + 300))
        {
            led_dis_time = GetTick();
            volt         = get_voltage_value(); // 获取电压值
            curr         = get_current_value(); // 获取电流值
            set_voltage_current(volt, 0);       // 更新电压段码值
            set_voltage_current(curr, 1);       // 更新电流段码值
        }
        key_scan(); // 按键检测
    }
}
```

中断函数

```C
/**
 * @brief This funcation handles BTIM1
 */
```

```
void BTIM1_IRQHandler(void)
{
  static uint32_t cnt;
  /* USER CODE BEGIN */
  if (BTIM_GetITStatus(CW_BTIM1, BTIM_IT_OV))
  {
    BTIM_ClearITPendingBit(CW_BTIM1, BTIM_IT_OV);
    get_adc_value();

    cnt++;
    if (cnt > 2)
    {
      //3ms 显示一个数码管
      cnt = 0;
      display_pro();
    }
  }
  /* USER CODE END */
}
```