第十三章 我的电动车共享充电柜创业项目

这一章振南要讲一下自己的创业经历，感觉有很多话要说，百感交集，但是又不知道从何说起……

在经过一周的考虑之后，我想我找到了合适的切入口和风格基调，以便大家通过本文的描述融入到振南的创业项目之中，如临其境，感同身受，同时在技术和认知上有所收获。如果你也正在创业的道路上，那么希望我的经历成为你的参考。

创业不易，九死一生。也许你认为你已经足够努力，甚至超负荷运转。但是在残酷的大环境下，在泥石俱下的洪流之中，你的努力可能仍然微不足道。

电动车共享充电柜是我在 2017 年之前，为数不多的创业经历中投入精力最大、运作最为正式、距离成功最近的一次。它的概念足够新颖而且迎合时宜；它的市场需求和商业模式被实实在在的用户数据一再验证；它的技术实现非常系统化，从机械控制、CAN 总线、4G 通信、后端和小程序以及数据分析等等。

要让市场和技术能力，升华为价值，也许我们还是差了一些。

1、关于创业思维

我认为：创业成功者，必然是作了“对”的事情，而且是一系列“对”的事情。这个“对”不是绝“对”，它与大环境有关，作得迎合时宜，需而求之，即是“对”。但是我们要作对的事情，首先要有对的思维。思维如建筑，并非几日就能够构建起来。所以需要积累、感悟、养成、修正、迭代（除非你天生就顺应这个社会，或者从小受到相关的熏陶）。这一过程并不是每个人都能作到的，或因愚钝，或因懒惰，或因性格相悖。了解了这个道理，你就明白了为什么非常努力，但是还是会失败。

1.1 原始思维

一切以盈利为目的的行为都是创业，而创业没有贵贱之分。在这种定义之下，我的创业行为可以追溯到大三。“你不上学吗？还有时间搞别的？”自我评价：我从来不是个安分守己的人，总喜欢搞些什么。这种性格为我带了一些好处。从大二开始我已经觉得上课是在浪费时间，课程内容实用价值很低。于是我开始自学单片机和软件开发，基本每天都泡在实验室和工程训练中心。在此期间我经人推荐认识了郭天祥（它是 8 系信通，我是 6 系计算机，而且他比我大一届），当时他正在筹备中国第一届空中机器人（无人机）大赛。而找我主要是因为我当时自学了 VC++，可以帮他们作地面站软件（向无人机发送指令以及获取它的各种信息，比如位置、速度、规律等）。最后，这个比赛获得了全国一等奖。而我们学校（哈工程）有一个政策，获得全国比赛三等奖以上可以直接报送研究生。所以，我从大二开始已经获得了保研资格（当然这只是资格，最后到大四的时候还是要看成绩的，但是要求没那么高）。有这样的金甲护体，我有了更多的时间来作我自己的事情。当然，郭天祥也已经不用为考研奔波，也是一个自由之身。从那开始，我俩开始承接项目。他主要是外部联络，欲称拉活，而我还是主要关注技术，作项目技术研发。

我们经历的项目不少，比较大的有黑龙江省军区的一款火炮油量计算器，还有哈尔滨某公司的电池智能活化仪。你们应该佩服我的脑子，到现在快到 20 年了，我还记得。

这算是我最早的创业行为，构建了原始思维：我要接项目挣钱。除此之外，还没有别的想法。或者说，充其量顶多是一个大学生想打工挣点零花钱。

1.2 升华思维

我在创业的盈利意识上比较愚钝，更多时候我反而是在享受项目研发过程中的快感，导致很多项目到最后我根本就不提钱的事。从本质上来说，我热衷的是学习和实践技术，并乐

此不彼。这样的性格是对的吗？在学生时代，象牙塔内，它似乎是对的，我慢慢成为了技术明星，并一度被推到了比较高的位置，还引来了新闻机构的专访。导师也越来越器重我，开始委以重任，代表学校和斯坦福的团队一同参与合作项目。但是尽管这样，我的口袋并没有鼓起来，我还是一个穷学生，或者说是有一些光环的穷学生。这算创业？我觉得顶多算是品学兼优罢了。

不得不承认，郭天祥与我的不是同一类人。他有很强的利益意识，一切的付出都要有收益。我开始意识到这个问题，搞技术的目的不是玩和自嗨，而是换取价值，但是在今后的道路上，我发现我的性格与技术有巨大的粘性，意思就是总也放不下技术。一不小心就会陷入到追求技术完美的深坑之中，而忘记创业的核心目的是盈利。

2007 年前后，郭天祥围绕 51 单片机的一系列产品，如开发板、视频教程、书籍开始展露头角，开始被市场所接受，开始大卖。这个时候我才恍然大悟，原来这些可以撬动如此大的市场。于是我开始研发我自己的开发板，从天狼星 Sirius 到后来的 ZN-X 模块化开发板，以及相配套的视频教程《C51 单片机 C 语言》、《单片机基础外设 9 日通》等等。我自认为我作事情是比较认真而且有毅力的，在我手上烂尾的事情并不多。我的这些东西投到市场上，反响也不错，多的时候一天可以出货 50 套，但是我知道与郭天祥相比我还是差得远。因为我很多时候还是在犯老毛病，过度的对技术较真，总是想作新东西，而疏于经营和推广。但是因为思维的转变，我开始有了一些名气。应该说郭天祥对我的影响是比较大的。没有他，我现在也许仍然是一个默默无闻的工程师而已。随着思维方式得不断演化，其实说白了就是在潜移默化之间在效仿郭天祥的作法，并理解他的思维逻辑。我开始和很多的平台合作，比如 elecfans、21ic 等等，去推广我的视频和相关产品，当然这过程中也会有不少项目找上门来。我也开始招募小兄弟来一起作。这样，慢慢形成了我的一种模式。

1.3 思维的歧途

你活着，作事情，到底是你理智的思维为主导，还是性格为主导。我认为理智让你进入成功，随性让你慢慢迷失。我比较反感重复性的工作和一成不变的状态。一直在搞开发板，尤其还有千篇一律的卖板模式和技术支持，我开始觉得没什么意思。我开始变得随性，不再追求利益，想作一些让自己从心眼里高兴痛快的事情。我回归了技术，有 3 年的时间，我深陷 FAT32 文件系统的技术深坑之中，坚毅的性格和扎根心底那个技术狂魔趋势我创造了 znFAT，并开网站和写书来推广。虽然我知道，znFAT 要盈利是非常难的，但比起盈利，我的性格让我更倾向于把它作好，因为我的一个假想敌--FATFS。付出总会有回报，znFAT 又笼络了一大部分人群，尤其是围绕 znFAT 我发布了很多的实验，比如 SD 卡 MP3 播放器、录像机、视频播放器等等。我觉得这些，让我更有成就感，起码让我觉得我作不是那么低端的东西。我注定是不会对按照别人的道路去循规蹈矩，复制所谓的成功，而是要走自己的道路。

1.4 创业之心

znFAT 让我认性了一把，安安心心写代码，想问题，其实是一件非常幸福的事情。而且自此之后，我也进行了深入的考虑，创业之心是要有的，技术要换取价值才有意义。但我应该不会再去卖板子之类的。我确实也是这样去作的，我的创业之心也越来越大，而且我告诫自己“不要太过深究技术，你是要创业还是要随性？！”就这样，才有了后面的多个创业项目。“电动车共享充电柜”算是规模比较大的项目。

写了这么多关于创业的东西，希望它不算与本章跑题。

2、一切始于那一夜

2016~2019 年，一场围绕共享概念的创业热潮席卷而来，共享单车、共享汽车等等。有一次我走在路上手机没电了，我刚注册了 TOGO 共享汽车（品牌比较多，还有一度、Gofun 等），突发奇想，我可以扫开一个汽车，进去充会儿电。顿时，一个更深层的想法产生了：

共享充电宝。关于共享充电宝的创业经历，我放在专门的章节去讲。接下来，我就开始投入到了共享充电宝的研发之中。并希望拿着样机和一些公司合作。当时，这个概念还是很新颖的，而且可以预见具有非常好的盈利性，因为手机充电是刚需。

2.1 入伙创业

我当时的一个同事（后来去了美国，再也没见过）知道我在作共享充电宝，而且似乎创业无望（当时共享充电宝已经有大资金进入，来电、街电、美团、怪兽已经开始充斥市场，而我还陷在技术研发的细节之中）。他向我介绍了 WK，称他作了多年的市场销售，手上有很多渠道和资源，而他也正在作一个共享充电的项目，希望能找到技术合伙人。其实我很犹豫，我确实需要一个擅长市场关系推广运作的合伙人，因为我深深地感觉到自己在这方面的短板。我可以包揽所有的技术研发，包括机械，包括软件，但是我似乎有一点社恐。我现在看似颇擅沟通和调侃，那是因为我在刻意锻炼这方面的能力。但是，我向来对于搞市场销售的人员没有什么好的印象，因为他们会忽悠，不实在，不可信。这是矛盾的，但不管怎么样，我还是约了 WK 一起谈谈。

晚上 9:30，一家 KFC。

“于哥，我先介绍一下这个项目。电动车，我是说电动自行车或摩托车，现在公安部下通知，禁止在楼道内充电，所以我认为这是一个很好的机会。”

“可以把电池拆下来，拿到屋里去充啊！”

“这也是不允许的。可能北京这边骑电动车的没那么多，你还感觉不到，南方骑电动车的非常多，所以充电会是一个很大的问题。而且现在这个市场还没有被发掘，我们可以先占领市场，作电动车共享充电箱。”WK 的话似乎有很强的煽动性，让人听起来觉得这个点子不错。

“听起来不错！”

“我打算先从北京作起来。我本身原来是作小区物业安保设备的，所以有挺多小区的渠道资源，后期的投放不是问题。现在主要是缺技术这块，我已经有了一个作软件的，所以希望于哥能加入，负责硬件这块的研发。”WK 道出了他的目的和想法。

其实我是不太想加入的，一方面是对 WK 并不了解，这可能是一个坑，他可能还有他更多的考虑，反正多半不是什么好事；另一方面是对电动车充电这个行业并不了解，我甚至从来没有骑过电动车。

“我出技术没有问题，你这个项目资金投入会比较大。”

“资金都从我这出，而且肯定亏待不了你，于哥，算技术入股，给你 30%的股份”

这些所谓的股份其实一文不值，纯属的画大饼。但是仍然要正视这个项目。

“我需要考虑一下！”

“好的，于哥，我这有一套别人家的设备，你可以参考一下。”

“好，你留下吧，我看看。想好了答复你。”

2.2 决策成金

万事决策最重要。我常用一个例子来比喻：一个算法超级牛，但是它上面有一个 if，那它可能连运行的机会都没有。因为，if 里的条件才是最重要的，否则就是蛮干，结果很可能是错的。

所以对于这个项目，我要慎重的考虑。既不能太激进，又不能太排斥。首先去掉所有不切实际的憧憬和幻想，不要过分脑补。我开始查阅相关的政策，公安部确实有这样的发文。而且不久我就看到了小区物业张贴出的通知，严禁电动车入楼入户。然后就看到有些人从家里扯出电线给电动车充电。随后物业又贴出了禁止私拉电线的通知。这样一来，电动车的充电确实就成了问题。最叫苦不迭的是那些外卖员，他们原来是有两块电池的，轮流在公司室内充电，这样来保证电动车一直可跑。

这慢慢，甚至是已经变成了一个刚需，而且我从网上搜索相关的盈利性电动车充电品牌似乎还没有。这确实是一个市场空白点，有搞头。我初步认为这个项目可以干，值得投入。我又把这个事情讲给我身边的人，他们也觉得是一个好项目，尤其是我岳父。但是需要提醒的是：合作过程中对于技术资料一定要有所保留。

OK，我答复了 WK：我负责所有硬件的研发，把样机作出来。但是后期批量阶段，你要增加人手。还有，你要尽快打通你的渠道，不要卡在投放部署上。再就是，我们既然作就尽全力把它作大，所以你一定要有好的融资渠道。

3、智能充电柜的技术实现

上面所描述的算是一个市场调研和建立项目的过程，当然还有合伙人的评估和权衡。

接下来就要仔细进行需求分析和产品定义了。需求分析是一切的根基，要考虑产品在各种场景下的应用需求。

3.1 需求分析

需求分析一般需要由产品经理来完成，但是像我们这种合伙创业，哪来的产品经理，都需要我们自己来搞。在经过与 WK 充分研究之后，我们提出了以下主要功能。

1、采用柜式充电仓，用户将电动车的电池拆下，将电池放入充电仓中，进行充电；

2、充电仓中留有有可远程控制的插座，提供 220V 交流电；

3、不提供充电器，需要由用户自行携带充电器；

4、充电仓可远程控制开门，由人工进行关门，可对电池和充电器起到防盗的作用；

5、充电过程中，可对充电功率、仓内温度、亮度进行监测，手机端可配合报警或提醒；

6、柜仓应考虑实际电池的体积不一，可安排大仓和小仓；

7、充电柜应具备自推广或自说明功能，考虑安装屏幕，如广告机；

8、充电柜应考虑满负荷下的实际电力条件，柜仓数控制在 20 个左右；

9、充电柜应考虑雨淋和浸泡，加装遮雨顶棚以及基座；

10、充电柜应考虑采用后身可敞开的设计方式，以便调试与维护；

11、充电柜应充分考虑散热，包括后身与仓内；

12、充电柜应有独立的电量统计，以便了解总用电情况，并可考虑远程抄表；

13、充电柜应考虑可级联，包括电力线与通信，可实现多柜合并工作；

14、应有有效便捷的故障显示和诊断手段，便于快速定位问题；

15、柜仓应支持手动控制方式，以便在极端情况下可以开仓；

16、充电柜整体嵌入式软件可通过远程进行更新；

17、应预留有效的远程调试接口，以便远程解决现场问题；

18、整体设计应便于装配、运输和现场安装部署；

19、应有防倾倒设计，并有视频监控功能，以便追溯和观察分析现场情况；

20、充电柜应有电防护措施，整体产品应通过安规相关测试，达到市场准入的标准。

振南为了方便大家理解上面这些需求，作了一个示意图，如图 13.1。

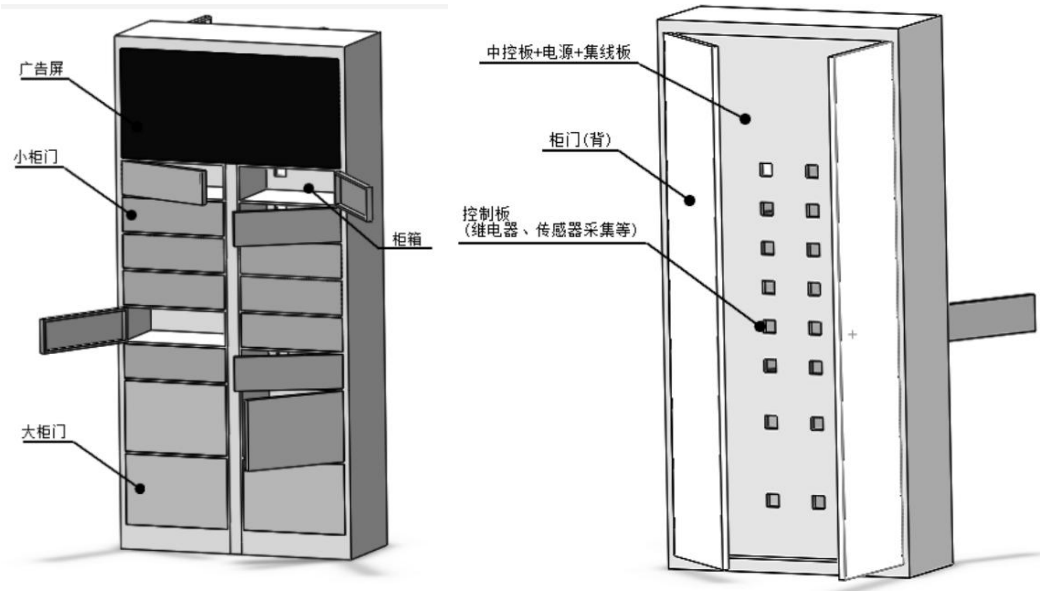


图 13.2 电动车充电柜的需求示意图

3.2 业务流程设计

3.2.1 充电仓位的开放

如图 13.3 所示。

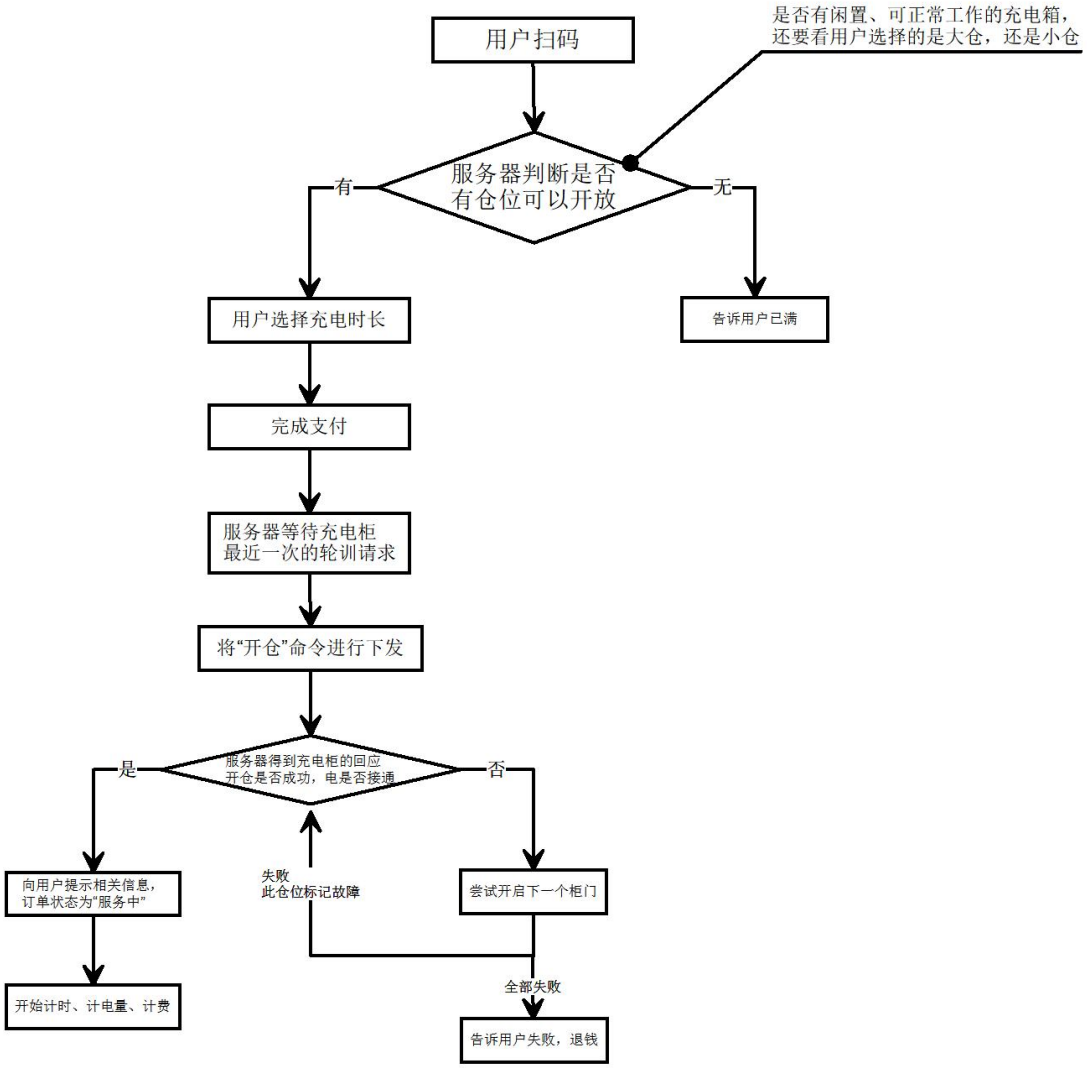


图 13.3 电动车充电柜的开仓业务示意图

3.2.2 服务中与异常处理

如图 13.4 所示。

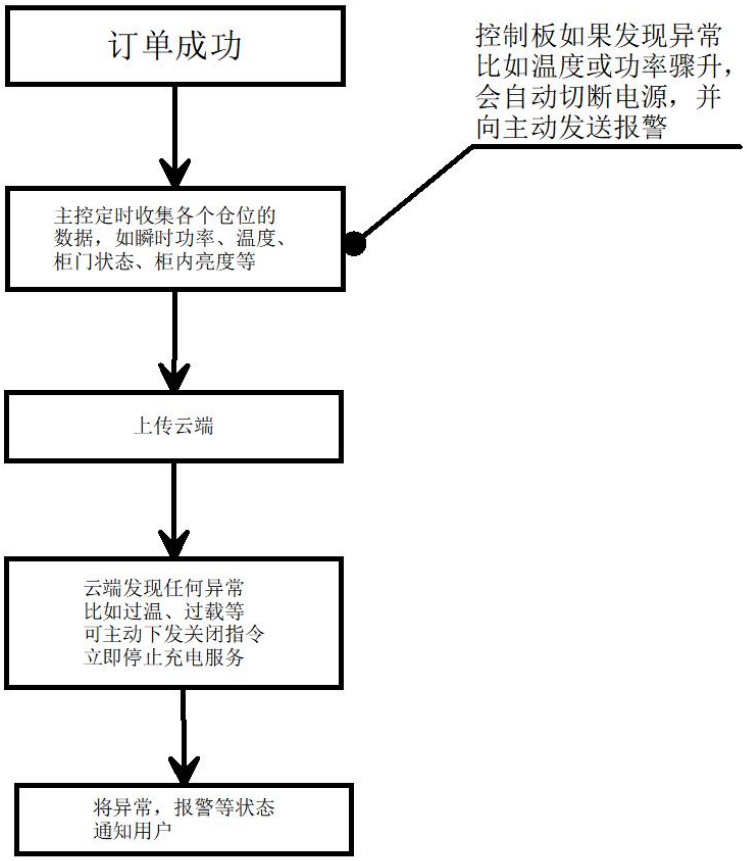


图 13.4 电动车充电柜的服务中与异常处理示意图

3.2.3 充电结束

分为几种情况：正常、意外与主动。

正常结束：服务器计时到时后，将关断相应仓位的电源。此时柜门并不开，等待用户要取出时，再次扫码（或者在手机上点开门），服务器向充电柜发送开门指令。用户取出东西，并成功关闭柜门，此次业务全部结束。

意外结束：充电过程中，因故障和紧急情况，导致某个仓位断电。服务器向用户告知，充电业务中止。

主动结束：用户在充电过程中，可随时中止充电，开门取出电瓶，业务结束。

我们自认为考虑还是比较完备的。其实，当时我们并没有把这些需求和业务流程都形成文档，只是在意识上达成一致。然后就开工了。（这一点在后来想想，确实是非常草率和不完备的）从后面部署之后的实际使用情况来看，我们考虑的还是有很多不到位的地方。但是，不到最后，谁也不能预知会出什么问题。

3.3 技术总体设计

3.3.1 人员分工

人员分工如表 13.1。

表 13.1 三人团队人员分工

人员	分工
WK	负责 柜体的设计、生产加工、厂家沟通、外观设计、广告机采购安装、展示设计、市场推广与部署渠道、柜体装配、以及后期的运

	营维护
小贾	负责 后端的开发、包括微信小程序、配合硬件完成相应的软件开发
于振南	负责 硬件电路设计、嵌入式软件开发、整体调试、通信协议设计、物料采购、批量化加工与测试、配合后期装配与现场部署

基本上都是身兼数职，这就是很多初创团队的状态。2-3 个人，3-5 个人，以一个产品为突破口，作好了大卖融资扩张，作不好就地解散。其实很多人可能都有这样的创业想法，但实际上这样的创业模式，对人员的要求是比较高的。每个人都要能够独挡一面，甚至可能一面还不行。而且这个“面”还不仅仅是技术，需要更综合的能力。

3.3.2 硬件总体设计

如图 13.1 所示。

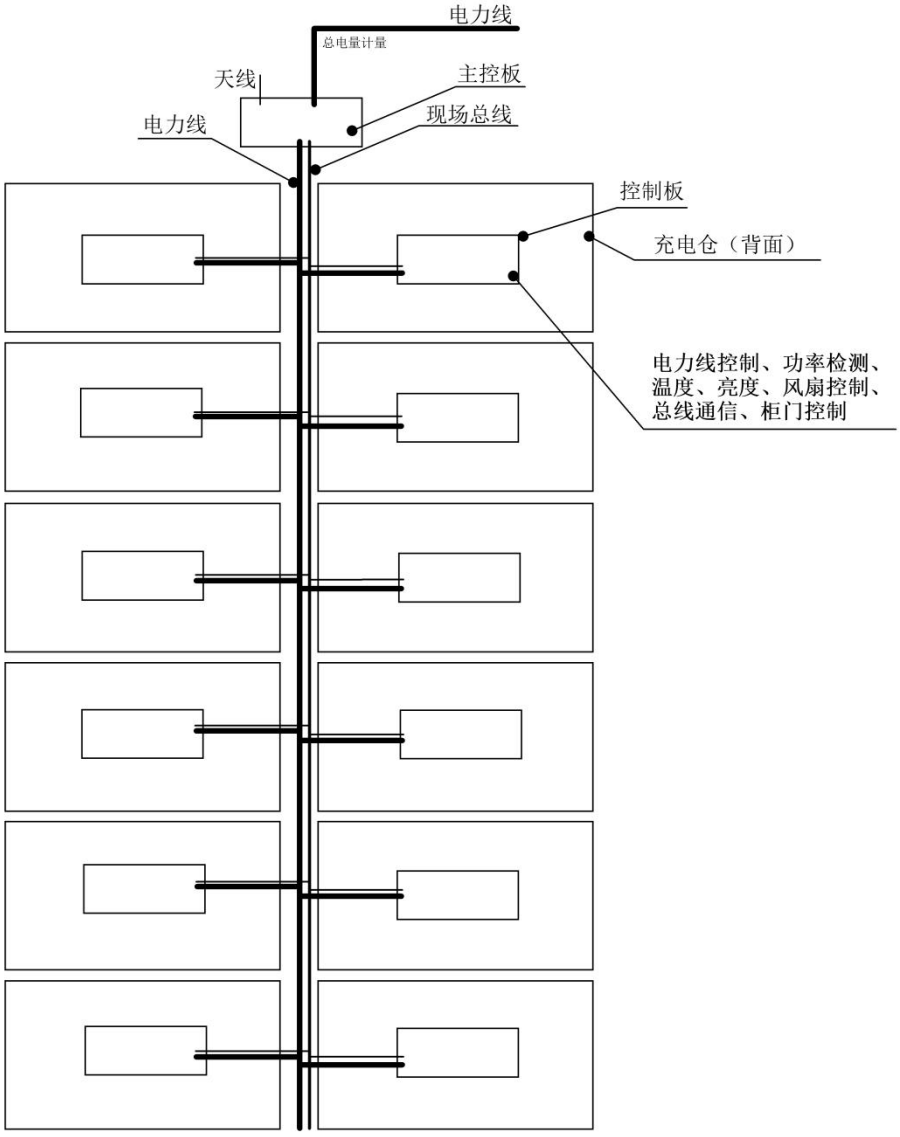


图 13.2 电动车充电柜的硬件总体设计

可以看到，整体采用一主多从的方式，主控向上通过 4G 与云端通信，向下则通过现场总线贯穿，与各个控制板通信，而每一个柜门有独立的控制板。架构看似很简单，但是真正

作稳定并不容易。可以看一下最终实现之后的效果，如图 13.3。

图 13.2 电动车充电柜的硬件总体效果图

这是最终的形态，可以看到多个充电柜是可以级联的。只是让大家先饱饱眼福。我们后面慢慢来讲。

3.3.3 电路原理

我本来不想铺原理图的，但是好像没有原理图，干讲也没什么意思。硬件还是要以图说话。那振南就把这个项目中的各个电路模块分解开，再辅以一定的说明。

1）主控板

1、主控板

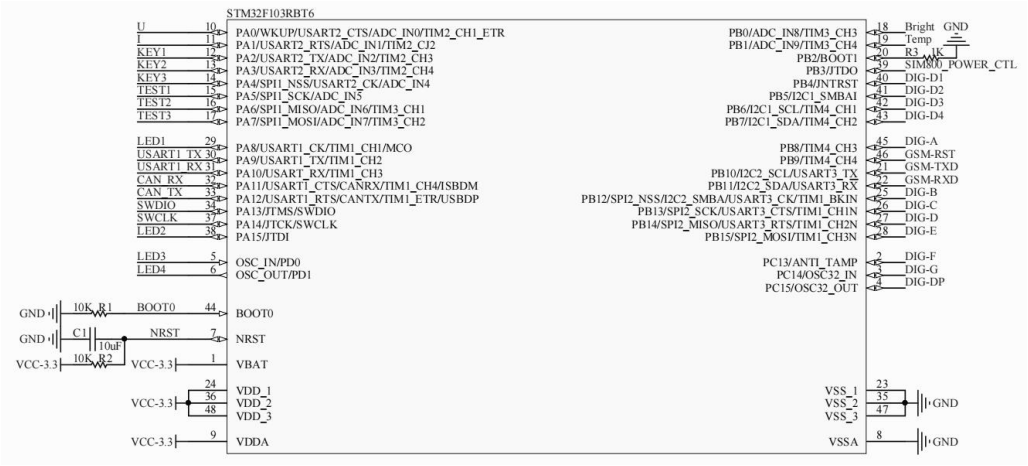
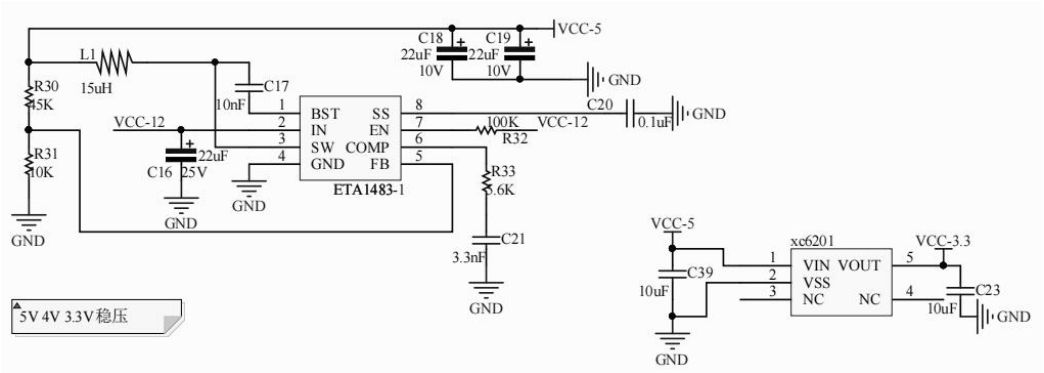


图 13.3 MCU、相关电路及相网络标号

成本考虑振南选用的是 ST 的 STM32F103C8T6 LQFP48 封装（图中是 RBT6，实际使用的是 C8T6）。可能你会问“C8T6 只有 64K 的 Flash，你前面说了还要实现远程升级，那么加上 Bootloader，你 Flash 够用吗？”其实这个选型逻辑可能恰恰相反。很多人在作硬件项目的时候，可能都喜欢习惯性的选型一些比较高端的芯片，尤其是单片机，他们的考虑是先用高端芯片实现功能，然后再作 costdown（降成本）。殊不知，市场的洪流和压力可能不会给你足够的时间去降成本，而会按照成本稍高的 BOM 直接上。这可能是经常发生在很多公司的现象。往往降成本会成为一个毒瘤或者严重问题被提出，而且历时弥长，劳心费事。我们应该从一开始就控制核心和频繁使用的器件的成本。Flash 吃紧？那你应该想办法去优化程序或者删减不必要的功能，而不是轻易更换更高档的芯片，起码不要高得那么离谱。

从图 13.3 中可见一斑，主控板上有按键（KEY）、LED、数码管、SIM800 模块、串口、亮度和温度采集、还有 U 和 I。

2、稳压电路与电源控制



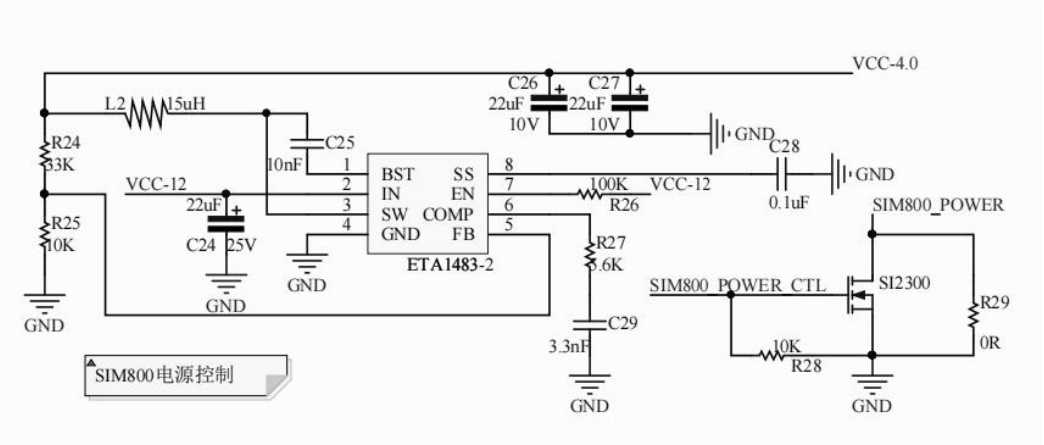


图 13.4 5V/4V/3.3V 稳压与电源控制

为什么用 ETA1483？而不用 MC34063、LM2576 这些常用的芯片？MC34063 这个芯片我很早就开始用了，它好像还有一个配套的计算小工具。它的问题主要是输出电流只有 1.5A，而 SIM800 的瞬时电流可能会比较高。而 LM2576 主要是因为体积比较大，虽然他可以提供 3A 的输出电流。还有就是它们都有一个共同的缺点，就是转换效率比较高，都不会超过 90%。

ETA1483 输出电流可以达到 2A，能量转换效率高达 95%。采用 SOP8 封装，体积比较小。还有一点就是 ETA1483 有很多的兼容型号，选型比较安全，比如 MP1482 等等。

3、CAN 收发器

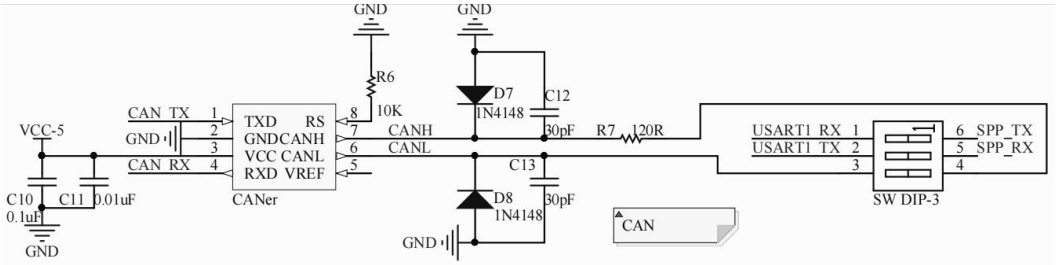


图 13.5 CAN 收发器相关电路

其实一开始我并没有考虑用 CAN，而是使用 485。因为 CAN 相对比较复杂，要用好并不容易。但是 485 在使用过程中，我发现了一些问题：因为我是 220V 交流电与 485 总线一同布线，所以 485 通信似乎受到了干扰。后来我发现很多人都遇到过这样的问题。

有一个哥们就曾被 485 坑惨过：他为工厂安装了一套监测设备（232 接口的），因为 PC 机房比较远，有 700 米。于是他使用了一对无源 232 转 485 转换器。安装、调试，顺利通过，走人。结果，过了一个月，厂家就打来电话，说 PC 读不到数据了，他怀疑是无源的转换器工作不稳定，于是带了一对有源的转换器，换上，工作正常，走人。又过了一段时间，厂家又来电话了，说读不到数据，他又怀疑是转换器设计不好，于是买了一对号称中国最专业的 232-485 转换器－武汉“波士”转换器。满怀憧憬的跑过去，安装、调试，工作正常，走人，他还心想：这次应该是最后一次了吧。但是他想得还是太简单了，几个星期后，电话又响了。他到现场检查了一下，这次更惨，PC 串口不能发数据了，设备里的主控芯片已经爆了。他瞠目结舌，但还是要静下心来仔细查找问题。他看了一下周围的环境，发现 485 线是架空的，离地面约 10 米，在其上方 10 米处有三相四线的电力线通过。现在他开始怀疑是感生电压通过 485 线路损坏了电路。最后没办法，大兴土木，挖了一条 30 公分深的小沟，铺上渡锌的自来水管，485 线从水管里过，渡锌管再通过铁块接地 2M 深，线路上接上瞬变

抑止二极管。最终问题终于解决了。

485 总线在实际工程应用时，有一些注意点：

1、485 线缆尽量使用屏蔽+双绞线；

2、在比较长的分支节点，差分线之间增加端接电阻；

3、在不影响系统响应前提下，尽可能降低波特率；

4、软件上增加校验机制，校验错误要重发。

5、在设备与设备之间，485 布线时尽量避开干扰源，感性设备、高频电路等敏感对象；

6、在收发器入口处，增加 TVS、空气放电管等保护措施，也能避免一些干扰发生。

当时我把 485 线换成屏蔽线或双绞线，可能问题就解决了。但是我的选择是直接改用 CAN 总线+屏蔽线。我使用的 CAN 收发器是常用的 TJA1050，而 STM32F103C8T6 是有内置 CAN 控制器的，这也是我选择使用 CAN 总线的一个原因（不会增加成本，又能使通信质量和相应速率有所保障）。

4、交流电压电流采集

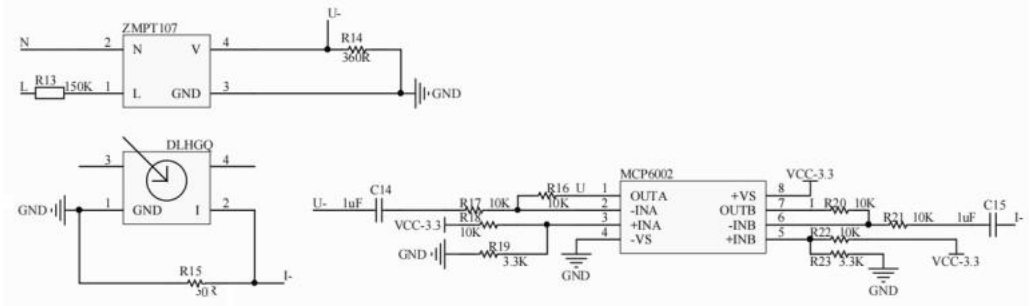


图 13.6 交流电压电流采集电路

主控需要对总体的用电量进行统计，所以它需要采集电力线根部的电压与电流。具体方法是使用电压和电流互感器+电压抬升，再进单端 ADC。

电压互感器，我采用的是 ZMPT107，如图 13.7 所示。

图 13.7 电压互感器 ZMPT107

如图 13.6 中所示是它的典型应用电路。在强电输入端串了一个限流电阻 R13，R14 是采样电阻，输出电压 U-的计算方法是 输入电压*R14/R13，这就是交流电压的变比。值得注意的是，我们平时所说的 220V，其实是指有效值，而单峰值则是一个周期内最高点的电压，约是有效值的 1.414 倍。所以市电 220V 的单峰值大约是 310V。峰峰值是正峰值与负峰值的差，即为 310V - (-310V) = 620V。按照图 13.6 中的变比，360/150000=0.0024，那么输出端的交流信号峰峰值即为 1.488V。

随后，我再用运放将交流输出抬升 1.5V，这样交流便就变成了脉动直流，可直接进入 STM32 的 ADC 了。那采集到波形，如果来计算其等效值呢？大家可以看一下《DPSD》一章。有人可能还会问：“我觉得变比可以再大一些，因为 ADC 0~3.3V 的采集范围还没有用满！”确实，那是因为振南还是比较保守，怕有突发高电压损坏 ADC，所以预留了一倍的余量。而且还在交流信号上加了保护，如图 13.8 所示。

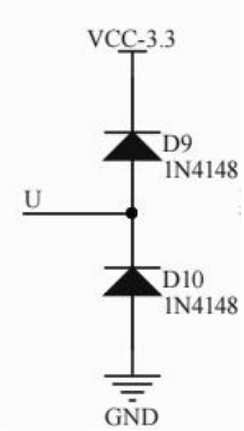


图 13.8 交流信号的保护电路

通过这样的电路，可以将脉动直流的信号限制在 0~3.3V 之间。

对于交流电流的采集，也是相同的作法，我使用的电流互感器是 QE168-1，如图 13.9 所示。

图 13.9 QE168-1 电流互感器

它主要考虑的指标是输入电流和变比，还有精度和线性度。使用的时候，将火线或零线从孔洞穿过即可。我要考虑在充电柜满负荷的情况下，电力线根部能够流过的最大电流。我们常见的电动车电池充电功率不会超过 300W，也就是 220V/1.5A。20 个仓位同时充电，那么电力线根部的电流为 20*1.5=30A。QE168-1 的输入电流最大为 50A（其实并不是说超过 50A 会怎么样，而是会影响它的线性度），变比为 1000:1，这样输出电流最大为 50mA。图 13.6 中，使用 20 欧的采样电阻，最终输出的交流信号峰峰值为 1.7V。抬升 1.5V 之后，可直接进入 ADC。当然，电流互感嘎啦输出也有相应的保护电路。

5、数码管、LED 与按键

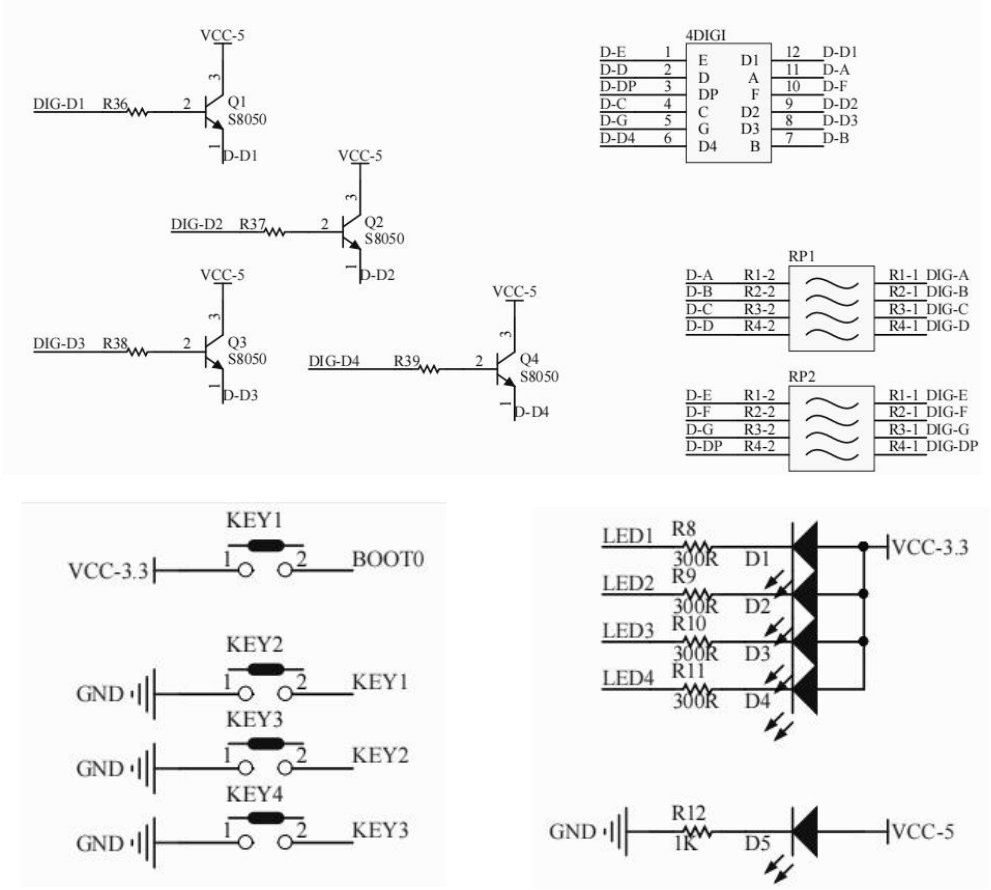


图 13.10 数码管、LED 与按键

我使用的是 4 位 8 段数码管，采用分位扫描的驱动方式，LED 和按键就不用多说了（KEY1 是用于串口 ISP 下载的，因为有 Bootloader 所以 KEY1 很少使用）。为什么要设计这些交互器件。主要是为了现场调试以及后期的远程运维。比如数码管可以轮循显示主控的各项重要内部参数（瞬时电压、电流、CPU 内核温度、链路健康度等等）；LED 可以指示运行状态，比如网络在线、数据收发等；按键可以让我们对主控进行简单的干预，比如快速翻阅特定参数等。

6、温度与亮度检测

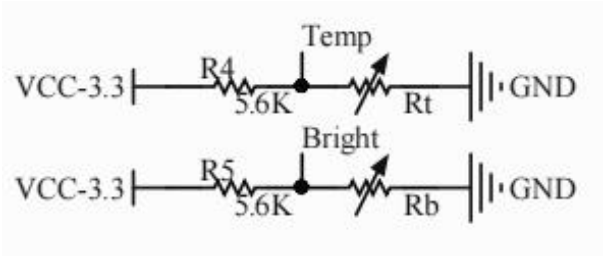


图 13.11 温度与亮度检测电路

先不说振南是通过什么方式来采集温度和亮度的，一个问题是：主控位于充电柜的后身，在那里检测温度和亮度有什么意义？充电柜后身内每一个仓位都有独立的控制板，它控制风扇将仓位内的热量抽出。我想，大家已经明白了。那亮度呢？充电柜后身是两扇门，用钥匙

打开即可维修检测电路。当这两扇门被无故打开时，亮度即会发生骤变，此时主控可向服务器报警。

“你要检测温度，并不需要用 NTC 这种方式，STM32 内部是集成了温度传感器的！”没错，但是这个内部温度传感器我用来检测 CPU 内核温度而非充电柜后身温度。如果你真得用过 STM32 的这个内部温度传感器，你就会发现，它检测的结果与外界温度是有很大出入的。在室温环境下，有时候这个温度能达到 40~50℃。其实你看到的是芯片内的温度。监测这个温度有利于我们了解芯片的硬件状态。有时候，芯片会出现部分烧毁的现象，比如静电、强电串扰等。此时，芯片会出现局部过热的现象。

温度与亮度我都使用敏感元件+ADC 来采集的，关于温度和亮度采集的更详细的内容，振南专门设有一章，在此不再赘述。

7、蓝牙无线串口

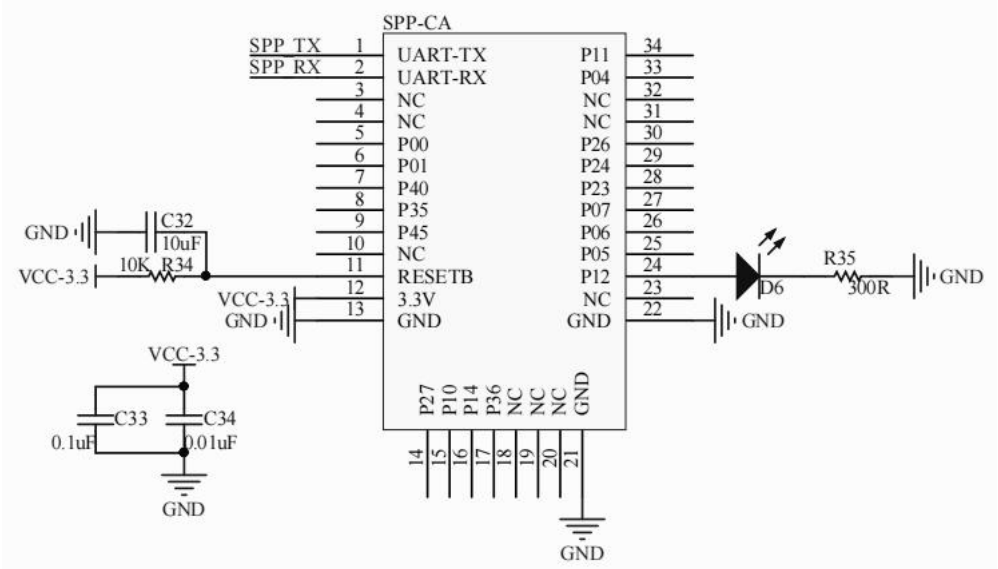


图 13.12 蓝牙串口模块

关于蓝牙串口这种远程调试方式，振南已经在多个章节有过介绍，这里不再赘述。再说一次蓝牙串口的好处吧，在这个项目里，蓝牙串口可以让我不开柜门就可以进行调试，而且无需带电脑。

OK，以上就是主控板的相关讲解，再来看看控制板。

2）控制板

1、继电器控制

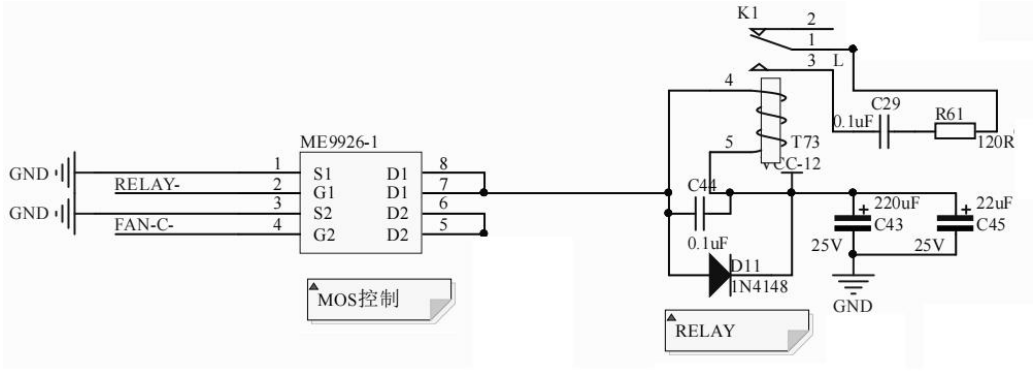


图 13.13 继电器控制电路

我使用的是最常用的 T73 继电器，如图 13.14。

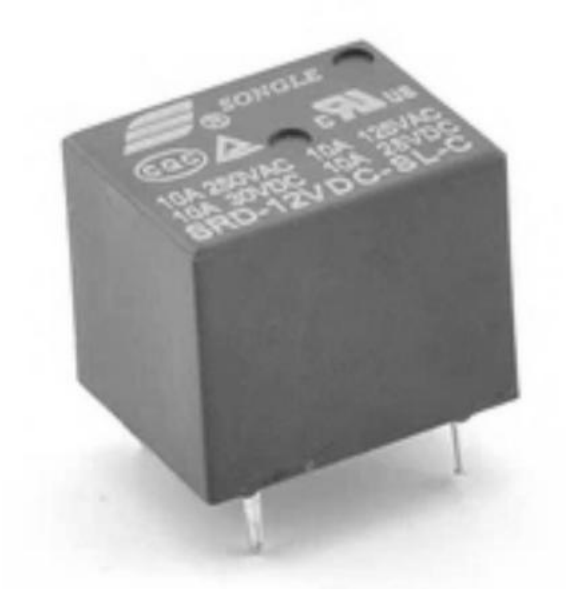


图 13.14 松乐继电器 SRD-12VDC-SL-C

继电器的控制看似非常简单，一个低压 MOS 管即可，但其实有一些细节。

a) 控制端反向电动势防护

要理解继电器控制端为什么会产生反向电动势，我们先来看一下继电器的内部结构和工作原理，如图 13.15。

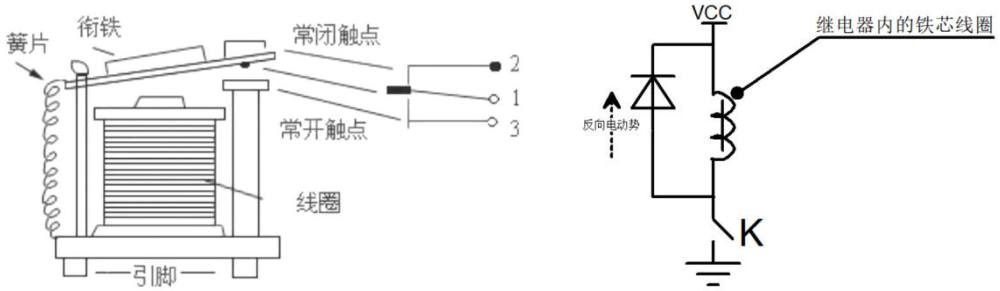


图 13.15 继电器的内部结构和工作原理

当我们在引脚上施加电压时，铁芯线圈中的电流所产生的磁场将吸附衔铁运作，从而接通或断开触点。那么在施加电压的瞬间，铁芯线圈上会产生一个很大的反向电动势（因为线圈是一个感性器件，它会产生反向电动势来尝试抗拒电流的突然输入），如果没有一个回路去将其释放，那它将直接作用于 MOS 管上，可能造成 MOS 的烧毁，从而影响电路寿命。

所以我们在铁芯片线圈的两端并联一个二极管，来构建释放通道。

b) 消弧措施

消弧电路主要是保护强电端的触点。我们曾经有这样的经历，当往插座上插插头的时候，有时会产生火花，这就是电弧，如图 13.16。

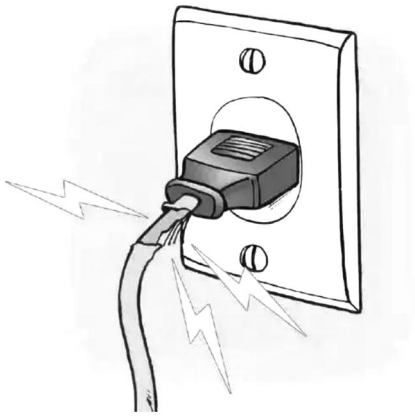


图 13.16 插座插接时产生的电弧

电弧的产生我认为是因为两个接触点产生机械振动而产生的，它类似于按键的抖动一样。电弧的危害主要体现在两方面 1、电弧产生的瞬时高压，可能会对电路系统产生冲击和干扰，这也是继电器控制一般都会加光隔离的原因；2、电弧产生的瞬时高温可能会让触点逐渐灼蚀，甚至烧结（两个触点就像是焊在一起一样，再也分不开）。所以，要有相应的保护电路来将电弧吸收掉。

图 13.13 中的 C29 和 R61 的功能就是吸收电弧，其基本原理是触点两端的电压不能突变。当然这个电容和电阻都需要耐压能力比较高。实际电路中振南选用的是 CBB 电容和水泥电阻（电容主要是吸收积聚能量，电阻则是将能量转化为热量，这仅代表振南个人理解）。如图 13.17。

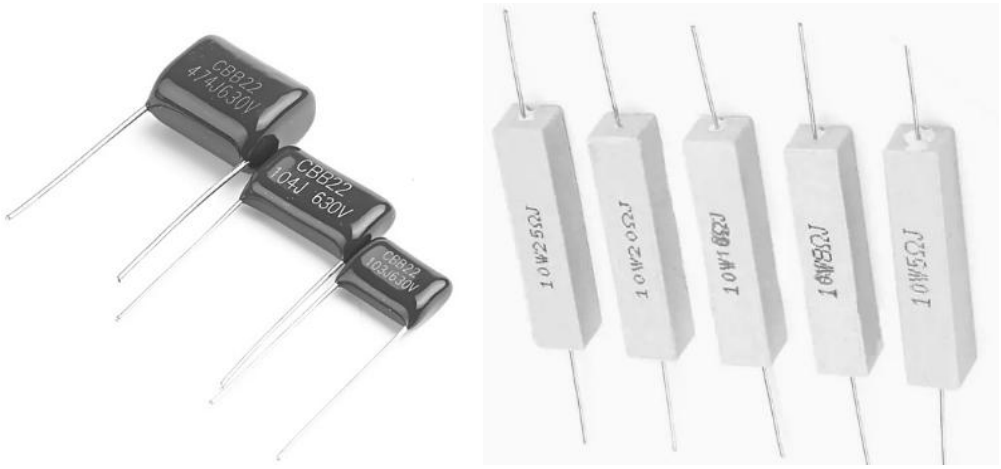


图 13.17 CBB 电容与水泥电阻

除了使用消弧电路，还有没有其它方法可以保护继电器的？有，采用软件方式。软件还能灭弧？听我来讲。

触点拉弧还有一个重要的原因，就是我们控制触点闭合的时机可能落在了交流波峰的附近，此时能量是比较高的。如果我们能够控制闭合的时机，正好落在零点附近，那么就不会产生电弧了。关于“过零检测”的实现，大家可以百度一下，这里就不再赘述了。

还说一句：关于反向电动势的问题，不光是继电器，所有的感性负载都有这个问题，比如风扇、电磁锁（用于柜门开启，如图 13.18）。

图 13.18 电磁门锁（锁头即为其铁芯）

关于电路似乎就这些了。

3.3.4 总体形态

总体形态就是充电柜最终呈现在人们面前的样子。振南直接铺图，如图 13.19-21。

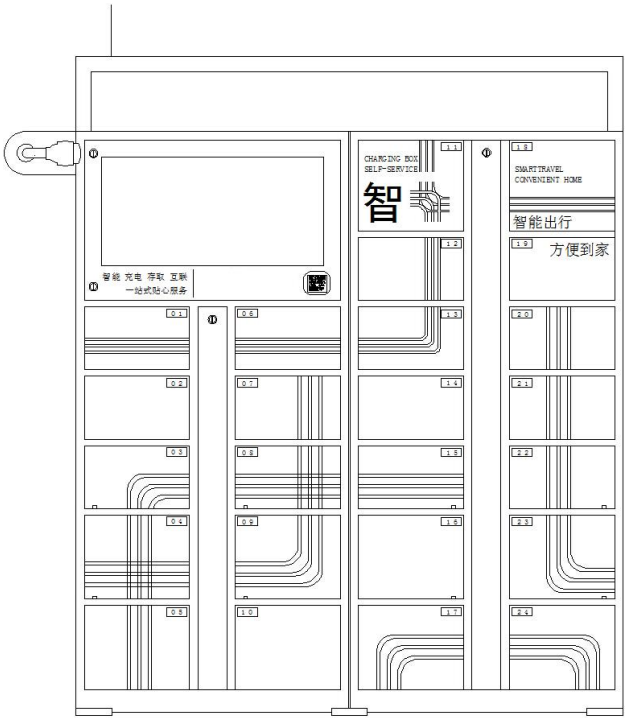


图 13.19 充电柜总体形态设计（主视）

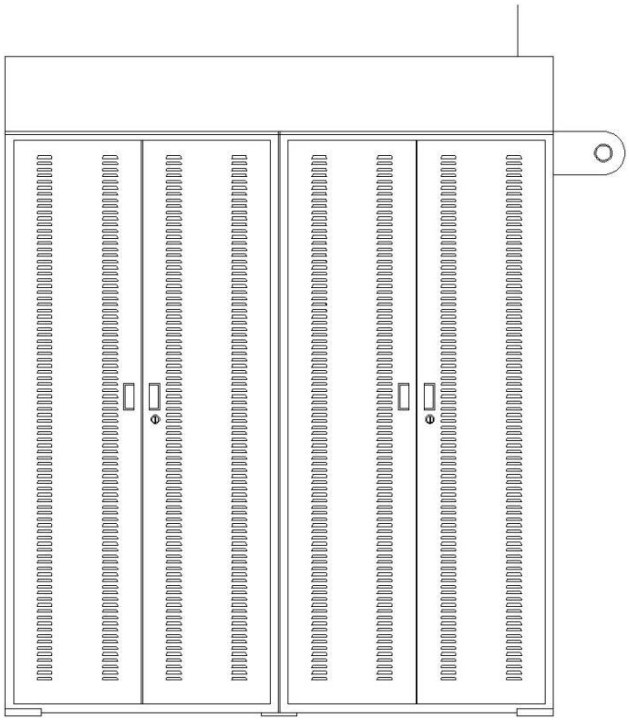


图 13.20 充电柜总体形态设计（后视）

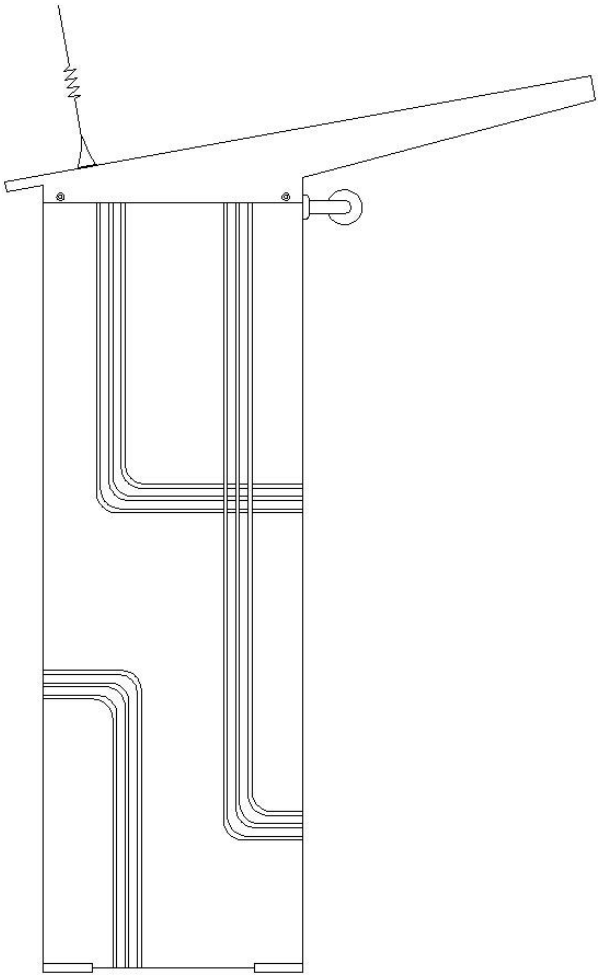


图 13.21 充电柜总体形态设计（侧视）

别光给我们看设计图啊，有没有实物图，当然有，如图 13.22。

图 13.22 充电柜总体形态设计（实物）

柜体的加工自然是委托给专门的工厂的。这个工厂在北京有一个仓库，具体地址是丰台区靛厂路九旭仓库，很多的快递柜都从这里中转发出。老板对我们的创业比较支持，在仓库专门开出一块地方给我们用于调试。这就是我们最初的“实验室”或者说“办公地点”。基于箱柜这种模式的创业者还是挺多的，在这个仓库里我们看到了另外几个创业团队，也在这里调试。我还记得其中一个是用箱柜来作植物培养的，他们要作的好像是要控制箱柜中的光照、温度、湿度这些。所以，在这个大“实验室”里，我们并不孤单。至于他们这些创业项目后来发展怎么样，那就不得而知了。

3.3.5 软件总体设计

在这一节振南不会大篇幅的罗列代码，那样毫无意义（还不如我把代码上传到 GitHub 大家来下载，自己研读）。关于软件，最重要的还是理解其设计思想，充分消化后最终为我所用。（对于产品的嵌入式功能实现，我相信每一个有一定经验的工程师都能搞定，但是你所秉承的设计理念和思想，则决定了你的产品是否稳定、是否优雅）

1、Flash 的划分

很多人在写嵌入式的时候，基本上都是更专注于功能实现和一些技巧的施展，而缺乏在开发之初的全盘思考，美其名曰叫“架构设计”。实际上这项工作非常重要：1、谋划好嵌入式的整体框架轮廓，想明白工程里应该包含哪些模块以及模块之间的关系；2、考虑清楚哪些部分是难点，哪些问题是需要首先解决的；3、考虑清楚如何便捷的调试功能；4、花足够的时间来作嵌入式的推演，写代码前作到整体心中有数；等等

Flash 其实是单片机中比较宝贵的资源，我们不要滥用。一般来说，我们会对 Flash 进行一个划分。不同的区域有着明确的分工。在充电柜这个项目中， 我把 STM32F103C8T6 仅有的 64K 的 Flash 进行了划分，如图 13.23。

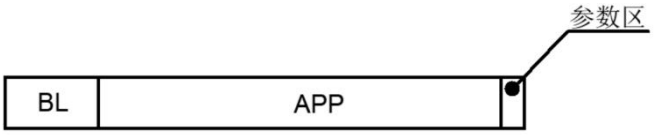


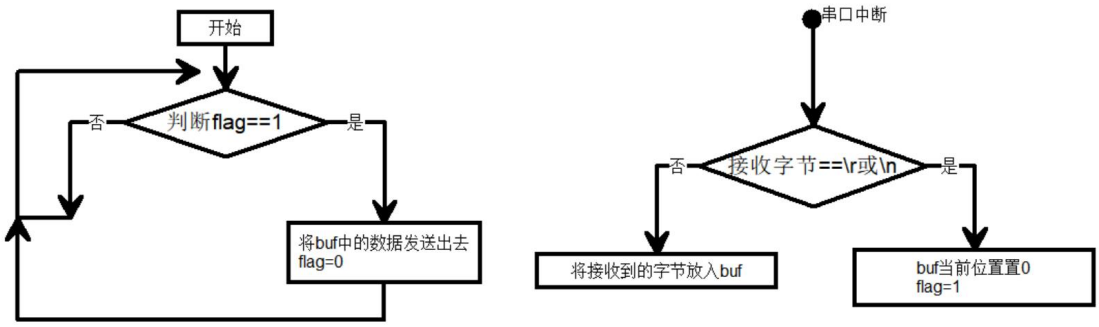
图 13.23 对于 Flash 的区域划分

这样的划分我在《Bootloader》一章中已经介绍过。唯独不同的是在后面还有一个参数区。振南把很多的参数都集中在这里进行管理。这样作有很多好处：1、尽可能多的将程序里的一些值，以参数的方式加入到参数区中，这样使得程序高度可配置。后期调试过程中，如果通过修改参数即可解决的问题，那就不需要重新编译烧录了。2、如果需要批量化的大量修改参数，我们可以以数据块的方式（补丁方式）进行烧录，等等。

2、前后台与 Shell

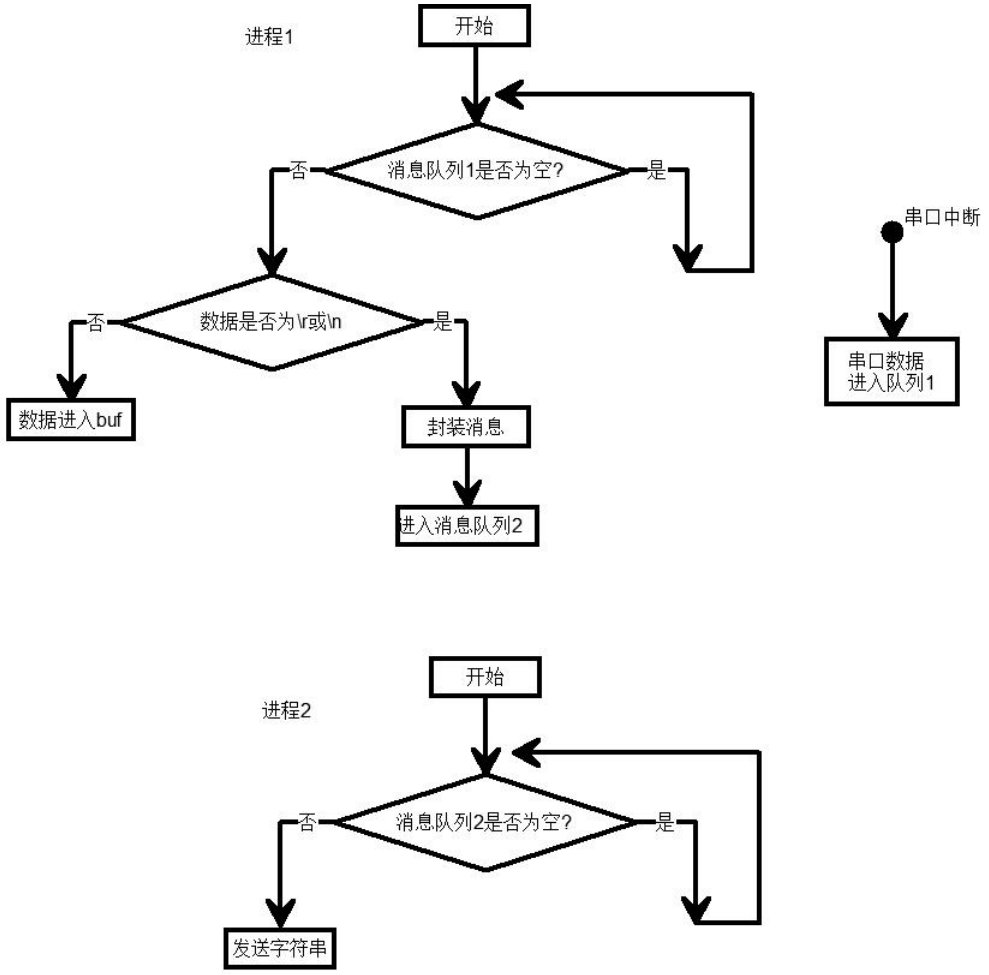
什么是前后台程序？有一定经验的嵌入式工程师应该都知道。它是区别于嵌入式操作系统来说的。前后的程序模式，一般是在 main 中放置一个大循环，在循环中判断一些标志位，从而影响其运行逻辑或分支。有一些比较成型的前后台编程方案，大家可以基于它来开发自己的程序，比如 TI 的 OSAL（很多人管他叫操作系统，但其实它本质上只是一个前后台的管理器）。而这些标志位是由后台的中断服务程序（ISR）来修改的或置位的。通过这样的一种方式，来实现最终的整体功能。

我来举个例子：串口接收到一个完整字符串（以\r\n 结尾），然后把这个字符串再发送出去，采用前后台方式的实现方法，如图 13.24。



13.24 前后台的实现方式

“那使用嵌入式操作系统怎么实现相同的功能呢？”我就知道你要问这个，如图 13.25。

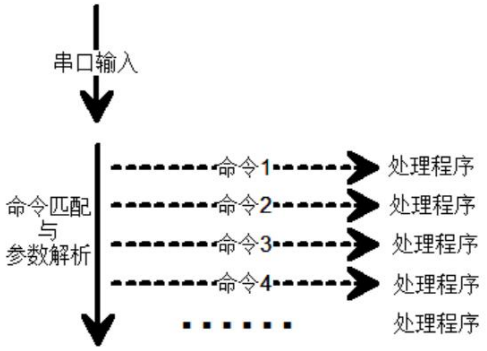


13.25 使用嵌入式操作系统的实现方式

振南使用了消息队列，当然还有很多其它实验方式，比如邮箱、信号量等等。如果串口数据量比较大，还可以考虑使用 DMA。

充电柜项目所有的嵌入式我都是采用前后台方式来实现，因为 MCU 的 Flash 实在有限。

Shell 本身并不区分前后台还是使用用操作系统，它主要是为了提高我们程序的可交互性。从 2012 年之后，我所开发的嵌入式项目，包括一些 DSP 项目，都一律加入了 Shell。甚至，在 Shell 的开发上所花费的时间占到了总体项目的 1/3。关于 Shell，如图 13.26。



13.26 Shell 的工作示意图

我们可以为 Shell 添加丰富的命令以及灵活的参数，这样可以使我们的程序极为灵活和友好。在现场调试时，只要 Shell 设计得足够强大，我们就可以操纵一切，而只需要一条串口线，如果再加上强大的 BL 和蓝牙串口，那我们基本上可以把电脑和仿真器扔掉了。

在充电柜项目中，我设计了如下的命令，如图 13.27。

```
printf("L CMD:\r\n");
printf("  1 smid:S MAID\r\n");
printf("  2 cmid:C MAID\r\n");
printf("  3 smdn:S D N.\r\n");
printf("  4 cmdn:C D N.\r\n");
printf("  5 sfc:S FCOM\r\n");
printf("  6 cfc:C FCOM\r\n");
printf("  7 sa:S Args\r\n");
printf("  8 ea:E Args\r\n");
printf("  9 la:L Args\r\n");
printf("  10 rs:Rs\r\n");
printf("  11 rsim:Rs SIM800\r\n");
printf("  12 sw:C. Work\r\n");
printf("  13 cant:CAN S REQ&CMD\r\n");
printf("  14 sfsi:S CntL\r\n");
printf("  15 cfsi:C CntL\r\n");
printf("  16 sbfd:S BFD\r\n");
printf("  17 cbfd:C BFD\r\n");
printf("  18 sp:S Com Pr.\r\n");
printf("  19 cp:C Com Pr.(\r\n");
printf("  20 stt:S TT\r\n");
printf("  21 ctt:C TT\r\n");
printf("  22 sot:S OT\r\n");
printf("  23 cot:C OT\r\n");
printf("  24 snn:S N_N\r\n");
printf("  25 cnn:C N N\r\n");
printf("  26 ffr:F BF RUN\r\n");
printf("  27 ffs:F BF STOP\r\n");
printf("  28 ffq:R FAN_B CAN\r\n");
printf("  29 sftt:S FAN TT\r\n");
printf("  30 cftt:C FAN TT\r\n");
printf("  31 spon:SIM800 PON\r\n");
printf("  32 spoff:SIM800 POFF\r\n");
printf("  33 srpon:SIM800 RePON\r\n");
printf("  34 td:T D\r\n");
printf("  35 grad:G ADC Data\r\n");
printf("  36 cvip:S&C VIP\r\n");
printf("  37 scid:S C CAN ID CAN\r\n");
printf("  38 rcid:Rs C CAN ID CAN\r\n");
printf("  39 rstc:Rs C\r\n");
printf("  40 d:DL C FW via CAN\r\n");
printf("  41 ccid:R CCID\r\n");
printf("  42 wd:WEBDOWN\r\n");
printf("  43 sdt:S DPSD CT\r\n");
```

13.27 充电柜主控的所有 Shell 命令

这些还是全部的命令，实际比这还要多。有些命令还是非常强大的，比如 wd，它是可以从云端拉取最新固件进行升级；cant 可以向任何一个控制板发送 8 字节的 CAN 数据帧。有人可能发现了，为什么这些命令的说明都是缩写，而且缩的这么厉害，甚至已经失去了说明的意义。这是无奈之举！Flash 划分中 APP 区只有 55KB 的空间，而整个主控程序编译后体积超了，只能把一些 log 或帮助信息进行缩写或注释掉。还有控制板的 Shell 命令，如图 13.28。

```
printf("List All Command and thus Detail:\r\n");
printf("  1 opendoor:Open Door\r\n");
printf("  2 checkdoor:Check Door Open or Closed\r\n");
printf("  3 turnon:TurnON Power\r\n");
printf("  4 turnoff:TurnOFF Power\r\n");
printf("  5 cant:CAN Transmit Test\r\n");
printf("  6 canr:CAN Receive Test\r\n");
printf("  7 tbui:Sample Temprature Bright U & I\r\n");
printf("  8 rs:Reboot\r\n");
printf("  9 cvip:Caculate U&I and P\r\n");
printf("  10 suitdrbbcc:Stop Sampling UITDRBBCC[U I T DOOR RPM B BIGDOOR TEMP_CHIP U_CHIP],Manual Se
printf("  11 sd:Set DEBUGOUT is Disable or Enable\r\n");
printf("  12 caa:Check All Arguments & Working Status\r\n");
printf("  13 stt:Set Temp_Threshold\r\n");
printf("  14 ctt:Check Temp_Threshold\r\n");
printf("  15 sot:Set Overload_Threshold\r\n");
printf("  16 cot:Check Overload_Threshold\r\n");
printf("  17 stts:Set Temp_Threshold_Check Sensitive\r\n");
printf("  18 ctts:Check Temp_Threshold_Check Sensitive\r\n");
printf("  19 sots:Set Overload_Threshold_Check Sensitive\r\n");
printf("  20 cots:Check Overload_Threshold_Check Sensitive\r\n");
printf("  21 tl:Test LEDs\r\n");
printf("  21 tf:Test FAN\r\n");
printf("  22 ff:Force FAN RUN or STOP or EXIT FORCE STATUS\r\n");
printf("  23 cso:Check POWER(RELAY) should be open or not??\r\n");
printf("  24 cts:Check TurnON Status\r\n");
printf("  25 sts:Set TurnON Should Status\r\n");
printf("  26 scid:Set CAN ID in FLASH\r\n");
printf("  27 scei:Switch CAN ID config Source(1 for External BOMA Switch,0 for Inner FLASH)\r\n");
printf("  28 sbd:Set BIGDOOR in FLASH\r\n");
printf("  29 sbdei:Switch BIGDOOR config Source(1 for External BOMA Switch,0 for Inner FLASH)\r\n");
printf("  30:slr:Set LR in FLASH\r\n");
printf("  31:slrei:Switch LR config Source(1 for External BOMA Switch,0 for Inner FLASH)\r\n");
printf("  32:s:Show Whole Config Arguments in FLASH!\r\n");
printf("  33:rbi:Read Two Backup Digital Input Channels\r\n");
printf("  34:sslt:Set Silent Mode\r\n");
printf("  35:adjrc:Adjust RC TRIM\r\n");
printf("  36:sdpsd:Show DPSD time\r\n");
```

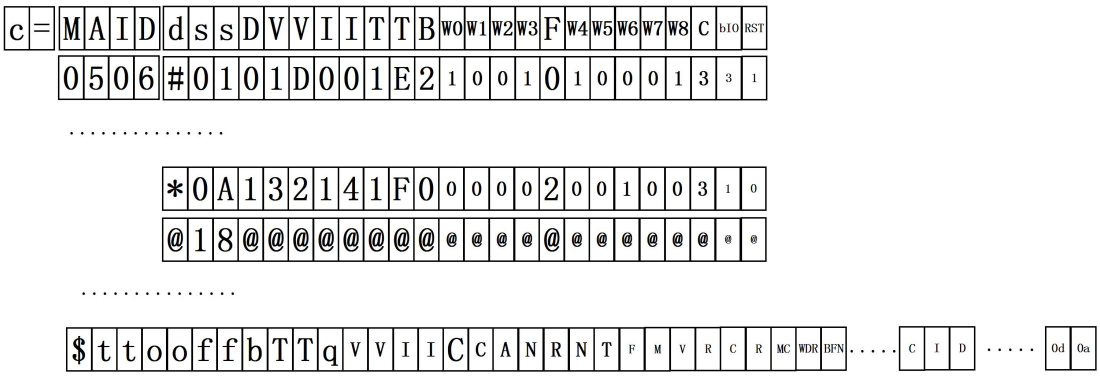
13.28 充电柜控制板的所有 Shell 命令

opendoor 就是打开柜门，turnon 就是接通电源，tf 就是测试风扇，s 就是显示在 Flash 参数区中的所有参数，等等。还有 BL 的一些 Shell 命令（BL 的命令甚至可以通过 CAN 总线向控制板烧录固件）。通过这近百条 Shell 命令，极大的方便了现场调试。（在整个充电柜项目中，我基本上抛弃了 Jlink，只需要一部手机）

有人可能会问：“那手机没电了怎么办？如果全靠手机的话！”别忘了我们作的是什么，共享智能充电柜，OK？只要带上充电器，电不是问题。

3、设备的自我诊断

我们来看看主控到底都在向云端传哪些数据，来看一下通信协议。如图 13.29。



13.29 充电柜主控与云端的通信协议

对协议的详细说明如表 13.2。

表 13.2 充电柜主控与云端通信协议详细说明

c=	数据帧开始符	
MAID	机器 ID，4 个字符	如 0502　1607
d	大小门标记	#小门，*大门
ss	门号	从 1 开始，最大 99
D	门开关状态	1 为关闭，0 为未关闭
VV	强电电压检测值	HEX 表达　实际电压/4　即单位 4V
II	强电电流检测值	HEX 表达　实际电流*10 即单位 百毫安(100mA 或 0.1A)
TT	门内温度	HEX 表达　单位 摄氏度
B	门内亮度	0~3
W0	过温阈值与设置值不符	可能出现 FLASHROM 写入故障或通信故障
W1	过流阈值与设置值不符	可能出现 FLASHROM 写入故障或通信故障
W2	因过温电源切断	门内温度持续一定时间均界于过温阈值之上
W3	因过流电源切断	门内强电电源电流过大，持续界于过流阈值之上（可反映某一小段时间内的瞬时功率过大）
F	风扇转速级别	0-3，门内温度到达(过温阈值-20)时，风扇运行
W4	风扇转速低	可能因风扇老化或外力造成低转速
W5	风扇异转	风扇在无驱动状态下，自转达到一定转速，可能因驱动芯片或外力造成其转动
W6	CPU 过压	CPU 内核电压超过 3.6V，视为 CPU 稳压芯片异常，长期存在烧毁 CPU 风险
W7	CPU 欠压	CPU 内核电压超过 2.6V，视为 CPU 稳压芯片异常，长期最终导致 CPU 不工作
W8	CPU 过温	CPU 内核温度过超过 35 度，视为 CPU 老化或损坏(CPU 电路损坏常伴有芯片温度上升)
C	功率因子	1 位 HEX 表达，0~F，乘以 5.625，结果为角度，此角度的 cos 值，即为功率因子，其参与有效功率计算，P=V*I*C
bIO	备用 IO 数字采集通道	为今后功率扩展预留，可采集两路数字信号，0=00 1=01 2=10 3=11
RST	控制板复位	1 说明控制板刚刚完成一次自动重启 有以下可能造成控制板自动重启 1、CPU 跑飞死机(如 CPU 受到强电磁干扰等)，自恢复机制会

		在 120 秒后，强制 CPU 重启； 2、较长时间无内部数据链路通信，CPU 为排除自身硬件原因，会自重启； 3、CAN 总线通信出现较高错误率，CPU 自重启，试图重新建立稳定的 CAN 通信； …… 一定时间内,控制板复位的次数,一定程度上体现了工作稳定性
	注:若某仓位数据帧因故障无效,则其对应的数据帧内容用@填充(如门号外)	
$	机柜整体信息帧开始符	后面的数据用于描述除柜门之外其它的各项参数
tt	当前机柜中生效的在使用的过温阈值	其应与服务器端下发的过温阈值相符
oo	当前机柜中生效的在使用的过流阈值	其应与服务器端下发的过温阈值相符
ff	当前机柜中生效的在使用的通信频度因子	其应与服务器端下发的过温阈值相符
b	机柜后箱中的亮度	辅助检测后箱门无故打开
TT	机柜后箱中的温度	辅助检测与评估散热效果
q	网络信号强度，RSSI	HEX 表达，0~F，即 0~15，最高信号越好，稳定的通信，此值应在 8 以上
VV	强电根电压	HEX 表达　实际电压/4　即单位 4V
II	强电根电流	HEX 表达　实际电流*10 即单位 百毫安(100mA 或 0.1A)
C	强电根功率因子	1 位 HEX 表达，0~F，乘以 5.625，结果为角度，此角度的 cos 值，即为功率因子，其参与有效功率计算，P=V*I*C
CANRNT	主控对 CAN 总线通信的错误检测,并对 CAN 总线进行干预和维护的次数	此值越大,说明 CAN 通信质量越差，6 位 HEX 表达
FMVR	主控固件版本	如 1000，11ab 等，用于鉴别网络升级固件时的固件标识，升级成功后，则数据帧中的 FMVR 相应改变
CR	主控 CPU 自动重启的次数	可能造成主控 CPU 重启的情况 1、网络通信故障，导致主控在一段时间内频繁出现数据收发不畅，CPU 排除自身硬件因素，则自动重启 2、主控 CPU 跑飞，因自动复位机制，强行重启 3、因各种原因，主控进入死循环，自恢复机制，强行重启 4、因网络更新固件，成功后，主控 CPU 重启以使新固件生效 ……
MC	最近一次网络更新固件,向主控下载的固件是主控固件,还是控制板固件	1、主控最近一次下载的固件为主控固件 0、主控最后一次下载的固件为控制板固件 协助区分网络固件更新时的固件类型，可通过主控间接向控制板进一步更新固件
WBDR	最近一次网络更新固件，其下载过程结束后	1 位 HEX 表达　0~F 0：通过网络更新固件成功

	的错误号	1：因 GPRS 网络离线或掉网造成固件升级失败 2：网络 HTTP 协议承载设置 1 错误 3：网络 HTTP 协议承载设置 2 错误 4：打开 GPRS 协议场景失败 5：场景开启后，IP 未获取成功 6：HTTP 协议初始化成功 7：设置 HTTP 协议参数 1 错误 8：设置 HTTP 协议参数 2 错误 9：访问 WEB 服务器目标文件，即 BIN 文件，失败或超时 A：下载过程中数据接收中断 B：下载后固件数据校验失败(下载数据有错) F：无网络更新固件的操作
BFN	大风扇列数	大风扇装于后箱柜门内侧，每一分机柜装 6 个大风扇，每一个分机柜称为 1 列，BFN 即为列数（BFN 现阶段取 0，即初期暂不使用大风扇）
CID	SIM 卡的 CCID	用此号码用于查询 SIM 卡流量情况

这个协议也不是一蹴而就的，而是在研发过程中不断生长出来的，加入了越来越多的一些考虑。W0-W8 主要是一些异常警告，再加上诸如复位次数、网络更新错误等，可以认为是设备的自诊断。

设备的自诊断是非常重要的，这是工程师们应该引起重视的，即所谓的“反向设计”。通常来说，设备的正常功能只占研发工作的一半，甚至更少。更多的时间我们在考虑各种异常情况。我们一定要转变对功能设计的认识：正常功能是小概率事件，是脆弱而需要百般呵护的。任何一个异常都可能是灾难性的，直接让设备宕机。我们永远想不到所有的异常情况，但是作为工程师，我们应该尽力而为之。而且我们应该预留足够的调试接口，甚至是 backdoor（后门），最好是远程可干预的，以便在未知异常出现的时候，可以救我们一命。背着电脑经常跑现场解决问题的工程师，看似很辛苦，会受到不明就里之人的赞赏。但其实是多半是无能之辈，前期设计没有作好。而且费劲周折来到现场，也只是为了烧录个程序或修改个参数，这种事情完全可以远程来作。我知道很多项目，就是因为频繁出差解决问题，而吃掉了本就不多的利润，甚至还要倒贴。

“这样看来技术是很重要的，对创业起到了决定性的作用，你怎么说技术只占 10%呢？”这是一个相辅相呈的关系。技术是敲门砖，如果你连这块砖都作不好，脆弱不堪，根本敲不开市场之门，那又何谈成功？当然，就算这块砖你作得坚固无比，轻松敲开了市场之门，在激烈的市场竞争之中，在商业层面上，你是否还能保持坚挺，这就属于那 90%的范畴了。

“项目时间那么紧，哪有时间考虑那么完善？尽快上市，占领先机才是最重要的！”没错，但凡有些运营概念的人，都会秉持市场优先的原则，所以研究周期经常被一压再压，这也就是工程师加班谢顶的根本原因。

高手是什么？用最小的代价完成尽可能优秀的任务，这才是高手。用子弹喂出来的狙击手不是好狙击手！

4、协议设计

充电柜的协议分为两大部分 1、主控与各控制板之间的 CAN 通信协议（还包括通过 CAN 来烧录固件的文件传输协议，这部分请详见《Bootloader》一章）；2、主控与云端的通信协议。振南着重说一下后者。有人看到图 13.29 中所示的协议，可能会有些疑问：“为什么协议要设计成这样？全部用 ASCII，为什么不用二进制呢，这样不是更节省流量吗？最前面的

c=是作什么的？”这主要是考虑后端便于解析，主控与云端采用 HTTP POST 的方式来传输数据（关于 HTTP 协议大家可以百度学习一下，在网络通信这方面还是有很用的）。振南把主控与云端通信的 log 给大家看一下。

```
POST /status.php HTTP/1.1
Host: 59.110.127.207
Content-Type: application/x-www-form-urlencoded
Content-Length: 214

c=0a21#01@@@@@@@@@@#02@@@@@@@@@@#03@@@@@@@@@@#04@@@@@@@@@@#05@@@@@@@@@@#0
6@@@@@@@@@@#07@@@@@@@@@@#08@@@@@@@@@@#09@@@@@@@@@@#10@@@@@@@@@@#11@@@@@@@
@@@#12@@@@@@@@@@*13@@@@@@@@@@*14@@@@@@@@@@*15@@@@@@@@@@*16@@@@@@@@@@
SEND OK
HTTP/1.1 200 OK
```

云端的 HTTP 服务在收到这个 POST 请求之后，会使用 status.php 这个脚本对其进行处理。在 PHP 中，我们可以简单的通过$_POST["c"]来获取到 c 这个参数的值，也就是 c=后面的那个长长的字符串。而之所以使用 ASCII 传输，一方面是因为 HTTP 是一种文本传输协议，比较适合于传输 ASCII（当然它也是可以传输二进制的，大家可以了解 Base64 编码）；另一方面是服务器后端脚本处理字符串比处理二进制要方便高效。（我本科时候是搞软件的，除了 VC++，像 ASP、PHP 这些脚本语言也是自学的）

5、CAN ID 的自动分配

既然使用了现场总线的硬件架构，那就躲不开地址或 ID 分配的问题。主控好说，CAN ID 固定为 1，主要是数量众多的控制板。为了解决这一问题，我在控制板上设置了拨码开关，如图 13.30。

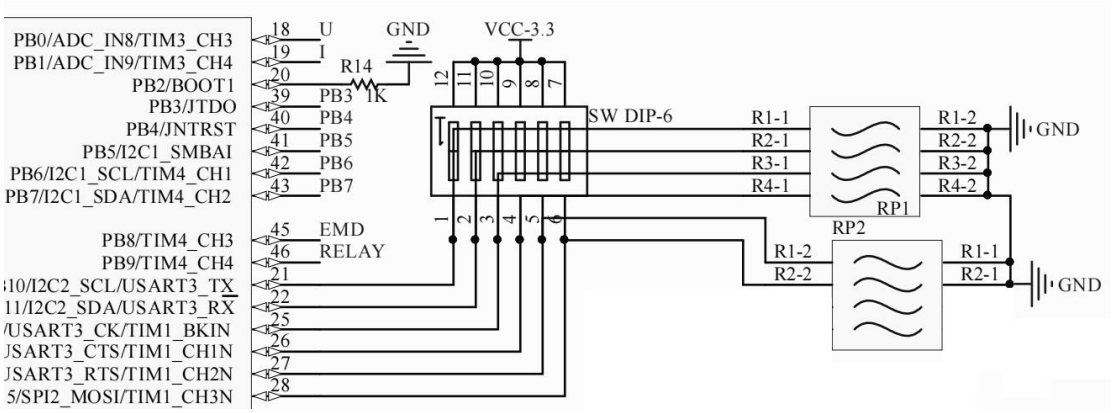


图 13.30 充电柜控制板上的 6 位拨码开关

后来我发现靠拨码开关根本不现实，因为人是会犯错的，尤其还是二进制的。所以，我在考虑如何实现 CANID 的自动分配。请看图 13.31。

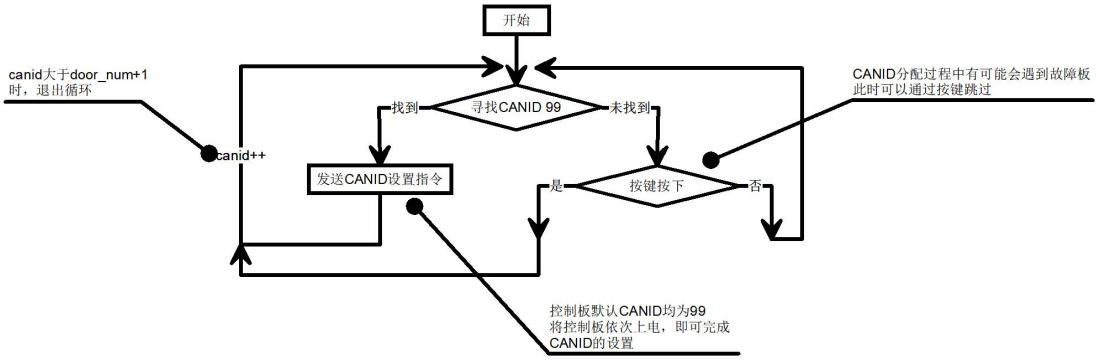


图 13.31 控制板 CANID 的自动分配方法

这样，只需要通过 Shell 命令让主控执行 CANID 自动分配程序，然后再将控制板按编号顺序依次上电，即可实现 CANID 的设置。这种方法极大的提高了后期的出货效率。

关于柜电柜技术方面的内容振南就讲这么多，其中的设计思想和方法希望可以对您的项目起到参考作用。

4、智能充电柜的市场投放