深圳市天贝物联科技有限公司

miyatsu@qq.com

摘要

[通过迷人的摘要吸引您的读者。它通常是文件的简短摘要。   
当您准备好添加内容时，只需单击此处并开始键入。]

共享VR眼镜柜

硬件中控设计开发文档

目 录

[1. 版本历史 3](#_Toc503896002)

[2. 项目简介 4](#_Toc503896003)

[3. 硬件描述 5](#_Toc503896004)

[3.1. 总体设计 5](#_Toc503896005)

[3.1.1. 结构说明 6](#_Toc503896006)

[3.2. 中控硬件设备 6](#_Toc503896007)

[4. 软件描述 7](#_Toc503896008)

[4.1. MCU操作系统 7](#_Toc503896009)

[4.2. Bootloader 7](#_Toc503896010)

[5. GPIO及红外感应 8](#_Toc503896011)

[5.1. GPIO 8](#_Toc503896012)

[5.1.1. Zephyr提供的API 8](#_Toc503896013)

[5.1.2. 包裹函数设计 8](#_Toc503896014)

[5.2. 红外感应 9](#_Toc503896015)

[6. 步进电机控制设计 10](#_Toc503896016)

[6.1. 硬件结构设计 10](#_Toc503896017)

[6.2. 代码设计 12](#_Toc503896018)

[6.2.1. PWM输出 12](#_Toc503896019)

[6.2.2. 定位设计 12](#_Toc503896020)

[7. 柜门控制设计 13](#_Toc503896021)

[7.1. 硬件结构设计 13](#_Toc503896022)

[7.1.1. 硬件功能说明 13](#_Toc503896023)

[7.2. 柜门工作流程 14](#_Toc503896024)

[7.3. 编程模型 14](#_Toc503896025)

[8. 业务逻辑设计 16](#_Toc503896026)

[8.1. VR眼镜借还流程概述 16](#_Toc503896027)

[8.2. 对接接口设计 16](#_Toc503896028)

[8.2.1. 开门接口设计 16](#_Toc503896029)

[8.2.2. 关门接口设计 16](#_Toc503896030)

[9. 耳机出售模块 17](#_Toc503896031)

[9.1. 售卖耳机模块结构设计 17](#_Toc503896032)

[9.2. 售卖耳机模块工作流程 18](#_Toc503896033)

[9.3. 代码设计 18](#_Toc503896034)

[9.3.1. 霍尔元器件定位逻辑设计 18](#_Toc503896035)

[9.3.2. 其他功能逻辑设计 18](#_Toc503896036)

[10. 固件升级 19](#_Toc503896037)

[10.1. Bootloader及其API 19](#_Toc503896038)

[10.2. 固件下载 19](#_Toc503896039)

[10.2.1. URL解析 19](#_Toc503896040)

[10.2.2. HTTP发送请求及固件写入 19](#_Toc503896041)

[10.3. 固件校验 20](#_Toc503896042)

[10.4. 固件版本设计 20](#_Toc503896043)

[10.5. 固件编译及打包 21](#_Toc503896044)

[10.5.1. 固件编译 21](#_Toc503896045)

[10.5.2. 固件打包 21](#_Toc503896046)

[11. 日志系统 23](#_Toc503896047)

[11.1. Zephyr系统级日志 23](#_Toc503896048)

[11.2. 自定义日志处理函数 23](#_Toc503896049)

[11.3. 扩展功能 24](#_Toc503896050)

[12. 工厂测试 25](#_Toc503896051)

[12.1. 接口设计 25](#_Toc503896052)

[12.1.1. 接口变更可扩展性 25](#_Toc503896053)

[12.2. 接口字段说明 26](#_Toc503896054)

[13. 对外接口 28](#_Toc503896055)

[13.1. 获取状态接口 28](#_Toc503896056)

[13.2. 正常开关门接口 29](#_Toc503896057)

[13.2.1. 开门接口 29](#_Toc503896058)

[13.2.2. 关门接口 29](#_Toc503896059)

[13.3. 管理员维护相关接口 30](#_Toc503896060)

[13.3.1. 管理员开门 30](#_Toc503896061)

[13.3.2. 管理员旋转仓体 31](#_Toc503896062)

[13.3.3. 管理员关门 31](#_Toc503896063)

[13.4. 出售耳机相关接口 32](#_Toc503896064)

[13.4.1. 购买耳机 32](#_Toc503896065)

[13.4.2. 耳机补货 32](#_Toc503896066)

[13.4.3. 重新检测耳机库存数 33](#_Toc503896067)

[13.5. 固件升级相关接口 33](#_Toc503896068)

[13.5.1. 请求版本 33](#_Toc503896069)

[13.5.2. 请求升级 34](#_Toc503896070)

[13.6. 日志上报相关接口 34](#_Toc503896071)

[13.7. 工厂测试相关接口 35](#_Toc503896072)

[14. 参考文献 36](#_Toc503896073)

# 版本历史

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修改时间 | 修改人 | 备注 |
| v0.0.1 | 2018-01-05 | 丁 涛 | 第一版 |
| V0.0.2 | 2018-01-16 | 丁 涛 | Bootloader固件校验算法由RSA2048更换至AES256，与v0.0.1版本不兼容；明确描述采用的mcuboot版本号。 |

# 项目简介

本项目是 **深圳市行影科技有限公司** 委托 **深圳市和宇智能科技有限公司**  所开发发的 **VR眼镜租赁管理平台**  中，由 **深圳市天贝物联科技有限公司** 提供人力资源辅助 **深圳市和宇智能科技有限公司** 开发 **VR租赁柜控制模块** 相关任务。

共享VR眼镜柜，借还。

# 硬件描述

本章简单描述硬件部分设计，及硬件限制。

## 总体设计

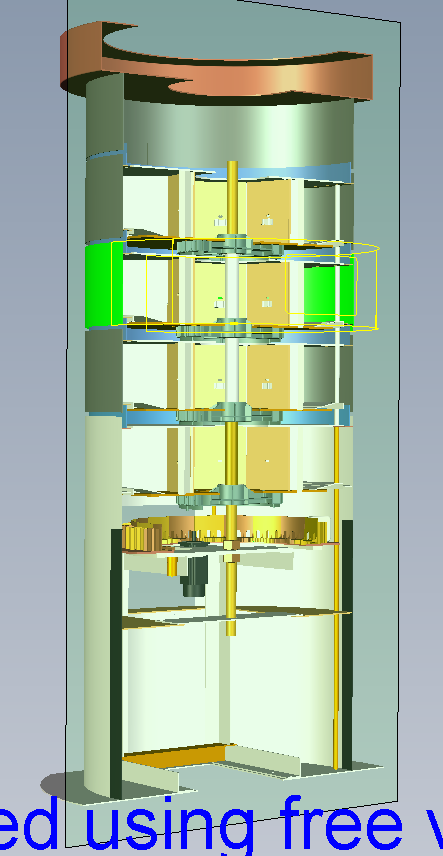
硬件总体设计如图 3‑1所示。

图 3‑1硬件结构设计横截面图

### 结构说明

柜子包含两个核心模块，物品租赁相关功能组件，售卖耳机相关功能组件。

#### 物品租赁相关结构

物品租赁功能设计为仓位形式，共设计有4层，每一层为圆形仓位盘，每一个仓位盘均分为7个仓位，共有4\*7=28个仓位。每个仓位设计有红外感应元器件，可检测仓内是否为空，供应用层使用。

#### 售卖耳机相关结构

售卖耳机相关结构相对简单，详细设计可直接阅读**耳机出售模块**章节。

## 中控硬件设备

中控设备采用“ALIENTEK探索者STM32F407开发板”（以下简称中控主板），所有引出IO口按分类进行分配，具体分配表可参考“IO分配表手册”和各模块源代码内的注释部分及GPIO抽象部分。

# 软件描述

本章简要描述采基于上述硬件所采用的软件名称，及版本信息。

## MCU操作系统

本项目软件平台需要使用RTOS作为操作系统，以向上提供硬件资源抽象。业界内常见RTOS有μC/OS[1]、FreeRTOS[2]及RT-thread[3]等。其代码从可维护性及跨平台可移植性方面考虑，均有各自的缺点。

Zephyr[4]为wind river设计的开源实时操作系统，专注于MCU嵌入式CPU，于2016年加入Linux基金会，现由Intel开源部门维护更新。其系统代码抽象层次高，逻辑结构清晰，拥有良好跨平台设计框架，很好的克服了μC/OS等主流RTOS可移植性差的缺点。从技术可持续性及软件可拓展性方面来考虑，本项目决定使用Zephyr为开发平台，软件版本与社区主分支（master）同步，直至最新版发布（zephyr-v1.10.0）。

## Bootloader

在项目需求中，设备需要支持进行固件自动升级。任何设备若需要支持固件升级都必须要有bootloader的支持。本项目启动初期，zephyr只最新版为zephyr-v1.8.99，暂时还不支持固件升级功能。Apache基金会下mynewt开源RTOS项目在设计初期就加入了bootloader的支持，随着zephyr的崛起，mynewt开发组将bootloader分离出来成立mcuboot项目，该项目在v0.9.0版本发布时，对zephyr进行的移植。

# GPIO及红外感应

本章简要描述GPIO包裹函数及红外感应相关设计方案。

## GPIO

本项目所有针对硬件部分进行管理的，均需要用到GPIO，其中有些需要当输入使用（如仓内红外感应），有些需要当输出使用（如步进电机驱动器PWM输出）而有些需要配置成中断模式（如门到位开关）。

Zephyr提供了GPIO相关API，适用在支持pin脚复用（pin multiplex）的STM32上，向上提供的接口使用上较为复杂，因此在本项目在设计时充分考虑到各模块之间对GPIO相关API调用比较繁琐，将所有GPIO相关API包裹成单一函数。向上提供配置，读，写操作。

### Zephyr提供的API

Zephyr所有硬件功能均抽象成struct device的结构体，该结构体由字符串常量唯一标识，可以使用函数struct device \*device\_get\_binding(const char \*)获取到唯一设备指针。例如要使用UART功能，则可以调用函数struct device \* uart\_dev = device\_get\_binding(“uart1\_name”); 获取uart1的设备句柄。

在开启引脚复用功能时，若要对PA.1和PB.1进行访问，则必须使用GPIOA的句柄访问PA.1脚，使用GPIOB的句柄访问PB.1。不能使用GPIOA的句柄访问PB.1。而唯一标识GPIO相关句柄的字符串常量在Kconfig文件中配置，写出的代码可读性较差。

### 包裹函数设计

分析：所有引脚访问为两个key值，第一为引脚分类PA,PB,PC等；二为引脚数字编码，PA.0,PA.1PA.2等。

设计：设计结构体，包含PA,PB,PC,PD等分类及引脚编号。为减少多个相同字符串占用较多ROM空间，字符串抽象成gpio\_group\_dev\_name\_table，使用enum gpio\_group\_e定位字符串常量所在地址。

关键数据结构：

const char \*gpio\_group\_dev\_name\_table[] = {“GPIOA”, “GPIOB”…}; //唯一标识GPIO设备

enum gpio\_group\_e { //对GPIO设备字符串进行定位

GPIO\_GROUP\_A = 0,

GPIO\_GROUP\_B,

GPIO\_GROUP\_C,

…

};

struct gpio\_group\_pin\_t { //开放给其他模块用于定义访问的引脚

enum gpio\_group\_e gpio\_group;

uint8\_t gpio\_pin;

};

为保持GPIO相关逻辑统一，程序内所有输入输出GPIO默认为高电平，接地时为低电平。

## 红外感应

仓位内的红外感应元器件为简单电平驱动元器件，一次性读入所有红外感应元器件GPIO状态，并保存在本地数组中供业务层读取。红外感应部分相对简单，详细设计思路请阅读程序源代码。

# 步进电机控制设计

本章简要描述仓体转动传动步进电机的结构及软件设计。

## 硬件结构设计

主控电机用于控制柜体内置物架的转动，置物架在结构设计上设有4层7列，共28个仓体。结构如图 6‑1所示。

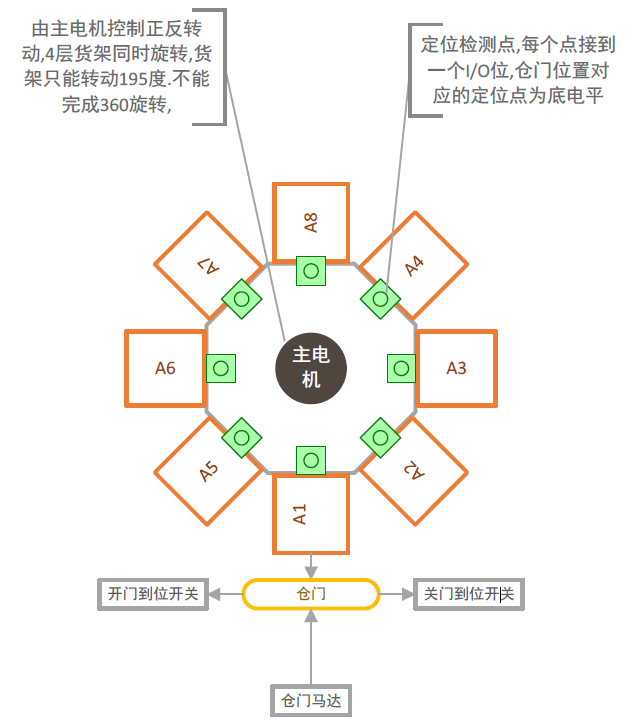


图 6‑1主控电机部分结构设计示意图

**注意**：该设计图为早期设计图纸，最终设计每一层货架为7个仓体。

考虑到中轴在旋转时，柜体内线束容易发生缠绕，因此需要在逻辑上将中轴旋转角度限定在一定范围内。

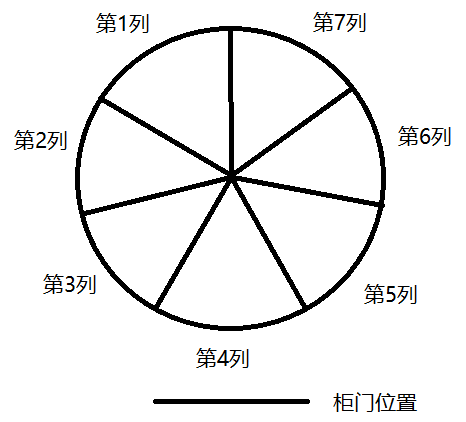
逻辑设计：以第4列为逻辑中心，向左旋转最多转至第1列；向右旋转最多转至第7列。采用这种方式进行旋转角度控制，可避免复杂业务逻辑设计编码。如图 6‑1所示。

图 6‑2仓体旋转示意图

定位点物理位置设计在每一列中心，柜门位置出口设计有金属触点用于定位触发，开发原型机定位部分设计如图 6‑3所示。

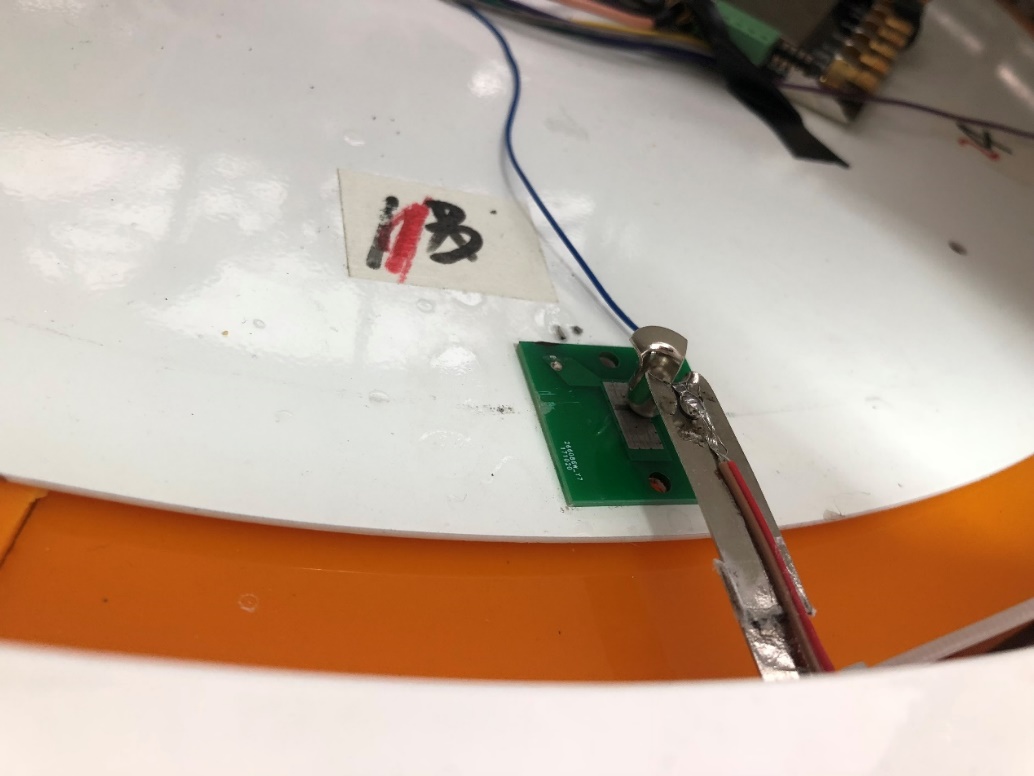


图 6‑3开发原型机中轴定位设计

金属弹片固定在柜门出口位置，步进电机启动时，绿色弹片随着橙色的仓体和白色的固定金属板一起转动，一旦下一片绿色弹片转动到该定位弹片时，GPIO电平会发生相应的变化，由此来判断仓体位置。

## 代码设计

### PWM输出

PWM接口可阅读zephyr内核源代码，PWM相关接口，或参考zephyr内核提供的sample程序fade\_led和servo\_motor两个project。

步进电机驱动器预留有使能接口，本程序未使用该接口，

### 定位设计

根据以上小节描述定位结构设计，本定位软件设计上采用全中断的方式判断，为避免定位GPIO从低电平往高电平跳转时导致触发，先启动步进电机，在定位点不在任何一个点上时再打开对应位置的中断，这样可以避免当前位置弹片误触发。

仓体在转动的时候，偶尔会出现不在定位点时中断误触发，为避免类似中断误触发的情况再次发生，中断处理函数还另外添加了当前仓体位置判断代码，未转到位的情况下不停止转动，直到当前位置与期望位置相符。更多详细信息可阅读程序源代码，源代码内含有大量注释，可为维护人员提供一定参考信息。

# 柜门控制设计

## 硬件结构设计

结构设计上，每个柜门使用5组信号，分别为：电机开门输出信号，电机关门输出信号，门开启到位开关输入信号，门关闭到位开关输入信号，防夹手开关输入信号。门结构设计如图 7‑1所示：

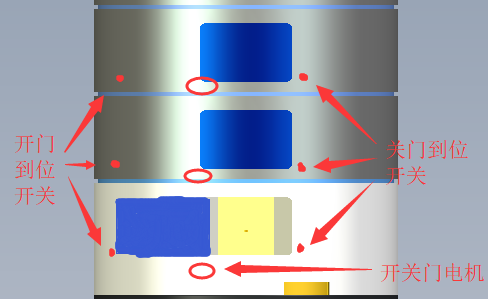


图 7‑1柜门部分结构设计

注意：其中防夹手功能未在图中画出，位置在关门时，门运动方向最前沿的平面上。

### 硬件功能说明

* 电机开门输出信号：用于控制柜门开启
* 电机关门输出信号：用于控制柜门关闭
* 门开启到位开关输入信号：用于检测柜门开启状态，是否完全打开，若已完全打开，中控主板可接收到完全打开的信号输入，由此判断停止开门时机或门状态是否正常。
* 门关闭到位开关输入信号：用于检测柜门关闭状态，是否完全关闭，若已完全关闭，中控主板可接收到完全关闭的信号输入，由此判断停止关门时机或门状态是否正常。
* 防夹手开关输入信号：柜门开关门电机为简单直流步进电机，无法利用外力使电机反转或停机，为使用过程中用户的安全考虑，柜门设计上配有红外感应装置用于输入，供中控主板判断是否继续转动开关门电机。

从产品可用性上考虑，以上5个输入输出GPIO缺一不可，因此，4层机柜共需要使用中控主板GPIO资源20个。在引脚分配上使用P5所有16个引脚，及P4部分4个引脚。分配情况如表 7‑1所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 层、功能 | 马达信号1 | 马达信号2 | 开门到位开关信号 | 关门到位开关信号 | 防夹手开关信号 |
| 第一层 | PD.10 | PD.9 | PD.8 | PE.15 | PE.14 |
| 第二层 | PE.13 | PE.12 | PE.11 | PE.10 | PE.9 |
| 第三层 | PE.8 | PE.7 | PD.0 | PD.1 | PD.15 |
| 第四层 | PD.14 | PD.4 | PD.5 | PD.6 | PD.7 |

表 7‑1柜门部分GPIO分配表

## 柜门工作流程

开门流程：

第一步，检测门是否已经**完全打开**，若未完全打开，则执行步骤二，否则返回打开成功。

第二步，打开门电机开门，执行开门操作。

第三步，等待柜门运动，触发开门开关闭合或超过给定的时间，停止转动电机。

第四步，检测门状态，是否完全打开，并返回结果。

关门流程：

第一步，检测门是否已经完全关闭，若未完全关闭，则执行步骤二，否则返回关闭成功。

第二步，打开电机开关，执行开门操作。

第三步，等待柜门运动，判断运动期间有无异物，。若检测到柜门关闭有异物阻挡，则迅速执行开门操作，一段时间后再执行关门操作，否则等待关门开关是否触发或超时并停止转动电机。

第四步，检测门状态，是否完全打开，并返回结果。

## 编程模型

为简化硬件部分设计，硬件中控不关心用户借/还操作，仅向上提供开门，关门服务。

逻辑上来说，柜门开关检测应当采用中断触发这种异步编程模型对柜门进行控制以降低程序时间复杂度，但在GPIO中断功能完整性测试上，未能通过测试（GPIO中断处理函数调用次数不可控）。为最大程度保证软件输出一致性，在门的“开/关门到位开关”上同时使用中断及轮训的方式判断柜门触发状态，两种测试方式通过信号量进行同步。

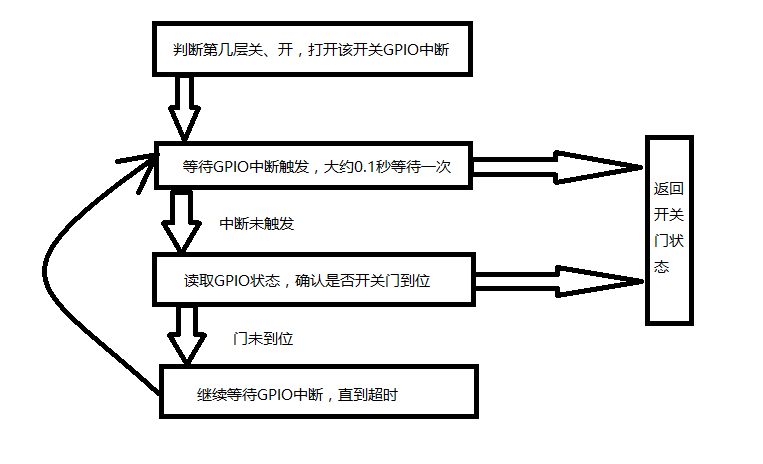


图 7‑2开关门设计流程图

# 业务逻辑设计

## VR眼镜借还流程概述

借流程：用户扫描二维码，借一个VR眼镜，所有与计费业务相关的计算全在云端完成，最终上位机确定好可租借的VR位置，将参数发送给控制板，控制板将仓体旋转到指定位置，然后打开对应仓门。此时，上位机将立马发送关门指令，控制板需要等待用户取走VR眼镜，等待15秒后，关闭仓门。

还流程：用户扫描二维码，还一个VR眼镜，所有与计费业务相关的计算全在云端完成，最终上位机确定好空的仓未位置，将参数发送给控制板，控制板将仓体旋转到制定位置，然后打开对应仓门，用户归还VR眼镜，等待上位机发送关门指令（无超时），读取红外感应状态，关闭仓门。

## 对接接口设计

为降低接口指令复杂度，控制板仅开放开关门功能，其他功能在上位机发送的ext域中进行扩展。

### 开门接口设计

无论用户是借还是还，所有计费相关的计算全由上位机完成，控制板只负责开关门。根据前一小节描述的借还流程可知，对于开门操作而言，无论接还是还，对柜子的操作都是一致的。

首先将仓体旋转到上位机要求的位置，操作成功的情况下，打开对应的仓门，并返回最终操作结果。

为简化上位机的编码工作，借还流程的开门返回值还新增字段”cordinate”: [Number1, Number2]，（上位机开发者要求新增，从软件可维护性上来说，我个人认为这是个非常错误的设计。）其中Number1为当前上位机要求的仓体转动位置（即position），Number2为仓门编号（即layer）。

### 关门接口设计

关门流程上，与开门有所不同，根据前一小节描述的流程可知，在关门时还需要判断当前关门指令是借操作还是还操作。来判断关门等待延时时间。

为保持数据接口的干净整洁，区分借还操作的flag放到上位机ext域中。区分方式：ext域中为json字符串，其中包含有cmd字段；该cmd字段的值若为”borrow”字符串则为借操作，该cmd字段的值若为”back”字符串则为还操作。由此来判断当前开关门指令是用户执行借操作还是还操作。

# 耳机出售模块

## 售卖耳机模块结构设计

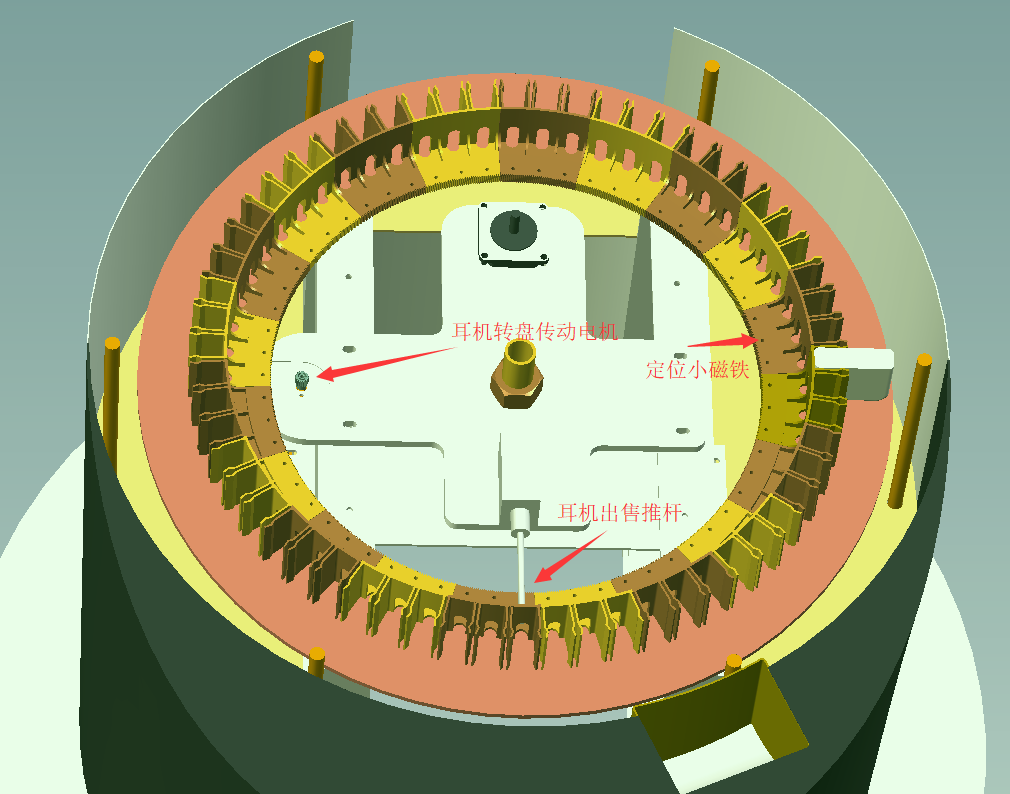
 本柜体要求提供额外耳机出售功能模块，结构设计如图 9‑1和图 9‑2所示：

图 9‑1售卖耳机模块结构设计1

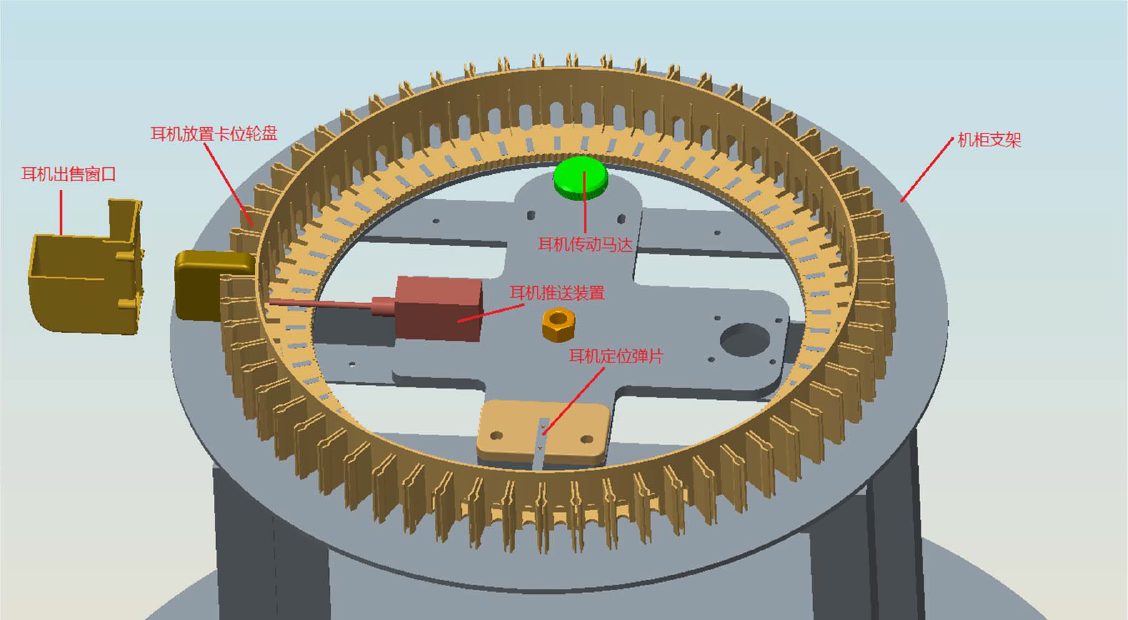


图 9‑2售卖耳机模块结构设计2

注意：用于耳机转盘定位的 “霍尔感应元器件”，出货口检测耳机是否有货的“红外检测元器件”两项，未显示在图 9‑1中，柜体背后缺口设计为耳机补货窗口。

## 售卖耳机模块工作流程

售卖耳机模块，一次推出一个耳机。

1. 首先启动“耳机转盘传动电机”，放置耳机的转盘开始转动，带动“定位小磁铁”一起转动。
2. 当“霍尔感应元器件”检测到磁场变化时，闭合开关，关闭“耳机转盘传动电机”，此时“耳机出售推杆”正好位于“耳机转盘”上的某一个仓位的中心位置。
3. 检测出货口红外感应状态，若存在耳机，则控制推杆工作，推出耳机。
4. 延时一段时间，收回推杆，再次检测出口红外感应状态，若不存在耳机，出售成功，否则失败。

## 代码设计

### 霍尔元器件定位逻辑设计

霍尔感应元器件在有磁场靠近时，开关闭路，否则为开路。元器件连接在中控主板上的单一pin脚，该pin脚默认为高电平，另一端接地。当为避免转盘转动时使霍尔感应元器件离开磁场范围而断路产生异常中断信号，程序设计上，优先转动转盘，待霍尔感应元器件完全断开时再打开对应pin脚的GPIO中断功能，等待中断。一旦中断触发，则表明转盘已经转到下一个耳机仓位。执行其他逻辑处理代码。

### 其他功能逻辑设计

出售耳机功能按照**售卖耳机模块工作流程**小节设计，程序流程相对简单。其他功能包括：获取耳机库存，耳机库存补货等。详细设计请阅读程序源代码，本小节不再详细说明。

# 固件升级

本项目固件升级功能设计参考了阿里云物联网组件中的固件升级方案，阅读本章前请先阅读阿里云固件升级接口帮助文档<https://help.aliyun.com/document_detail/58106.html>

本项目采用与阿里云一致的设计方案，由云端通过MQTT向控制板发送固件下载地址，固件大小及固件MD5校验值。STM32在收到该固件下载地址后，通过HTTP协议将固件下载至本地flash内并进行MD5校验，一切正常便会进行升级操作。

## Bootloader及其API

固件升级需要Bootloader的支持，Zephyr目前仅支持mcuboot。关于mcuboot固件升级详细设计方案请访问mcuboot仓库<https://github.com/runtimeco/mcuboot>，阅读doc中设计相关章节。

Zephyr对mcuboot的接口在zephyr源代码目录树下${ZEPHYR\_BASE}/subsys/dfu/，其中包括针对固件下载后保存的flash操作API及mcuboot升级触发API。详细细节请阅读Zephyr固件升级功能API文档及mcuboot bootloader设计文档。

## 固件下载

固件升级模块接收单一参数固件下载URL，使用HTTP协议从服务器上下载固件，并调用Zephyr提供的flash API写入到bootloader升级指定区域内。

### URL解析

Zephyr提供的HTTP相关API是需要建立正常的HTTP连接之后，通过path进行发送请求，因此需要将URL部分进行解析，host部分用于建立HTTP连接，path部分用于传入HTTP请求函数。对于一些不支持HTTP 1.0特性的服务器，还需要在HTTP请求头部中加入host字段。

URL解析组件使用Zephyr内置http\_url\_parser模块，详细API请阅读Zephyr源代码。

### HTTP发送请求及固件写入

Zephyr的HTTP异步请求支持传入用户自定义指针，使用异步接收响应时，可在响应回调函数中访问该用户自定义指针。在初始化所有HTTP相关组件后，向HTTP响应回调中注册固件写入flash的数据结构句柄，在响应处理的同时，将响应的数据结果写入到flash中。

异步接收响应完全是考虑到内存大小不够而设计，而非性能。为避免发生内存越界问题，栈空间开辟的缓存区应当等所有响应处理完毕后再释放，因此采用信号量对请求发送函数，及响应回调函数进行同步。当且仅当响回调函数应不再被调用（即缓存区不再使用），请求发送函数才能返回。

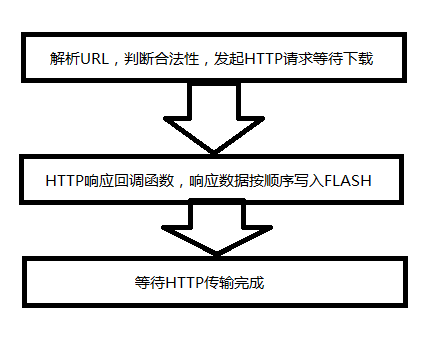


图 10‑1固件下载流程图

对于响应的HTTP报文，本项目采用flash\_img\_context对该报文进行写入操作。相关API请阅读Zephyr源代码，DFU部分。

## 固件校验

HTTP标准采用TCP作为传输层，但在写入FLASH时可能会发生错误。因此固件必须经过校验确认之后，才能进行升级操作。本小节描述的固件校验只针对固件传输及数据保存过程中可能发生的数据异常做校验，并不保证与预期上传固件一致。负责固件升级的bootloader会针对固件进行再一次的校验，以保证固件的一致性。

本固件校验功能直接使用了内置的mbedTLS加密库中的MD5校验函数，相对简单。详细功能可查阅程序源代码。

## 固件版本设计

本项目内核层跟随zephyr社区master分支，**应用层上不区分内核版本**。考虑到软件兼容性及可扩展性，应用层版本号设计应该要符合Semantic Versioning 2.0.0（<https://semver.org/>）。

为避免再次熟悉应用层版本号，应用层版本号设计在符合Semantic Versioning 2.0.0的情况下，跟zephyr内核版本号设计几乎一致，采用major.minor.patch-revision的形式。

固件包命名版本号设计：

prefix\_v1.2.3-5\_debug.bin

prefix\_v1.2.3-1\_alpha.bin

prefix\_v1.2.3-2\_beta.bin

prefix\_v1.2.3-3\_beta.bin

prefix\_v1.2.3\_release.bin

prefix前缀标识项目名称，v1.2.3三位版本号，非release版还带有revision小版本号，用于标识开发测试阶段的固件版本，生产环境忽略revision小版本号。

不带revision文件名设计：prefix\_v1.2.3\_release.bin

带revision文件名设计：prefix\_v1.2.3-[0-255]\_[debug, alpha, beta, rc1, rc2....].bin

## 固件编译及打包

### 固件编译

本小节不提供zephyr编译相关资料，Zephyr编译环境搭建及相关文档请访问zephyr官网：<http://docs.zephyrproject.org/getting_started/getting_started.html>

Zephyr编译后，在cmake输出目录中的zephyr子目录（${CMAKE\_BUILD\_DIR}/zephyr/）会生成zephyr.bin的二进制文件（${CMAKE\_BUILD\_DIR}/zephyr/zephyr.bin），该文件即为可执行的程序二进制固件。

### 固件打包

固件升级功能由mcuboot提供，同时mcuboot为保证固件一致性，也提供固件签名加密功能（AES256和RSA2048）。本项目使用了该加密功能，加密算法采用AES256。

有关固件加密方法，请访问mcuboot代码仓库<https://github.com/runtimeco/mcuboot> ，仔细阅读${MCUBOOT\_BASE}/docs/imgtool.md，本小节不再赘述。

注意：本项目bootloader采用mcuboot v1.1.0版本进行编译，编译时zephyr commit id: 9851b1ac21，请维护人员测试升级功能时，务必将mcuboot切换至v1.1.0版上进行测试。

#### 固件签名密钥

目前本项目使用mcuboot公开加密密钥对，密钥信息如下：

公钥：

const unsigned char root\_pub\_der[] = {

0x30, 0x59, 0x30, 0x13, 0x06, 0x07, 0x2a, 0x86,

0x48, 0xce, 0x3d, 0x02, 0x01, 0x06, 0x08, 0x2a,

0x86, 0x48, 0xce, 0x3d, 0x03, 0x01, 0x07, 0x03,

0x42, 0x00, 0x04, 0x2a, 0xcb, 0x40, 0x3c, 0xe8,

0xfe, 0xed, 0x5b, 0xa4, 0x49, 0x95, 0xa1, 0xa9,

0x1d, 0xae, 0xe8, 0xdb, 0xbe, 0x19, 0x37, 0xcd,

0x14, 0xfb, 0x2f, 0x24, 0x57, 0x37, 0xe5, 0x95,

0x39, 0x88, 0xd9, 0x94, 0xb9, 0xd6, 0x5a, 0xeb,

0xd7, 0xcd, 0xd5, 0x30, 0x8a, 0xd6, 0xfe, 0x48,

0xb2, 0x4a, 0x6a, 0x81, 0x0e, 0xe5, 0xf0, 0x7d,

0x8b, 0x68, 0x34, 0xcc, 0x3a, 0x6a, 0xfc, 0x53,

0x8e, 0xfa, 0xc1, };

const unsigned int root\_pub\_der\_len = 91;

私钥：

-----BEGIN EC PRIVATE KEY-----

MHcCAQEEINeY1S+DASQ701QrflXtTHRhGQCw+VBagk/h6OwGO8/xoAoGCCqGSM49

AwEHoUQDQgAEKstAPOj+7VukSZWhqR2u6Nu+GTfNFPsvJFc35ZU5iNmUudZa69fN

1TCK1v5IskpqgQ7l8H2LaDTMOmr8U476wQ==

-----END EC PRIVATE KEY-----

# 日志系统

日志信息供开发维护人员查阅，用于故障排查使用。日志若存于本地将非常不方便获取日志消息，因此在日志系统上，需要将日志以特定的格式发送至云端。

## Zephyr系统级日志

Zephyr系统设计有日志功能，定义为SYS\_LOG\_\*宏簇，包括SYS\_LOG\_ERR、SYS\_LOG\_WRN、SYS\_LOG\_INF、SYS\_LOG\_DBG，以上SYS\_LOG\_\*宏最终调用printk函数向Console输出日志。

为支持日志模块的可移植性，该SYS\_LOG\_\*宏支持另外定义新SYS\_LOG\_\*底层钩子函数。由CONFIG\_LOG\_HOOK开启。用户可自定义钩子函数实现，可将SYS\_LOG\_\*调用传入的参数（日志信息）写入文件或发送至日志服务器等。

钩子函数定义原型需与printf(const char \*format, …);一致，并调用日志系统提供的注册函数syslog\_hook\_install(size\_t (\*)(const char \*format, …))注册钩子函数。

## 自定义日志处理函数

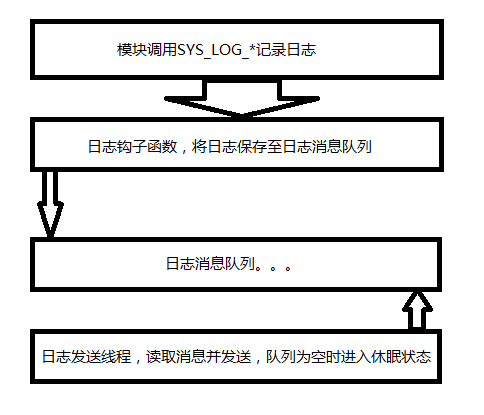
 考虑到日志信息是通过网络进行发送，可能存在一定延时或网络不通。为让中断处理函数也支持记录日志，日志发送过程将由另外的独立线程发送至云端。当前需要记录日志的线程将调用SYS\_LOG\_\*，最终调用自定义函数log\_hook。Log\_hook函数内使用va\_list(), va\_start(), va\_end()等变参遍历宏将日志信息存储到内存空间中，并日志信息加入日志发送队列中。日志消息接收函数将不断从日志队列中获取日志消息，并形成格式，发送给云端。日志系统处理流程如图 11‑1所示：

图 11‑1日志处理模块流程图

## 扩展功能

为了日志处理模块的可扩展性，日志模块还提供将未能正常发送的日志写入文件操作，用户可根据需要将文件内的日志消息添加至日志消息队列中，也可在网络中断时将日志消息队列中的日志消息写入文件中保存。

注意：写入文件操作在使用NFFS文件系统的STM32F4上测试不成功，软件维护人员在未完全测试正常的情况下不应该开启日志写文件功能！

# 工厂测试

本章介绍工厂测试相关细节。

## 接口设计

上位机发送指令格式：

{

“cmd”: “factory\_test”,

“component”: “String1”,

“operation”: “String2”,

“parameter(optional)”: Number,

“ext”: “Application Specification String”

}

控制板返回指令格式：

{

“cmd”: “factory\_test”,

“error\_code”: Number,

“ext”: “Application Specification String”

“Other information (optional, depend on [component/operation/parameter]).”

}

其中：

component为机柜部件名称，类型为字符串，取值范围为：axle、door、etc.

operation为部件执行的操作，类型为字符串，取值由component决定。

parameter为执行操作参数，类型为数字，取值由componet和operation决定。

ext为上位机定义字符串，供上位机使用。

error\_code为操作执行结果，类型为数字，取值范围为-255到255；返回0为正常，其他均视为返回结果异常。

### 接口变更可扩展性

上位机在发送测试指令前，应该通过其他指令获取控制板版本信息，根据控制板固件版本信息，参考技术接口文档，来确定可支持的测试接口。

## 接口字段说明

|  |  |  |  |
| --- | --- | --- | --- |
| component | operation | parameter | 说明 |
| door | open | 1 | 打开第1层门 |
| 2 | 打开第2层门 |
| 3 | 打开第3层门 |
| 4 | 打开第4层门 |
| close | 1 | 关闭第1层门 |
| 2 | 关闭第2层门 |
| 3 | 关闭第3层门 |
| 4 | 关闭第4层门 |
| open\_no\_stop | 1 | 打开第1层门(触碰开门开关不停止转动) |
| 2 | 打开第2层门(触碰开门开关不停止转动) |
| 3 | 打开第3层门(触碰开门开关不停止转动) |
| 4 | 打开第4层门(触碰开门开关不停止转动) |
| close\_no\_stop | 1 | 关闭第1层门(触碰关门开关不停止转动) |
| 2 | 关闭第2层门(触碰关门开关不停止转动) |
| 3 | 关闭第3层门(触碰关门开关不停止转动) |
| 4 | 关闭第4层门(触碰关门开关不停止转动) |
| stop | 1 | 停止转动第1层门马达 |
| 2 | 停止转动第2层门马达 |
| 3 | 停止转动第3层门马达 |
| 4 | 停止转动第4层门马达 |
| open\_all | - | 打开所有门 |
| close\_all | 关闭所有门 |
| stop\_all | 停止转动所有门马达 |
| axle | lock | - | 步进电机刹车解锁 |
| unlock | 步进电机刹车上锁 |
| rotate\_desc | 仓体向逆时针方向转动，门正对的仓体位置编号递减方向（刹车自动解锁，无需干预） |
| rotate\_asc | 仓体向顺时针方向转动，门正对的仓体位置编号递增方向（刹车自动解锁，无需干预） |
| rotate\_stop | 停止转动步进电机（刹车自动上锁，无需干预） |
| position | 获取当前仓体位置编号（正对仓门的编号） |
| relocatioin | 让程序自动将仓体移动到合适的位置 |
| rotate\_to | 1 | 转动仓体到位置1 |
| 2 | 转动仓体到位置2 |
| 3 | 转动仓体到位置3 |
| 4 | 转动仓体到位置4 |
| 5 | 转动仓体到位置5 |
| 6 | 转动仓体到位置6 |
| 7 | 转动仓体到位置7 |
| infrared | refresh | - | 刷新所有4\*7=28个仓位内的红外检测结果并将结果作为返回值发送给上位机，新增cabinets域返回，格式与get\_status中的格式一致。 |
| headset | rotate | - | 启动耳机转盘马达 |
| stop | 停止耳机转盘马达 |
| push | 推动耳机推杆，0.2秒后收回 |
| infrared | 获取出货口红外元器件状态，返回1有物品，0无物品 |
| accuracy | 不断模拟购买流程，最多60次 |

说明：个别指令外带其他返回域如下：

axle.position: 返回值带”position”字段，类型为数字，0为未读到，[1-7]为仓体位置。

infrared.refresh: 返回值带”cabinets”字段，类型为数组，数组内有4个数组类型的值，每一个数组内有7个数字，0为空，1为非空。”cabinets”: [[1, 1, 1, 1, 1, 1, 1], […], […], […] ]。

headset.infrared: 返回值带”infrared”字段，类型为数字，0为空，1为非空。

# 对外接口

## 获取状态接口

上位机可发送指令获取柜子当前状态，数据格式：

{

“cmd”: “get\_status”,

“ext”: “Application Specification String”

}

控制板返回数据格式：

{

“cmd”: “status”,

“ext”: “Application Specification String”,

“axle”: Number,

“door”: [Number, Number, Number, Number],

“cabinets”:

[

[Number, Number, Number, Number, Number, Number, Number],

[Number, Number, Number, Number, Number, Number, Number],

[Number, Number, Number, Number, Number, Number, Number],

[Number, Number, Number, Number, Number, Number, Number]

],

“headset\_stock”: Number

/\*\* 任何指令返回值均带有ext, axle, door, cabinets, headset\_stock 5个域，除非特别说明，以下将这5个域统称为”Other status fields” \*/

}

说明：

ext域为上位机自定义字符串，控制板返回时将原封不动的返回该字符串（除**固件升级相关接口**和**工厂测试相关接口**以外，其他所有上位机指令均需要带ext域，且返回值包含的：ext, axle, door, cabinets, headset\_stock域（5个域以下简称”Other status fields”）。注意：**日志上报相关接口**不需要上位机发送指令，因此不在考虑范围内）。

axle域为步进电机控制的仓体状态；1为正常工作，0为出现异常，无法工作。

door域为仓门状态；1为正常工作，0为出现异常，无法工作；按顺序以此为第1层，第2层，第3层，第4层。

cabinets域为28个仓位红外检测状态，1为有物品，0为无物品（空）

headset\_stock域为耳机库存量，-2为未初始化状态，-1为结构故障，不可出售，0为库存量为0，其他大于0的值为当前柜子内耳机剩余数量，且处于可出售状态。

备注：以上5个域在收到上位机的指令时，均会返回给上位机，以下统称为”Other status fields”。

## 正常开关门接口

### 开门接口

开门指令格式：

{

“cmd”: “open”,

“position”: Number,

“layer”: Number,

“ext”: “Application Specification String”

/\*\*

\* “ext” is json message string, it must contain “cmd” field,

\* more detail please refer to **开门接口设计**

\*/

}

返回数据格式：  
 {

“cmd”: “open\_ok”,

“coordinate”: [Number1, Number2],

“Other status fields”

}

{

“cmd”: “open\_error”,

“error\_code”: Number,

“coordinate”: [Number1, Number2],

“Other status fields”

}

说明：ext域必须为json字符串，其中必须包含”cmd”域且该域取值必须为”borrow”或”back”，详细说明请参阅**开门接口设计**章节。返回coordinate为上位机开发人员要求，方便上位机实现程序（维护人员需要注意此域，不得留空），Number1为请求消息的position值，Number2为请求消息的layer值。

### 关门接口

关门指令格式：

{

“cmd”: “close”,

“layer”: Number2,

“ext”: “Application Specification String”

}

返回数据：

{

“cmd”: “close\_ok”,

“Other status fields”

}

{

“cmd”: “close\_error”,

“error\_code”: Number,

“Other status fields”

}

说明：无

## 管理员维护相关接口

### 管理员开门

将仓体移动到固定位置，并将4个仓门都打开，指令格式：

{

“cmd”: “admin\_fetch”,

“ext”: “Application Specification String”,

“position”: Number

}

返回数据格式：

{

“cmd”: “admin\_fetch\_ok”,

“Other status fields”

}

{

“cmd”: “admin\_fetch\_error”,

“error\_code”: Number,

“Other status fields”

}

说明：打门成功时，cmd域为” admin\_fetch\_ok”，失败时cmd域为” admin\_fetch\_error”并附带error\_code域，类型为数字。以上两组返回结果均带Other status field.

### 管理员旋转仓体

将仓体移动到固定位置，指令格式：

{

“cmd”: “admin\_rotate”,

“ext”: “Application Specification String”,

“position”: Number

}

返回数据格式：

{

“cmd”: “admin\_rotate\_ok”,

“Other status fields”

}

{

“cmd”: “admin\_rotate\_error”,

“error\_code”: Number,

“Other status fields”

}

说明：旋转成功时，cmd域为” admin\_rotate\_ok”，失败时cmd域为” admin\_rotate\_error”并附带error\_code域，类型为数字。以上两组返回结果均带Other status field.

### 管理员关门

关闭所有4个仓门，指令格式：

{

“cmd”: “admin\_close”,

“ext”: “Application Specification String”,

}

返回数据格式：

{

“cmd”: “admin\_close\_ok”,

“Other status fields”

}

{

“cmd”: “admin\_close\_error”,

“error\_code”: Number,

“Other status fields”

}

说明：旋转成功时，cmd域为” admin\_close\_ok”，失败时cmd域为” admin\_close\_error”并附带error\_code域，类型为数字。以上两组返回结果均带Other status field.

## 出售耳机相关接口

### 购买耳机

上位机请求数据格式：

{

“cmd”: “headset\_buy”,

“ext”: “Application Specification String”

}

控制板返回数据格式：

{

“cmd”: “headset\_buy\_ok”,

“Other status fields”

}

{

“cmd”: “headset\_buy\_error”,

“error\_code”: Number,

“Other status fields”

}

说明：购买耳机成功时，cmd域为”headset\_buy\_ok”，失败时cmd域为”headset\_buy\_error”并附带error\_code域，类型为数字。以上两组返回结果均带Other status field.

### 耳机补货

上位机请求数据格式：

{

“cmd”: “headset\_add”,

“ext”: “Application Specification String”

}

控制板返回数据格式：

{

“cmd”: “headset\_add\_ok”,

“Other status fields”

}

{

“cmd”: “headset\_add\_error”,

“error\_code”: Number,

“Other status fields”

}

说明：购买耳机成功时，cmd域为”headset\_add\_ok”，失败时cmd域为”headset\_add\_error”并附带error\_code域，类型为数字。以上两组返回结果均带Other status field.

### 重新检测耳机库存数

上位机请求数据格式：

{

“cmd”: “headset\_recount”,

“ext”: “Application Specification String”

}

控制板返回数据格式：

{

“cmd”: “headset\_recount\_ok”,

“Other status fields”

}

{

“cmd”: “headset\_recount\_error”,

“error\_code”: Number,

“Other status fields”

}

说明：购买耳机成功时，cmd域为”headset\_recount\_ok”，失败时cmd域为”headset\_recount\_error”并附带error\_code域，类型为数字。以上两组返回结果均带Other status field.

## 固件升级相关接口

固件升级接口类似阿里云固件升级，现提供两个功能，请求版本和请求升级两个。注意，固件升级相关接口不包含”Other status fields”。

### 请求版本

上位机请求数据格式：

{

“cmd”: “dfu”,

“sub\_cmd”: “version”

}

说明：上位机发送的json数据**必须**包含cmd及sub\_cmd域，其中，cmd域必须为字符串”dfu”，sub\_cmd域必须为字符串字符串”version”，其他任何数据域将会被忽略。

控制板返回数据格式：

{

“cmd”: “dfu”,

“sub\_cmd”: “version”,

“version”: “version\_string”

}

说明：version域类型为字符串，用于版本对比，详细版本号设计请查阅固件升级章节。

### 请求升级

上位机请求数据格式：

{

“cmd”: “dfu”,

“sub\_cmd”: “upgrade”,

“url”: <http://hostname/path/to/new/firmware/vX.Y.Z.bin>

“size”: 56789,

“md5”: “4a1da493a1bdf5bcf048e9df8eeb5567”

}

说明：以上所有域为必须要求，其中cmd域必须为字符串”dfu”；sub\_cmd域必须为字符串”upgrade”；url域为固件下载链接，暂时**不支持https**，不支持ipv6，不支持域名，不支持非局域网链接（测试失败）；size域为固件大小；md5域为固件MD5校验字符串（不区分大小写）。

控制板返回数据格式：

{

“cmd”: “dfu”,

“sub\_cmd”: “upgrade”,

“error\_code”: Number

}

说明：error\_code域为数字，取值为：-4，升级失败（FLASH写入异常）；-3，MD5校验错误（实际MD5与所下固件MD5不一致）；-2，固件下载错误（HTTP传输错误或FLASH写入错误）；-1，数据格式错误；0，一切正常，开始重启升级。

## 日志上报相关接口

注意，固件升级相关接口不包含”Other status fields”。

控制板产生的日志信息，均会生成json格式数据发送给上位机，数据接口格式：

{

“cmd”: “error\_log”,

“msg”: “Log message string”

}

说明：cmd域为字符串”error\_log”，msg域为日志消息字符串，上位机仅需保留msg域即可。

## 工厂测试相关接口

工厂测试接口可参考**工厂测试**章节。注意，固件升级相关接口不包含”Other status fields”。

# 参考文献

1. Micrium | Real Time Operating Systems, <https://www.micrium.com/>
2. FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions, <https://www.freertos.org/>
3. RT-Thread, RTOS, 物联网操作系统 - RT-Thread物联网操作系统, <https://www.rt-thread.org/>
4. Home - Zephyr Project, <https://www.zephyrproject.org/>