基于 STM32 的超级电容模块设计

姓名	学号	角色	本人在团队中承担工作说明
李钊兴	1753466	组长	系统架构 硬件设计 上位机开发
李昀青	1752138	组员	嵌入式软件设计 车体运动分析
游韧舆	1751322	组员	嵌入式软件设计 模块功能开发

目录

0 摘要

超级电容器（Supercapacitors）是能够快速存储和供应高功率电力且经过大量循环不会性能衰减的电化学电容器。主要用于能源储存，而非通用电路元件，特别适用于精密能源控制和瞬间负载设备。超级电容器也有作为能量储存和 KERS 设备在电动汽车使用。在新型电动汽车以及机器人领域有着很好的应用前景。

本项目是为全国大学生机器人大赛机甲大师赛(RoboMaster)2020 赛季设计的超级电容管理模块，通过给底盘并入超级电容的方式在一定时间内突破比赛的功率限制，提高车体的机动性从而在比赛中获取更大优势。本项目包括电路设计、代码设计、上位机设计三部分，经测试功能正常但还待比赛验证。本设计对于小型机器人底盘应用超级电容有一定参考价值。

1 课题意义

1.1 潜在社会和市场应用价值分析

近年来，随着科技的进步，新能源汽车技术得到了飞速的发展，日益成为主流。但就新能源汽车而言，电池仍是问题。过去新能源汽车的电池主要是锂电池，这种电池往往充放电时间长、循环寿命短、工作温度范围窄。电池电量用尽后需要数个小时的充电才能再次使用，关键时刻会造成很大的麻烦。如果使用超级电容器来蓄能就能避免这一缺点。超级电容器（Supercapacitors）是能够快速存储和供应高功率电力且经过大量循环不会性能衰减的电化学电容器。具有充电电流、电压大，速度快的特点，在最新的超级电容电池汽车中，只需要几分钟的充电，就能正常行驶数十公里，非常的灵活。

但超级电容仍然具有体积大、能量密度小等缺点。用其作为电池的汽车一般都无法长距离行驶，或是行驶一会就要停下充电，虽然充电灵活但又带来了新的不便，所以超级电容器目前的主要作用是在汽车启动加速时提供传统锂电池无法提供的大电流，或是为一些只需要短时间工作的机器人等供电。总体上超级电容器有着很好的发展前景，甚至可以说未来的新能源汽车领域是属于超级电容的，对其进行控制和研究有很大的现实意义。

1.2 竞赛作用

超级电容在许多竞赛中也发挥着重要的作用，甚至可以决定比赛成败。在全国大学生机器人大赛机甲大师赛(RoboMaster)中，参赛队伍使用的机器人有底盘功率限制，超出功率就会受到相应惩罚。而超级电容则可以在机器人底盘输出功率较小时将额外的能量存储，在机器人加速、爬坡时使用存储的能量，提高了车体的机动性。[1]

本设计给出了一套完整的超级电容应用方案，包括电路设计、嵌入式代码设计、上位机设计。使得超级电容可以正常的在比赛中使用，并且可以通过上位机监测模块工作状态以及车体运动情况，进而做出相应的优化。

2 目标

2.1 目标清单

内容	需求陈述	指标	备注
功能需求	充电功率控制	0~120W 可控	必备
	放电路径选择(电池/电容)		必备
	总输入功率与电容充电功率监测		必备
	CAN 总线控制		必备
	工作状态指示(OLED,WS2812)		辅助
	配置信息存储与恢复(EEPROM)		增强
	车体运动分析(MPU9255)		增强
	PC 上位机监测		增强
成本需求	硬件成本控制在 500 元以内		
可靠性需求	操作方便简单，易于使用		
	能够适应各种复杂的工作环境 测试方便，损坏后易于更换		

2.2 目标分析

该模块首先要实现的便是**可控的电容充电功率**，由于比赛规则限制底盘最大供电功率为 120W，本模块的最大充电功率也设计为 120W。其次便是**放电路径选择**，此功能可以使灵地模块灵活地切换输出状态，在某些情况下直接直接使用电池供电。通过板载的**功率监测**器可以对功率控制做出反馈，并实现在额定功率不变的情况下既给底盘供电又将剩余的功率给电容充电的功能，这一部分的实现参考了港科的开源方案。[2]

控制方式主要是通过 CAN 总线与车体主控通信，接收命令并实时反馈状态。也可使用为此模块开发的上位机进行控制，由于上位机主要在调试时使用，故其有比主控更高的控制优先级，目前与上位机的通信是通过在串口通信基础上加入自定义通信协议完成的。

由于该模块将在比赛时同时使用 4 个以上，操作简便并且易于检查就成了另一个设计目标。为了便于检查模块的工作状态，模块上预留了一个 **OLED** 接口，只要讲屏幕接入模块便可以直接查看当前模块的工作状态。同时在模块一边预留了 3 枚 WS2812 **RGB 彩灯**，可以通过三枚彩灯的颜色和闪烁状态快速地知道模块目前的剩余电量、与车体主控的连接状态以及整个系统的运行情况。加入**上位机**则可以更为清晰全面地了解车体运动时的模块工作状态，并可以进行快速地测试。

为了便于调试，该模块预留了给 IMU 使用的 SPI 接口，可以在主控使用不同功率控制策略时对车体的加速度与速度进行回传并通过上位机进行可视化展示以评估其效果。加入 EEPROM AT24C02 则可以将部分配置与调试数据持久保存，不会由于掉电丢失，解决了每次调试完成后需要修改代码才能修改参数的麻烦。

3 架构设计

3.1 硬件架构

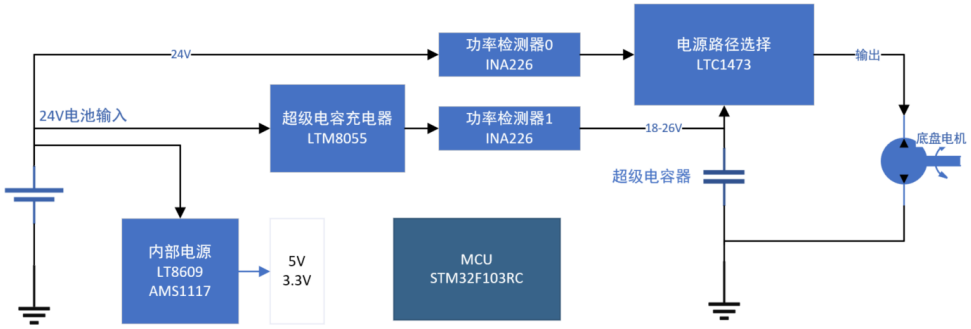

图 3.1 主要电源路径

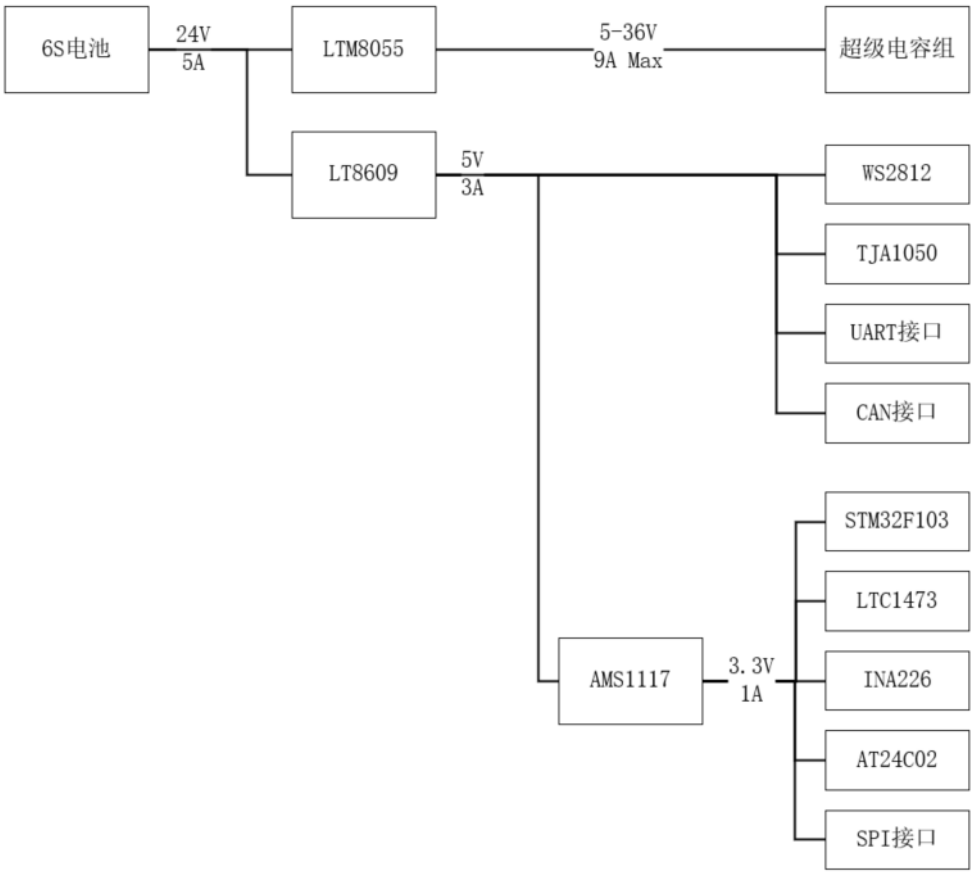

图 3.2 Power Tree

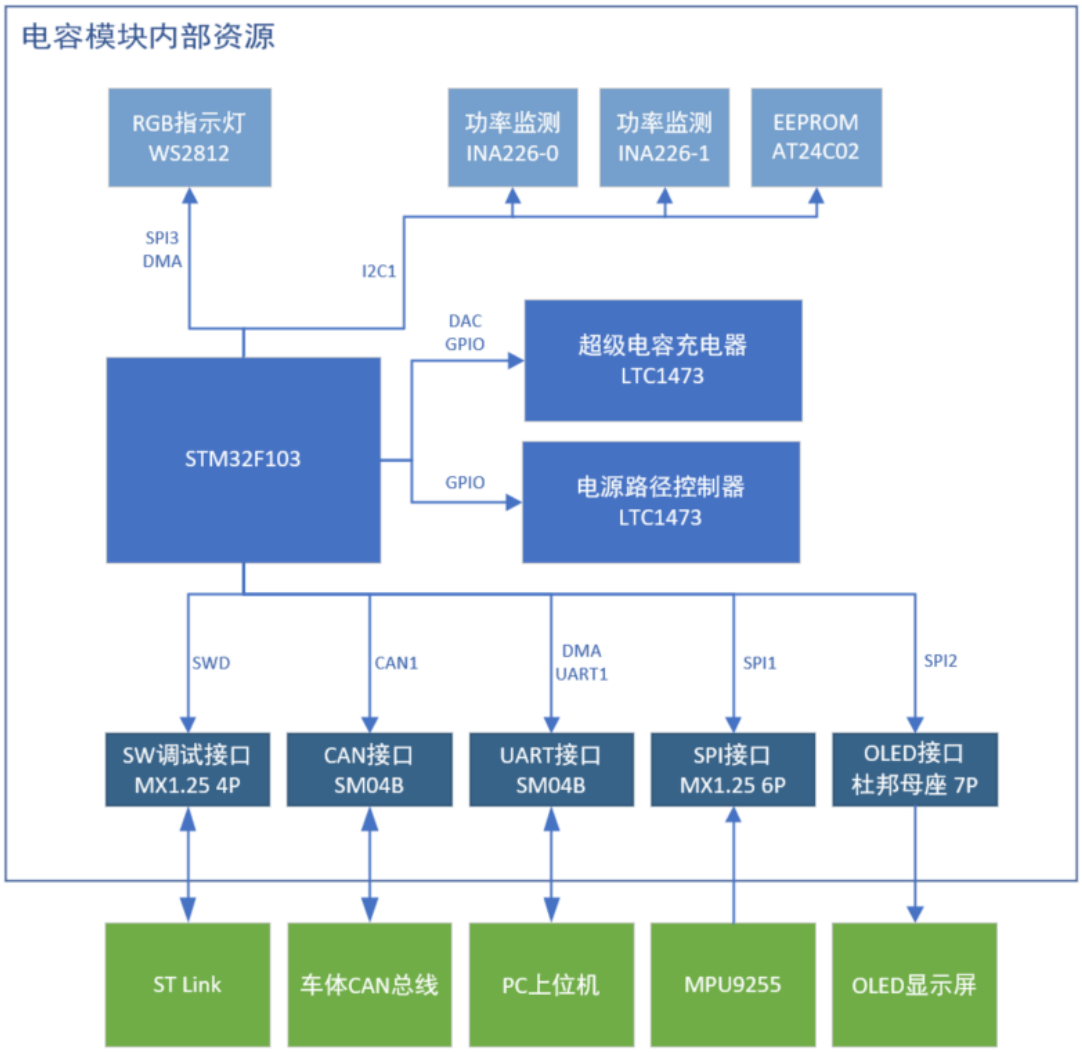

图 3.3 硬件资源框图

3.2 嵌入式软件架构

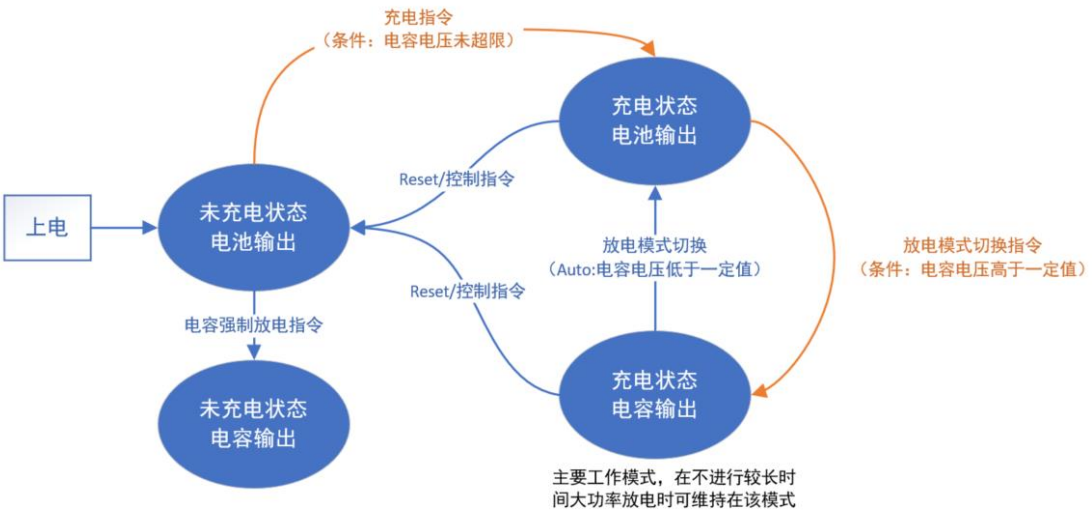

图 3.4 电容模块控制状态图

本项目嵌入式软件设计的核心就是按上图逻辑正确切换系统工作状态。首先需要对各个硬件资源进行正确的初始化并控制，其次则是可以正确的监测模块的运行情况并接收来自 CAN 总线或串口的控制指令，进而对模块的工作状态进行相应的切换。同时要将系统的当前状态准确及时地告知主控(通过 CAN)，上位机(通过串口)或是进行现场调试的人(通过 RGB 灯、OLED)

3.3 上位机软件架构

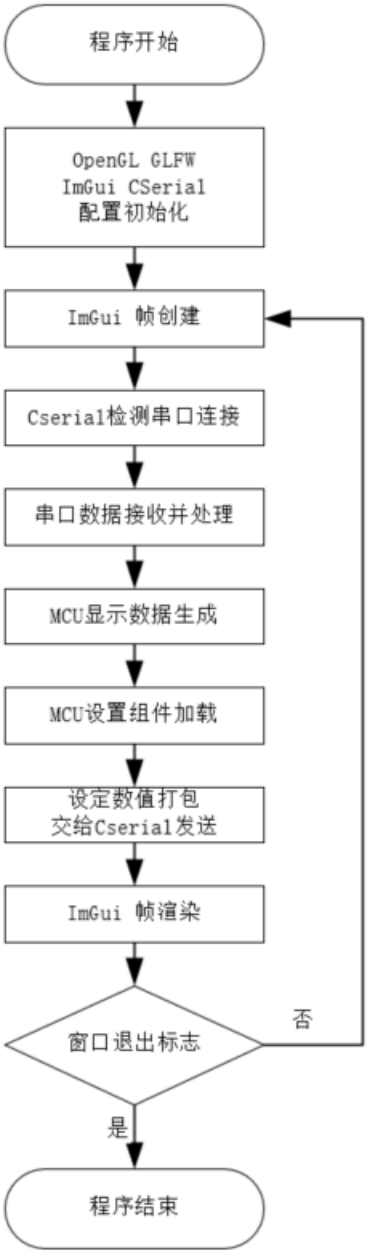

图 3.0 上位机程序流程图

4 硬件设计

设计所用的软件: Altium Designer 19.0.4

4.1 核心器件选型

本项目使用的芯片按功能可大致分为：MCU、充电控制器、功率监测器、电源路径控制器、内部供电芯片、存储芯片、CAN 收发器等、IMU。下面给出芯片的具体选型与介绍。

MCU：STM32F103RCT6 这是一款中低端的 32 位 ARM 微控制器，CPU 最高速度达 72 MHz，RAM 48KB FLASH 256KB，具备项目所需的所有外设且价格低廉，满足我们的需求。同时该芯片与本课程讲授使用芯片类型相似，可加强组员对课程学习的理解。

充电控制器：LTM8055 这一款具有 5V 至 36V 输入范围的降压-升压型稳压器，并可容易地通过并联以扩展负载电流能力。该器件的四开关降压-升压型拓扑具有高效率，同时允许输入电压低于、等于或高于输出。LTM8055 能够调节一个恒定电压 (CV) 和一个恒定电流 (CC) 输出，以及具有输出电流的模拟编程。在项目中的主要利用了其可编程输出电流的特点，在 24V 输入的情况下，对于工作在 18~26V 的电容组进行可设定的恒功率充电。在使用单独一个 LTM8055 的情况下最大可输出 180W，满足项目需求。由于其在紧凑的封装中集成了完整的电源功能，且仅需少量外围器件支持，极大的简化了本课程设计中电源部分的调试与设计难度，加快了开发速度。

功率监测器：INA226 这是一款 36V、双向、超高精度的 I2C 输出电流/功率监控器。该器件监视分流压降和总线电源电压。其可编程校准值、转换时间和取平均值功能与内部乘法器相结合，可实现电流值（A）和功率值（W）的直接读取。使用 INA226 芯片可以简单且精确地测量模块的功率信息，便于更好的控制与后期的优化。

电源路径控制器：LTC1473 这是 ADI 公司的双通道 PowerPath 开关驱动器。该芯片通过驱动两组背对背 N 沟道 MOSFET 实现了开关功能，使用全 N 沟道开关管可降低功率损失和成本。通过 LTC1473 可以快速灵活地切换模块的输出模式，消除了可能由于模式切换对于底盘运动的影响。

内部供电芯片：

LT8609 这是宽输入电压范围 42V、3A 同步降压稳压器，在本项目中提供 5V 模块内电源，并可对部分外部模块提供较大电流的供电。

AMS1117-3.3 这是一款使用非常广泛的 3.3V 线性降压器，可提供至多 1A 的输出电流，在本项目中提供 3.3V 模块内电源

CAN 收发器：TJA1050 这是 NXP 的高速 CAN 收发器，提供了控制器区域网络(CAN)协议控制器和物理总线之间的接口。该器件为总线提供差分发射能力并为 CAN 控制器提供差分接收能力。

IMU：MPU9255 是内部集成陀螺仪、加速度级、磁力计的 9 轴姿态传感器。通过陀螺仪检测物体处于运动状态时的角速度，再利用算法结算成当前处于的角度。加速度计和磁力计分别测量加速度和磁感应强度。本项目中主要是测量模块所在底盘的加速度并进行速度计算，评估模块对于车体运动性能的提升，并对功率控制算法的优化提供了依据。

4.2 硬件原理图

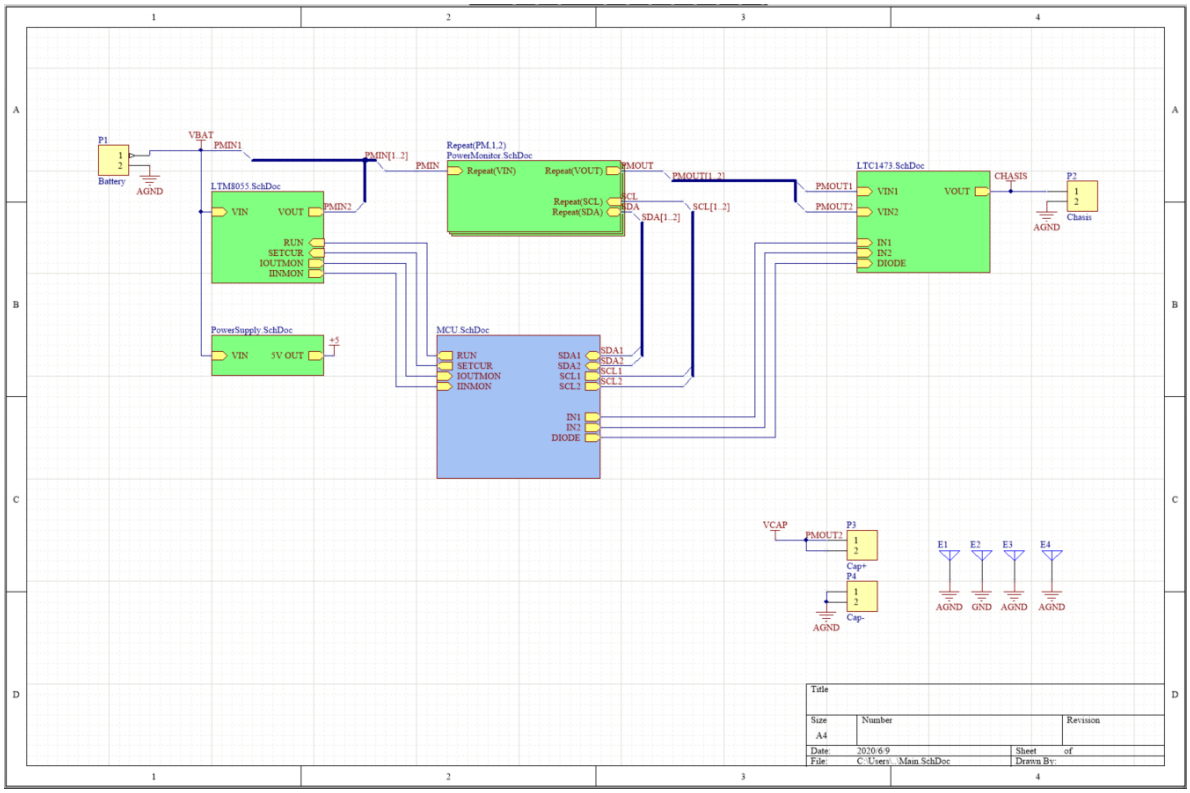

图 4.1 电容模块母图(总览)

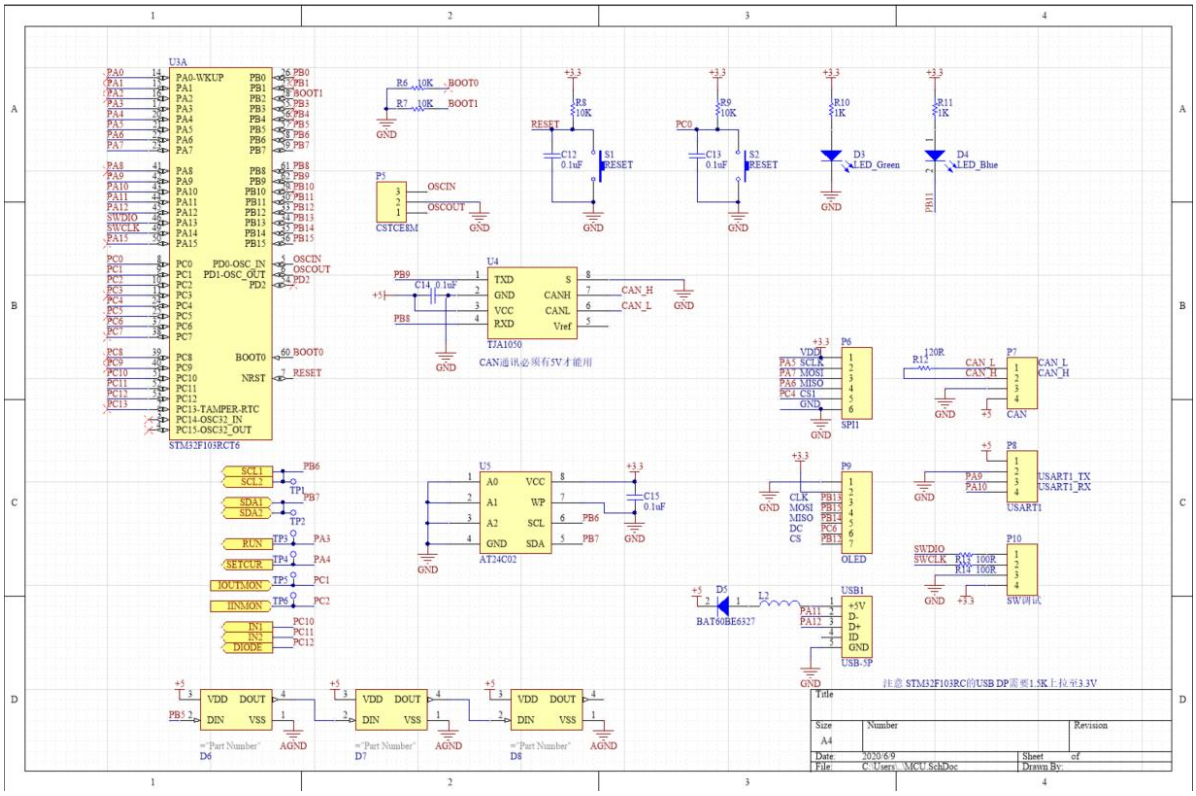

图 4. 2 电容模块子图 1: MCU

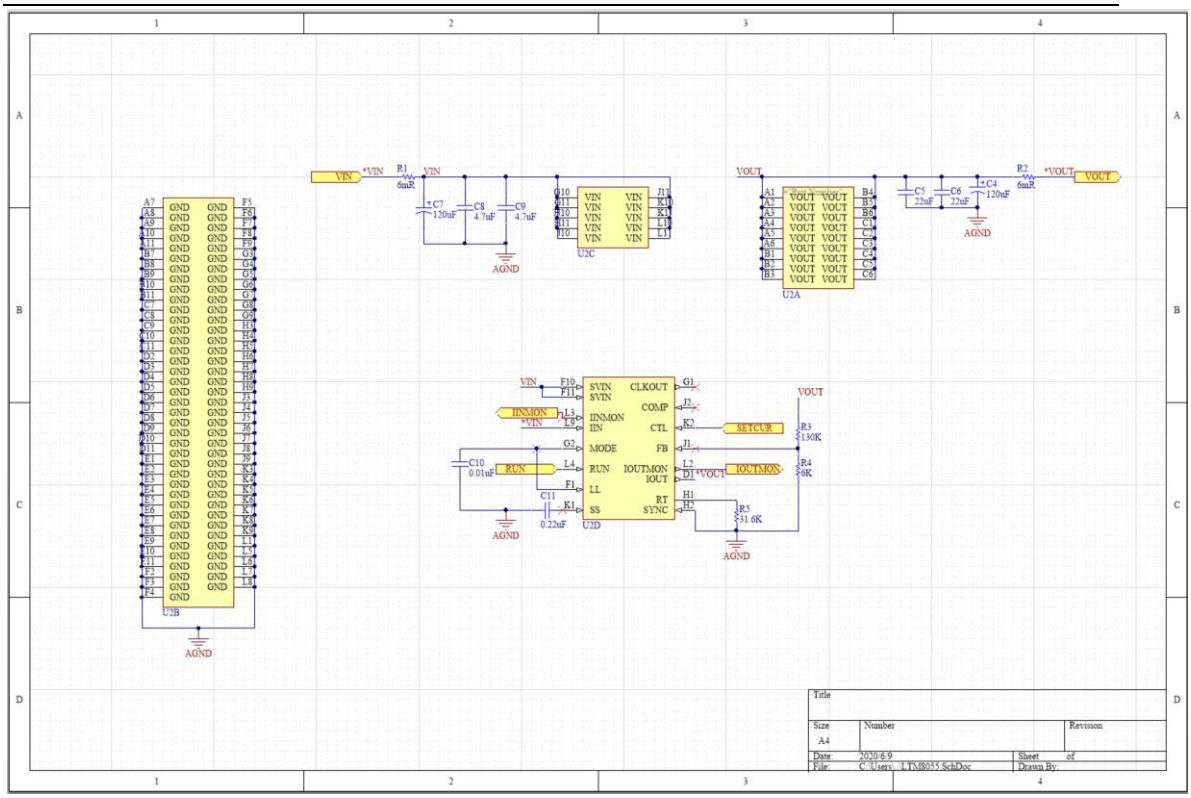

图 4.3 电容模块子图 2：充电控制器

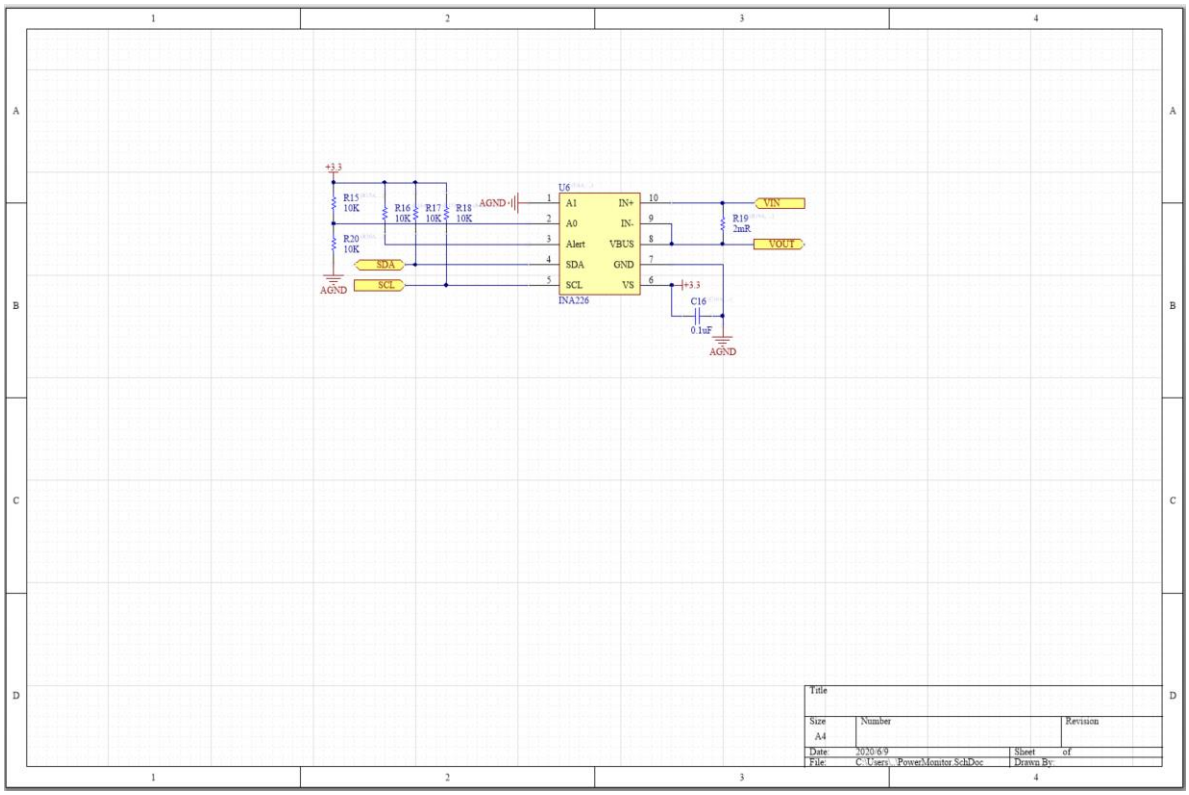

图 4.4 电容模块子图 3：功率监测器

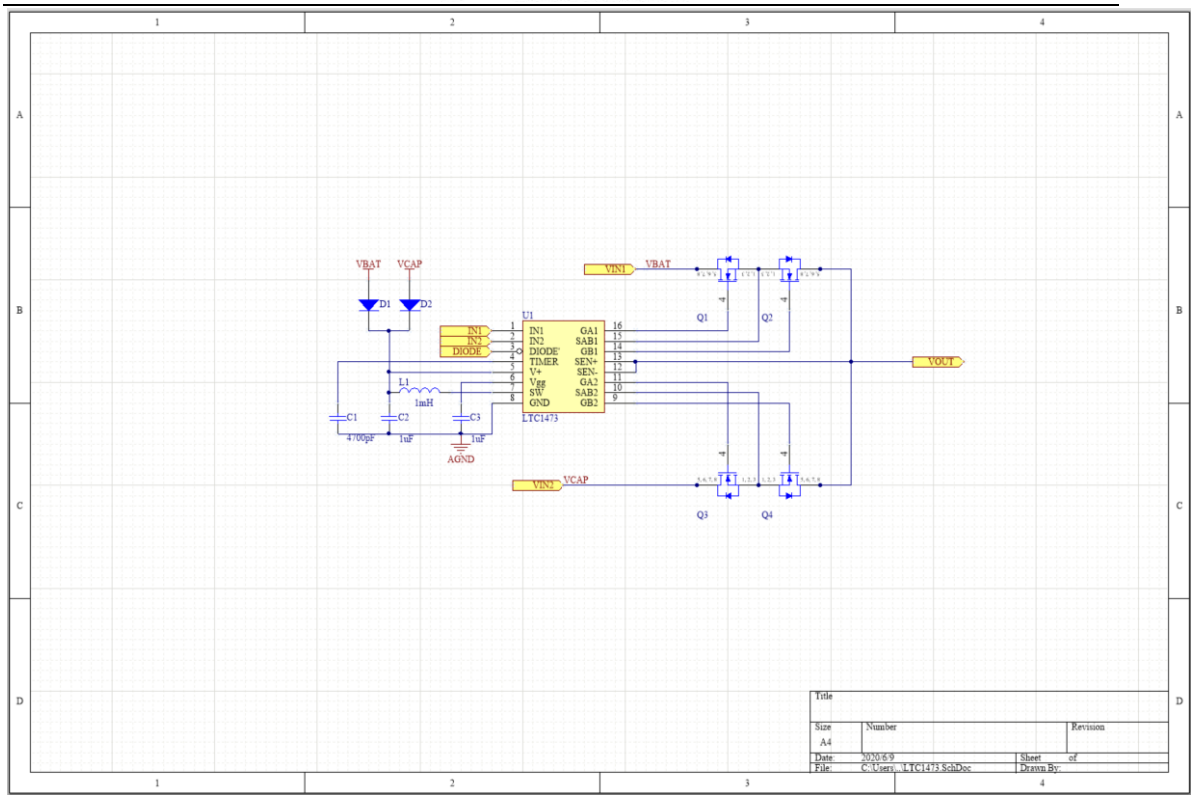

图 4.5 电容模块子图 4: 电源路径控制器

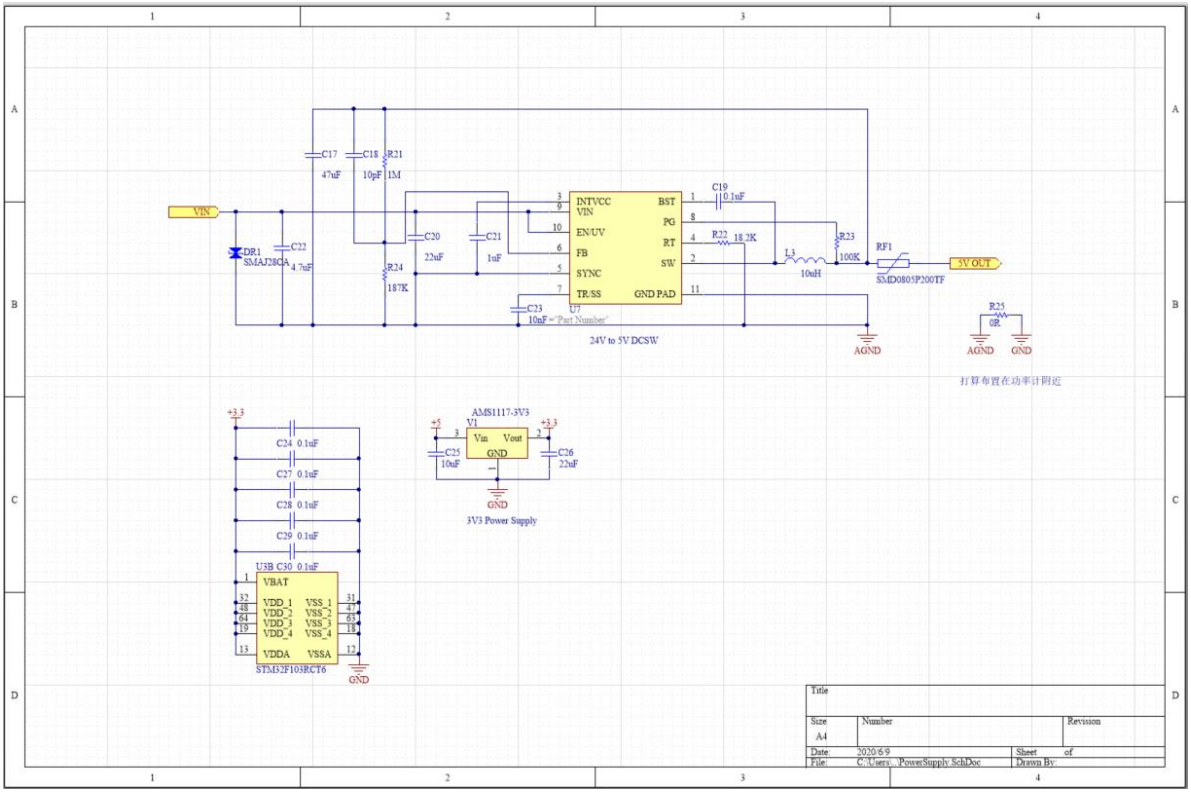

图 4.6 电容模块子图 5:内部电源

4.3 原理图清单及功能说明

原理图文件名	名称	功能说明
MAIN.SCHDOC	原理图母图	连接各个子图，构成整体
MCU.SCHDOC	微控制器	整个系统的主控核心
POWERSUPPLY.SCHDOC	模块供电	模块 5V、3.3V 供电
POWERMONITOR.SCHDOC	功率监测模块	实时检测电压、电流、功率
LTM8055.SCHDOC	LTM8255 μ Module 稳压器	超级电容充电器
LTC1473.SCHDOC	LTC1473 电源路径控制模块	模块电源输出路径控制

表 4.2 模块清单及功能

4.4 PCB Layout

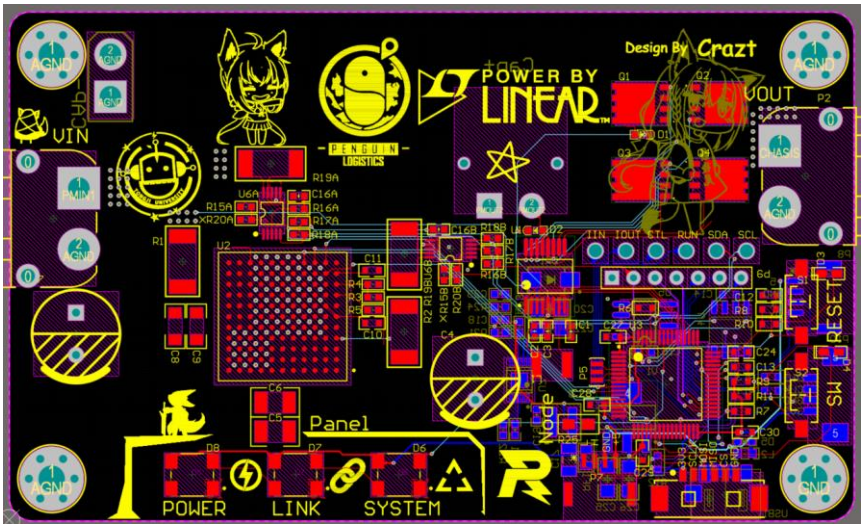

图 4.7 隐藏铺铜后的 PCB 2D 视图

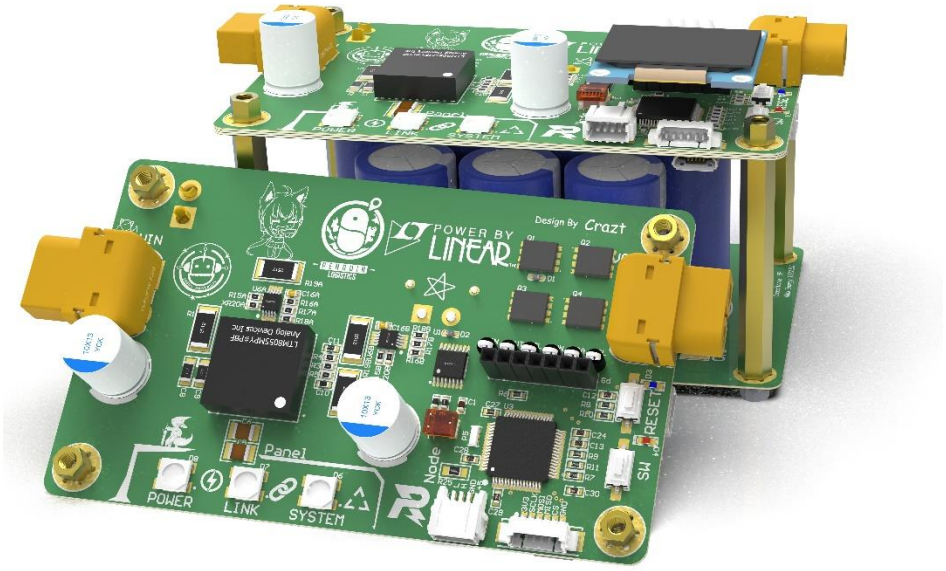

图 4.8 3D 渲染图

4.5 原材料清单

1	Comment	Value	Footprint	Description	Designator	Quantity
2	Cap	4700pF	C 0603_L	X5R 50V	C1	10
3	Cap	1uF	C 0603_L	X5R 50V	C2, C3, C20	10
4	Cap	22uF	1210C	X5R 50V	C5, C6	5
5	Cap	4.7uF	1206C	X5R 50V	C8, C9	5
6	Cap	0.01uF	0603C	X5R 50V	C10	5
7	Cap	0.22uF	0603C	X5R 50V	C11	5
8	Cap	0.1uF	0603C	X5R 50V	C12, C13, C14, C23,	50
9	Cap	0.1uF	C 0603_L	X5R 50V	C15A, C15B, C18	5
10	Cap	47uF	C1206	X5R 25V	C16	5
11	Cap	10pF	C 0603_L	C0G 50V	C17	5
12	Cap	22uF	C 1206_L	X5R 50V	C19, C25	10
13	Cap	4.7uF	C 1206_L	X5R 50V	C21	5
14	Cap	10nF	C 0603_L	C0G 50V	C22	5
15	Cap	10uF	C 1206_L	X5R 25V	C24	10
16	LED0		LED 0603B	Green	D3	5
17	LED1		LED 0603B	Blue	D4	5
18	Res1	6mR	R 2512_L	Resistor	R1, R2	30
19	Res1	130K	R 0603_L	Resistor	R3	30
20	Res1	6K	R 0603_L	Resistor	R4	30
21	Res1	31.6K	R 0603_L	Resistor	R5	30
22	Res1	10K	0603R	Resistor	R6, R7, R8, R9	30
23	Res1	1K	0603R	Resistor	R10, R11	30
24	Res1	120R	0603R	Resistor	R12	30
25	Res1	100R	0603R	Resistor	R13, R14	30
26	Res1	10K	R 0603_L	Resistor	R15A, R15B, R16A, F	30
27	Res1	2mR	R 2512_L	Resistor	R19A, R19B	30
28	Res1	1M	R 0603_L	Resistor	R21	30
29	Res1	18.2K	R 0603_L	Resistor	R22	30
30	Res1	100K	R 0603_L	Resistor	R23	30
31	Res1	187K	R 0603_L	Resistor	R24	30
32	Res1	0R	0805R	Resistor	R25	30
33	SMD0805P200TF		R 0805_L	PPTC自恢复保险丝	RF1	30
34	WS2812		Adafruit_1655_0	RGB LED	D6, D7, D8	10
35	Diode		SOD-523	SS14	D1, D2	10
36	Inductor	mH	L 0603	MPZ1608S181ATAH0	L2	50
37	LTC1473CGN		SSOP16_N		U1	3
38	LTM8055EY		DWG# 05-08-1891		U2	2
39	STM32F103RCT6		STM-LQFP64_N	ARM Cortex-M3 32-bit MCU, 256 KB Flash	U3	4
40	TJA1050		SO8	CAN收发器	U4	5
41	INA226AIDGSR		MSOP10_N		U5A, U5B	5
42	LT8609EMSE#PBF		LT8609EMS		U6	3
43	AMS1117-3V3		SOT223_N	三端稳压芯片	V1	10
44	BSC028N06NS		INF-PG-TDSON-8-	N-Channel OptiMOS 2 Power-Transistor, 1	Q1, Q2, Q3, Q4	10
45	CSTCE8M00G52-R0		CSTCE16M	Header, 3-Pin	P5	10
46	SMAJ28CA		SMA	瞬态电压抑制二极管	DR1	10
47	BAT60BE6327		SOD-523	Diode	D5	10
48	Battery		TX60PW-M20	XT电源接口	P1	2
49	Chasis		TX60PW-F20	XT电源接口	P2	2
50	Cap+		XT30PW-F	XT电源接口	P3	2
51	Cap-		XT30U-M	XT电源接口	P4	2
52	SPI1		MX 1.25-WS-6P	Header, 6-Pin	P6	2
53	CAN		SM04B	Header, 4-Pin	P7	2
54	USART1		SM04B	Header, 4-Pin	P8	2
55	OLED		HDR2.54-M-LI-7P	OLED Connector 7P杜邦母头	P9	2
56	SW调试		MX 1.25-WS-4P	Connector MX1.25 4P	P10	2
57	USB-5P		USB-MICRO_E	微型USB母座	USB1	5
58	RESET		TSW SMD-3*6*2.5	Switch	S1, S2	5
59	Cap Pol1	120uF	CS 5.0*10*13 - C	Polarized Capacitor (Radial)	C4, C7	5
60	TestPoint		TP_Black(1.6mm)	PCB测试点	TP1, TP2, TP3, TP4, T	10
61	Inductor	10uH	1812LS-105XKBC	1812LS-105XKBC	L1	10
62	Inductor	10uH	XAL4040	XAL4040 10uH	L3	10

表 4.1 原材料清单（含冗余量）

5 嵌入式软件设计

设计所用的软件: ST Cube Mx、ST Cube IDE

5.1 嵌入式软件架构图

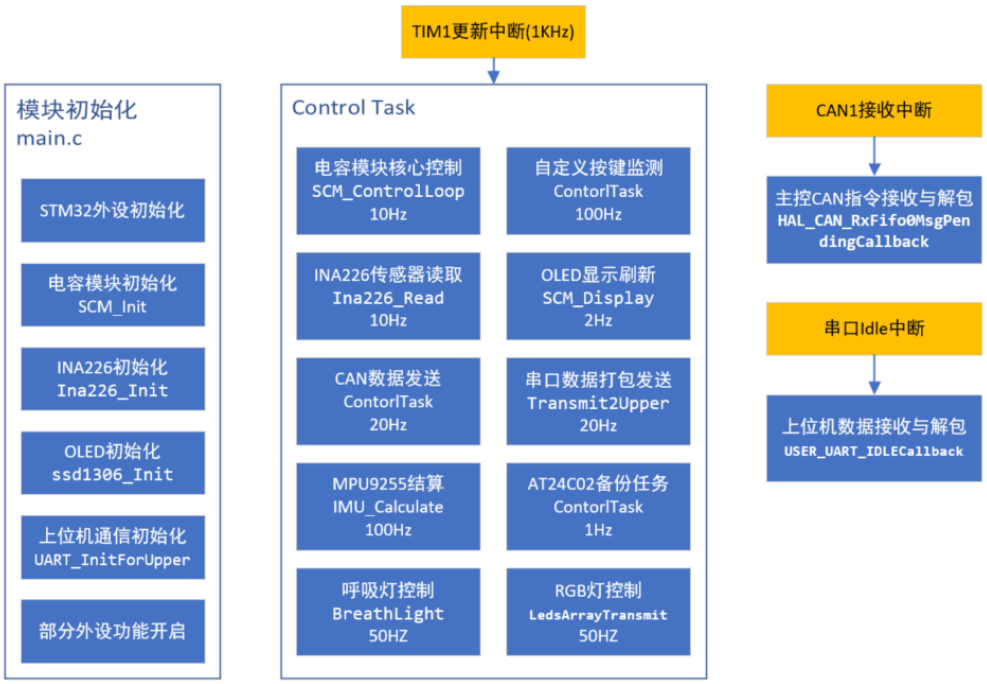

图 5.1 嵌入式软件架构图

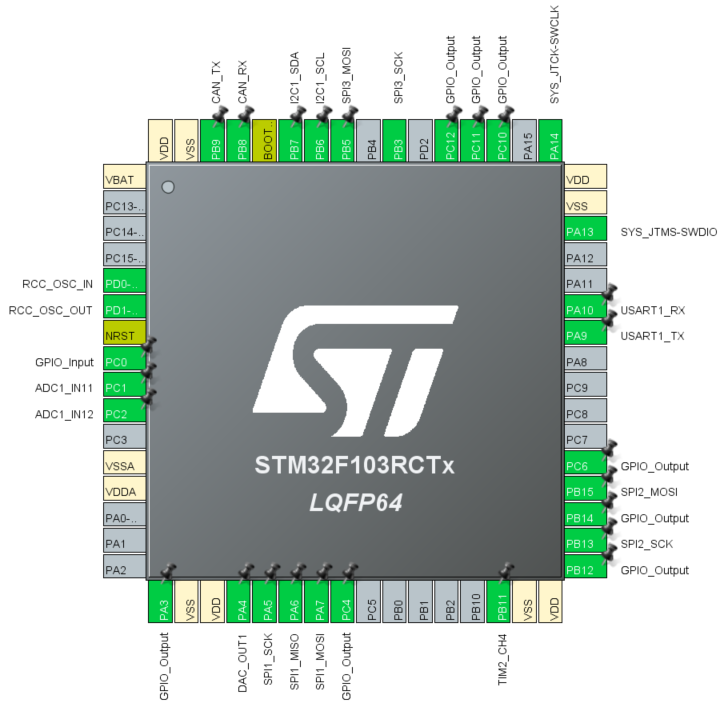

图 5.2 STM32 引脚配置

5.2 核心代码概述

超级电容模块核心功能控制
文件：SCM.h　SCM.c
描述：电容模块核心功能程序，负责控制
主要函数：

```
void SCM_Init(void);//超级电容模块初始化
void SCM_ControlLoop(void);//超级电容控制主循环
void SCM_SetCurrent(int32_t SetCur);//设置电流值
void SCM_SetPower(int32_t SetPow); //设置功率值
void SCM_MonitorUpdate(void);//电容、电池的电压电流功率数据更新
void SCM_Display(void); //在屏幕上显示电容、电池的电压电流功率数据
```

控制任务：
文件：ControlTask.h　ControlTask.c
描述：电容模块的核心控制程序，模块大部分功能在这里完成，由 TIM1 更新中断调用 ControlTask 来定时执行，周期为 1ms
主要函数：

```
void ControlTask(void);//主任务循环
void ButtonSense(void);//按键检测
```

上位机：使用串口进行通信
文件:upper.h　upper.c
描述：电容模块与上位机通信的相关函数，包含串口收发以及数据的打包与解包。
主要函数：

```
void UART_InitForUpper(UART_HandleTypeDef *huart);//上位机通信串口初始化
void USER_UART_IRQHandler(UART_HandleTypeDef *huart);//串口中断处理函数
void USER_UART_IDLECallback(UART_HandleTypeDef *huart);//串口 idle 回调函数
void USER_UART1_DMA_Transmit2Upper(); //通过串口 1 以 DMA 方式向上位机发送数据
uint8_t crc_check(SerialDataType* Data,uint16_t Len);//对接收数据进行 crc 校验
```

陀螺仪运动结算：通过 SPI 进行通信
文件：MPU9255.h　MPU9255.c
描述：对 IMU 进行初始化、读取原始值、转化为真实值、去零飘、静止自检等
主要函数：

```
void mpuCsEnable();
void mpuCsDisable();//片选引脚使能和禁用
uint8_t SPI2_WriteReadByte(uint8_t TxData);//spi 读写函数
void MPU9255_Init(void);//mpu9255 初始化
uint8_t MPU9255_Write_Reg(uint8_t reg,uint8_t value);//spi 发送
```

```
uint8_t MPU9255_Read_Reg(uint8_t reg,uint16_t *value);//spi 读取
void MPU9255_Read(struct MPU9255_RAW_DATA *MPU9255_raw_data);//mpu9255 原始数据读取
int MPU9255_ConvertData(struct MPU9255_RAW_DATA raw_data, struct MPU9255_REAL_DATA *real_data,int cnt);  //mpu9255 转化为真实数据
void MPU9255_ZeroOffset(void);//去零飘
void MPU9255_Print(struct MPU9255_REAL_DATA real_data);//串口打印 //调试
```

功率监测模块 INA226：使用 IIC 进行通信
文件：ina226.h　ina226.c
描述：Ina226 的驱动程序，用于电压电流功率监测
主要函数：

```
void Ina226_Init(void);//Ina226 初始化函数
void Ina226_Read(void);//Ina226 读取电压、电流、功率
```

OLED 显示模块 SSD1306：使用 SPI 进行通信
文件：ssd1306.h　ssd1306.c　ssd1306_font.h
ssd1306_font.c　ssd1306_test.h ssd1306_test.c
描述：常见的 OLED 64*128 模块的驱动程序，用于模块状态显示
主要函数：

```
void ssd1306_Init(void); //oled 屏幕初始化
ssd1306_Fill(SSD1306_COLOR color);//使整个屏幕为指定颜色
ssd1306_UpdateScreen(void);//更新屏幕
void ssd1306_DrawPixel(uint8_t x, uint8_t y, SSD1306_COLOR color);
//在屏幕缓冲区中画一个像素
char ssd1306_WriteChar(char ch, FontDef Font, SSD1306_COLOR color);
//在屏幕缓冲区中画一个符号
char ssd1306_WriteString(char* str, FontDef Font, SSD1306_COLOR color);
//在屏幕缓冲区中画一个字符串
void ssd1306_SetCursor(uint8_t x, uint8_t y); //定位游标
void ssd1306_Reset(void); //重置 OLED
void ssd1306_WriteCommand(uint8_t byte); //向命令寄存器发送一字节
void ssd1306_WriteData(uint8_t* buffer, size_t buff_size); //发送数据
```

EEPROM AT24C02 模块：通过 IIC 进行通信
文件：at24c02.h　at24c02.c
描述：存储器的驱动程序，该存储器主要用于备份及恢复模块的参数设定，放止掉电丢失
主要函数：

```
void SetATbuffer(void);//设置要发送的电容设置数据及校验码
void AT24C02_Write_Settings1(void);//向 EEPROM 写入要发送的数据（缓存 1）
```

```
3.  void AT24C02_Write_Settings2(void);//向 EEPROM 写入要发送的数据（缓存 2）
4.  void AT24C02_Read_Settings(void); //根据校验码从 EEPROM 中读取正确的设置数据
```

RGB WS2812 数控彩灯：通过 SPI 进行通信
文件：WS2812.h　　WS2812.c
描述：WS2812 的彩灯驱动，主要用于模块状态显示
主要函数：

```
7.  void RGB_LED_Write_Byte(uint32_t byte);
8.  //把 32 位设置数据转换为驱动 WS2812 的 0 码和 1 码并存入数组
9.  void RGB_LedsArrayTransmit(void);
10. //通过 spi+dma 的方式传送数据给 WS2812 模块
```

crc 校验：
文件：crc32.h　　crc32.c
描述：用于串口接收数据后计算数据帧的校验值是否正确
主要函数：

```
1.  uint32_t crc32(unsigned char* s, int len)//crc 校验码计算
```

5.3　电容模块与车体主控的通信协议

为了使用 CAN 总线实现电容模块与车体主控的信息交互，我们确定了一套基于 CAN 通信的通信协议，主要用于状态信息的交互和控制命令的发送。具体细节如下：

电容模块接收报文：

电容模块接受报文		
标识符	0x300	
帧类型	标准帧	
帧格式	DATA	
DLC	8	
DATA[0]	ControlData	配置见下方
DATA[1]	RESERVE	
DATA[2]	充电功率高八位	
DATA[3]	充电功率低八位	LSB=25mW
DATA[4]	RESERVE	
DATA[5]	RESERVE	
DATA[6]	RESERVE	
DATA[7]	RESERVE	

其中 ControlData 的约定控制效果如下

		Value=0	Value=1
ControlData[0]	充电使能	关闭	开启
ControlData[1]	电源路径	使用电池	使用电容
ControlData[2]	电容强制放电	否	是
ControlData[3]	RESERVE		
ControlData[4]	RESERVE		
ControlData[5]	RESERVE		
ControlData[6]	RESERVE		
ControlData[7]	RESERVE		

电容模块发送报文：

模块反馈报文		
标识符	0x301	
帧类型	标准帧	
帧格式	DATA	
DLC	8	
DATA[0]	FeedbackData	配置见下方
DATA[1]	电容可用电量	LSB=1%
DATA[2]	充电功率高八位	
DATA[3]	充电功率低八位	LSB=25mW
DATA[4]	底盘功率高八位	有极性
DATA[5]	底盘功率低八位	LSB=25mW
DATA[6]	电容电压高八位	有极性
DATA[7]	电容电压低八位	LSB=1.25mV

其中 ControlData 的内容约定如下

		Value=0	Value=1
FeedbackData[0]	充电使能状态	关闭	开启
FeedbackData[1]	电源路径状态	使用电池	使用电容
FeedbackData[2]	RESERVE		
FeedbackData[3]	RESERVE		
FeedbackData[4]	RESERVE		
FeedbackData[5]	RESERVE		
FeedbackData[6]	RESERVE		
FeedbackData[7]	RESERVE		

6 上位机软件设计

开发所用的软件: Visual Studio 2019 Community

6.1 上位机简介

上位机的项目命名为：SuperPowerClient，该项目是为了方便对 TongJi-SuperPower 战队超级电容模块进行调试而搭建的上位机，同时也可拓展加入其他调试功能。目前上位机的主要功能有以下两个：

- 实时显示电容模块的使用状态，并可以通过上位机进行部分功能的设置
- 通过 IMU 对于车体运动进行分析，评估电容模块的实际使用效果

上位机主要由以下功能模块组成

使用的库/工具	功能模块	资源
OPENGL 3	图形绘制	参考教程[4]
DEAR IMGUI	用户界面	Github 项目地址[5]
CSERIAL	串口通信	CSerial A C++ Class for Serial Com[6]

表 6.1

通信方式目前采用串口，只需要连接蓝牙模块就可以简单的实现无线传输，并且帧尾校验的加入也使得在通信连接情况较差的情况下尽量减少使用错误的数据。

6.2 上位机界面与功能模块开发

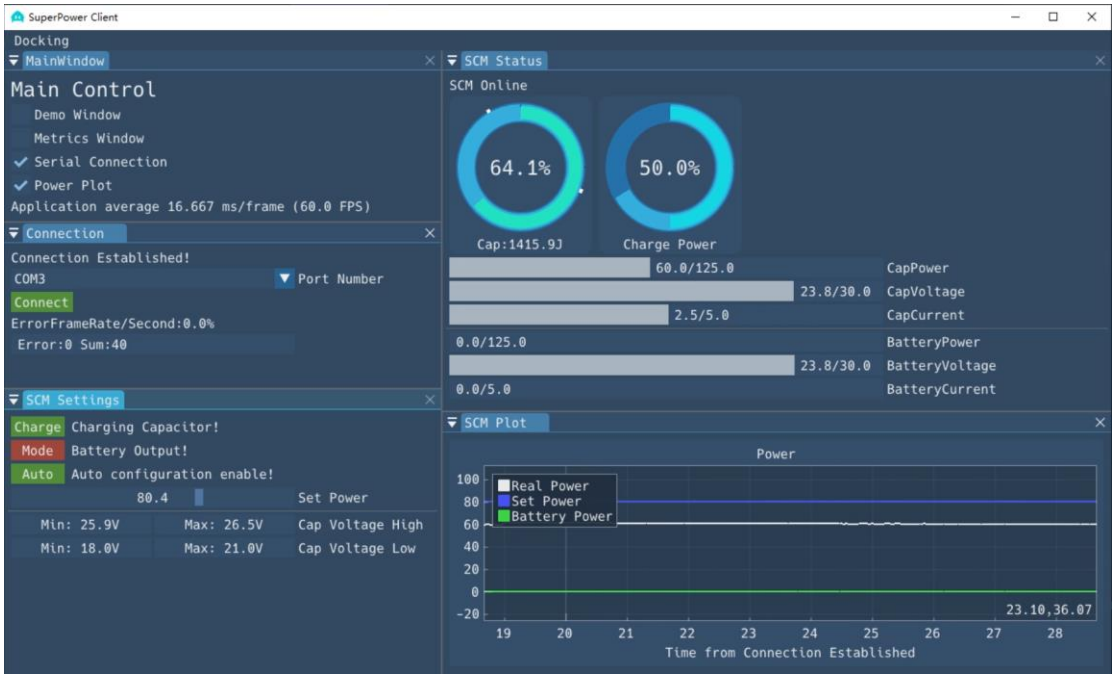

图 6.1 上位机界面图

该程序直接点击 Release 中的 SuperPowerClient.exe 就可直接运行

上图便是上位机目前的界面，左半边主要是功能设置，右半边则是采集到的数据展示。设置中的 Main Window 负责控制 UI 界面的窗口开关，为总控窗口。Connection 则负责串口连接控制，可以选择并连接我们需要连接的串口，连接成功后会统计通信中的错误帧数及错误帧率。SCM Settings 则是电容模块的主要控制窗口，可以控制电容模块的基本运行情况。右侧的数据显示部分则可以实时显示电容当前电量、充电的电压电流功率，以及电容模块电池直接输出的电压电流功率，下方的绘图窗口则可观察自己需要的变量，便于调试分析。

上位机的的窗口使用 GLFW 创建并采用 OpenGL 绘制，使用了 GLAD 加载函数位置，这是一套较为常见的图形学程序框架。在按教程配置好窗口后，将 Dear ImGui 的基础文件整合进我们的项目文件中，再进行一些调试后就可以正确的绘制出 ImGui 的 Demo 界面了。

为了便于使用，本项目采用了 Dear ImGui Docking Branch ，这样可以灵活地调整窗口的布局，也可以将部分组件拉出渲染窗口，在屏幕的其他部分显示。同时编写了 imgui_customs 文件来存放我们需要的部分小组件，比如上图中的圆环双层进度指示器就是自行编写的，其他大部分组件已经在 UI 中自带或是可以在 Github Issue 中找到。

6.3 MCU 与上位机的通讯协议

为了在 MCU 与上位机间收发多种数据，需要拟定一个协议进行数据收发。我们自行拟定了一种通信协议，每个数据帧（Data frame）由帧头、数据部分、帧尾三部分组成。我们可以通过帧头识别出每种帧的数据长度，并且有差错校验以及帧的匹配算法，经测试可以较为稳定的传输。**协议标准**如下

帧组成	**说明**	**长度 BYTE**
“SP”	每一段数据的起始，SuperPower 首字母	2
帧类型编号	以一个 Byte 来确定发送内容的相应类型，同时确定之后数据的长度	2
内容	实际传送的内容	可变
CRC 校验	此帧中前面内容的 CRC 校验值	4

表 6.2

备注：表中的“SP”与帧类型编号共同构成了帧头

为了便于开发，这里统一每 4Byte 为一个小包，为一类数据的最小打包单位，在编程时使用 C Union 可以简洁地进行数据转换。在使用较常用的串口波特率 115200，并配置为 1 停止位无校验位时，每秒最多可发送约 115200/9=12800Byte 信息，如果每个包长为 20，那么每秒大约可发送 640 个包，大约可以使信息的更新频率维持在数百 Hz，可以满足基础的调试需要。

实际使用时按数据发送方向将协议分为以下两种

6.3.1 上位机对下位机发送的具体协议

帧类型	*帧头*	*帧内容*	*帧总长 Byte*	*备注*
控制帧	“SP00”	控制信息	4+1*4+4	
IMU 帧	“SP01″	accel[3] gyro[3] temp	4+7*4+4	此方向禁用
电容帧	”SP02“	超级电容状态及控制	4+2*4+4	
电容配置帧	”SP03“	超级电容部分配置	4+4*4+4	

不同帧的具体内容如下

SP00 控制帧 4Byte	*数据类型*	*位置 bit in 4Byte*
IMU 数据传输使能	BOOL	0
电容数据传输使能	BOOL	1

SP02 电容帧 6*4Byte	***数据类型***	***位置 Byte***
电容模块控制状态	BOOL	0-3
电容充电功率设定	FLOAT	4-8

电容模块控制状态 包含以下内容:

- bit 0 Buck 控制电容模块是否充电 0 否/1 充电
- bit 1 Mode 控制放电模式 0 电池/1 电容
- bit 2 ForceUseCap 设定是否强制电容放电 0 否/1 是 。该设定主要用于调试时强制电容放尽电量，否则模块在电容电压过低时会自动切换回电池放电

SP03 电容设置帧 4*4Byte	***数据类型***	***位置 Byte***
Cap_MaxVol_H	FLOAT	0-3
Cap_MaxVol_L	FLOAT	4-7
Cap_MinVol_H	FLOAT	8-11
Cap_MinVol_L	FLOAT	12-15

6.3.2 下位机对上位机

帧	***帧头 4Byte***	***帧内容***	***帧总长 Byte***	***备注***
控制帧	“SP00”	控制信息	4+1*4+4	此方向禁用
IMU 帧	“SP01″	float accel[3] gyro[3] temp	4+7*4+4	
电容帧	”SP02“	超级电容模块状态及控制	4+8*4+4	
电容配置帧	”SP03“	超级电容部分配置	4+4*4+4	
其他帧	”SP0x″	电池电压，其他传感器数据		暂定

不同帧的具体内容如下

*SP02 电容帧 8*4Byte*	*数据类型*	*位置 Byte*	*备注*
电容模块实际控制状态	BOOL	0-3	与之前相同
*电容**设定**充电功率*	FLOAT	4-7	单位：W
电容实际充电功率	FLOAT	8-11	单位：W
电容实际充电电流	FLOAT	12-15	单位：A
电容实际电压	FLOAT	16-19	单位：V
电池实际输入功率	FLOAT	20-23	单位：W
电池实际输入电流	FLOAT	24-27	单位：A
电池实际输入电压	FLOAT	28-31	单位：V

*SP03 电容配置帧 4*4Byte*	*数据类型*	*位置 Byte*
Cap_MaxVol_H	FLOAT	0-3
Cap_MaxVol_L	FLOAT	4-7
Cap_MinVol_H	FLOAT	8-11
Cap_MinVol_L	FLOAT	12-15

7 测试和效果评估

7.1 测试项目

测试主要依据功能需求逐项测试并整合

内容	需求陈述	备注	测试结果
功能需求	充电功率控制	必备	正常
	放电路径选择(电池/电容)	必备	正常
	总输入功率与电容充电功率监测	必备	正常
	CAN 总线控制	必备	正常
	工作状态指示(OLED,WS2812)	辅助	正常
	配置信息存储与恢复(EEPROM)	增强	已测试 待整合
	车体运动分析(MPU9255)	增强	已测试 待整合
	PC 上位机监测	增强	正常

7.2 测试内容与评价

充电功率控制测试 功能测试正常

充电功率控制的原理是通过 DAC 控制 LTM8055 CTL 引脚的电压便可对其输出的最大电流进行线性调节(见下图)。在确立了充电功率的情况下，通过 INA226 读取此刻电容两端的电压，就可以计算出当前需要设定的最大输出电流。因为电容充电器的输出是直接连接在电容两端的，且超级电容组的内阻相当低(约数百毫欧)，只需要较小的压差充电电流就会达到模块的最大输出电流值，然后便会进行恒流充电。

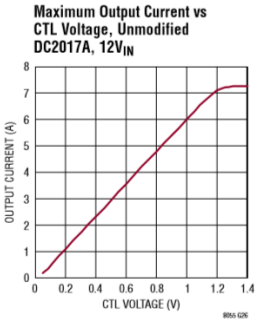

图 7.1 最大输出电压与 CTL 引脚电压的关系[3]

在按上图的规律编写了相应的电流与功率设置函数后，在测试后发现设定功率与实际功率有一定偏差，且对于不同的模块偏差值不太一样，推测这一方面是由于控制参数还未调整好以及 LTM8055 本身的差异性引起的。对于每个模块单独校正即可。

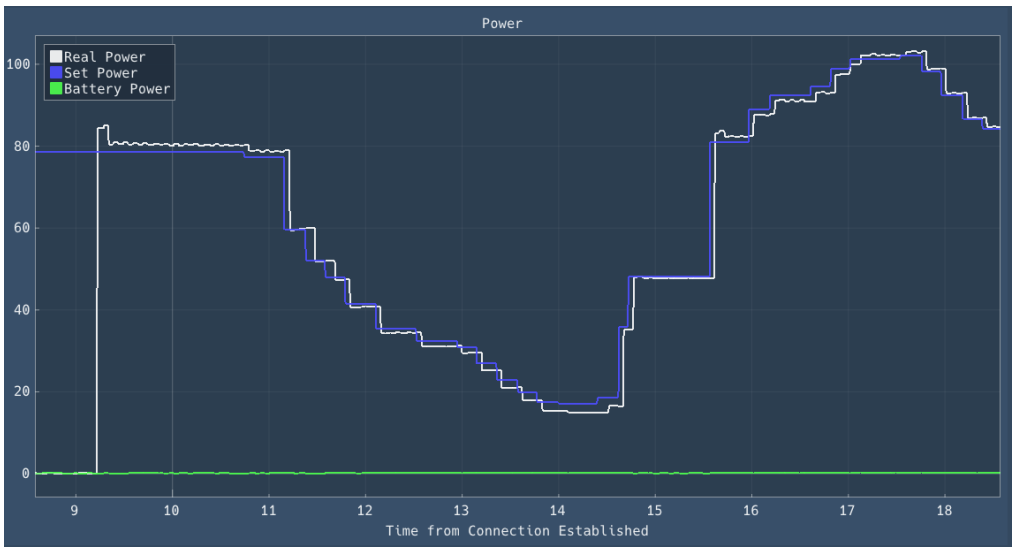

图 7.2 校正后的实际功率响应

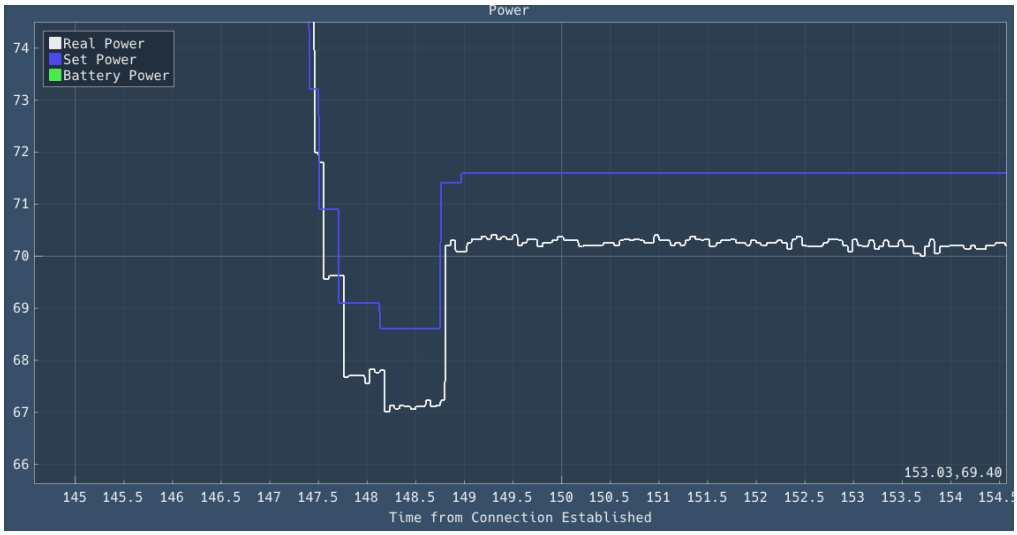

图 7.3 设定值为 71.6W 时的实际功率

经校正后实际充电功率在设计工作条件下的误差不超过 3W，已经较为准确。若想进一步校正需要将采集到的数据进行分析，调整参数，或是考虑加入闭环控制。

放电路径选择测试 功能测试正常

在输出端连接电阻，使用上位机或 Debug 切换模块放电状态。可以正常的在电容放电与电池放电间切换。

总输入功率与电容充电功率监测与 OLED 联合测试

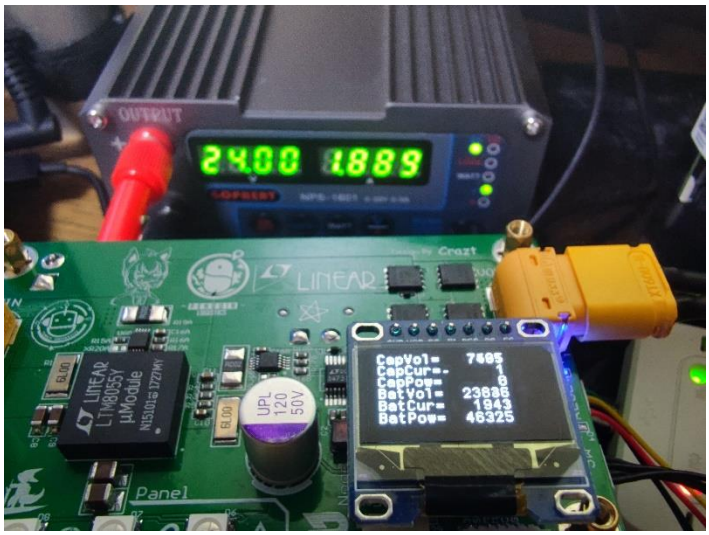

图 7.4 电源示数 1.889A 模块测量值 1.943A 负载为水泥电阻

该部分功能主要是在正常使用时观察 OLED 显示的数值与实际数值是否吻合，在使用电源路径放电的情况下，模块的测量示数与电源的示数相差不超过 100mA,较为准确。更精确的测试在家中缺乏设备，暂时无法进行。

CAN 总线控制 功能测试正常

CAN 总线测试正常。测试首先在 Debug 模式下在 CAN 接收中断里设置断点，然后在模块以及主控端分别进行 Debug，确定二者均可收到对方的数据，再检测数值是否正确。该部分已经过测试。

PC 上位机测试 功能测试正常（综合测试）

主要测试上位机的所有功能是否能正常使用，使用上位机对电容模块进行的测试可见最终调试视频(在提交的文件中，或在线观看)

8 改进、增强与创新设计

8.1 电容模块硬件改进

1. STM32 型号升级

可以考虑使用 F4 系列的芯片，主频更高，外设更为丰富。

该项目已有部分功能受限于 STM32 型号而无法使用：在硬件设计时为了使用虚拟串口，给电容模块加入了 USB 接口。但是在程序编写时发现 STM32F103 的资源有限，无法同时使用 CAN 与 USB 功能。若想同时使用只能更换芯片为 F105 或 F4 系列功能更丰富的 STM32 芯片。

2. 电源路径控制处的设计问题

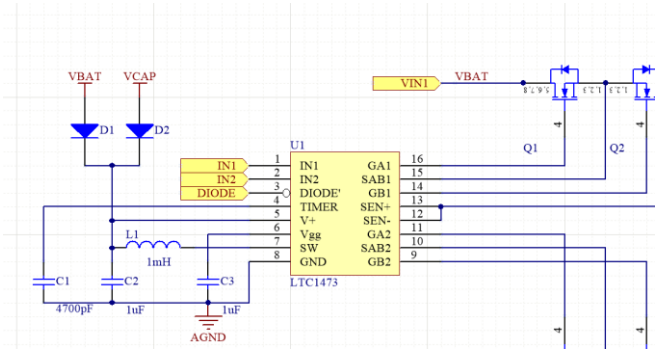

图 8.1 LTC1473.SchDoc 部分原理图

如图在电源路径控制处，使用了两个二极管提供 V+引脚的工作电压。虽然可以保证在两个电源电压差异较大时能正常的切换。但是一旦整个模块断电，芯片不受控制，电容仍会给底盘供电。

解决方案目前拟定有两种：

a)不使用二极管，直接用电源输入为 V+供电，但这又有另一个问题，当电池在使用一段时间后会跌落至 22V 甚至更低的电压，而此时的电容可能有 26V 的电压，这时 MOS 管的栅极电压可能无法让 MOS 管完全导通。这是只能根据电池电压相应地调整电容的充电电压

b)将该芯片的控制引脚全部下拉，这样在系统断电后可能实现 MOS 管全部关断的效果，但该方法还待验证。（可以之后使用飞线尝试）

3. RM 电容管理模块的对接

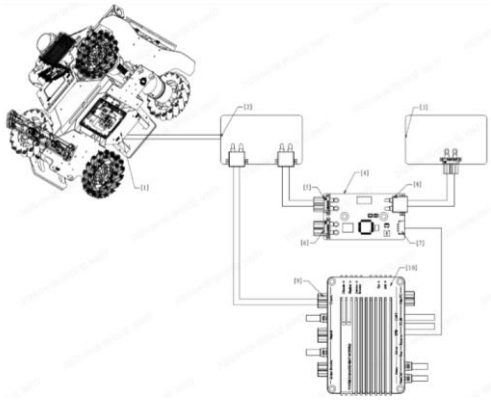

图 8.2 Robomaster 电容管理模块接线示意图

在本电容模块基本完成之后，官方更新了机器人制作规范手册,在电容模块与电容组间加入了电容管理模块用于检测超级电容的容值，以及检测比赛过程中超级电容的容量。虽然本模块可以通过重制连接线使用管理模块，但这会带来额外的麻烦，最好还是按照官方模块在下一版统一电源接口

8.2 电容模块嵌入式程序改进

1. 电容模块控制优化

目前电容模块虽然接管了部分的控制权限，如电容过压保护、低压切换输出模式等，但并没有实际地涉及如何合理的控制充电功率与输出模式以匹配车体的运动。不过若是能较长时间地维持在边充电边放电的模式下，控制也不复杂，今后可以考

虑由此模块接管充电功率的设定以及放电模式的切换，主控只负责提供必要的信息以及简单的控制指令，增强模块的易用性。

2. **对电容充电功率进行闭环控制**

当前电容模块的功率使用开环控制，虽然效果也比较理想，但需要对于每一个模块进行单独调参，较为麻烦。如果使用了闭环控制，如 PID 控制，则在基本参数相同的情况下在不同的模块上均会有较好的效果。

3. **IMU 模块性能优化**

目前虽然可以读出 IMU 数值，但现有滤波算法不够完善，不能得到更精确的实际数值，通过积分计算出速度值误差较大，还需要进一步优化。

MPU9255 还有磁力计等功能，但需要使用 IIC 进行读取，之后可以讲 MCU 预留接口配置为 IIC 模式读取磁力计数值。

8.3 PC 端上位机改进

1. **上位机通信协议的改进**

虽然目前已经明确了协议内容并初步地完成了应用，但由于该协议没有较好的参照一些已有的方案，有一定闭门造车的性质。该协议目前来看还是较为混乱，给使用以及 Debug 带来了一定的麻烦。在完成协议后我才发现已经有相关的实现以及论文，但这部分前期工作却被忽视了，如果能提前多做了解肯定可以制定出更规范易用的串口协议。这也是之后上位机改进的方向之一。

2. **上位机通信方式的改进**

虽然已经使用串口实现了基础的通信功能，可以满足基本的调试需求，但串口传输距离较短，且带宽有限，无法实现数据量较大的采集以及图像类数据的回传，这便限制了上位机的应用场景。

比如在该比赛中加入了飞镖这一兵种，需要使用摄像头来识别并制导打击目标单位，如果能完成图像回传就可以实时修改图像识别算法的参数并立即看到实际效果，极大地提高了飞镖控制的开发与优化效率。

目前可以考虑的该进方向是使用 ESP32 模块实现 STM32 与 PC 端的网络连接，实现多种数据的传输。

3. **上位机实现方式的改进**

目前上位机实现的底层渲染引擎是 OpenGL，使用 GLFW 辅助窗口相关事宜，开始采用改方案主要是个人想了解一些图形学相关的知识并做简单的应用，但 GLFW 有一个在 Windows 上的固有缺陷。一旦拖动主窗口，整个渲染就停止了，这虽然不是很影响上位机的功能，但希望能在之后解决这个问题。目前已知使用 DirectX 引擎是没有该问题的。

8.4 课程设计答辩时提出的问题分析(串口误码率高)

董老师在答辩的问答环节指出，**模块与上位机的串口通信存在误码率较高的问题**，虽然通过在通信协议中加入 CRC 校验部分避免了错误数据的影响，但只使用一个 USB 与串口的转换模块(项目中使用的是 ST Link V3SET 的 VCP 功能)直接与模块连接不应

该有这么高的误码率，接下来便对这一问题进行分析并试图给出解决方案。

- **统计实际使用时的包错误率(PER)**

使用上位机接收程序统计连续 30S 内的上位机错误数据，结果如下

PER	错误帧数	总帧数	PER	错误帧数	总帧数
0.9%	11	1228	1.2%	13	1091
1.1%	12	1098	1.6%	17	1091
1.1%	12	1099	1.4%	15	1096
1.2%	13	1101	1.3%	14	1096
1.3%	14	1098	1.1%	12	1098

表 8.1 连续 30S 内的错误数量（模块待机状态）10 组

- **使用逻辑分析仪观察实际通信情况**

型号：DreamSourceLab DSLogic Plus

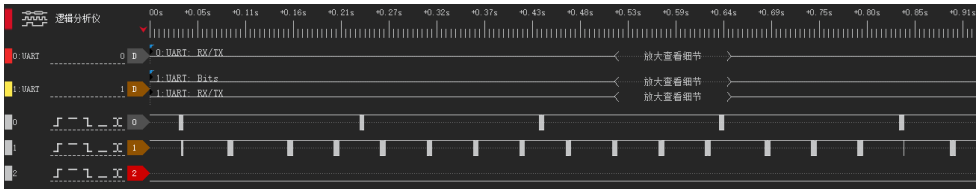

图 8.3 串口数据总览(CH0 模块接收，CH1 模块发送)

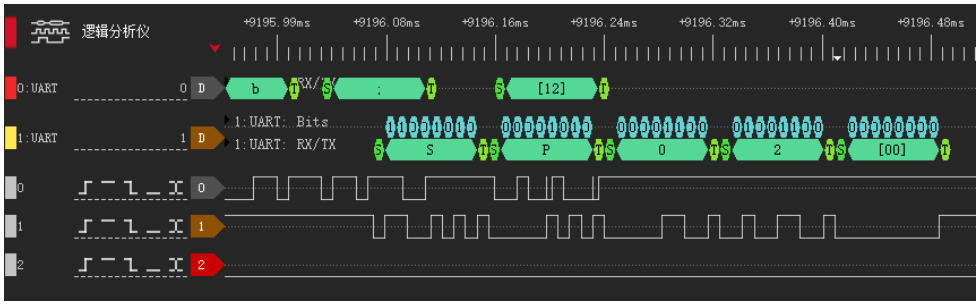

图 8.4 错误帧类型 I :不完整数据帧(帧头完整)

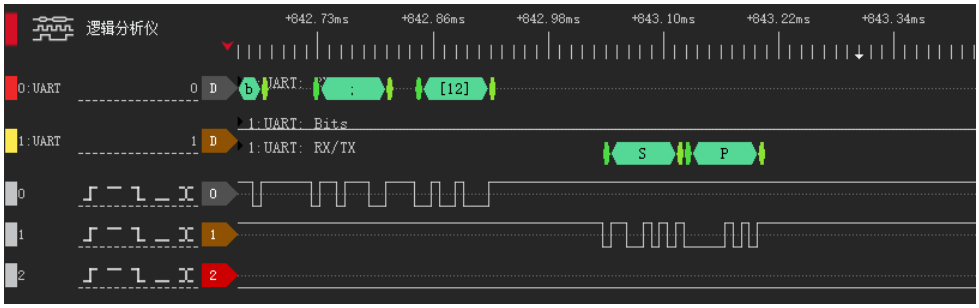

图 8.5 错误帧类型 I :不完整数据帧(帧头不完整)

经过目测在一次时长为 20S 的数据采集中，模块的 TX 端共发出错误数据帧共 9 个。按此数据估算，每 30S 会发出 13 个 TX 错误帧，这很接近上一步统计所得的数据，可以说大部分的错误帧由下位机的发送错误产生，而非因为干扰导致的比特错误。实际观察错误帧的内容，发现错误帧可能在帧头起始后的任意位置被截断，故而被判定为错误。

- **错误帧出现原因分析**

首先猜测的原因是串口的发送过程被其他中断打断，导致数据发送不完整，但串口的发送使用了 DMA，且调用串口 DMA 发送的任务只有一处，理应不受影响，该原因可能性较小。

仔细查看逻辑分析仪数据的内容，发现**错误帧必定伴随着模块的串口接收过程产生**。这时我想到董老师曾在上课时讲过串口的接收与发送共用了一个数据寄存器，那么**模块发送错误的原因**很可能是串口的接收中断突然出现在了串口发送任务执行时，导致了串口发送的异常

- **解决上一步分析出的问题**

方案 1：由于上位机的发送周期确定，可以指定模块在收到上位机的指令后的一段时间内回传数据，避免发送与接收的碰撞。但该方案需要按上位机的实际发送周期确定下位机的相应发送时限，而且需要取消上位机检测模块在线后才能发送数据的限制。

方案 2：降低串口接收的优先级，防止数据发送过程被打断。但就目前的情况而言，为了及时执行上位机的控制指令，串口接收的优先级应当比发送要高。故该方案也略有不妥，而且实际如何实现我也不太清楚。

方案 3：不做处理。因为协议中加入了校验部分，发送数据的错误不会对上位结算到的数据产生影响，而上位机的指令也能及时发送给下位机，对实际功能的影响不大。

方案 4：修改硬件接线，使用 2 个不同的串口分别负责发送与接收，互不干扰。例如使用 USART1 接收数据，使用 USART2 发送数据。但该方法需要修改硬件线路，暂时无法使用。

- **问题解决与效果**

目前使用**方案一**对该问题进行临时解决，再次测试效果如下。

PER	错误帧数	总帧数
0.3%	5	1990
0.3%	6	2114
0.2%	4	2091
0.4%	9	2098
0.3%	9	2088
0.2%	4	2105

表 8.2 连续 60S 内的错误数量（模块待机状态）6 组

可见错误帧数量已经大幅降低，问题得到了一定的解决，但无法避免由于干扰产生的比特错误。同时解包与封包的方法也可能有一定问题导致错误。对于 PER 优化还需要进一步花费时间进行深入，错误帧目前也不影响实际功能。

下一步可以分析上位机接收到的错误帧内容判断错误类型，如：是同种错误，还是由于解包方法异常导致的错误，或是还有其他原因。进一步的分析由于临近期末，时间有限，暂不进行。

9 对本课程的体会和建议

1、 该课程的课题较为宽泛，又偏向实践，适合我们这个阶段的学生学习。既可以加深对于之前的电路、软件、微机课程的学习，又可以培养实践技能，增强我们在资料查找、自主学习以及团队协作等多种能力。

2、通过一学期的学习，我们认识到嵌入式系统是十分庞大、综合的一个学科，它涵盖了C/C++等计算机语言知识，涉及到了实际元件、芯片的实际使用方法，是目前为止我们学习过的最偏向于实践的一门课程，也是一门相当实用课程。对我们今后自主解决实际问题大有裨益。

3、 在查阅资料过程中，我们了解到相当一部分软件指南（如 ST Cube）、芯片手册及应用等资料均为英文，通过搜索引擎得到的中文资料相对不完善且重复性高，很多都没有价值。如果没有相关的知识经验，对各种工具链以及开发中常见问题缺乏了解，很有可能面对一些莫名出现的问题无从下手，甚至在耗费大量时间后也无法解决。这使我们认识到学习知识必须深度与广度皆备才能更好地将其应用，解决实际问题。

4、 前几周在 MOOC 上进行的网课课程侧重于嵌入式的初步讲解，虽然课程的内容基础而全面，但是没有硬件条件进行实践学习，在前半个学期基本处于一种似懂非懂的状态，甚至连入门都达不到。直到接近课程结束才开始统一采购开发板，助力我们完成课设并实践学习，造成了课程前半较闲且无法深入，课程后半忙忙碌碌的问题。虽然是因为疫情的缘故，但还是希望学校以此为鉴，在日后在遇到特殊情况可以有更好的应对方法。

10 参考文献

[1] DJI: RoboMaster 2020 机甲大师赛机器人制作规范手册 V1.2 (20200527)
[2]香港科技大学 ENTERPRIZE 超级电容方案开源
https://bbs.robomaster.com/forum.php?mod=viewthread&tid=7007
[3] Analog Devices Data Sheet: LTM8055 36VIN, 8.5A Buck-Boost µModule Regulator .Rev C
[4] Joey de Vries, Blumia: OpenGL 中文教程 https://learnopengl-cn.github.io/
[5] ocornut: Dear ImGui https://github.com/ocornut/imgui
[6] Tom Archer and Rick Leinecker: CSerial - A C++ Class for Serial Communications