具有无线充电的变电站巡检机器人无线充电子系统

目录

一、 系统组成.....1
二、 硬件设计.....2
2.1 发射端硬件设计.....2
2.1.1 概述.....2
2.1.2 原理图.....3
2.2 接收端硬件设计.....10
2.2.1 概述.....10
2.2.2 原理图.....10
三、 软件设计.....11
3.1 发射端软件设计.....11
3.1.1 概述.....11
3.1.2 发射端主要代码.....13
3.2 接收端软件设计.....18
3.2.1 概述.....18
3.2.2 接收端主要代码.....18
四、 上位机软件.....19
4.1 概述.....19
4.2 功能框图.....19
4.3 主要代码.....20
五、 系统调试.....20
5.1 概述.....20
5.2 硬件调试.....21
5.2.1 概述.....21
5.2.2 发射端硬件测试.....21
5.3 软件调试.....22
5.4 上位机与下位机联调.....23
5.5 系统联调.....24
六、 系统运行.....26
七、 结束语.....27

一、系统组成

如图 1.1 所示，系统的总体构架包括：用户层、服务层、现场层及操作员站。

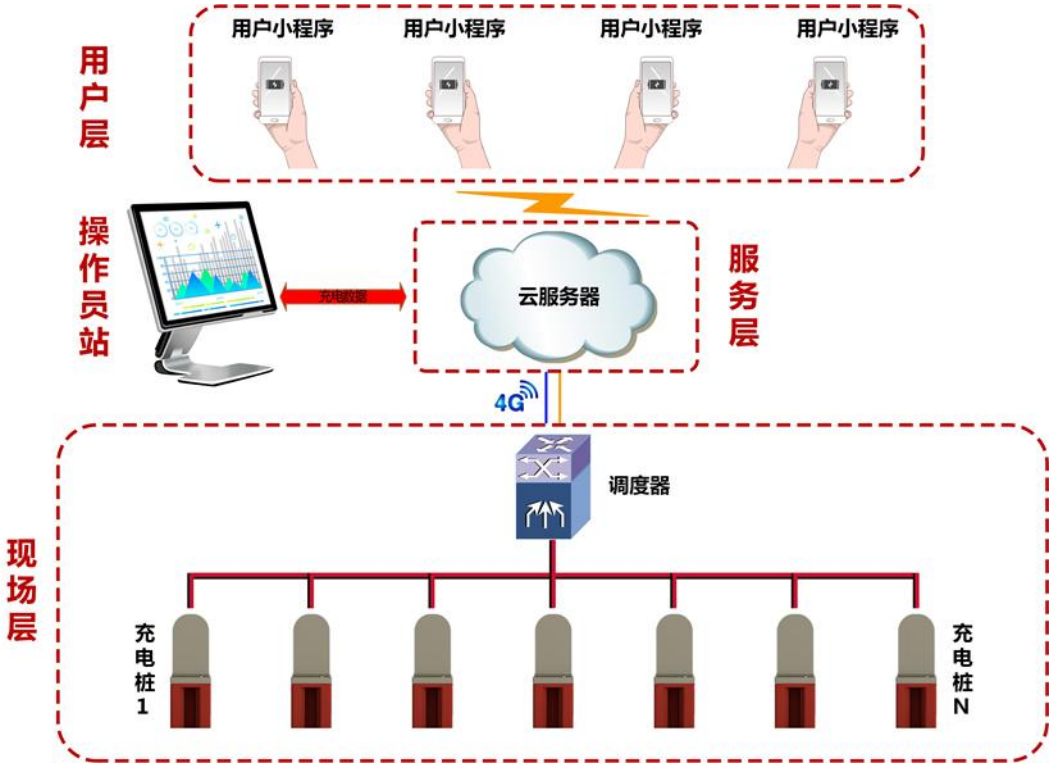

图 1.1 系统总体架构

用户层：在车辆上安装一个接收器，以无线的方式接收电能，并在处理后充入电池；通过红外向发射器实时报告自身信息。一个充电桩能够绑定多个用户。用户首次安装时，填写用户基本信息、车辆编号、绑定桩编号、电池型号等信息；用户通过小程序或 APP 能够实现对自己车辆状况了解和掌控。

服务层：服务器是连接用户、操作技术员、总控调度器的机构。其主要功能包括：接受总控发送的桩体信息，通过后台展现给技术人员查看，或者通过小程序展现给用户；接受用户通过小程序下发的指令，或接受技术人员通过后台下发的指令，然后转发给总控；各种异常情况的判断与短信告警；数据的记录与报表的生成；电池充电参数的汇总与修改。

现场层：主要包含总控调度器和单体无线充电桩。总控器是充电桩和服务器的连接枢纽，一方面通过 485 轮询得到下属充电桩的数据，并通过 4G 转发给服务器；另一方面接受服务器的指令，并转发给对应的充电桩。充电桩为用户提供充电服务，接受总控器的指令开启充电，能够在充电完成、指令结束、异常等情

况下自主结束充电；同时，通过红外获取停放车辆的全部信息，并通过 485 轮询实时上传给总控器。

对于单体无线充电桩，其结构组成如图 1.2 所示。系统主要包括：直流供电电源、发射端控制器、发射端线圈、接收端线圈、接收端控制器、以及负载电池。流供电电源来自于电源柜中的电源总线，所有充电桩都挂在一条电源总线上。发射端控制器主要进行发射功率控制，并通过红外获取接收端控制器中的全部信息，通过 485 总线与总控器连接，汇报自身的信息与接收到的接收端控制器中的信息。发射端线圈和接收端线圈之间存在磁场耦合，通过高频磁场实现能量的无线传输。接收端控制器对接收到的电能进行处理，然后供给电池，并通过红外与发射端控制器联系。

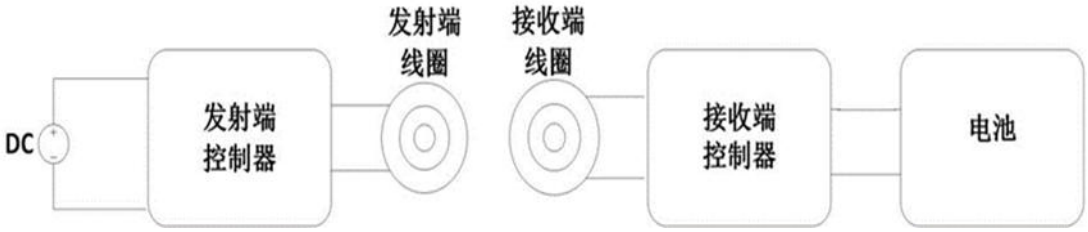

图 1.3 充电系统组成

二、硬件设计

2.1 发射端硬件设计

2.1.1 概述

发射端的核心功能是将直流电转换为交流电后通过线圈将能量发射出去。在核心功能之外，为了提升其易维护性以提升用户体验，又在核心功能的基础上增加了一系列附加功能。这些附加功能包括但不限于：和上位机进行通信，与接收端通信，电压电流监测与保护等。首先，核心功能的实现依赖于完整的电源供给，完整的实质意义在于提供各器件所需要的电压等级，比如 12V、5V、3.3V 等。再者，电流从直流连续转换到交流发射需要经历一系列过程，这个过程除了包括各级电压电流保护外，其主要内容是通过 BUCK 斩波电路控制发送电压，通过半桥逆变电路将直流电转换为交流电。核心功能以及附加功能的详细介绍将主要在原理图基础上一一介绍。发射端控制电路原理示意图如图 2.1 所示。

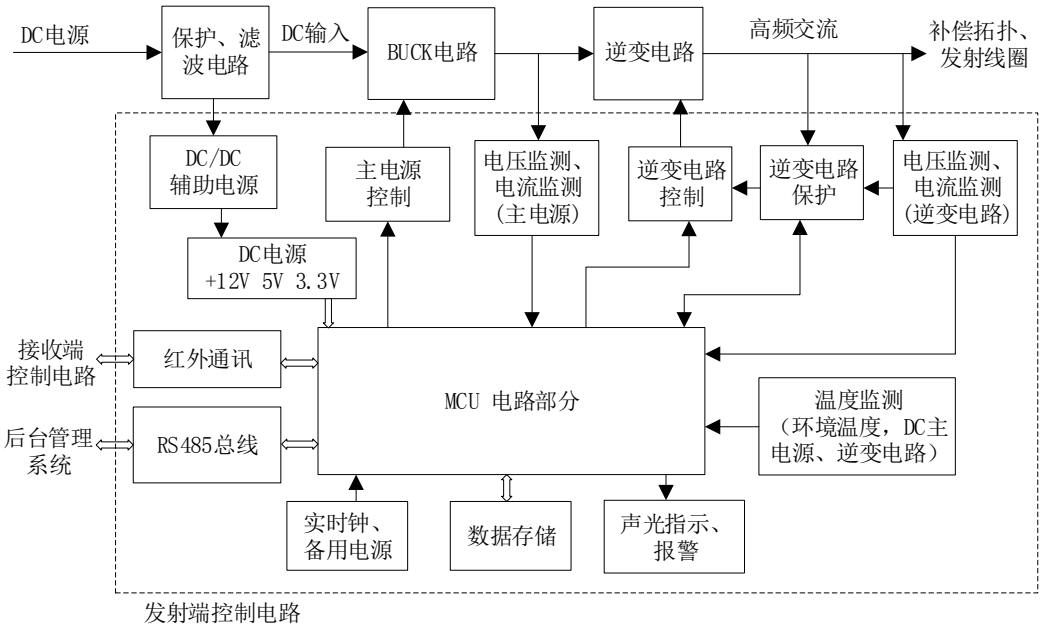

图 2.1 发射端原理示意图

2.1.2 原理图

1）MCU 主控制模块

本项目发射端主控芯片为 STM32F103RCT6。F1 是 STM32 基础型芯片，相对于 F0 级芯片而言功能较为完善，即外设接口较多，而在价位上相对于 F4、F7 而言又十分低廉。但是究其根本，选择该芯片的原因还是在于功能足够，价钱最低，这是选择一款工业配件的基本原则。该芯片型号倒数第三位的 C 表示闪存即 FLASH 容量为 256KB，和 STM32F103C8T6 的 64KB 闪存相比有了两个计算机量级的提升，这对于本项目而言十分重要，因为许多系统运行的信息有赖于保存到 FLASH，以便系统下次开机调用。

发射端用到的 MCU 外设功能包括：串口通信，用于同上位机通信，主要使用场合为调试和投放后运行时同系统主机的通信，当然通信是通过 RS485 接口以半双工方式进行；SPI 通信，用于同频率合成芯片 AD9833 通信，可以生成如正弦波、方波、三角波等波形，虽然 AD9833 的通信接口为三线制串口通信，并非标准的 SPI 四线制接口，但是可以兼容 SPI、QSPI、MICROWIRE、DSP 等串口通信标准，虽然频率合成这一功能连鸡肋都不如，但是这确实是项目内容，没办法不介绍；PWM 波生成，这是 MCU 自带时钟的其中一个功能，还是十分强悍的，用好了可以事半功倍，用不好就事倍功半了，在本项目中只用到了时钟 5 的通道 1，用

来生成 PWM 波控制 BUCK 电路的 MOS 管，以产生目标输出电压；AD 采样，AD 采样可以说是本项目发射端的一个核心功能，因为这是完成控制的一个关键点，原理图中从 AD12 到 AD15 共四个 AD 采样通道，用以采样电压、电流、温度等数据；普通 IO 外设，用于实现按键控制、LED 灯控制、485 收发控制等功能。另外，程序下载与调试接口并不在外设范畴内，本项目虽然引出了 ST-LINK 调试所需的所有接口，但是最终只用到了其中的四根线，即四线制 SW 标准，只需要 SWDIO、SWCLK、3.3V 以及 GND 即可快速下载程序和完美进行程序调试，程序下载器为 JLINK-OB，相比 ST-LINK 价格低很多。

一块芯片能够运行，有赖于其最小系统，最小系统即芯片能够成功运行所需的最基本的硬件模块。MCU 能够运行，需要保障以下硬件的有效连接：①电源，包括所有 VSS/VDD 接口、VBAT 接口、AVDD 以及 AGND 接口。滤波电容是必不可少的，并且在画 PCB 时应让对应滤波电容尽量靠近对应电源引脚以提升运行稳定性。许多人在最开始做板时总忘了连接模拟电源接口而导致芯片不能成功运行。②BOOT 引脚，BOOT 引脚的连接决定了芯片运行起始位置，为了让芯片能够成功运行程序，我们需要将 BOOT0 以及 BOOT1 引脚均通过 10k 电阻连接到地，即置 0。③晶振电路。晶振是 MCU 的心脏，本项目采用的晶振是 8MHz 无源晶振，通过 MCU 自带的倍频及分频功能可以将运行频率提升到 72Hz，即 F1 系列 MCU 的标准运行频率，但各外设的运行频率各有不同，详情见参考手册的时钟介绍部分。低频时钟晶振为 32.768KHz，并非最小系统范畴。④复位电路。复位电路可以外接按键，以手动复位系统，亦可不接按键，每次上电时系统即复位，或通过调试接口复位系统。复位的触发机制可以自己查，基本原理就是保证复位引脚有一定时间被置为 0。

2) 电源

发射端电路唯一外部电源输入为直流输入，该直流输入一方面为逆变提供能量，另一方面又需要保障整个系统的运行。为了保障输入电源的稳定性，采用了多种电源保护方法，在下图所示的原理图中，采用了保险丝、热敏电阻、压敏电阻、稳压管来保证电源电压、电流低于限值，电压限值为 150V，电流限值为 3A。采用了电解电容与独石电容相结合的方式保证电压稳定，因为电解电容虽然足够大，但由于其不能很好地过滤高频电压波动，所以采用独石电容互补。在直流源

输出端采用两个 330K 电阻串联，是为了在系统停止工作后，泄放电容中保存的电荷。

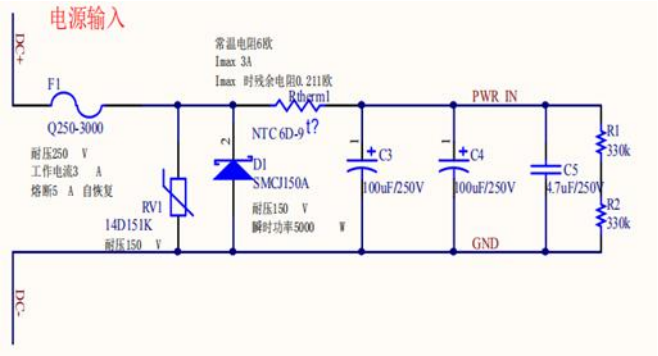

图 2.2 电源输入

12V 电源产生电路采用了某国产公司的 XD308H 芯片。之所以采用这一型号芯片，可能实在是没办法了，因为进口芯片没这么大的电压输入范围，基本最高能到 72V 就顶天了，但是这一款芯片声称电压输入范围是 18V 到 600V，可真是说出来都惊掉下巴。当然，福兮祸所依，伴随着宽电压输入范围的是差输出稳定性，通过电阻配置好输出电压后，当输入电压改变时，输出电压也会随之改变，并不能很好的稳定在期望的输出电压。

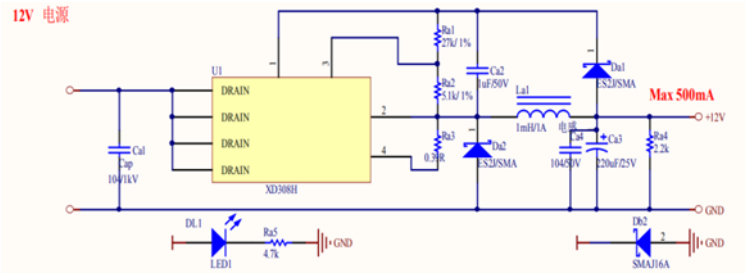

图 2.3 12V 电源

12V 电源在本项目发射端中主要用作逆变驱动和 BUCK 驱动。但是实际上，根据所使用的 MOS 管电气参数来看，栅极驱动电压越高，漏源极电阻越小，本项目使用 12V 电压进行逆变驱动，通过升压芯片作 BUCK 驱动，使用两种驱动电压其实没有太大必要。同时，12V 电源也用于后级变压，以产生 5V 电压。

5V 电源其实只是作为过渡电压级，以便后级芯片将 5V 电压降压产生 3.3V 电压。使用 MP2359 的好处在于其宽输入电压范围，类似于 XD308H，电压输入范围很大，为 4.5V-24V。而其输出也同样是通过电阻进行配置，这其实没有很大必要，因为我们所需的电压只是 5V 这一特定电压。3.3V 电压通过 AMS1117-3.3 产生，这款新品其实也同样有着比较宽的电压输入范围，最大输入电压为 18V，如

果在初级变压阶段能够将电压较好地稳定在 12V，可以不需要 5V 这一中间变压级，可以直接使用 AMS1117-3.3 产生 3.3V 电压。所以这一设计的缺陷就在于，采用了 XD308H 这一款芯片，导致后级麻烦很大，添加了一些不必要的器件。至于 MP2359 和 AMS1117-3.3 这两款芯片的介绍，可以参见 MP2359、AMS1117-3.3。

3）逆变控制

关于频率合成，这是一种以数字方法控制的模拟输出方式，使用 AD9833 这一芯片，可以输出不同模拟波形，如正弦波、方波、三角波等。但是实际上，所需求的波形只是方波即可，并不需要其他波形输出，因此这一芯片的使用就是功能多余，如果可以直接利用 MCU 的互补波形输出功能，还自带死区控制，相对于本项目的逆变控制方式而言会有巨大提升。

MCU 通过 SPI 接口对 AD9833 进行控制，由 AD9833 输出 SIGNAL 单一信号，通过死区控制后进行半桥逆变控制。SIGNAL 信号先通过 UA9638D 产生两路互补信号 SIG A 和 SIG B。以 SIG A 为例，之所以能够产生死区时间，主要原因在于或门之前的串联 RC 电路。由于或门的特性，当 SIG A 信号拉高时，或门直接输出高信号，但当 SIG A 信号拉低时，由于 RC 延时的存在，或门输出并不能即时拉低，即当 SIG A 信号拉低时，或门另一输入端仍输入高信号，当此高信号电平慢慢降低直到达到一定电压水平（此电压水平由或门芯片 SN74HC32D 以及芯片供电电压决定，其确定值为供电电压的一定百分比，当供电电压为 4.5V 时，3.15V 以上即为高电平）时，或门输出低电平。继而，使用 SN74HC32D 反相器，对信号进行反相。读者可以自己根据这一过程画出时序图，由于采用了死区控制，可以保证一定时间内逆变电路驱动电压均为低，防止出现由于关断不及时导致电流过大而烧坏电路板。

关于逆变驱动，本项目采用了 IR2110S 这款芯片，这一芯片驱动能力较强，可以同时驱动两片 MOS 管，并且其输入信号可以是 3.3V 信号，需要注意的是，有的逆变驱动芯片亦即 MOS 管驱动芯片，需要的输入信号电压级别较高如 10V。IR2110S 实际上可以直接由 MCU 驱动，因此就产生了一种新的硬件连接方式，通过 MCU 产生带死区的 PWM 互补信号，再直接驱动 IR2110S 进行逆变控制，这一控制方式相对于现有方案而言有相当大的优势。本方案中，逆变驱动电压级别为 12V，实际上，如果能够使用 15V 驱动电压，对于降低漏源极间内阻会更有好处。

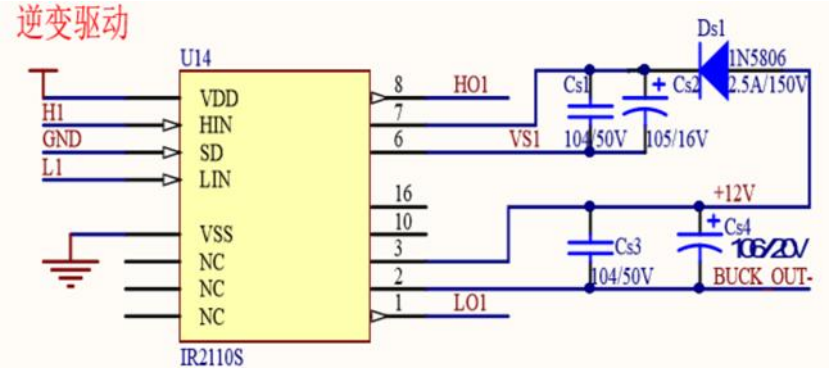

图 2.4 逆变驱动

关于半桥逆变电路，由于其自有特性，相对于全桥逆变而言，有优势亦有劣势。优势在于可以采用更少的器件来节约成本、方便设计制作和维护，劣势在于输出最高电压只有电源电压的一半，会降低发射功率。关于板桥逆变所采用的元器件，核心元器件为 IRFSL4618PBF，此芯片漏源极电压最大为 150V，对于半桥逆变电路而言，电源电压可以达到 300V，但设计电源电压是不超过 150V 的。漏源极电阻典型值为 34.5Ω，这一电阻值并不小，最大漏极电流为 33A，相对于本项目而言完全足够。同时，对 MOS 管的参数关注还有一个重点，就是开启延迟时间、上升时间和关闭延迟时间、关闭时间，这四个参数关系到死区时间的设置。此 MOS 管上升延迟时间为 15ns，上升时间为 35ns，下降延迟时间为 25ns，下降时间为 20ns，这几个参数在英飞凌的 MOS 产品中并非最适合用作逆变控制的，但也勉强过得去，毕竟电动巡检车无线充电所要求的功率级别并不高，对效率要求也没那么严格。而半桥逆变电路中的其他元器件，其功能现在将一一进行介绍。栅极串联 10Ω 电阻，是在开启状态下起到保护作用的，而其并联的二极管，是在 MOS 管关断时，由于 MOS 管与驱动芯片之间组成电流回路，并联二极管可以让电流快速通过以减小关断时间。栅极与源极之间的 20K 电阻是为了防止由于漏源极间加高电压时，由于寄生电容的影响，使得栅极电压过高而直接开启了 MOS 管，而此 20K 电阻的存在可以泄放掉栅极电荷，以免误开启 MOS 管。至于 20K 电阻所并联的稳压管，是为了在栅极电压过高时，可以通过稳压管快速放电，通过稳压管放电远快于电阻放电。逆变电路 VS1 端口分别与逆变电路电源的高电压级和低电压级之间连接了 RC 串联电路，电容的存在是半桥逆变的基本操作。而将 RC 串联，是利用了滤波原理，当漏源极间电压过大且不稳定时，即可通过 RC 串联电路放电，以免损坏 MOS 管，也能输出较为稳定电压电流波形。

4） BUCK 斩波

BUCK 电路是发射控制的重要一环，是属于负反馈控制的控制端。发射端通过红外端子可以获取接收端充电电压电流信息，再通过 BUCK 电路调节逆变输出电压。BUCK 斩波电路是一种较为简单粗暴的电压调节方式，只需要调节 MOS 管控制端控制信号占空比，即可调节电压，比如占空比为 50%，BUCK 输出电压即为输入电压的一半。当 MOS 管开通时，续流二极管不通过电流，当 MOS 管关断时，续流二极管便接入电流通路，和电感、电容形成电流回路。关于电感和电容的选取，电感越大，输出电流稳定性越高，电容越大，输出电压纹波越小，但电感电容选取过大会导致 BUCK 响应速度变慢。BUCK 电路所采用的续流二极管对 BUCK 效率影响非常大，当电流为 10A 时，若二极管导通电压为 0.7V，则会有 7W 的功率损失，这个损失不可以说不大，因此在某些情况下，采用 MOS 管代替二极管作续流，可以大大降低 BUCK 损耗。

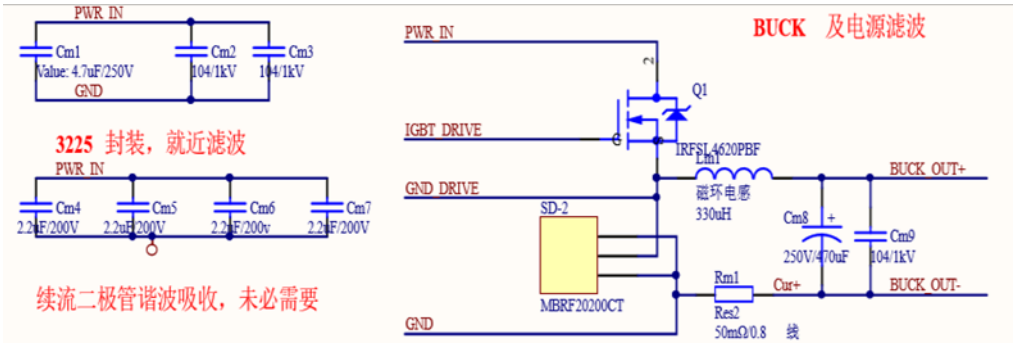

图 2.5 BUCK 斩波电路

5） AD 采样

关于电压电流采样，本方案中大量的采用了差分放大电路进行 AD 采样。本方案采用康铜电阻进行电流采样，使用此类电阻的好处在于可以在电流较大、发热量较大时更快地释放热量，但是有其他方案可以优于采用康铜电阻，因为康铜电阻的采用不利于电路板贴片，且安全性较低。由于采样电阻两端电压较小，需要采用差分放大电路对电压信号进行放大，再通过 MCU 自带的 AD 采样采集电压信号。本方案对 BUCK 电路输入电压、输出电压和输出电流进行监测，均采用了差分放大电路。同时也通过热电阻测量方式对电路板散热片温度进行了监测。

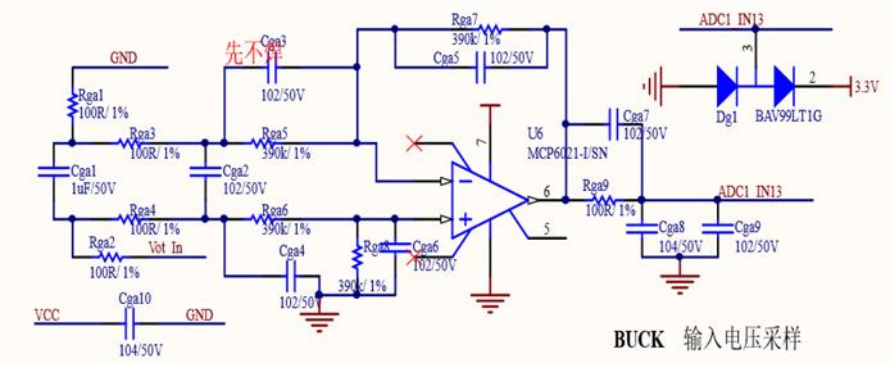

图 2.6 AD 采样电路

6）通信

关于 RS485 通信，这是一种工业现场总线通信，可以挂在 255 个从通信端，但是通信稳定性对硬件设计要求较高，在很多情况下 RS485 的通信稳定性并不是很高。这是一种半双工通信方式，发送和接收不能同时进行，通过 485 通信收发控制端口进行控制。因此，485 通信一般采用查询方式，主机主动发动对从机的查询，从机返回对应信息。收发控制方面，当 RS485_EN 引脚为高电平时，可以进行数据发送，当该引脚为低电平时，可以接收数据。因此，在软件控制方面，一般在需要进行数据发送时，将引脚置为高电平，而数据发送一旦完成则立即将该引脚置为低电平，以备数据接收。485 为两线制差分通信，当 A、B 端口间压差达到一定水平时，芯片解析为高电平，当压差低到一定水平时，芯片解析为低电平，这是一种防干扰能力较强的通信方式，但通信速率较低，且受通信距离影响较大。

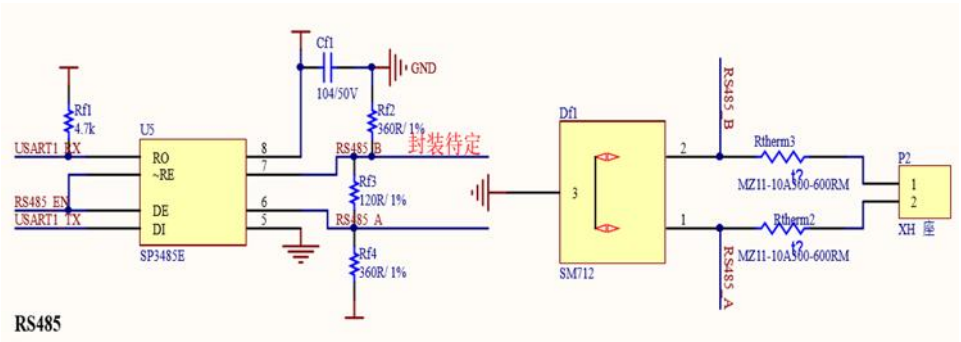

图 2.7 485 通信电路

至于发射端与接收端之间的通信，本方案中是一种单向通信，也就是说发射端与接收端只有一个能够发送数据，只有一个能够接收数据。在这里，发射端是只有红外接收头没有发射头的，只能被动接收信息。接收端可以往发射端回传消

息，比如电池电压、充电电流等。使用红外通信的优势在于，可以节约一定成本，但是劣势在于调试比较困难，通信要求较高，且如果要进行双向通信，复杂度将会提升一倍。在很多无线充电场合，比较受欢迎的通信方式是 2.4g 无线通信，一般是可以进行双向通信的，对通信模块安装位置的要求较低，调试难度也能大大降低。

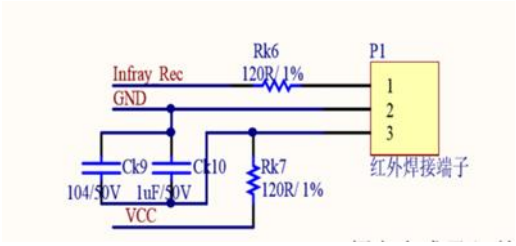

图 2.8 红外通讯

2.2 接收端硬件设计

2.2.1 概述

接收端的核心功能是将线圈接收到的高频交流电整流为直流电后给电动巡检车蓄电池进行充电。接收端的控制电路原理示意图如图 2.9 所示。

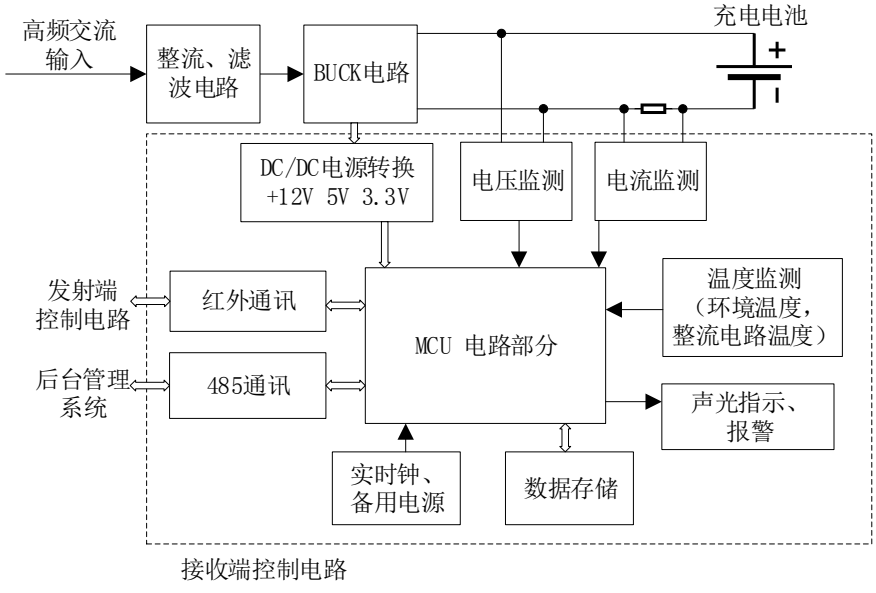

图 2.9 接收端原理示意图

2.2.2 原理图

鉴于接收端电路与发射端电路有较多相同之处，故在原理图部分重复的地方不做赘述，参阅发射端硬件设计。

1）整流电路

接收端采用的是由四个肖特基二极管 SS5200 组成的单相桥式整流电路，辅以滤波电容以平稳高频交流电整流后的输出波形，其基本原理如图 2.10 所示。

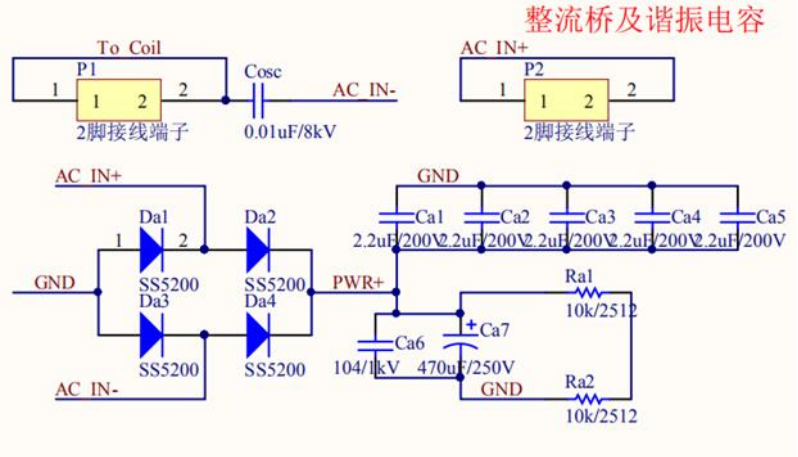

图 2.10 接收端整流电路

2）红外发射

红外发射电路跟普通的 LED 灯电路没有区别，只不过是将 LED 灯换成了红外发射管，由 MCU 直接控制红外发射管实现红外发送，发射端的红外接收器探测红外信号实现信息传递。在本设计中只能由接收端发射红外，接收端接收红外，为单工通信，串联的电阻起到限流的作用。

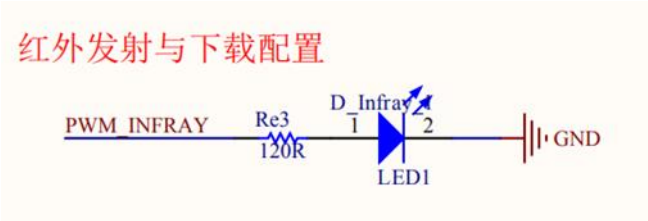

图 2.11 红外发射

三、软件设计

3.1 发射端软件设计

3.1.1 概述

发射端下位机软件设计是以 CUBEMX 下的硬件配置为基础，以 HAL 库作为基本库的。一般而言，有如下几种硬件配置的方式，第一，通过直接配置 MCU 寄存器完成 MCU 初始化和控制、读取等操作；第二，使用标准库（区别于 HAL 库的另

一种官方库），但标准库需要对外设、时钟等手动编程配置，配置难度略高，但低于直接使用寄存器配置的方式；第三，使用 HAL 库，并搭配 CUBEMX 使用，这种配置方式是嵌入式程序员的福音，因为只需要在 CUBEMX 提交外设需求，该软件就能直接配置好头文件及 C 语言初始化代码。这三种配置方式的优越度是递进的关系，也越来越契合当今程序员需求。

关于下位机程序结构，变量及 diy 函数基本在 main.c 文件中，同时，该 C 文件中的主函数即 main 函数相对而言也是比较长的，main 函数先完成整个系统外设的初始化，包括 IO 口、DMA、ADC、串口、时钟、看门狗。DMA 是一种流数据传输方式，不需要程序干预即可自行完成数据传输，本项目看门狗采用了软件看门狗方式，在 main 函数中无限循环代码部分即 while(1){…}的最后阶段“喂狗”，以保持程序运行，如果超时没有“喂狗”，系统就会重启。系统初始化后，是变量初始化部分，由于在变量定义时没有赋予初值，因此另外定义了 Variable_Init 函数来初始化变量。之后，就是开启一系列需要在程序初始运行时需要开启的中断、功能，包括红外接收功能、BUCK 控制功能、RS485 接收功能、看门狗功能。

在 main.c 文件中，while(1)函数后，就是大量的自定义函数、部分中断调用函数以及时钟配置函数等，但时钟配置函数不需要自行编写，已由 CUBEMX 进行了配置。

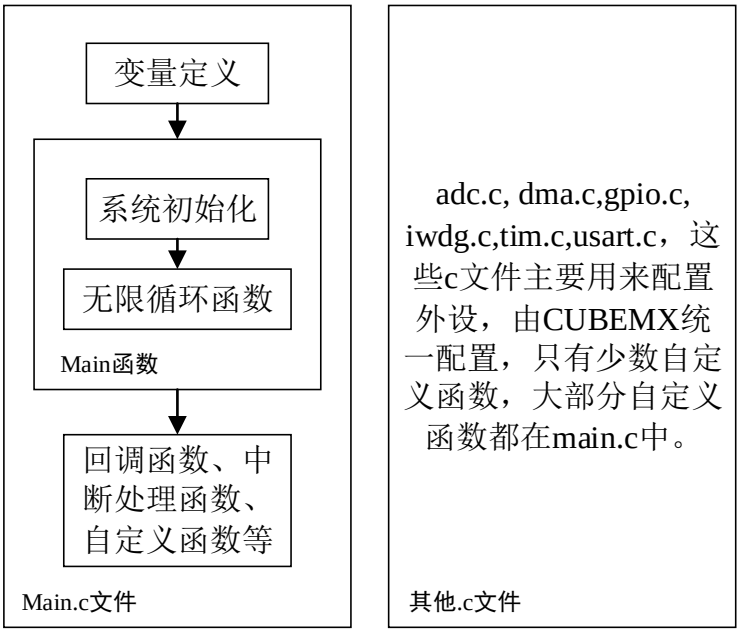

图 3.1 软件结构图

3.1.2 发射端主要代码

在无限循环下，有许多判断语句，包括 if…else 结构和 switch…case 语句。这些语句是按照时间顺序一个个执行的，一个判断语句运行完毕后才运行后续语句。一般而言，任务运行有两种方法，一种是以时钟中断作为任务驱动源，每隔一定时间判断应该执行哪一个任务，这种方式的好处是可以掌握任务运行的节奏，而这种方式的劣势在于需要程序员有敏锐的时间嗅觉，能够掌握程序运行的时间节奏，根据各部分程序运行时间及项目需求安排程序运行间隔。第二种方式就是将大部分任务置于 while 循环中，按照时间顺序执行任务，这种方式的好处在于可以减少程序员工作量，不仅可以简化代码，也不需要程序员去思考太多关于时间节奏的问题。本项目的任务执行方式就是以第二种方式为主，以中断和 DMA 方式为辅。

1) 第一个判断语句 if(Flag_StartByte){…}是对检测字头标志位进行判断，Flag_StartByte 这一标志位是在串口接收中断中置位即置为 1 的，当接收到指令的第一个字节为 0x55 时，证明已经接收到了字头，即将 Flag_StartByte 这一标志位置为 1。而本判断语句是为了检测是否在接收到字头后是否在 50ms 内完成了消息接收，如果没有完成则说明数据接收有问题，需要重新接收数据，则将字头接收标志位 Flag_StartByte 置为 0，将 RS485 接收时间 TimeCnt485 置 0，将字节数 Byte_Cnt 置 0。

2) 第二个判断语句 if(Flag_485_Reset)是为了判断 485 是否在工作状态，Flag_485_Reset 这一标志位是在 while 循环中另一个 if 语句中每隔 5s 被置 1 一次，即每隔大概 5s 需要运行本判断语句。HAL_UART_GetState 这一函数是 HAL 库自带函数，可以判断串口外设是否运行正常，如果返回值为 0x01 即 HAL_UART_STATE_READY 时，表面串口正在正常工作，若返回值为 HAL_UART_STATE_RESET，表明串口还没有被初始化，需要重新初始化。若返回值为 0x04，则表明发生了未知错误，一般解决办法也是重新初始化外设。返回值类型有多种，不同返回值代表的含义如下：

```
HAL_UART_STATE_RESET      = 0x00,    外设未初始化
HAL_UART_STATE_READY      = 0x01,    外设可以使用
HAL_UART_STATE_BUSY       = 0x02,    正忙于某一内部进程
```

HAL_UART_STATE_BUSY_TX = 0x12, 串口正忙于发送

HAL_UART_STATE_BUSY_RX = 0x22, 串口正忙于接收

HAL_UART_STATE_BUSY_TX_RX = 0x32, 串口正忙于发送和接收

HAL_UART_STATE_TIMEOUT = 0x03, 超时

HAL_UART_STATE_ERROR = 0x04 错误

在这一判断语句中，Rece_485_En 这一函数十分简单，就是将 RS485 收发芯片的收发控制端口置为 0，即可开启接收模式，等待接收数据。HAL_UART_Receive_IT 函数原型有三个形参，第一个为所用串口，第二个为数据放置位置，第三个为接收数据字节数。在这里，我们使用串口 3 来进行 485 收发，将数据存储于 Data_ReceSingle 中，且每次只接收一个数据。至于 MX_USART3_UART_Init 函数则为串口初始化函数，是通过 CUBEMX 配置的。

3) if(Flag_Temp_Refresh)这一判断块是为了每隔一定时间更新温度对应的功率上限值。众所周知，MOS 管的功率潜力是非常大的，只要散热做的好，管子就往死里搞。Flag_Temp_Refresh 这一标志位是和 485 检测标志位在同一个判断语句内置 1 的，不同的是 Flag_Temp_Refresh 是每隔 10 秒置位一次，即每隔 10s 来对功率上限作一次判断，至于判断公式，应该是从 MOS 管的说明手册中得出来的。

4) 在数据接收完成后，需要对接收数据进行解析。LED 闪烁、延时函数这种小儿科就不多说了。清零接收完成标志后，判断第二个字节即 Data_ReceArray[1]的反码是否等于第一个字节，当然，帧头 0x55 是排除在这一数组外的。后面就捡重点说，switch 语句用来判断数据包的第三个字节即功能码的含义。0x0F 意为查询状态功能码，开启 485 发送后，即打包状态数据，并通过 DMA 将数据发送出去。0x1E 为开启充电码，后面这一大堆看程序跟注释就好了，代码已经写的很清楚了。0x2D 意为停止充电，并返回状态。0x3C 意思就是开始扫频，扫频的意思就是以不同频率发射能量，看以什么频率发射能够获得最大发射功率。至于 0x4B，0x5A，0x69 这一系列功能码，重要性上倒也还好。另外，还有一个修改发射端 ID 的功能码，但是这个功能码并不是第三个字节，而是第一个，如果第一个字节为 0xFF，就说明上位机想修改这一充电桩或者说发送端 ID 了，ID 应该大家都懂，就是 identity-身份的意思。然后后面就是一系列的操作了，把第三个

字节作为 ID，把 ID 写到 flash 里面，开启 485 发送，打包状态数据并通过 DMA 发送。

5) if(RmtSta&(1<<6))这种操作其实意义很明确，就是看 RmtSta 这个状态标志从右边数过来第六位是不是 1，如果是 0 的话，0&1 为 0，只有这一位是 1，才能返回 1 这个结果，if 语句条件才能成立。Infrared_Rec_Succ_Cnt 这一计数标志位的原理很简单，成功一次就加一次，而且这是一个无符号 32 位整形数据，要想让这个数据加满，那还是要费些功夫的，如果加满了，那这一数据就变成了 0xFFFFFFFF，再取反为 0x0，这个时候这一变量才不会继续增加，否则每次红外接收成功都要加 1。Data_Decode 是红外数据解析函数，后文会接着介绍，这里就先不多说了。

6) Flag_Infrared_Error 是红外接收错误标志，就是接收到的红外光线连 MCU 都不知道是啥玩意时，就把这个标志位置 1，每次发现错误，就给错误计数标志位 Infrared_Rec_Error_Cnt 这一标志位加 1。

7) 如果超过 10s 每接收到接收端数据，就认为车没停过来，这还是很好理解的。

8) 如果超过 60s 没接收到红外信号，那直接放弃了。

9) 如果检测到车辆了，就打开 LED，否则关闭 LED，这个 LED 是裸露在充电桩外让用户看车有没有停好位置的。

10) Charger_State 这一标志位的第 6 位和第 7 位是用来判断充电桩工作状态的。一个字节有 8 位，8 位即两个半字节代表的数值分别为 8421|8421，0x60 即 0420|0000，以二进制码表示即为 01100000。在这个 switch 函数下，对这两个状态位进行判别，若为 0x00，则表示这两位为 00，是停止状态。后面再进行一系列操作，具体什么操作就不多说了。如果这两位为 11，那么表示现在是正常充电状态，可以根据一系列情况去改变这一状态。

在正常充电情况下，即充电状态标志位第 6 位和第 7 位均为 1 的情况下，对系统运行情况有一系列的判断。在 case:0x60 块下，有一些状态标志位其实是需要特别关心的。比如 Infrared_Charging_Waiting_Cnt，这是红外延时计数标志，因为在系统滴答时钟即 sys_tick 下，每隔 1ms 会唤醒系统时钟回调函数，并给该标志位自加 1，也就是说，红外接收间隔时间是作为充电正常与否的判别条件

的。如果间隔时间在 1s 以内，那么表征充电完全正常。用时间间隔来表征充电状态，这是需要接收端程序配合的。另外，Duty_Rec_Buck 这个标志位是表征接收端功率是否过高的标志，其数值越大，功率越小，这搞得我有点迷了。判断充电是否应该在恒流段的方法是，和最大充电电流进行比较，若偏小则快速增加功率，如 Power_Trans_Refer+=10，若充电电流比较大，那么功率增加幅度就比较小。如果电压比较接近电池最大电压，那就是恒压段，功率增加值也偏小。当然，Power_Trans_Refer 这一变量不可能无限增大的，因为功率限值 Power_Trans_Refer_Upper 这一标志位的存在。Duty_Rec_Buck 这一标志位就很奇怪，数字越小说明接收端越过载，需要降低发射端 BUCK 控制占空比，即降低发射输出电压。

至于充电完成的判断，主要有两个指标，一是电池电压要接近最大电压，二是充电电流要足够小，具体判断指标可以在程序中看。并且，需要有 100 次数据都表明，充电完成了，可以停止充电了。这种判断方式，就需要接收端在进行电压电流测量时做好滤波，不然非常容易导致误判，充电永远都完成不了的情况也有可能会发生。

充电功率和效率的计算比较简单。充电功率是通过接收端电压和电流相乘计算出来的，当然，还乘了一个矫正系数：K_Power_Rece。效率计算也比较简单，就是接收功率除以发射功率。当效率低于 50%时，说明其中有那么些问题，需要将问题判断一下再记入故障记录。后续就是对红外间隔的判断了，间隔越高越说明需要降低发射功率了，特别时红外间隔超过 30s 的话，就需要停止充电了。

11) 关于系统时钟的设置，均是使用 CUBEMX 完成的。这里系统时钟使用的时钟源是内部低速时钟 LSI 或外部高速时钟 HSE，另外附带说一下，8MHz 晶振是为 HSE 即高速外部时钟提供心跳的，32.768KHz 晶振是为 LSE 即低速外部时钟提供心跳的。另外，值得注意的是，这里系统时钟中断的优先级是（0，0），也就是最高级，说明系统时钟回调函数的计时是十分准确、不会被其他中断打断的。在这里，中断优先级的设置函数是 HAL_NVIC_SetPriority(SysTick_IRQ, 0, 0)。

12) 系统时钟回调函数为 HAL_SISTICK_Callback。这个函数是中断回调函数，根据系统时钟的设置，这个中断每隔 1ms 触发一次。在这个回调函数下，首先对 485 接收时间进行了计时，若一个消息未在 50ms 内完成接收，则放弃该数

据。接着，对红外总体等待时间计时，每次加 1ms。然后，每隔 10s 根据温度更新功率限值。接着，在停止充电状态下和正常充电状态下对红外发射的三种帧计数。

13) 关于红外接收，是在 PWM 捕获下进行的。关于红外的数据捕获可以自行网络查找，因为这种通信方式实际上比较麻烦，并且无法向红外发送端反馈信息，是一种只能被动接收的通信方式。

14) 变量初始化函数无需赘述。

15) 串口发送完成中断回调函数的功能在于，置位发送完成标志位，主要还是在于使能接收，因为 485 通信必须要保证数据发送完成后尽快使能接收。

16) 串口接收中断，这一回调函数在每次接收到一个字节数据时均会触发，可能有些人认为接收完成中断回调函数是一帧数据接收完毕后才触发的，实际上并不是这样。

17) 关于红外解码，主要工作量在于校验，第二工作量在于解析。实际上，对于红外接收数据有三种解析方式，根据功能码判断应该以哪种方式解析数据。从接收端获取的三类数据分别为：一是心跳包，二是充电电压电流数据包，三是对二类帧的实时性判断。第二类数据和第三类数据解析代码大部分是一模一样的，只是第三类帧的解析多了几行判断第二类帧实时性的代码，若第三类帧与第二类帧之间间隔时间小于 1s，说明红外数据是没问题的，这种判断方式还是挺新奇的。

18) 关于数据打包，这些在此不多赘述，在上位机介绍部分有关于数据格式的介绍。

19) 关于 AD 采样，这在 MCU 中是比较简单的部分，只需要将采集到的数字数据转化为模拟数据就行，但是在本项目中并没有先将数字结果转化为模拟数据，所以可能得到了 AD 采样数据还需要自行或者通过上位机进行换算。根据 AD 采样得到的电压电流值，需要对发射端运行进行调节，若输出电压或电流过大，需要降低 BUCK 电路 MOS 管控制端的占空比，以减小输出电压。若在占空比很小时输出电压很大，说明 MOS 管已经失去了对 BUCK 的控制，MOS 管可能烧坏了等等。至于充电准备函数和充电停止函数，内部代码都比较简单，就不多赘述了。

20) 关于 flash 的读写操作，实际上代码不多也不复杂，基本原理就是提供

需要读写的地址，即可进行读写或者擦除操作等。

3.2 接收端软件设计

3.2.1 概述

接收端的下位机亦是以 Cubemx 配出来的代码为基础，再进行代码设计得来的，这种方式的好处是让码农更加专注于代码应用的部分，底层的代码就交给 Cubemx 生成，该方式生成的代码使用的是 HAL 库。如果能看懂发射端的代码，那么接收端代码看起来就容器很多，代码的结构跟风格都比较类似，完成初始化之后开启串口接收，保持 Buck 满开，进而在 while（1）里面对各种标志位进行判断，转而执行相应的程序，完成 485 通讯、充电状态判断。紧接着就是各类自定义函数、中断回调函数，在回调函数中实现对中断事件的处理。由于接收端软件与发射端软件亦有较多相似之处，故重复之处不做赘述。

3.2.2 接收端主要代码

while（1）里面的 switch 语句，SystemState 是工作状态标志位，用 siwtch 语句的好处是在选择分支较多的时候 switch...case 语句会提升效率（虽然目前只有 0 和 1 两个 case），而且代码结构比较明晰，适用于处理这种字符型或者数字型的变量。当系统 SystemState 标志位为 0 时，为半休眠态，每间隔三秒用红外发送一次定时信息，因为发射端需要接收到接收端发来的红外信号才能判定有车来了，再进行充电。同时在半休眠态的时候会持续的对 Buck 电路的输入电压进行判断，若持续的高于设定阈值超过一秒，即判断为可开启充电，将 SystemState 标志位置 1，如果检测到电压值过低，会清零电压过高状态标志位，电压的检测是在 ADC 中断里面完成，在 case 语句里面对检测到的数据进行判断。当系统 SystemState 标志位为 1 时，为充电状态，执行红外发送，同时也对输入电压进行检测，过低的时候将标志位置 0，变为半休眠态，在这里 switch 语句主要是做状态的检测与切换以及发送红外。

定时器更新事件回调函数，这里用到了基本定时器 TIM7，根据发送标志位的值来判定开启或是关闭 TIM3 的红外发送，同时也可以修改 TIM7 的 ARR 值来改变 TIM7 的事件周期，即也是红外发射的载波周期。

定时器输出比较中断回调函数，开启了 Buck 控制的同时也在回调函数里用

DMA 的方式开启了 ADC。

ADC 完成中断回调函数，获得电压电流的采样值，根据采样值进行 PID 运算，运算结果写入 TIM1CH4 的 CCR，即也是改变了 BUCK 控制的占空比，通过这种方式实现对 Buck 的控制。

四、上位机软件

4.1 概述

为了对下位机的功能进行测试，检测是否能够实现无线充电各指令的功能，特使用 VS2012 开发了上位机的发射端和接收端软件，对发射端和接收端进行指令测试，便于后续对发射端和接收端的程序进行调试和改进。

4.2 功能框图

发射端框图如图 4.1 所示。

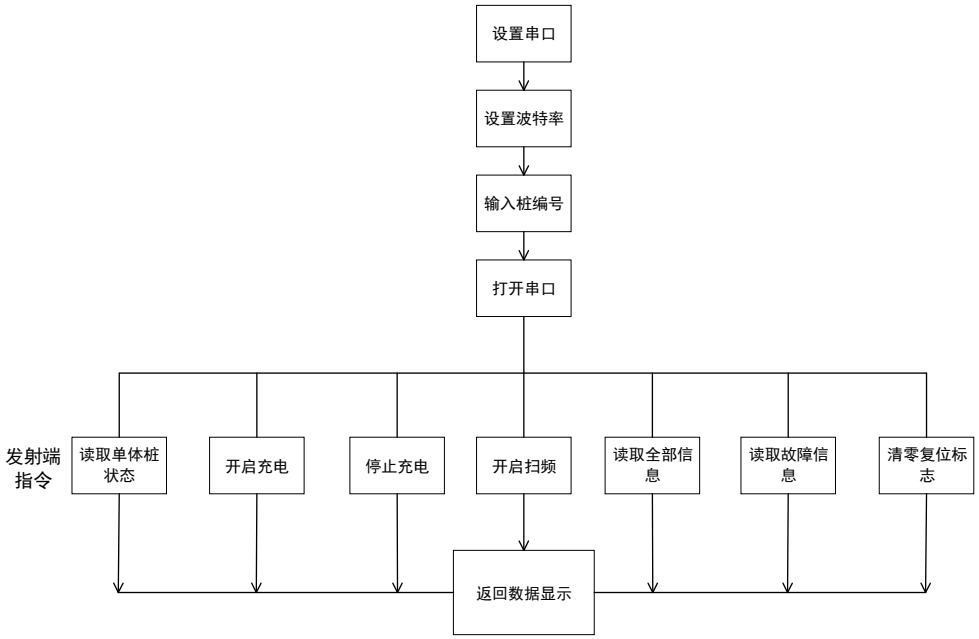

图 4.1 上位机发射端框图

接收端框图如图 4.2 所示。

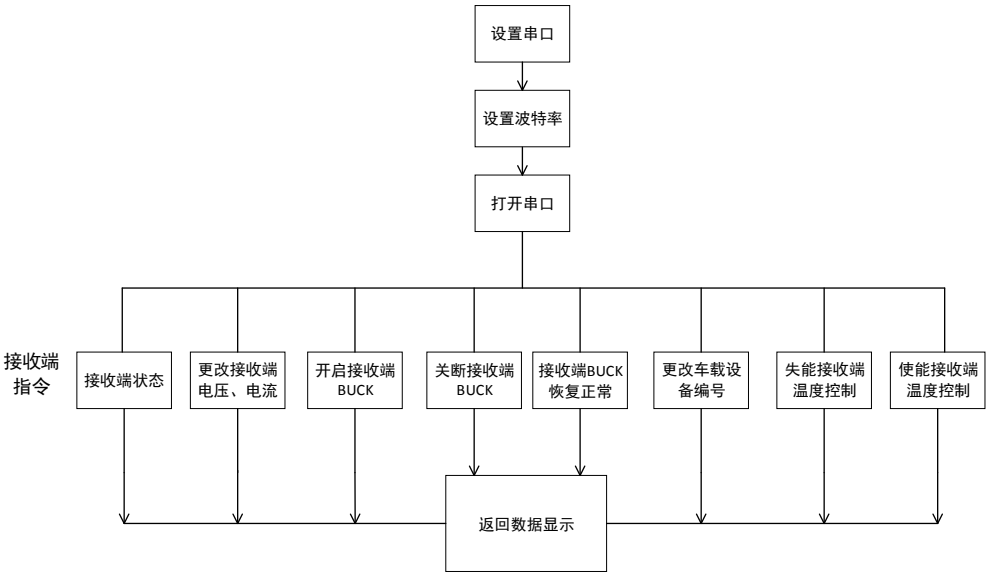

图 4.2 上位机接收端框图

4.3 主要代码

发射端完整代码 https://github.com/xiaohui96/Send_end.git

接收端完整代码 https://github.com/xiaohui96/Receive_end.git

五、系统调试

5.1 概述

系统调试是对整个系统的测试，将硬件、软件、操作人员看作一个整体，检验它是否有不符合预期目标的地方。这种测试可以发现系统分析和设计中的错误。

针对本项目的产品调试主要包含以下几个方面：

1）功能测试：主要测试硬件电路的功能模块是否能正常工作，例如本系统的电源部分是否稳定，整流电路、逆变电路能否正常工作，无线充电系统能否共振传输能量，红外通信是否正常，485 通信是否正常，采样模块能否正常采样电压电流。常用工具万用表、示波器。

2）程序测试：系统中有控制功率和保护功能，这些功能都是由程序控制硬件实现，所以要保证程序能够实现目标。通常会人为加入和制造干扰，以让程序能够促发控制条件。如果在这个过程中，程序没有启动相应控制，说明程序中还由 bug 进行修复。

3）参数确定：在确定硬件功能和程序功能正常情况下，还需要保证控制的

精准性。例如要让电池按照充电曲线进行充电，在已知三段式电压电流情况下，应该确认精准的电压电流对应的采样值，这是因为理论计算与实际存在偏差。还有温度控制、启动条件、停止充电等都需要确定精准的控制参数。

4）安全测试：测试安全措施是否完善，能不能保证系统不受非法侵入。比如高温 100 度下，系统能否正常工作；是否能够在雨雪、雷电天气下正常工作；车辆突然推走，系统能否及时作出反应停止充电；非绑定用户是否能够进行充电等安全问题。

5）可靠性测试：主要是保证系统能够稳定可靠运行。设计的系统需要在高温、潮湿环境中连续以最大负荷功率运行 72 小时。

5.2 硬件调试

5.2.1 概述

电路板的调试往往时硬件软件协调测试的，至少软件的调试时必须结合硬件的。而硬件单独的测试，无外乎就是探针测试，各级电压测试等。探针测试在电路板制作好后就已经完成了，而拿回电路板之后要做的除了用眼睛看，最多就是上电后对各电压级别进行测试，使用万用表或者示波器。

5.2.2 发射端硬件测试

一般拿回电路板后，首先要对其进行细致观察，看看是否该焊接的元器件都已经焊接上去。另外，如果完成了贴片且可以为 MCU 上电的情况下，可以先为 MCU 上电看程序下载是否正常，能否进行程序调试。

为谨慎起见，可以逐级焊接电源器件，焊接完后使用示波器观察该级电平是否正常，主要观察电压是否稳定，纹波情况，上电和断电后是否出现电流突变等情况。在逐级焊接好电源器件后，还需要再测试一次，以确认各级电平正常。

接下来，焊接好所有元器件后，可以加上电源电压，电源电压应该慢慢往上加，在电压提高的过程中，仍需要监测各级电压是否正常，若没有问题，则可以将电源电压加到超过设定电压值，看系统能否自动断电，断电后能否自行恢复。

在本项目的发射端设计方案中，有一项需要重点关注的硬件测试项目，就

是 BUCK 电路的实时响应性和稳定性。尽管在确定参数时，已经在仿真软件比如 SIMULINK 上对 BUCK 电路进行了仿真，但是仿真与实际情况肯定是不一样的，至于差距到底有多少，还需要进行实际测试。可以在程序中将 BUCK 电路 MOS 管控制端占空比设置在一个比较小的值，看 BUCK 电路输出电压是否能随着控制端占空比的变化而有一个比较快的响应，在慢慢提高占空比的过程中，也需要实时监控 BUCK 电路输出电压。

另外，在 BUCK 电路之外，半桥逆变电路的测试也是重中之重。在为发射端连接好线圈、匹配电容后，应当对半桥逆变电路电气特性进行测试。具体测试方法为，使用差分探头和示波器测量逆变电路输入输出电压，查看死区时间的设置是否合适，是否有尖峰电压。若死区时间太长，可以适当减小死区模块中 RC 延时电路的电阻值，以减小死区时间，因为死区时间太长会影响发射功率。若出现尖峰电压，可能是滤波电容或逆变电路缓冲 RC 电路有问题。当然，如果输出波形完全不正常，完全超出意料，那么有可能是逆变驱动信号、MOS 管甚至是匹配电容和发射线圈的问题。这些问题都需要逐一排查，在这里就不多赘述了。关于其他硬件问题，需要同软件一同调试，因为有些引脚只有在 MCU 的控制下才能打开。接收端硬件测试

5.3 软件调试

软件调试和硬件调试是相辅相成的，在本方案中没有单独的软件调试，因为软件只有下载到 MCU 运行之后才知道到底有没有问题。

软件调试的主要方法是设置断点、逐行调试或者让程序运行到目标位置。在程序运行时，需要监测各变量值，看变量的变化是否符合预期，若与预期不符，则需要一步步排查问题。

在本系统的软件调试过程中，通信部分需要注意一些校验环节，因此发送指令时应该严格按照通信规范说明里来。

总体而言，软件的调试还是突出一个逻辑正确性，只要程序在逻辑上是正确的，硬件能够供给的电平是没问题的，那么软件运行也不会有问题。

另外，关于软件各部分代码的介绍在前文以及代码文件中有不少，只需要完成对代码的理解，逐行调试代码运行，如果没有特别突出的硬件问题，软件运行起来应该会比较顺利。软件的运行是需要搭配硬件的，在硬件软件设计全部都已

经完成的情况下，软件调试并不是那么重要，因为这是经过前人充分验证的，不会出现大的逻辑错误。

5.4 上位机与下位机联调

使用 485 转 USB 线将下位机与笔记本相连，打开笔记本中的上位机软件，选择对应指令发送，查看数据显示。

发送端调试界面如图 5.1。

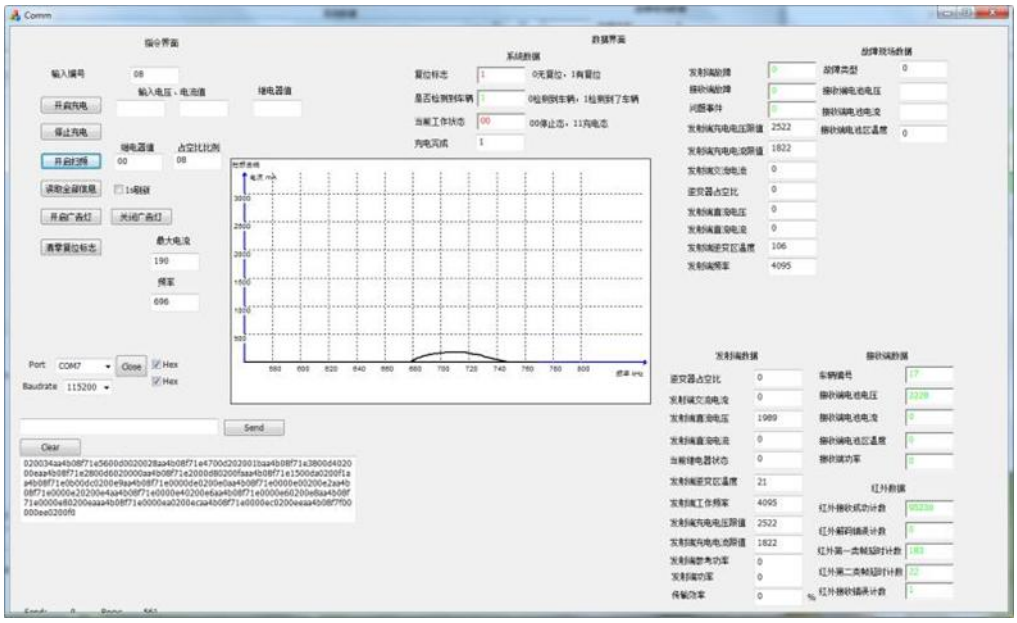

图 5.1 发射端上位机调试界面

接收端调试界面如图 5.2。

图 5.2 接收端上位机调试界面

5.5 系统联调

将单套系统挂载在电源总线与 485 总线上，联调时将在一条总线上挂载 20 套系统进行测试。在联调时首先调整发射端线圈与接收端线圈的位置，接收端将车辆信息通过红外发送给发射端，再由发射端通过 485 通讯传至总控，最后由总控通过 4G 发送给后台，使后台可以检测到车辆信息。当车辆停放状态正常时可由后台下达扫频指令，扫频正常后开始充电，扫频过程可由上位机或后台显示图像。如图 5.3 所示。

图 5.3 扫频界面

充电过程中，接收端将实时监测电池的电流与电压等状态变量，并将数据通过红外发送至发射端，发射端通过 485 通讯传至总控，最后由总控发送给后台，则可通过后台实时展现给技术人员，或者通过微信小程序向用户展示。技术人员可通过该界面了解到充电的起始时间、结束时间等信息。如图 5.4 所示。

充电事件

序号	开始充电时间	结束充电时间	电池型号	消耗电量	充入电量	充电时长	充电效率	操作
1	2019-09-08 19:39:14		铅电48V-20AH					充电曲线/刷新
2	2019-09-07 22:26:14	2019-09-08 00:32:23	铅电48V-20AH	0.24	0.21	2.10	87.5%	充电曲线/刷新

图 5.4 后台观测界面

在车位信息系统中，可以查询桩体接受端和发射端变量，判断车辆的停放状态；当发生故障时，可以查询故障及故障变量，得到故障时发射端和接收端的数据，判断故障的类型。

在发射端状态管理系统中，首先输入总控制器编号和控制器编号，然后点击查询，会出现该控制器编号下的所有桩体编号。

通过发射端状态可以查看充电桩当前状态，是否有故障、有何种故障、有无车辆停发、是否正在充电、就绪车辆电池型号、当前车辆电量、所需连接端口数和当前连接端口数。

在车位历史记录中，记录了总控重启时间和桩位历史信息，输入控制器编号和单体桩编号，点击查询，桩位历史记录会更新为当前桩位车辆状态信息。

图 5.10 车位历史记录系统查询界面

在将硬件挂载在总线后可通过以上界面对系统整体工作过程进行联合调试。

六、系统运行

如图 6.1 为系统运行基本流程图，具体为：首先用户将需要充电的车辆放置至充电桩指定位置处，此时由安装在车辆上的接收控制器通过红外向发射端控制器发送车辆信息，如车辆编号、车辆电池型号、车辆剩余电量等，发射端控制器再将信息通过 485 通信发送到总控器，总控器接到信息后，通过 4G 网络将信息上传给服务器。当车辆信息与登记在后台的数据匹配时，则可由后台服务器检测到车辆，此时可以发送充电指令，充电指令可由用户通过微信小程序或者开发者直接通过后台发送，总控器在接收到服务器的充电指令后通过 485 通信将指令下达给发射端控制器，此时发射端控制器接通电源，调整占空比开始充电。在充电过程中接收端控制器不断将接收端信息通过红外发送给发射端，发射端控制器将根据接收到的信息反馈调整占空比，使得接收端工作在正常状态，同时发射端将接收到的接收端信息与发射端信息一起发送给总控，通过总控上传至服务器，使得开发者或者用户可在后台观测到实时充电状态如电池电量曲线等。如果在充电过程中发生故障，系统将对故障进行分类，并将数据上传至服务器，同时由服务器下发停止充电指令。当接收端检测到电池电量充满时，发射端检测到该信息后将会停止充电，充电完成。

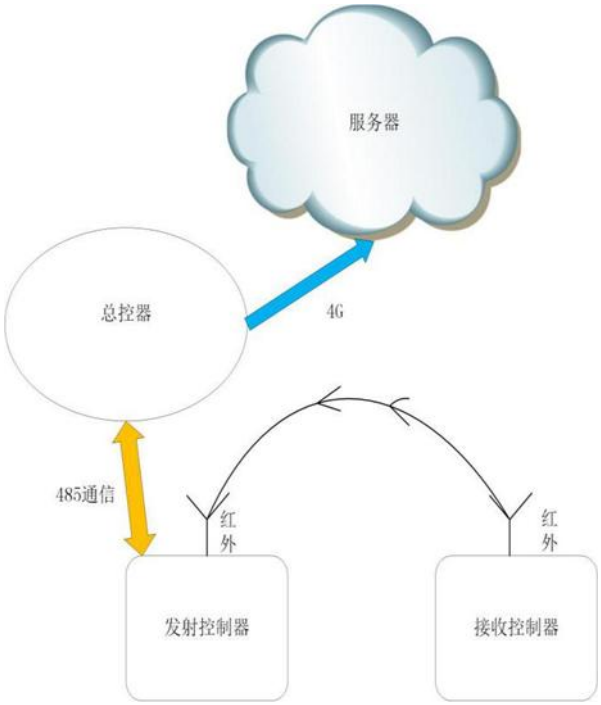

图 6.1 系统运行基本流程

七、结束语

本文档是具有无线充电的变电站巡检机器人无线充电子系统，是无线充电巡检机器人资料的第五册。文档不足之处恳请指正。