



**本科毕业设计（论文）**

**基于嵌入式Linux系统的地铁AFC设备的AGM读写器设计与实现**

|  |  |
| --- | --- |
| **学 院** | **信息工程学院** |
| **专 业** | **通信工程** |
| **年级班别** | **2019级（3）班** |
| **学 号** | **3119002470** |
| **学生姓名** | **梁丞奂** |
| **指导教师** | **郝禄国** |

2022年11月

**基于嵌入式Linux系统的地铁AFC设备的AGM读写器设计与实现梁丞奂信息工程学院**

摘要

智能读写器是用于自动售检票系统（Automatic Fare Collection）的智能集成电路卡（Integrated Circuit Card）读写模块，它是为轨道交通自动售检票系统设计和开发的，在自动售检票系统中担任着至关重要的角色。作为票卡交互的部件，它的主要功能是读写票卡信息，完成设备与票卡的读写接口。随着自动售检票系统标准读卡器应用日趋广泛，智能读写器性能和功能的进一步提升已成为发展趋势，以满足乘客日益增长的需求

本文作者深入研究地铁自动检票设备并掌握了嵌入式系统应用软件开发技术，以此为基础设计开发了基于嵌入式Linux系统的装配在自动检票闸机上的智能读写器。论文综合调研了当前地铁自动检票系统、国内智能式读写器的研究现状以及市场上对读卡器性能提升的需求，从通信技术、硬件方案和软件协议等角度对智能读写器进行系统的阐述。特别地，本文详细介绍了软件设计方案，包括研究内容和设计开发成果，并通过轨道交通相关的测试软件对读写器软件进行了测试。

测试结果表明，智能读写器能够满足不同轨道运营商系统和城市通卡运营商系统的应用，支持各种新型票卡的验票交易，以及自动检票闸机验票和开关闸等业务功能的报文命令交互。

**关键词：**嵌入式，Linux系统，自动检票闸机，智能读写器，串口通信

Abstract

The intelligent reader-writer module is designed and manufactured for the railway automatic ticket sales and checking system application project, playing an important role in the railway industry. As a component of ticket-card interaction, its main function is to read and write ticket-card information and complete the reading and writing interface between the device and the ticket card. With the AFC system standard card reader application becoming increasingly widespread, the further improvement of the performance and functionality of the intelligent reader-writer is the trend to meet the growing needs of passengers.

The author of this article has conducted in-depth research on the automatic ticket checking equipment in the subway and mastered the application software development technology of embedded systems. Based on this, the author has designed and developed an intelligent reader-writer, which is based on embedded Linux system and installed on the automatic ticket checking gate machine. The paper comprehensively investigates the current automatic ticket checking system of the subway, research status of intelligent reader-writers in China, and the demand for improving the performance of existing card readers, and elaborates on the reader-writer from the aspects of communication technology, hardware solutions, and software protocols. Especially, the software design scheme is detailly introduced in this paper, including the research content and design development results, and the software of the reader-writer is tested with the railway-related testing software.

The test results show that the intelligent reader-writer can meet the applications of different railway operators' systems and city transport card operators' systems, support the verification and transaction of various new types of cards, and complete the message command interaction of business functions such as automatic ticket checking gate verification and switching gate.

**Key words**：Embedded Linux, Automatic Gate, Intelligent Reader and Writer, Serial Communication

目录

[摘要 3](#_Toc129955578)

[Abstract 4](#_Toc129955579)

[1 绪论 1](#_Toc129955580)

[1.1 选题背景及目的 1](#_Toc129955581)

[1.2 国内研究现状 2](#_Toc129955582)

[1.3 论文结构及内容详细安排 4](#_Toc129955583)

[2 方案设计与基本理论 5](#_Toc129955584)

[2.1 需求分析 5](#_Toc129955585)

[2.1.1 总体功能需求 5](#_Toc129955586)

[2.1.2 模块化功能需求 6](#_Toc129955587)

[2.2 基础理论和原理分析 6](#_Toc129955588)

[2.2.1 开发环境 6](#_Toc129955589)

[2.2.2 开发工具 7](#_Toc129955590)

[2.2.3 读写技术 7](#_Toc129955591)

[2.2.4 无线通信标准 8](#_Toc129955592)

[2.3 方案设计 8](#_Toc129955593)

[2.3.1 整体方案设计 8](#_Toc129955594)

[2.3.2 硬件方案设计 9](#_Toc129955595)

[2.3.3 软件方案设计 12](#_Toc129955596)

[2.3.4 方案设计标准 19](#_Toc129955597)

[2.3.5 方案总结 19](#_Toc129955598)

[3 概要设计 20](#_Toc129955599)

[3.1 读写器软件概要设计 20](#_Toc129955600)

[3.1.1 公用接口子模块设计 20](#_Toc129955601)

[3.1.2运营参数模块设计 22](#_Toc129955602)

[3.1.3 自动检票闸机业务功能模块设计 23](#_Toc129955603)

[3.1.4 票卡读写模块设计 25](#_Toc129955604)

[3.2 寄存器数据设计 26](#_Toc129955605)

[3.2.1 寄存器数据存储 26](#_Toc129955606)

[3.2.2 AGM机寄存器数据设计 26](#_Toc129955607)

[4 详细设计 29](#_Toc129955608)

[4.1 软件总模块详细设计 29](#_Toc129955609)

[4.2 初始模块详细设计 35](#_Toc129955610)

[4.3公用接口模块详细设计 38](#_Toc129955611)

[4.2.1串口通信编程 38](#_Toc129955612)

[4.2.2 设备初始化 43](#_Toc129955613)

[4.2.3 版本信息获取 47](#_Toc129955614)

[4.2.4 获取SAM卡信息 47](#_Toc129955615)

[4.2.5 设备降级模式 47](#_Toc129955616)

[4.2.6 获取票价 47](#_Toc129955617)

[4.2.7 参数管理 48](#_Toc129955618)

[4.2.8 时间同步 48](#_Toc129955619)

[4.2.9 获取寄存器值 48](#_Toc129955620)

[4.2.10 获取读卡器当前信息 48](#_Toc129955621)

[4.4运营参数模块详细设计 49](#_Toc129955622)

[4.4.1参数下载 49](#_Toc129955623)

[4.4.2 参数生效通知 51](#_Toc129955624)

[4.4.3 参数加载 53](#_Toc129955625)

[4.5 AGM机业务功能模块详细设计 55](#_Toc129955626)

[4.3.1通道类型处理 55](#_Toc129955627)

[4.3.2入闸处理 57](#_Toc129955628)

[4.3.3出闸处理 58](#_Toc129955629)

[5 系统测试 59](#_Toc129955630)

[5.1 测试环境 59](#_Toc129955631)

[5.2 功能测试 59](#_Toc129955632)

[5.3 测试结果 59](#_Toc129955633)

[总结 60](#_Toc129955634)

[参考文献 60](#_Toc129955635)

[致谢 61](#_Toc129955636)

# 绪论

## 选题背景及目的

集成电路卡 (Integrated Circuit Card，IC卡)，也称智能卡(Smart Card)、智慧卡(Intelligent Card)、微电路卡(Microcircuit Card)或微[芯片](https://baike.baidu.com/item/%E8%8A%AF%E7%89%87/32249?fromModule=lemma_inlink" \t "_blank)卡等。是一种具有微处理器或存储器等集成电路技术的智能卡，可用于存储数据、进行加密认证、进行电子支付等功能。IC卡的使用方便、安全、可靠，并且可以通过更新电路来提升其功能。它是将一个[微电子](https://baike.baidu.com/item/%E5%BE%AE%E7%94%B5%E5%AD%90/1410350?fromModule=lemma_inlink)[芯片](https://baike.baidu.com/item/%E8%8A%AF%E7%89%87/32249?fromModule=lemma_inlink)嵌入符合ISO7816标准的卡基中，做成卡片形式。IC卡与读写器之间的通讯方式可以是[接触](https://baike.baidu.com/item/%E6%8E%A5%E8%A7%A6/5692?fromModule=lemma_inlink" \t "_blank)式也可以是非接触式。集成电路卡（IC卡）在金融、交通、医疗、身份认证等各个领域有广泛的应用。它将微电子技术和计算机技术有机结合，为现代化的工作和生活提供便利。

随着电子科技的飞快进步，短短几年内，IC卡技术在轨道交通自动售检票（Automatic Fare Collection）系统迅速得到了大规模的实际应用。非接触式IC卡通过其储存量大、保密性强、支持一卡多用等特性逐步替代了磁卡，成为世界各大城市轨道交通收费系统的首选票质媒介。智能读写器是专为轨道交通自动售检票系统应用项目设计和生产的智能式IC卡读写模块产品。在轨道交通自动售检票（AFC）系统中，智能读写器是票卡交互的核心部件，智能读写器作为票卡交互的设备，承担着票卡读写和接口处理的基本职责，在轨道交通自动售检票系统中具有举足轻重的地位和作用。

在AFC系统中，标准的读写器需要实现一系列的功能。首先，它需要实现与票卡之间的通信，包括对票卡数据的读、写和访问等功能。其次，它需要为票卡提供所需的工作能量。此外，读写器还需要与主控设备（上位机）进行交互通信。当然还要具有存储大量数据的能力，例如日志、交易数据和参数文件等，并且应该具备防冲突机制（寻卡多卡判定）等功能，以确保系统的稳定与安全。在软件应用层面，现代化的读写器已经从传统的模式逐步发展为大读写器模式。这种模式中，读写器内部集成了完整的票卡业务处理程序，能够实现诸如票种、票价表等票卡参数的使用，以及各种票卡业务处理逻辑，如车票数据读取、售、补、充、进出站等业务，并能够执行这些业务并产生相应的交易数据。

随着城市轨道交通行业的发展，智能式IC卡读写模块产品因具有小巧、完善功能和易安装等特点成为了自动售检票系统终端等终端设备的理想选择。AGM（Automatic Gate Machine）作为城市轨道交通AFC设备的重要组成部分，需要配备高效智能式读卡器。如今，网络化建设和运营需求对关键核心设备的互换性提出了更高的要求。AFC智能读写器需要按照线网级应用进行标准化设计和开发，票务处理软件需要内置到读写器中，以解决不同供货商设备的互通和互换问题。同时还应适应不断出现的新票种、新业务和新应用的需求。因此，开发一种高效智能式读卡器是城市轨道交通AFC设备中不能缺少的组成部分。

## 国内研究现状

地铁作为现代城市的主要轨道交通工具，是大部分居民出行的优先选择。伴随着地铁轨道交通行业的快速发展，自动售检票(AFC)设备的数量也成倍增长，运营效率极大提高的同时，日益增长的客流也对AFC设备提出了更加严峻的考验。AFC系统作为直接面向乘客的系统，能够完成自动售票、检票等基础服务。同时，AFC系统还能够进行票务管理、统计清分、财务结算等数据统计分析，为轨道交通科学合理地运营提供有力的数据支持。

我国城市轨道交通车站的自动售检票设备，最初是来自外国，近年来我国已进行了大量的开发研制工作，提出了多种形式的产品，技术水平也在不断提高。主要表现在以下几点：1、自动化程度提高：近年来，地铁自动售检票设备的自动化程度越来越高。这些设备的设计和制造采用了先进的自动化技术和智能化控制系统，大大提高了设备的运行效率和精度。2、安全性能提高：为了保障乘客的出行安全，在地铁自动售检票设备的设计和制造中越来越注重安全性能方面的技术创新。例如，采用多重身份验证的安全措施，从而确保数据和信息的安全性。3、创新技术的应用：现代科技的快速发展为地铁自动售检票设备的升级和升级打开了无限的可能。简单举例来说，人脸识别、人工智能等新技术的应用，能够在确保安全的前提下进一步提升售检票设备的智能化和自动化程度。总之，地铁自动售检票设备研究现状在不断地变化和提升，从自动化程度、安全性能到科技应用等方面都有着卓越的发展，让城市轨道交通更加便利和高效，也为我国智慧城市建设发展奠定了坚实的基础。

一般地铁的轨道交通自动售检票系统基本架构分为五个层次，分别为清分中心、线路中央计算机层、车站计算机层、车站终端设备层和车票五个层次。其中与乘客直接接触的主要是车票与车站终端设备，车站终端设备主要包括检票闸机、自动加值机、自动售票机与半自动售票机等，这些设备都与读卡器关系密切，尤其是其中的自动检票闸机（Automatic Gate Machine）。

自动检票机是地铁自动售检票系统中的重要设备，能够通过刷IC卡等方式进行快速识别和验证乘客身份，并完成售票过程。它是一种通道阻挡装置(通道管理设备)，用于管理人流并规范行人出入，自动检票闸机主要应用于地铁闸机系统、收费检票闸机系统、景区检票系统等、其最基本最核心的功能是实现一次只通过一人，可用于各种收费、门禁场合的入口通道处。自动检票机的研究现状在不断进步和创新，其中从硬件和软件技术的不断发展到拥有更广泛的应用场合和增加的投入资金和人力等方面都得到了持续提升。随着未来科技的不断进步，自动检票机的智能化也将更加完善。

AGM机读卡器目前的技术改变和研究进展主要表现在以下几个方面：1、读取速度有所提升：新一代AGM机的读卡器在读取速度方面有了较大的提升，甚至实现了无需停站刷卡，实现快速通行的功能，这大大提高了AGM机的验票和收费效率。2、可靠性更高：AGM机的读卡器采用了更加先进的读卡技术，如RFID、NFC和蓝牙等技术，提高了读卡器的可靠性和稳定性，减少了因读卡器故障引起的停机、误操作和票务信息错误等问题3、安全性能更好：随着支付卡技术的不断发展，AGM机的读卡器提供了更高的安全性能，包括硬件方面的加密存储技术、数据传输的安全协议，以及更为严格的身份验证和授权机制，进一步保障了数据和交易的安全。4、智能化程度更高：AGM机的读卡器采用了更为智能化和自动化的技术，能够自动识别不同种类的支付卡，并实现自动计费和自动结算等功能。此外，读卡器还能够实现票务信息的长效存储和自助查询等功能，便于乘客日后查询。

在硬件上，AGM机读卡器采用了先进的传感器、CPU、嵌入式系统等技术，改善自动检票机的运行性能和安全性能；在软件上，开发了基于物联网技术的操作系统，增强设备的智能化和自动化程度。综上所述，AGM机的读卡器研究现状在不断进步和创新，新技术的不断发展，不断提高了读卡器的读卡速度、可靠性、安全性、智能化程度等方面的性能，未来，读卡器会越来越智能化，越来越广泛地应用于各个领域，同时它的设计、功能和维护也将更人性化，更符合用户的需求。让AGM机不断适应城市轨道交通的快速发展。

## 论文结构及内容详细安排

本文的主要内容是关于基于嵌入式Linux系统的地铁AFC设备的AGM读写器的设计与实现。本文的章节层次及内容安排如下：

1、绪论：本章介绍本课题的选题背景，并对国内研究现状进行分析，以确定本论文的框架结构。

2、方案设计与基本理论：本章首先对目前地铁AFC设备对读写器的需求和读写器的功能进行概述以及分析。然后概述本文读卡器的设计原理和基本理论，从而确定相关技术选型方案。最后对读写器设计方案进行概述和相关设计标准的总结。

3、概要设计：本章将读写器的各个功能模块分解为小模块，绘制出运行逻辑图和调用关系图，并提出寄存器和功能模块之间的调用关系。最后对概要设计进行总结。

4、详细设计：本章详细说明读卡器的具体业务逻辑并画出软件代码函数流程图。接着对不同的实现功能的代码模块进行详细解析说明。最后对详细设计进行总结。

5、测试实现：本章采用地铁AFC设备的专用测试软件来测试读写器各项功能，并对测试结果进行分析和补充说明。

6、论文总结：本章全面细致地总结本课题，确定需要改进的地方，并得出本课题的收获。

# 方案设计与基本理论

## 需求分析

### 总体功能需求

轨道交通系统拥有许多专业领域，然而，AFC系统的重要性不言而喻。该系统为乘客提供了进站自动检票和出站自动售票功能，并为轨道交通运营公司提供了可靠的数据，成为科学管理的有力支持。作为轨道交通AFC系统设备闸机的主要组成部分，读写器需要具备高效、可靠和灵敏的响应能力，并具有良好的稳定性能。它是电子收费终端中智能读卡设备的关键组件。智能读写器是专为轨道交通自动售检票系统应用项目设计和生产的轨道交通行业智能式IC卡读写模块。该产品结构小巧、功能完善、易于安装等特点，可适用于自动售检票系统终端等城市轨道交通行业终端，并可在金融IC卡接入、城市一卡通等领域应用。读卡器的设计需要满足各种相关的技术标准，符合国际标准和国家标准。

读写器采用模块化结构设计，严格遵守轨道交通AFC标准进行设计，对外接口统一，设计的读写器满足以下需求：1、满足不同轨道交通运营商系统和城市通卡运营商系统的应用需求：读写器底层预留了大容量存储器，方便扩展和满足现有和将来的不同应用需求。它是一个功能独立且整体功能统一的软硬件综合体，能在安全认证模块的配合下完成车票分析和车票交易处理，无需外围设备参与读写操作和密钥计算功能，整个流程统一，接口一致。读写器能够独立完成所有与业务无关的票卡应用处理，包括非接触式IC卡检测、信息读取和写入、防冲突、选择IC卡应用、票卡密钥计算、密钥双向认证、票卡逻辑数据恢复、交易验证码产生、信用管理和交易流水号管理。2、满足地铁AFC设备AGM机的接入和处理其报文的需求：读写器能够独立完成交易的全部处理过程，只需与上位机程序通过简单接口（如串口）硬件连接，无需中间层软件。3、满足系统引入新功能和新票种（或新卡型）的需求：读写器预留有新功能接口，包括手机支付功能接口，可以满足系统引入手机支付功能和新票种（或新卡型）的要求，并适应票卡数据结构的升级变化和跨行业的应用。4、满足连接智能装备并进行功能交互的需求：智能设备是传统电气设备与计算机技术、数据处理技术、控制理论、传感器技术、网络通信技术、电力电子技术等相结合的产物。读卡器及核心模块可通过串口通讯的方式与外围的智能设备进行连接,例如人脸识别、智能运维等以适合不同用户的需求。

### 模块化功能需求

**1、读卡器独立工作功能需求**

该读卡器具备强大的联网和脱机工作能力，可以在管理中心监控电脑关机或通讯线路故障时正常地脱机工作，并不受任何影响。在联网模式下，由管理层判断卡片是否合法，并触发业务，例如开门动作。读卡器搭载的AFC系统软件界面可以弹出持卡人的照片、余额、次数等信息。

**2、接入终端设备功能需求**

该读卡器能够与外部设备进行交互，并获取外部模块传递的数据。它可以设置设备标识、识别和查询设备，安全地连接设备，检测通信状态以获取设备数据。后续可对数据进行分析处理、安全地断开设备并保留数据通信日志等。为了实现此需求，读卡器需要具备专门的串口设置。

**3、读卡器基本票务处理功能需求**

该读卡器能够满足不同AGM设备对票卡的进出站、更新、查验等应用操作要求。它用于实现票卡的读写，在AFC系统的AGM机中得到广泛应用。检票机读写器应可以完成票卡的进出站检票交易功能。读卡器能够将票卡处理的业务流程部署于内部独立完成，使票卡业务的操作流程由读卡器独立完成。此外，终端设备的工控机不涉及票卡业务的具体实施流程。工控机和读卡器之间仅涉及操作命令、信息反馈等信息交互。

## 基础理论和原理分析

### 开发环境

嵌入式开发是一种专门针对嵌入式系统开发的软件开发方式。嵌入式系统通常是一种特定目的的计算机系统，由计算机硬件和软件构成，它被嵌入到特定设备中以执行特定任务，如电子产品、汽车、医疗设备、安全系统等。。

嵌入式开发是一种高度专业化的领域，需要熟悉硬件和软件系统的开发，经常需要高度优化的代码，因为嵌入式系统通常具有资源受限的特点，例如小型的处理器和少量的内存和存储空间等。嵌入式开发涵盖很多方面的知识和技术，包括硬件设计和驱动程序、嵌入式操作系统、应用程序开发、网络和通信协议等等。通常，嵌入式开发需要针对特定应用程序进行开发，在硬件和软件之间找到最佳的平衡点，以满足性能、资源和成本等方面的要求。

### 开发工具

**1、C++**

C++是一种带有对象编程特性的编程语言，可以用于开发底层操作系统、高级图形用户界面、桌面应用程序、科学计算等各种应用程序。C++语言是由Bjarne Stroustrup在20世纪80年代所发明的。它是在C语言的基础之上发展而来的，因此C++包含了C语言所拥有的所有传统特性，同时还添加了许多新的功能，例如：类、虚函数、函数重载、运算符重载、继承和多态等高级特性，这些特性使得C++语言成为了一种非常强大和灵活的编程语言。

C++语言的优势是其能够处理不同类型和复杂数据结构，也能够面向对象地编程和应用泛型编程技术，使得程序的设计更为模块化和抽象化，从而提高可维护性和可扩展性。此外，C++还拥有一些内存管理功能，如指针、动态内存分配等，能够精确地控制程序申请和使用的内存空间，提高程序的效率和可靠性。

C++是软件开发公司及程序员们广泛采用的编程语言之一，例如：游戏开发、物联网应用、移动应用、大数据、皮肤外科医学影像处理、量化金融等。

**2、ARM**

Linux是一种免费、开源、通用的操作系统内核，ARM则是一种处理器架构，广泛应用于嵌入式系统和智能设备领域，如移动设备、智能手表、智能家居、机器人、汽车等。由于其低功耗、高性能和低成本等优势，越来越多的设备采用ARM的处理器架构。

于是，针对ARM处理器架构的Linux发行版就应运而生。ARM平台指的是在ARM处理器架构上运行的Linux操作系统。ARM平台的Linux发行版，拥有与传统桌面版Linux相似的文件系统结构，但它还针对ARM特定的硬件和设备进行了深度优化和适配。同时，ARM平台Linux发行版还支持包括网络、文件系统和运行状况监控在内的各种特性和服务。

ARM平台的Linux发行版不仅具有高度的灵活性、可定制性和免费的特点，而且能够在不同的嵌入式系统和智能设备上运行，从而实现标准化和兼容性。

### 读写技术

**1、NFC简介**

NFC通信是指“Near Field Communication”（近场通信），是一种短距离的高频无线通信技术。它通常用于智能手机和其他电子设备之间的数据传输，例如付款、门票或交通卡的传输。NFC使用的是13.56 MHz的无线电频率，距离通常不超过4厘米。NFC通信技术可以让设备之间进行无需触摸的通信，只需要将设备靠近即可进行通信。它是一种高速、安全、简单和方便的通信技术。

**2、NFC工作模式**

NFC技术可以工作在三种不同的模式下：（1）读卡器/写卡器模式：在这种模式下，NFC设备被设置为传输数据，另一设备作为被动接收设备。例如，可以使用智能手机作为读卡器，读取银行卡或门禁卡上的信息。（2）卡模拟模式：在这种模式下，NFC设备被设置为一个被动的设备，它能够模拟智能卡行为，可用于代替传统的IC卡，例如交通卡、门票和身份证等。（3）点对点模式：在这种模式下，两个NFC设备可以直接相互通信。数据可以直接在两个设备之间传输，例如通过NFC技术进行文件传输。这种模式需要两个NFC设备都支持点对点模式。总之，通过这三种模式，NFC技术可以方便快捷地进行数据传输、支付、门禁等应用。

### 无线通信标准

ISO/IEC 14443是一种无线通信标准，用于非接触式智能卡和读卡器之间的通信。该标准由国际标准化组织（ISO）和国际电工委员会（IEC）联合制定，并于2001年发布。ISO 14443定义了一个用于非接触式智能卡和读卡器的接口标准，其中卡片和读卡器之间使用13.56 MHz的无线电信号进行通信，距离通常在2-10厘米之间。此标准对于无线门禁、公共交通票务系统、支付系统和其他类似应用都具有重要意义。

ISO 14443包含了4个部分：1、Physical characteristics（物理特征）2、Radio frequency power and signal interface（无线电功率和信号接口）3、Initialization and anticollision（初始化和防碰撞机制）4、Transmission protocol（传输协议）ISO 14443标准确保了不同厂家生产的卡片和读卡器之间的互操作性和兼容性，这使得广泛应用的非接触式智能卡系统能够相互间正常通信。

## 方案设计

### 整体方案设计

专用读卡器用于轨道交通的自动售票系统（AFC），对于其设计要求包括体积、功耗和系统可靠性等方面具有严格限制。为了能够满足这些限制，嵌入式系统成为了一种理想的选择。嵌入式系统的应用方向明确，以计算机技术为基础，软硬件可裁剪，可以完美满足AFC系统专用读卡器对功能、可靠性、成本、体积和功耗方面的严格要求。

嵌入式系统是一种具有嵌入性、系统性和智能性的计算机系统。它被嵌入到特定对象体系中，是一种以软硬件为一体的实体，能够智能地实现预定的功能。嵌入式系统的主要特点包括专用性和实时性。嵌入式系统采用专用处理器，不同的功能需要不同的处理器，这是其专用性的体现。由于嵌入式系统常被用作控制或辅助操作设备的装置，因此要求具有很好的实时性，以满足对象系统的要求。此外，嵌入式系统还具有系统设备电子化、编程语言低级化、系统性能可测化和开发系统专用化等特点。在嵌入式系统中，处理器通常是系统的控制核心，具备良好的性能。

因此，根据需求分析，采用嵌入式平台进行轨道交通AFC系统专用读卡器的设计，能够更好地满足该读卡器的严格要求。

### 硬件方案设计

**1、读卡器芯片方案**

主控电路板选用了运行于MX6UL-C微控制器上的ARMCortex-A7 CPU作为其核心。ARMCortex-A7是一款经典的工业级CPU，具有优越稳定的性能、高性价比、调试简便以及丰富的外设接口等特征，这些特性使得该CPU被广泛地应用于各个行业。

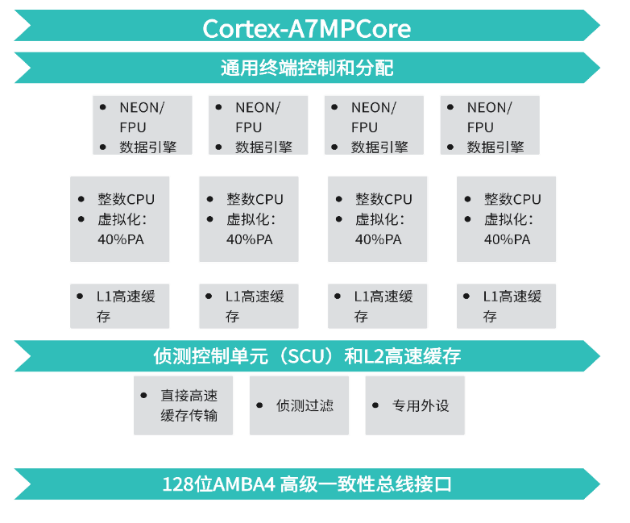


图2.1 Cortex-A7架构

ARM-Cortex-A7的良好性价比、满足大部分应用场景的功能和性能以及相对简单的生产工艺，使得该CPU成为一种十分优秀的选择。此外，该CPU拥有成熟的应用市场和高普及度，具有旺盛的生命力。鉴于此，考虑到需求分析的结果，本方案选用MX6UL-C作为读写器的主控制器。

**2、读卡器PCB方案**

**表2.1 读卡器配置及参数**

|  |  |
| --- | --- |
| **项目** | **配置和性能参数** |
| 中央处理器 | ARMCortex-A7 |
| 主频 | 528MHz |
| ROM | 4GB |
| RAM | 512MB |
| 天线模块 | 3个 |
| SAM卡槽 | 8个 |
| 串口 | TTL电平：6路 RS232电平：5路 |
| USB | Host：1路 OTG：1路 |
| 网口 | 1个· |
| SD卡槽 | 1个· |
| 铁电存储 | 1个· |
| 加密芯片 | 1个· |
| 蜂鸣器 | 1个 |
| 供电电源 | DC12V |
| 功耗 | ≤3W |
| 存储温度 | -40℃~125℃ |
| 运行环境温度 | -20℃~70℃ |
| 相对湿度 | 10%~90% 无凝露 |

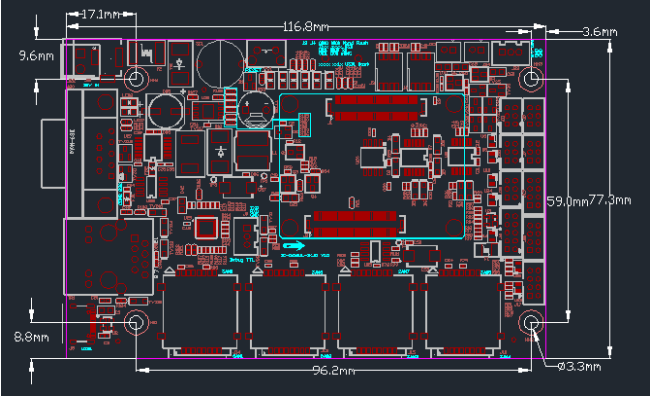


图2.2 读卡器PCB板布局图

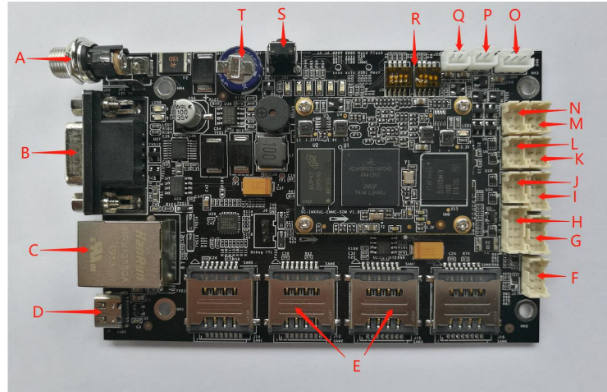


图2.3 读卡器实物图

**表2.2 读卡器实物对应接口**

|  |  |  |  |
| --- | --- | --- | --- |
| A | 电源接口 | F、G、I、H | 天线接口 |
| B | DB9 通讯口 | J、K、L | 串口 TTL |
| C | RJ45 网络接口 | N | 串口RS232(ttymxc4/5) |
| D | USB OTG 接口 | R | 拨码开关 |
| E | 8 个 SAM 卡槽 | S | 复位按键 |

### 软件方案设计

**1、读卡器整体软件架构**

由于读卡器的功能要求较为复杂，需要多任务处理，并对系统的安全性与稳定性有着较高的要求。嵌入式操作系统能够满足这些需求。嵌入式操作系统具有完整的编程规范和统一的接口函数，可使用高级程序语言，从而使应用软件具有良好的代码可重用性和可移植性。

在选择读卡器操作系统时，主要考虑以下因素。首先，选用主流的、成熟的操作系统，该系统需拥有强大的生命力，已经被广泛接受，将在可预见的未来仍会是主流操作系统。其次，应用软件可实现与城市轨道交通不同线路之间的兼容。最后，尽可能选用免费或低成本的系统，以便于未来的大规模生产发布。

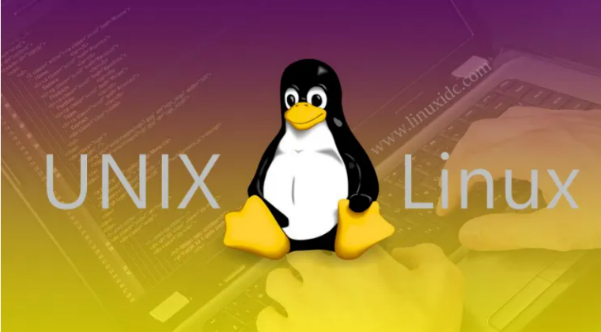


图2.4 Linux系统图标

嵌入式Linux是一种将日益流行的Linux操作系统进行裁剪修改，使其能够在嵌入式计算机系统上运行的操作系统。相对于传统的嵌入式操作系统，嵌入式Linux具有Linux开源的优点和嵌入式的特点。嵌入式Linux因其免费、性能优异、开源代码易于移植和应用软件开发周期短等特点，具有良好的安全性、实时性和稳定性。同时Linux系统也很易于开发：Linux系统采用了类Unix的操作系统结构，提供了一组强大的命令行工具和开发环境，以及许多其他的开发工具和资源，如开源库、开发文档等，使得开发人员可以更容易地创建和调试软件。此外，由于Linux系统非常流行，因此可以选择使用众多现成的开发工具和资源。

读写器选用了Linux系统，并在该系统上开发了驱动和应用程序。Linux操作系统分为内核空间和用户空间，其中各个硬件设备的驱动位于内核空间，而其他应用软件位于用户空间。其具体软件系统架构层次如图所示：

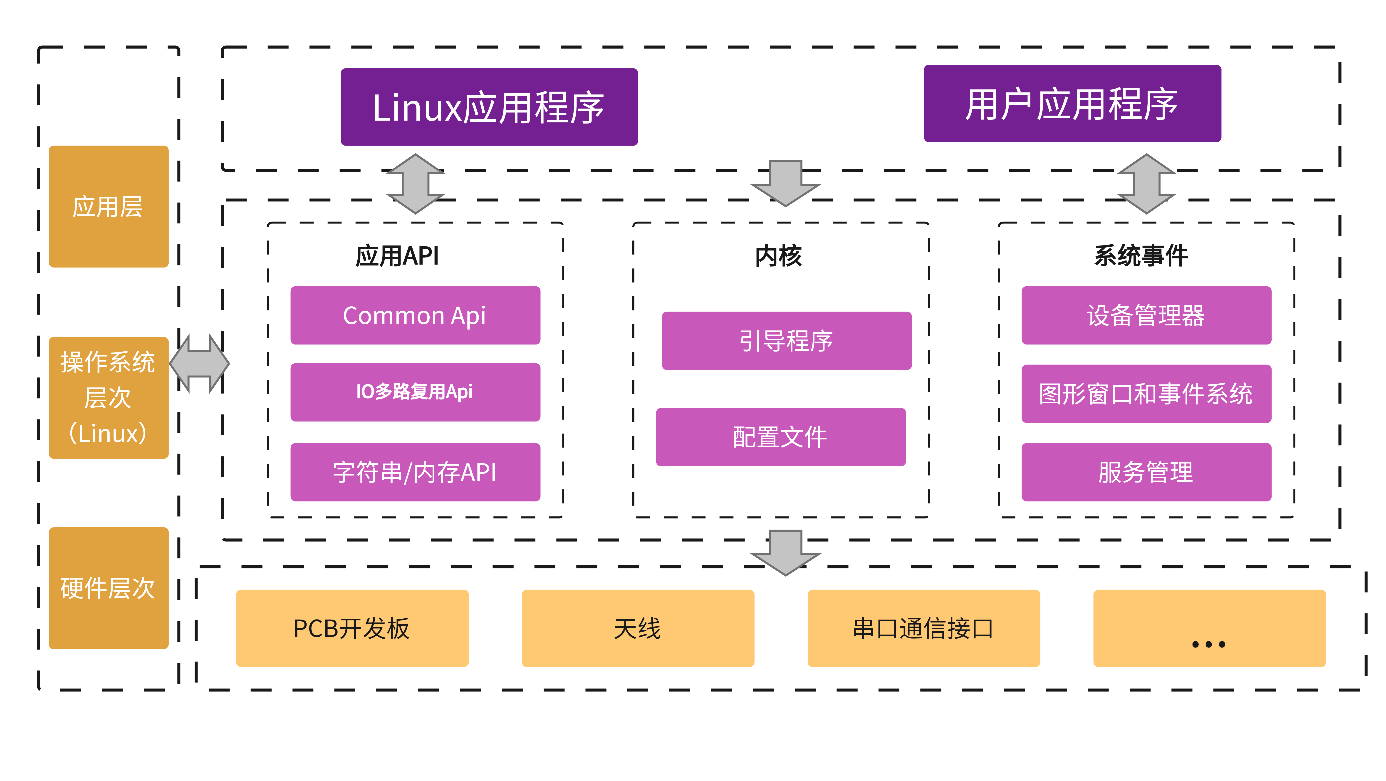


图2.5 Linux系统架构图

嵌入式 Linux 系统的分层架构模型，该模型采用层次化思想进行设计，分为三个层次，分别是硬件层、操作系统层和应用层。硬件层包括微处理器、各种外围设备和硬件功能模块。操作系统层为应用程序提供载体，并且可以定制化选用 Linux 操作系统的组件。应用层包括 Linux 应用程序和用户应用程序。这种架构模型在系统设计中对于优化性能有很大的作用。

**2、 操作系统及底层驱动方案**

根据Linux层次体系架构，并结合AFC系统AGM读写器的情况，我们可以得出AFC系统专用读写器的架构。此架构需要开发驱动程序协助大多数的功能硬件，但已包含在硬件封装内部的驱动程序不在此范围内。硬件驱动程序与操作系统和应用软件之间的结构关系如下图所示。

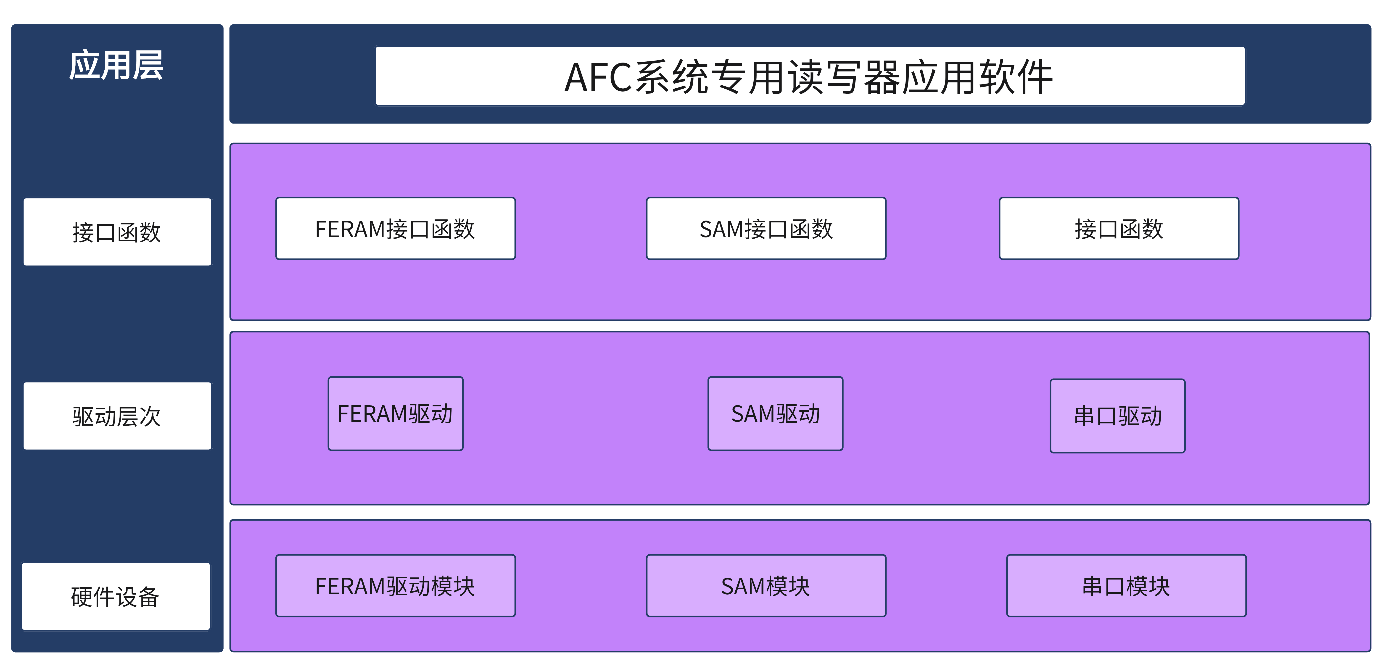


图2.5 AGM读卡器系统架构图

应用软件需要采用C/C++语言开发，这样程序具有很强的可移植性。软件能根据不同的操作系统、开发环境的变化而重新独立编译或移植。

硬件设备的控制需要设备驱动程序来将操作系统的请求转化为特定物理设备控制器所能理解的命令，并通过读写硬件寄存器来实现。作为操作系统和输入输出设备之间的桥梁，设备驱动程序的功能十分关键，它负责识别并管理硬件设备，在保证设备正常运行方面具有不可或缺的作用。一般而言，设备驱动程序是一种内核模块，能够管理硬件设备底层I/O操作。在操作系统中，标准接口函数是内核与设备进行交互的通道。

在Linux系统中，为了简化对外设的管理，所有外围设备被归为三类：字符设备、块设备和网络设备。字符设备包括鼠标、键盘、串口等，这些设备在存取时不使用任何缓存；块设备包括硬盘设备、Flash等，这些设备的读写都有缓存支持，并且可以进行随机存取；网络设备则作为专门处理的一类设备，其数据读写都需要系统提供的发送和接收缓存支持，同时不能通过简单的读写操作进行数据访问，例如以太网卡等。除了网络设备，字符设备和块设备都被映射到Linux文件系统的文件和目录中，访问字符设备和块设备需要通过文件系统的调用接口函数open、write、read、close实现。对于块设备而言，其具有比字符设备更为复杂的结构因此需要建立对应的磁盘/Flash文件系统以进行管理。不同的设备对应不同的文件系统。

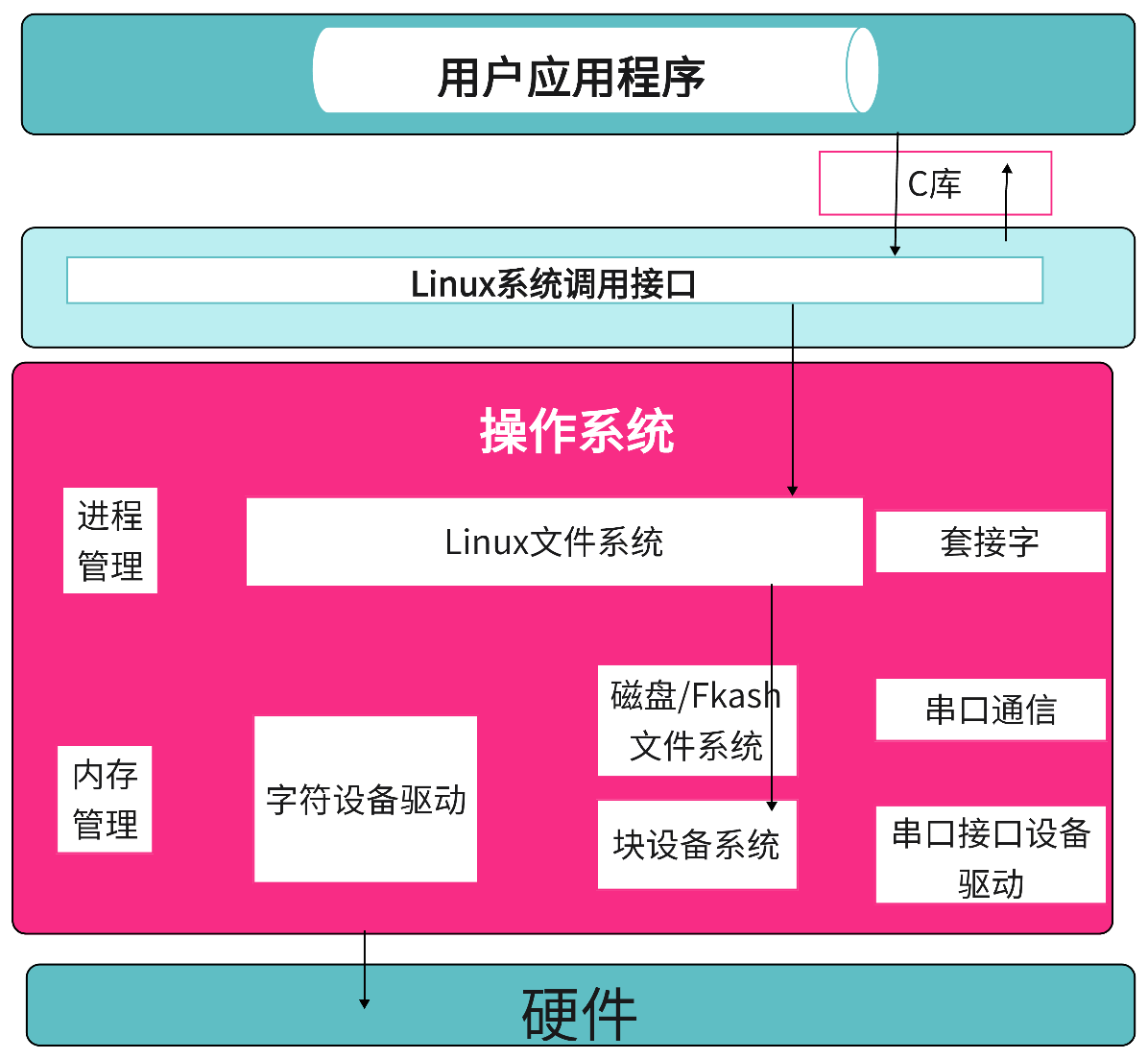


图2.6 Linux系统逻辑关系

Linux将所有外部设备看成特殊的文件类型，称之为设备文件。设备驱动程序则是Linux内核与外部设备之间的接口。通过设备驱动程序的实现，应用程序在操作硬件时与正常的文件操作并无特殊区别，从而简化了对硬件的操作。设备驱动程序的主要功能包括：设备的初始化和释放；设备参数设置和设备接口操作；实现应用程序与设备文件之间数据的传输和请求返回；以及对设备进行监测和纠错处理。

Linux操作系统分为内核空间和用户空间，这两个空间的存储空间不同。内核空间是指核心软件存放于受保护的内存空间中，具有较高的权限和对所有硬件设备的访问权限；用户空间则是用于运行其他应用程序的空间。在Linux系统中，用户空间和内核空间之间的转换主要是通过系统调用和硬件中断实现的。为了实现驱动和访问，许多设备驱动程序和相应的应用软件采用内核级别和用户级别程序的组合。驱动程序运行在内核空间，而用户空间的应用程序则可以通过设备文件在文件系统中与设备驱动程序进行交互。

**3、读卡器软件应用方案**

通过对AFC系统AGM读卡器在实际中应用的研究， AGM读卡器方案要得以实现，设计的软件要满足以下几个方面。兼容性：读卡器软件应该支持多种智能卡的规范和标准，如ISO、Mifare、Felica。可配置性：读卡器软件应该具有配置的灵活性和可扩展性，以方便系统的调整和升级。即时性：读卡器软件应该能够快速地对智能卡进行读取和验证。能够快速反应，高效稳定地管理电子票务。安全性：读卡器软件应该具备良好的安全性能，包括防止黑客攻击，保护个人信息。可维护性：读卡器软件应具备良好的维护性，包括易于维护和管理，提供日志以便于跟踪等等

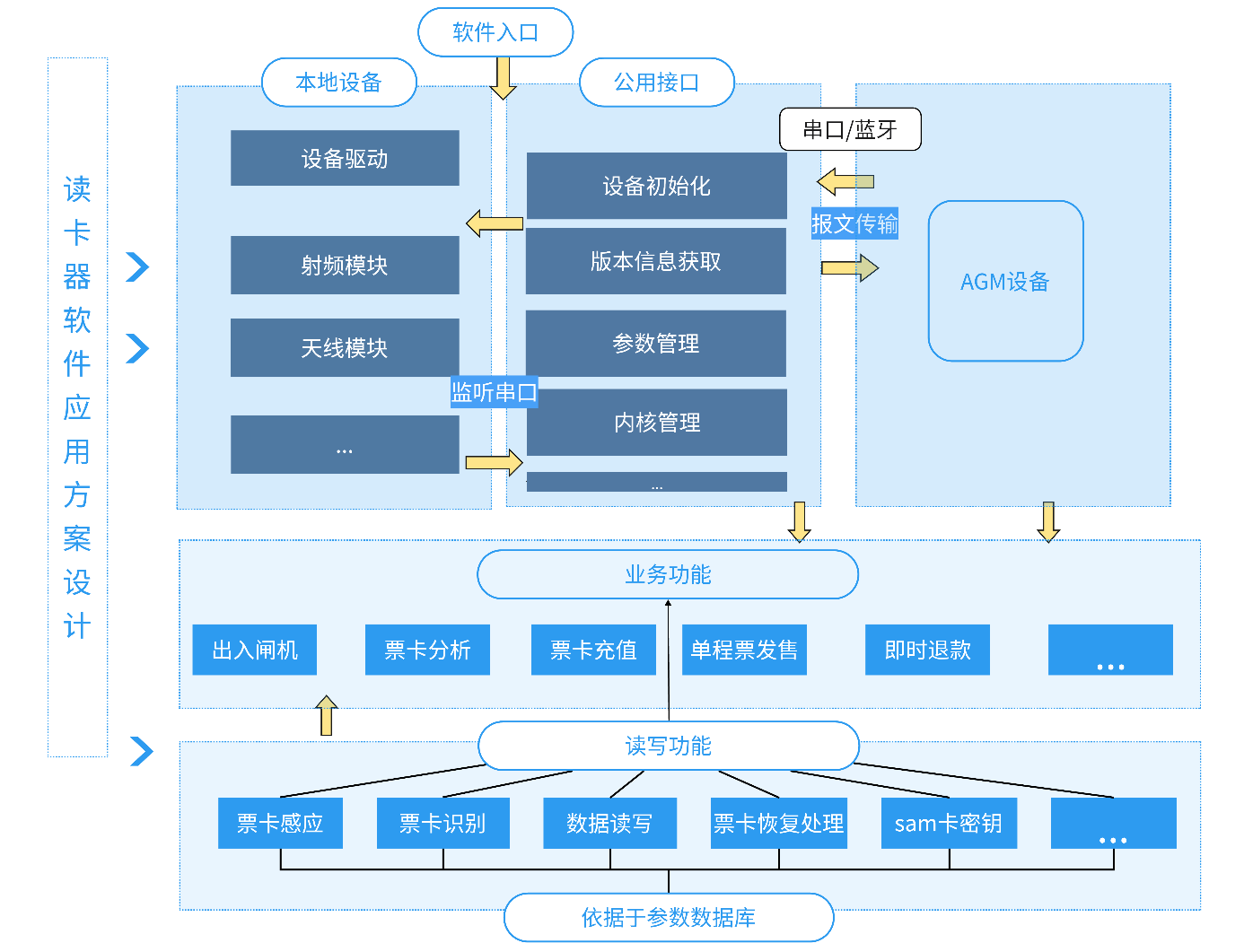


图2.7 AGM读卡器软件架构图

AFC（自动售票系统）专用读写器在城市轨道交通系统中的应用主要集中在自动检票机、自动售票机、半自动售票机和自动加值机等载体上，以与票卡进行通信。专用读写器的作用是通过监管、控制或更改票卡内的特定数据，帮助实现AFC系统对城市轨道交通的票务管理。根据这些设备的功能，票卡交易的过程通常包括充值、进站和出站等环节。因此，所有读写器应用层软件的设计都围绕着读写器的这一主要功能进行。。

在城市轨道交通的AFC系统中，AGM机器（检票闸机）的主要职责是提供检票服务。它通常安装在站厅的收费区和免费区之间。当乘客持有单程票或储值票从免费区进入收费区并通过检票闸机时，智能读卡器将车票信息读取并写入入站时间、车站编号等信息。如果车票符合规定，旅客便能够通过检票闸机。当乘客离开收费区时，检票闸机上的智能读卡器会再次读取车票信息并验证其有效性。对于有效的单程票，它们将被回收并允许离开；对于储值票，扣除本次乘车费用后才可以离开。

基于软件需求分析，该软件设计主要包括五大模块。第一，公用接口模块，用于与AGM机进行报文通信交互。第二，AGM机业务功能模块，能够接收AGM报文命令并实现开关出入闸机、识别和分析票卡等业务功能。第三，读写模块，能够读取票卡的相关数据并将其回送至AGM机。第四，参数管理模块，可用于下载和生效参数设置。最后，寄存器模块，除了存储本地设备数据外，还可以记录日志和储存错误码等相关功能。

**4、读卡器软件方案流程**

基于AGM闸的读卡器软件设计和实现，本智能读卡器软件方案业务流程如下：

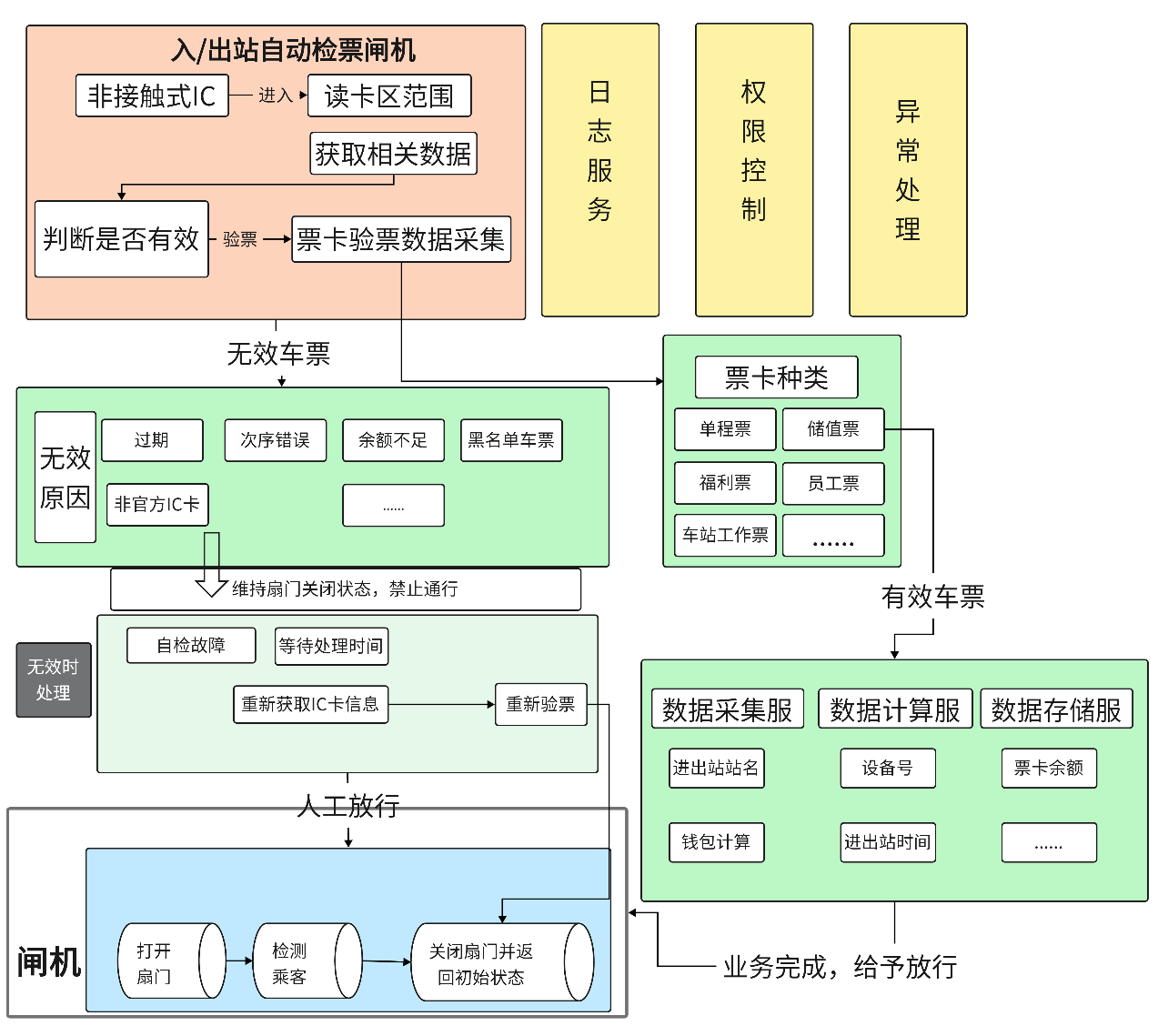


图2.8 AGM读卡器软件功能流程图

（1）进站检票机的工作流程如下：旅客使用一张非接触式 IC 卡进入读卡区范围。读卡器将对车票进行有效性检查。如果车票有效，则自动在车票上写入进站站名、进站时间和设备号等信息，然后打开闸机(扇门)。一旦检测到乘客通过，闸机便会关闭并回到起始状态。如果车票无效，包括票证过期、票证次序错误、余额不足、黑名单车票和非官方发行的 IC 卡等情况，则会提示该车票无效或进行报警，并维持闸机关闭状态，禁止旅客通过。

（2）出站检票机的工作流程如下：旅客使用一张非接触式 IC 卡进入读卡区范围。检票机会对车票的有效性和乘车费用进行检查。如果车票有效，则根据车票种类进行不同的处理：单程票、福利票和出站票将会自动写入注销信息并回收。定值票、储值票和计次票将扣除相应的乘车费用和乘次。员工票和车站工作票等免费车票将会写入相应记录。处理完成后，检票机会打开闸机(扇门)。一旦检测到乘客通过，闸机便会关闭并回到起始状态。如果车票无效或费用不足，则会提示该车票无效或欠费，并维持闸机关闭状态，禁止旅客通过。

（3）双向检票机的工作流程包括进站检票机和出站检票机两个部分。该设备可设置为进站检票机状态、出站检票机状态或进/出站检票机状态。当检票机处于进站状态时，设备自动执行进站检票机的工作流程；而当检票机处于出站状态时，设备自动执行出站检票机的工作流程。

### 方案设计标准

### 方案总结

# 概要设计

## 读写器软件概要设计

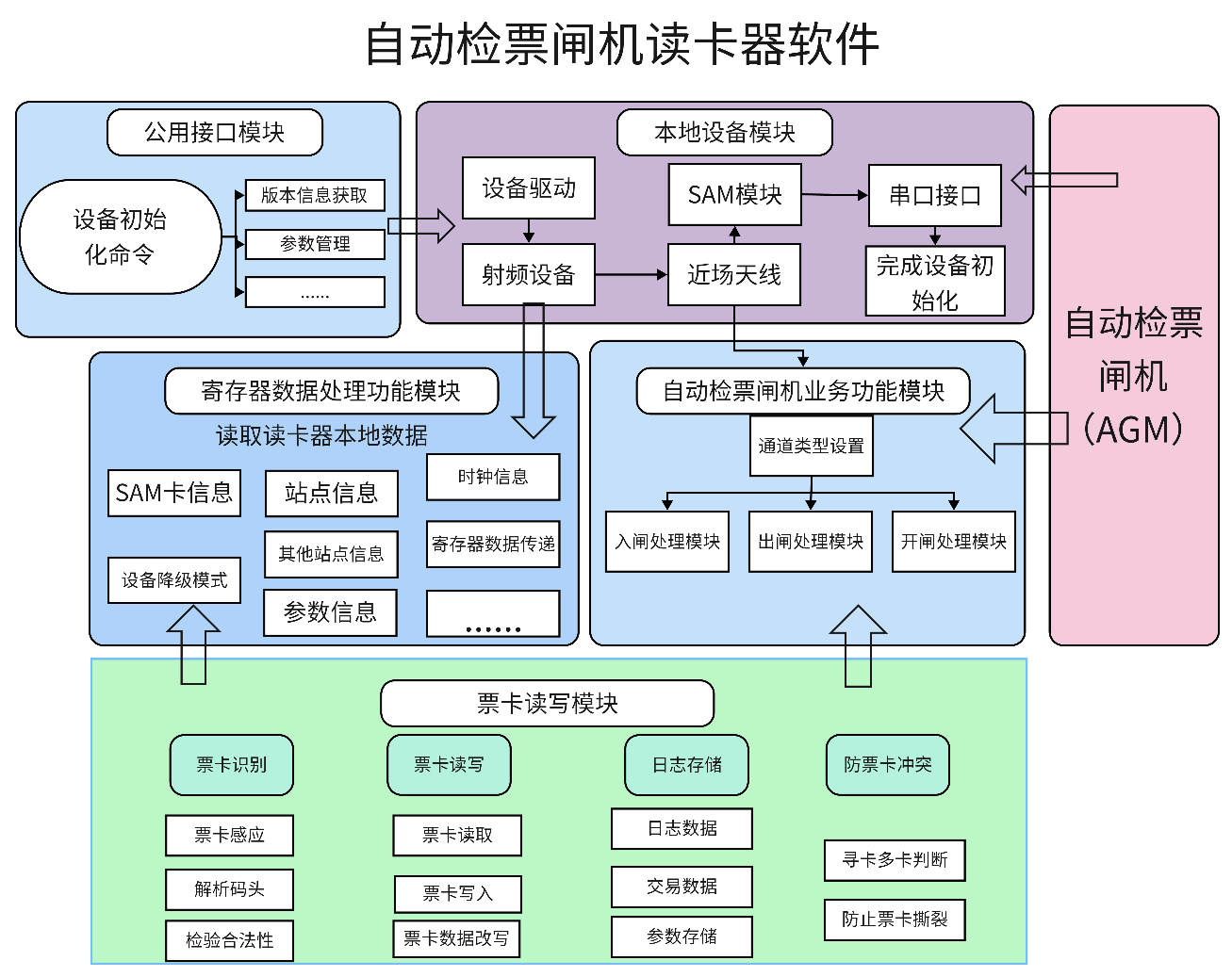


图3.1 AGM读卡器软件详细架构

本软件设计主要的三大模块：

1、公用接口模块：该模块负责实现读卡器与上位机之间的通信，同时也负责读卡器本地数据的读取和自初始化，为后续的开发工作打下了坚实的基础。

2、该模块主要用于实现参数列表的下载和相关参数生效的功能。通过该模块，我们可以更加方便地管理闸机系统中的各种参数，提高了系统的灵活性和可维护性。

3、自动检票闸机业务功能模块：该模块是我们软件设计的核心部分，主要实现闸机类型设置、出入闸机以及开闸处理等所有业务功能。在各种不同闸机之间，该模块可以帮助我们实现闸机业务流程的统一，提高系统的利用率和效益。

1. 公用接口子模块设计

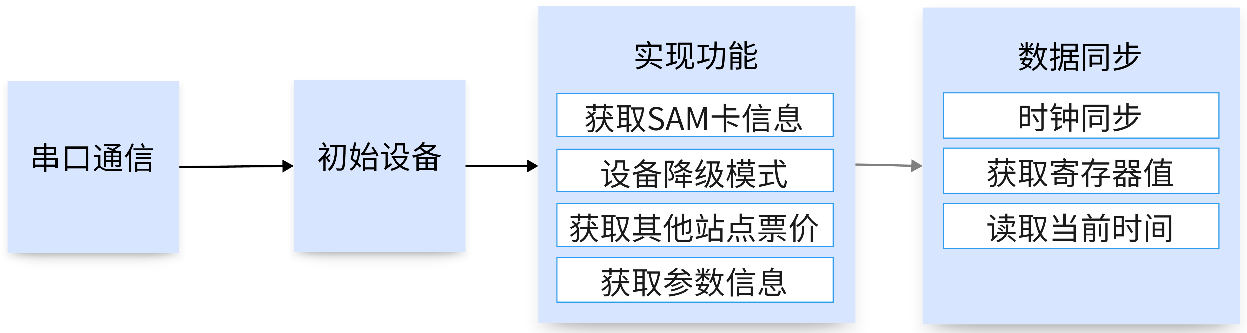


图3.2 AGM读卡器公用接口功能流程

**1、串口通信**

UART串口连接时，最简单的方式是采用TTL电平三线连接方式。其中，UART\_TXD线为发送数据线，需要连接到接收设备的UART\_RXD引脚上；UART\_RXD线为接收数据线，需要连接到发送设备的UART\_TXD引脚上；GND为共同接地，用于确保两个设备具有相同的参考电位。

所有业务逻辑均由非接触式读卡器实现。当需要进行读写器的设置或读写卡片时，终端向读写器发送相应指令，并等待读写器返回操作结果。在未收到支付交易命令时，读卡器应暂时禁止卡片寻找功能。除了从读卡器接收卡片信息以外，终端还可能需要执行其他功能，因此必须确保报文流程不会显著增加交易时间。

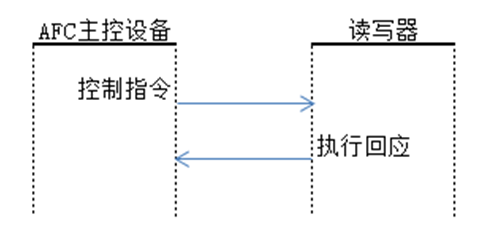
****

图3.3 AGM读卡器与上位机通信

**2、设备初始化**

在使用读卡器进行票卡处理之前，必须首先初始化读卡器的各个硬件模块。在初始化过程中，读卡器会根据接口传入的参数加载所需的设备参数，并申请内部资源。因此，在任何时候要调用读写器来处理票卡，都必须首先执行初始化调用。需要注意的是，在进行读卡器初始化之前，还需要使用‘Config\_Parameter’对设备必须的参数进行配置。

**3、版本信息获取**

读取读写器对应的版本信息。

**4、获取SAM卡信息**

读取读写器对应类型的SAM卡相关信息。

**7、获取其他站点到本站的票价**

获取当前时间其它站到本站的票价，做付费区无进站码更新时可能需要额外补款，可以通过本接口配合。

1. 运营参数模块设计

**1、参数下载**

当参数表有更新时，ECU上层应用程序将参数表文件下载到本地存储器中，并调用此接口函数通知API启用此参数文件。

参数下载流程:

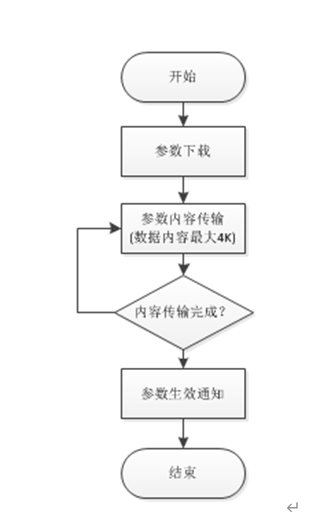


图3.4 AGM读卡器参数下载流程

参数内容传输：将参数文件分包下发到读写器，每次下发最大数据内容为4Kbyte。

**2、参数下载生效**

参数下载开始前和参数下载完成后调用。调用后同时加载单个参数。

**3、获取参数信息**

获取当前启用指定参数表的文件名，不包含全路径。

1. 自动检票闸机业务功能模块设计

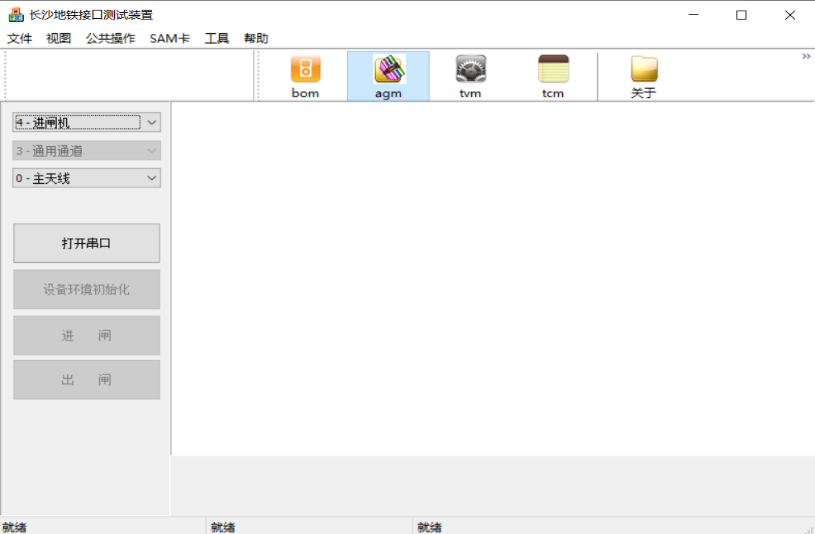


图3.5 AGM机业务功能

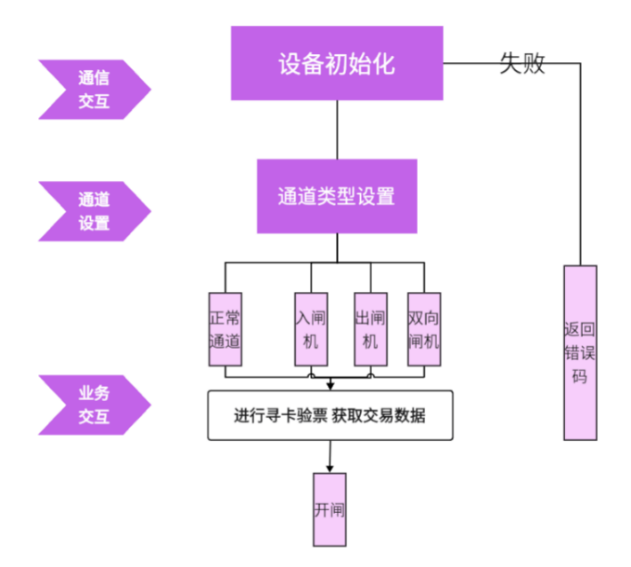


图3.6 读卡器与AGM闸机业务交互流程

**1、通道类型设置模块**

设定读写器当前所属通道类型，此函数只适用于闸机，闸机车票处理流程中读写器根据此通道类型去处理对应类型的车票。上位机应用软件调用完读写器初始化接口后紧接着调用此接口函数，如不调用此接口函数，在入/出闸处理中做为正常通道处理。

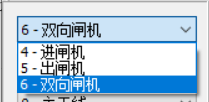


图3.7 AGM闸机通道类型

**2、入闸处理模块**

在入口闸机中调用此接口，成功进行交易后返回进闸交易数据，否则返回操作错误状态码。如果扩充保留字段为0x11,0x00，代表为电子车票业务，电子票入闸接口仅返回ITP验证电子票合法性的票卡结构，交易数据由开闸处理接口产生。

**3、出闸处理模块**

在出口闸机中调用此接口，根据应用情况指定主天线处理储值票或是副天线处理单程票，成功进行交易后返回出闸交易数据，否则返回操作错误状态码。电子票出闸接口仅返回ITP验证电子票合法性的票卡结构，交易数据由开闸处理接口产生。

**4、开闸处理模块**

当调用完入闸和出闸接口后，需要在调用此接口，获得开门指令以及交易数据。

## 寄存器数据设计

### 寄存器数据存储

(1)寄存器数据由车站设备产生，寄存器数值从设备投入使用之日起开始累计。

(2)寄存器数据采用四字节整型，金额单位为分，累计到最大值后清零重新开始累计。

(3)SC定时（定时间隔由SC参数设定，默认为15分钟）向设备发送查询寄存器数据的指令，设备（仅包含闸机、BOM、TVM三类设备）以SOCKET消息报告当前的寄存器数据，或SC操作员查询时，设备报告当前寄存器数据。

(4)SC以SOCKET消息方式向LCC转发所收到的寄存器数据，不存储本地数据库。

(5)LCC将收到的寄存器数据消息定时（定时间隔由LCC参数设定，默认为15分钟）打包成一个文件（按线路打包），以文件形式转发ACC，不存储到本地数据库。

(6)ACC将寄存器数据文件解包并存储到本地数据库。

AGM 审计数据的流向：AGM->SC->LCC->ACC。

**表3.1 数据英文说明**

|  |  |
| --- | --- |
| 英文及其缩写 | 说明 |
| ACC（AFC Clearing Center） | 清分中心系统 |
| LCC（Line Central Computer） | 线路中央计算机 |
| SC （Station Computer） | 车站计算机 |

### AGM机寄存器数据设计

**表3.2 AFC设备请求协议格式**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AFC设备主控制器发起报文-命令报文 | | | | |
| 数据名称 | 长度 | 偏移 | 数据类型 | 说明 |
| 消息头部 | 1 | 0 | byte | AA（设备主程序发送） |
| 节点编码 | 1 | 1 | byte | 接收端地址码，用于一个端口接多个读写器  默认为0x01 |
| 报文序列号 | 1 | 2 | byte | 由上位机管理，0开始，每次通讯完加1，到0xFF后重新归零。 |
| 指令类码 | 1 | 3 | byte |  |
| 命令码 | 1 | 4 | byte |  |
| 参数1 | 1 | 5 | byte[] | 预留 |
| 参数2 | 1 | 6 | byte | bit7:重试标志位(0:新的请求,1:重试获取上次应答内容)  bit6-bit4预留  bit3-bit2:蜂鸣次数最多三次  bit1-bit0:天线模式0=主天线，1=副天线，2=第三天线 |
| 时间戳 | 7 | 7 | bcd[] | 主控设备当前时间,bcd码表示,YYYYMMDDHHNNSS |
| 数据长度 | 2 | 14 | word | 数据域长度，可以等于0；最大为4KByte |
| 数据域 | N | 16 |  | 控制指令相关的数据内容，可等于0 |
| 报文校验码 | 2 | 16＋N | word | 以上所有数据的CRC16计算值  Crc多项式0x1021，Crc预值0x00  例: 0x31 0x32 0x33 0x34 计算的结果为 0xD789 |

**表3.3 读卡器应答协议格式**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 读写器回应报文-回应报文 | | | | |
| 数据元 | 长度 | 偏移 | 数据类型 | 说明 |
| 消息头部 | 1 | 0 | byte | BB（读写器发送） |
| 节点编码 | 1 | 1 | byte | 本设备地址码, 0x01 |
| 序列号 | 1 | 2 | byte | 填充当次处理指令收到的序列号 |
| 指令类码 | 1 | 3 | byte |  |
| 命令码 | 1 | 4 | byte |  |
| 长度 | 2 | 5 |  |  |
| 数据域 | N | 7 |  |  |
| 报文校验码 | 2 | 7＋N | word | 以上所有数据的CRC16计算值 |

**表3.4 协议格式中数据类型定义**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 长度 | 字节顺序 | 例子 |
| hex | 1 | Intel | 内存地址值:{0x0A} 表示整数10 |
| byte | 1 | Intel | 内存地址值:{0x0A} 表示整数10 |
| bcd | 不定长 | Motorola | 内存地址:{0x20,0x15,0x10,0x26,0x16, 0x38,0x29} 表示整数2015年10月26日16时38分29秒 |
| word | 2 | Intel | 内存地址:{0x01,0x23} 表示整数0x2301(8961) |

说明: Intel顺序表示为小端方式,即低位在前,高位在后。Motorola顺序为大端方式,即高位在前,低位在后。

**表3.5 SAM卡全局相关信息**

|  |  |  |
| --- | --- | --- |
| 字段名称 | 数据类型 | 说明 |
| keyidx | Unsigned char | 密钥索引 |
| len\_terminal\_id | Unsigned char | 终端号长度 |
| sam\_terminal\_id [8] | Unsigned char | 终端号 |
| len\_logical\_id | Unsigned char | 逻辑卡号长度 |
| offset\_logical | Unsigned char | 取逻辑卡号的偏移 |
| sam\_logical\_id [8] | Unsigned char | 逻辑卡号 |
| dsn | Unsigned Int |  |

**表3.6 错误提示信息结构**

|  |  |  |
| --- | --- | --- |
| 字段名称 | 数据类型 | 说明 |
| wErrCode | Uint16\_t | 错误码 |
| bNoticeCode; | Uint8\_t | 关联提示码 |
| bRfu[2] | Uint8\_t | 扩充保留字段 |

# 详细设计

## 软件总模块详细设计

1、读卡器软件CSReader

Main函数：

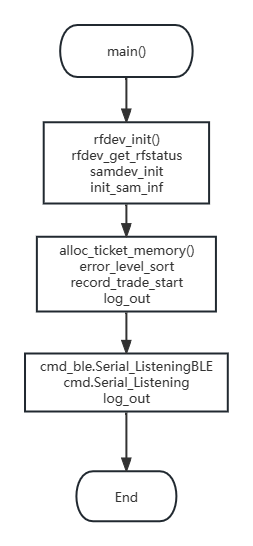


图4.1 Main函数流程图

软件从main函数开始进入，首先进入设备初始化功能模块，其中包含主要的函数rfdev\_inti()、rfdev\_get\_rfstatus()、samdev\_init()和init\_sam\_inf()。

第二个功能模块是内存管理和日志输出，其中包括函数alloc\_ticket\_memory()用于分配内存和error\_level\_sort、record\_trade\_start、log\_out等函数，用于写日志。其中，err\_code表示当前调用的错误码，而lvl表示当前指定的日志级别，只影响当前日志记录。只有当通过err\_code查找到的级别或者lvl\_appoint级别高于系统日志级别时，才会写入日志。

第三个模块是自动检票闸机（AGM机）报文命令的监听模块，主要通过监听串口接收的报文命令实现相关功能。

2、串口监听函数

int CmdSort::Serial\_Listening()：

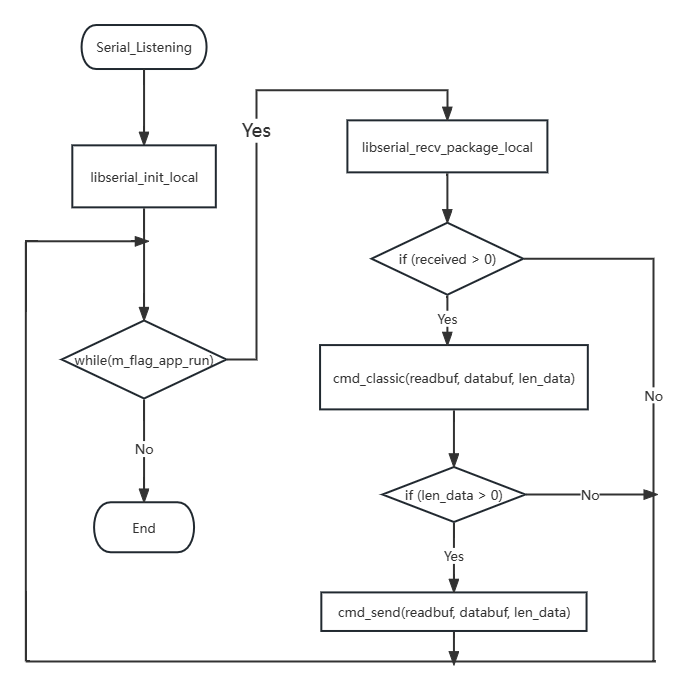


图4.2 串口监听函数流程图

cmd.Serial\_Listening是用于监听串口的函数，该函数首先对串口通信进行设置。在调用libserial\_init\_local函数进行串口初始化之后，根据m\_flag\_app\_run的值依次进入或结束while循环，用于持续监听AGM机的报文命令。

libserial\_recv\_package\_local函数用于从串口接收AGM机的报文命令，并判断接收到的数据长度是否大于0。如果接收到的数据长度大于0，则使用cmd\_classic(readbuf, databuf, len\_data)函数执行相关的报文命令操作。

cmd\_classic(readbuf, databuf, len\_data)函数是用于执行五种不同功能模块的报文命令操作函数。它包括AGM机呼叫命令、参数管理、系统呼叫命令、ITP参数下载和调试命令。

cmd\_send(readbuf, databuf, len\_data)函数用于发送数据报文，并检查返回的数据长度是否大于0。如果长度大于0，则使用该函数输出读卡器报文命令，并将结果返回给AGM机。如果长度小于0，则继续执行while循环体。

**表4.1 串口监听函数字段说明**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名称 | | 数据类型 | 说明 | 详细内容 | |
| COM1 | | Char | 通信串口 | 软连接至(char \*)/dev/ttyS0 | |
| m\_flag\_app\_run | Bool | true | 真值 |
| readbuf | | Uint8\_t | 读写缓冲 | 初始值为0 | |
| databuf | | Uint8\_t | 数据缓存 | 初始值为0 | |
| len\_data | | Uint16\_t | 发送的数据 | 初始值为0 | |
| received | | Int | 串口通信返回值 | >0:跳转报文命令  ≤0：继续监听串口 | |

3、功能操作函数

void CmdSort::cmd\_classic(uint8\_t \* p\_cmd\_recved, uint8\_t \* p\_data\_to\_send, uint16\_t& lentosend)：

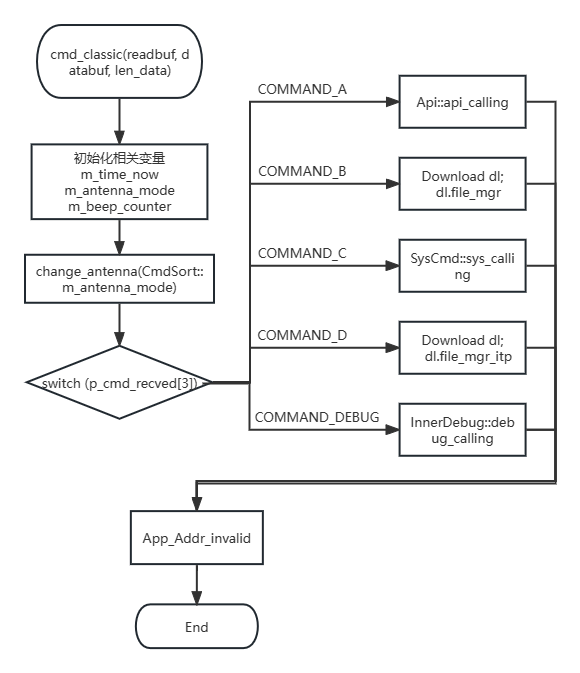


图4.3 功能操作函数流程图

CmdSort::cmd\_classic函数是用于执行各种不同功能模块的函数操作。首先，需要进行变量的初始化和变量的内存分配，然后开启天线模块，调用change\_antenna函数进行操作。

接着，根据传入的readbuf参数值，可以判断p\_cmd\_recved[3]的值，从而选择不同的功能模块。使用switch语句进行判断，如果是COMMAND\_A，则执行Api::api\_calling()来完成AGM机的报文命令操作；如果是COMMAND\_B，则执行dl.file\_mgr函数来进行参数管理；如果是COMMAND\_C，则执行SysCmd::sys\_calling函数完成系统呼叫命令；如果是COMMAND\_D，则执行dl.file\_mgr\_itp函数进行itp参数下发通知；如果是COMMAND\_DEBUG，则执行InnerDebug::debug\_calling函数用于人工调试命令。

最后，执行地址非法错误码返回函数Api::App\_Addr\_invalid。如果app\_valid的值为fales，则返回相应的错误码。

**表4.2 功能操作函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| m\_time\_now | Uint8\_t | 当前时间 | 从p\_cmd\_recved获得数值 |
| m\_antenna\_mode | Uint8\_t | 天线模块 |  |
| m\_beep\_counter | Uint8\_t | 蜂鸣器计数 |  |
| p\_cmd\_recved[] | Uint8\_t | readbuf | AGM机发送的报文 |
| COMMAND\_A | #define | (uint8\_t)1 |  |
| COMMAND\_B | #define | (uint8\_t)2 |  |
| COMMAND\_C | #define | (uint8\_t)3 |  |
| COMMAND\_D | #define | (uint8\_t)4 |  |
| COMMAND\_DEBUG | #define | (uint8\_t)0x10 |  |
| app\_valid | Bool | 初始值为true |  |
| lentosend | Uint16\_t& | Switch后功能返回值 | 执行不同情况功能后的返回值 |
| wErrCode | Uint16\_t | 错误码 |  |

4、AGM机报文命令操作函数

void Api::api\_calling(uint8\_t api\_addr, uint8\_t \* param\_stack, uint8\_t \* data\_to\_send, uint16\_t& len\_data, bool& valid\_app\_addr)：

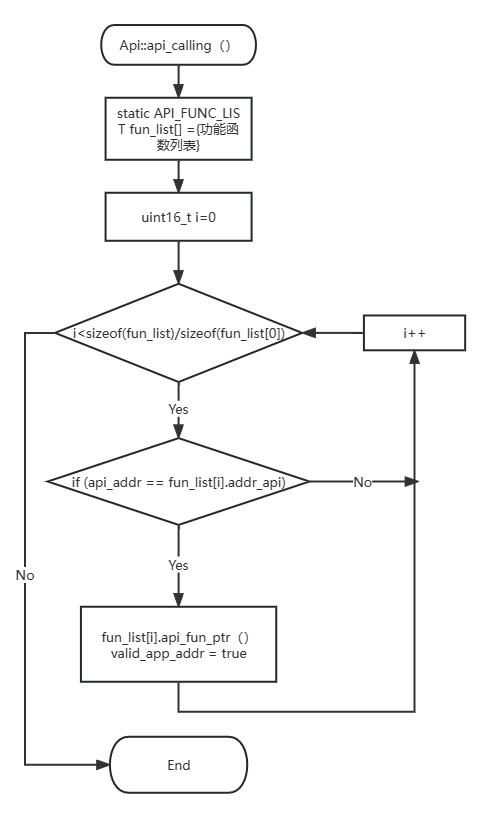


图4.4 AGM报文命令函数流程图

Api::api\_calling() 是 AGM 机报文命令功能操作函数。在执行该函数之前，需要为初始化变量赋值。其中，fun\_list[] 是功能函数列表数组，包含所有的函数。

在执行 for 循环之前，需要将变量 i 初始化为 0，然后判断 i 是否小于功能函数列表 fun\_list[] 中封装的函数个数。如果大于或等于，函数会直接停止运行。如果小于，则会执行对应的功能函数。在执行功能函数之前，还需要判断 fun\_list[i] 中的 addr\_api 是否等于从 AGM 机接收到的报文命令中的 p\_cmd\_recved[4]。

如果 addr\_api 不等于 p\_cmd\_recved[4]，则直接将 i 加 1，然后返回 for 循环开头。函数会不断循环检索封装列表中的功能函数，直至符合 AGM 机的报文命令指示后，启用对应的功能函数。

**表4.3 AGM报文命令函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| fun\_list[] | Typedef struct | 结构体类型 | 存储了功能函数的编号 |
| i | Uint16\_t | 变量 | 用于循环检索相应功能函数 |
| fun\_list[i].addr\_api | Uint8\_t | 编号 | 列表中的功能函数编号 |
| api\_addr | Uint16\_t | readbuf | AGM机发送的报文 |
| valid\_app\_addr | bool | True | 函数运行成功返回true值 |

**表4.4 功能函数列表说明**

|  |  |  |
| --- | --- | --- |
| i（fun\_list[i]）列表函数编号 | 功能函数名称 | 说明 |
| 0 | Common\_Initialize\_Device | 设备环境初始化 |
| 1 | Common\_GetVersion | 获取版本信息 |
| 2 | Common\_GetSamInfo | 获取SAM卡信息 |
| 4 | GetFare | 获取其他站点到本站的票价 |
| 5 | Common\_GetParamInfo | 获取参数信息 |
| 13 | Gate\_AisleModel | 通道类型设置 |
| 14 | Gate\_EntryFlow | 入闸处理 |
| 15 | Gate\_ExitFlow | 出闸处理 |

## 初始模块详细设计

1、射频设备初始化函数

rfdev\_init()：

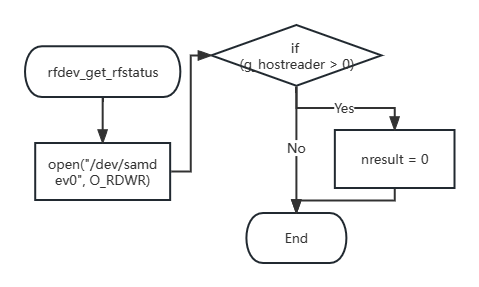


图4.5 射频设备初始化流程图

初始化射频设备首先定义 nresult 值。然后进行射频设备 rf531 的打开，然后判断 g\_hostreader 值。若返回值为 0，则射频设备打开成功，将 nresult 值置为 0。若返回值为 -1，则射频设备打开失败，nresult 值不变。初始化射频设备函数结束。

**表4.5 射频初始化函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| nresult | Int | 初始化射频设备返回值 | 0=成功  -1=打开设备句柄出错 |
| g\_hostreader | Int | 打开射频设备返回值 | 初始值：-1  打开成功：>0 |

2、天线设备初始化函数

rfdev\_get\_rfstatus：

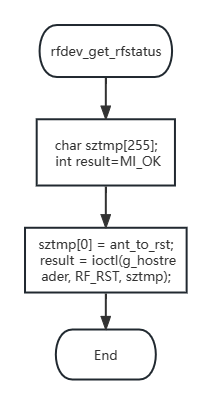


图4.6 天线设备初始化函数流程图

开始时需要对天线设备进行初始化操作。在这个过程中，需要定义一个名为 sztmp 的数组和一个名为 result 的变量。为了保证 sztmp 数组中的首位为1，我们可以进行赋值操作。之后再使用 ioctl 函数对设备的 I/O 通道进行管理和对设备的特性进行控制。

**表4.6 天线设备初始化函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| sztmp | Unsigned char | 初始化射频设备返回值 | 0=成功  -1=打开设备句柄出错 |
| result | Int | I/O设备管理及控制返回值 | 0=没有天线  1=RF1天线有效  2=RF2天线有效  3=RF1和RF2天线有效 |
| MI\_OK | #define | 0 |  |
| RF\_RST | #define | 24 |  |
| ant\_to\_rst | Unsigned char | 1 |  |

## 公用接口模块详细设计

4.2.1串口通信编程

1、串口（UART）设置函数

在Linux系统中，操作设备通常使用的接口主要由 open、ioctl、read 和 write 函数组成。在使用 UART 时，除了这些基本操作，还有针对 UART 所封装的许多功能函数。其中，最常用的是设置行规程。通过设置波特率、数据位、停止位、检验位、RAW 模式以及一有数据就返回等参数的不同值，可以实现不同的功能。在编程时，通常使用结构体来表示这些行规程参数。

int libserial\_init\_local(char \*dev, int speed, int parity, int databits, int stopbits, int hwf, int swf)：

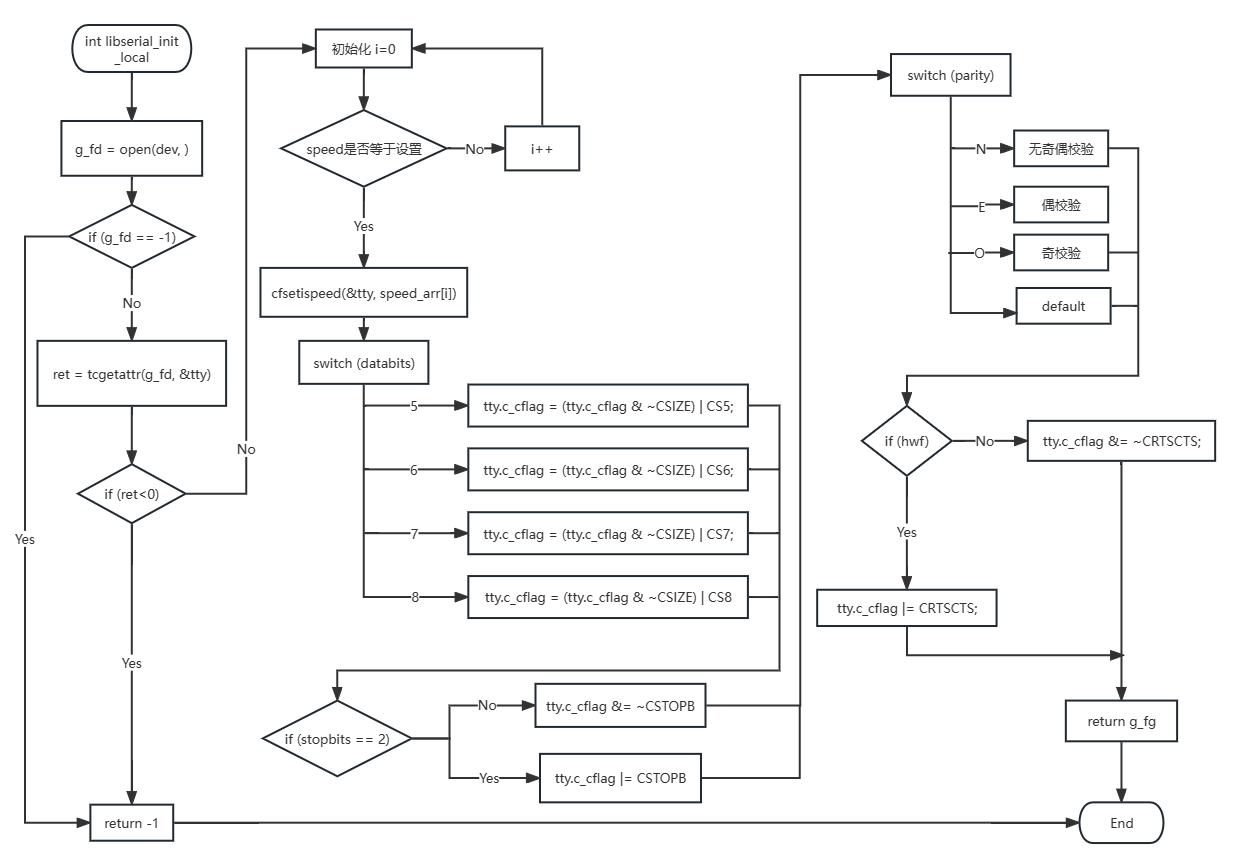


图4.7 串口设置函数流程图

（1）串口设置函数的第一步是调用 open() 函数来打开串口设备，并设置串口为可读可写以及非控制台状态。如果 open() 函数返回值为-1，则说明打开设备失败，函数将返回-1并结束。如果返回值不为-1，则调用 tcgetattr() 函数来获取与终端相关的参数，其中参数fd为终端的文件描述符，返回值保存在termios 结构体中。如果返回值小于0，串口函数将返回-1并结束；否则，将进行波特率的设置。

（2）设置波特率：要约定好传输速率（每一秒传输的数据位数，即波特率），一般选择9600,19200,115200等。进行for循环，初始化变量i，若i不等于相应的输入波特率，则i++，直到i等于需要的波特率。然后用cfsetispeed函数，将串口的波特率设置为当前i值。

（3）设置数据位：获取databits的值，为5、6、7或者8，并用tty.c\_cflag = (tty.c\_cflag & ~CSIZE) |databits函数设置好串口的数据位。数据位:可以有5、6、7或8位的数据，一般我们是按字节（8位）传输数据，发送方一位一位的改变数据线上的状态（高电平或低电平）将它们发送出去，传输数据时先传最低位，最后传送最高位。字符’A’的8位二进制字符是01000001，先发送最低位bit0，其值为1；再发送bit1，其值为0，如此继续；最后发送最高位bit7，其值为0

（4）设置停止位：停止位停止位（逻辑1）用来表示当前数据传输完毕。停止位的长度有三种：1位，1.5位，2位，这里选择2位。判断stopbits的值是否等于2，不为2则进行tty.c\_cflag &= ~CSTOPB用于停止位的设置。为2则进行一下步设置奇偶校验。

（5）设置奇偶校验：使用了奇偶校验功能，有效数据位发送完毕后，还要发送1个校验位（奇偶校验位）。有两种校验方法：奇校验，偶校验-------数据位连同校验位中，“1”的数目等于奇数或偶数。奇偶校验只能检错，不能纠错的。而且只能检测1位误码，检测出有错后只能要求重发，没法纠正的。判断parity值，为N则无奇偶校验，为O则是奇校验，为E则是偶校验。其余情况均非法结束。

（6）设置软流控制：如果需要软流控制，将 tty.c\_cflag |= CRTSCTS 设置为软流控制。函数返回 g\_fg 的值。

最后串口设置函数结束并返回g\_fg的值。

**表4.7 串口设置函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| dev | Char | 串口设备名 | 串口1是"/dev/ttyS0" |
| g\_fd | Bool | 设备打开标志 |  |
| ret | Bool | 设备参数返回值 | 参数返回值大于0则成功 |
| speed | Int | 串口波特率 | 可以是230400, 115200, 57600, 38400, 19200, 9600 |
| parity | Int | 奇偶校验 | 值为'N','E','O','S' |
| databits | Int | 数据位 | 值是5、6、7或者8 |
| stopbits | Int | 停止位 | 值是1或者2 |
| hwf | Int | 硬件流控制 | 1为打开，0为关闭 |
| swf | Int | 软件流控制 | 1为打开，0为关闭 |

2、串口数据发函数

发数据格式详见3.2.2AGM寄存器数据设计：读卡器应答数据报文表格

int libserial\_send\_package\_local(uchar \*buf, const int buflen);

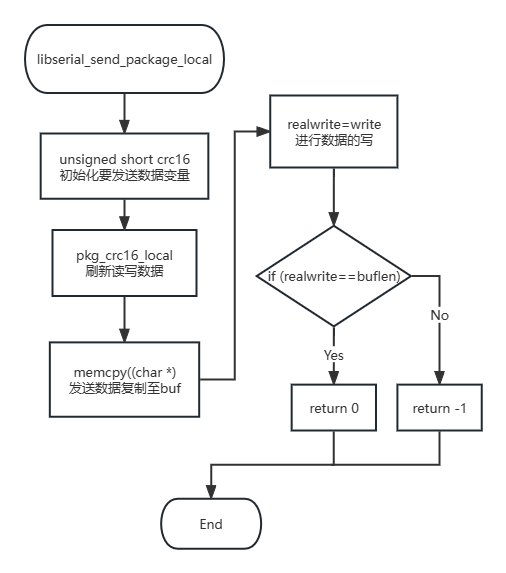


图4.8 串口发送数据函数流程图

串口发送数据函数，首先初始化发送的数据变量crc16，接着进行读写数据的刷新。然后用memcpy函数麻将传入串口发送函数的参数即crc16内容复制至要发送的buf当中。再之后进行realwrite=write(g\_fd, buf, buflen)函数，将buf的内容写至通信串口。

最后，对write函数值进行判断，如果realwrite==buflen，返回值等于相对的buf长度，则串口发送数据函数返回0并结束运行。若其不等，则串口发送函数返回-1并结束。

**表4.8 串口发送数据函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| crc16 | Unsigned short | 发送值 |  |
| buf | Bool | 发送缓冲 |  |
| buflen | Int | 数据长度 |  |
| realwrite | Int | write函数返回值 | 0=发送完成，-1=发送失败 |

3、串口收数据

收数据格式详见3.2.2AGM寄存器数据设计：AGM机发送数据报文表格

libserial\_recv\_package\_local(int maxrecv, int timeout, uchar \*buf)

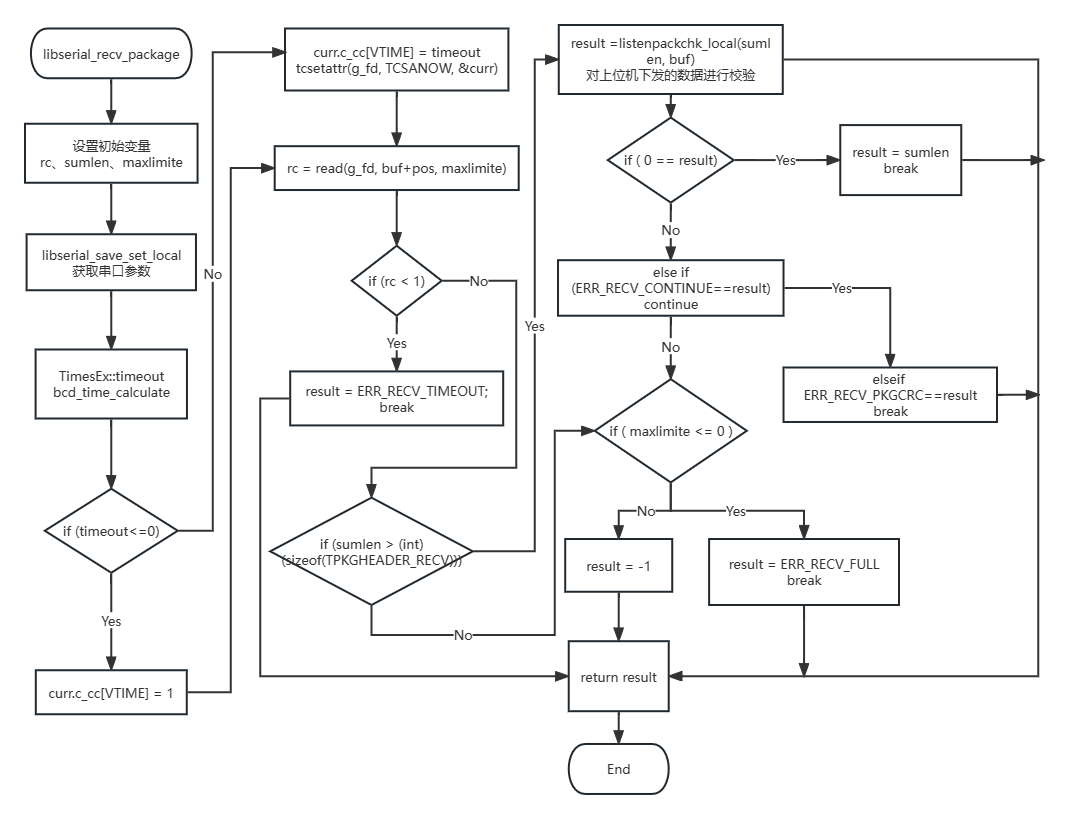


图4.9 串口收数据函数流程图

（1）串口接收函数 libserial\_recv\_package\_local() 首先初始化相关变量，并调用 libserial\_save\_set\_local() 函数获取设备的相关参数。接着，判断是否超时，并进行 BCD 编码的时间加减。如果接收时间超时，函数将返回 ERR\_RECV\_TIMEOUT，并结束运行。如果接收时间未超时，则进行下一步操作。。

（2）接下来，根据时间要求设置等待时间，并使用 read() 函数对上位机发送过来的字符进行接收。如果接收到的字符数不大于等于1（即 rc <1），则认为超时。将 result 的值设为 ERR\_RECV\_TIMEOUT，并返回 result 值并结束运行。如果 rc 大于1，则进行下一步处理。

（3）判断数据长度是否大于 sizeof(TPKGHEADER\_RECV)。如果是，则直接判断最大数据包长度 maxlimite 的值。如果 maxlimite 小于等于 0，则将 result 的值设为 ERR\_RECV\_FULL，并结束收数据函数。如果 maxlimite 大于0，则将 result 的值设为-1，并结束收数据函数。如果数据长度小于 sizeof(TPKGHEADER\_RECV)，则进行下一步处理。

（4）下一步调用 listenpackchk\_local() 函数对上位机发送的数据进行校验，当数据包合法时，对数据内容进行处理并回传。回传值有以下四种情况：0数据有效，1数据包头错，-2 长度不足，-3校验出错。然后对回传值进行判断，如果0 == result，则result = sumlen，数据有效，收数据函数返回result值并结束；如果ERR\_RECV\_CONTINUE==result，则判断maxlimite 是否小于等于0 ，若其小于等于0则result = ERR\_RECV\_FULL，返回result，并结束收数据函数运行。若maxlimite大于0，则result = -1，并结束收数据函数运行；如果ERR\_RECV\_PKGCRC==result，则报文校验码出错，收数据函数返回result值并结束。

**表4.9 串口收数据函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| maxrecv | Uint8\_t | 接收缓冲大小 |  |
| timeout | Bool | 超时时间 | 超时时间，单位为100ms |
| buf | Bool | 接收缓冲 |  |
| rc | Int | 接收字符 | 判断接收值，<1是超时 |
| maxlimite | Int |  | 最大接收长度，防止缓冲溢出 |
| TPKGHEADER\_RECV | struct | 报文接收结构 |  |
| ERR\_RECV\_TIMEOUT | #define | 接收超时 | -1 |
| ERR\_RECV\_FULL | #define | 数据接收缓冲区满 | -2 |
| ERR\_RECV\_PKGCRC | #define | 数据校验错 | -3 |
| ERR\_RECV\_CONTINUE | #define | 接收未完成，继续 | -6 |
| result | Int | 收数据函数返回值 | 0=执行无返回  -1=接收超时  -2=接收缓冲区满  -3=crc校验错  >0=有正确的数据报文件，返回整个报文长度 |

4.2.2 设备初始化

{0, Common\_Initialize\_Device},//设备环境初始化

读写器在进行读写操作前必须进行相应的设备初始化操作，只有正确地进行了设备初始化操作后，读写器才能够分别进行TVM入闸出闸处理、BOM型读写器票务处理等，跳过此步骤直接进行票卡处理会使得读写器无法匹配到相对应的票卡信息，使得返回码不为零，出现错误。

设备环境初始化根据接口传入的参数，加载设备所需参数，申请内部资源。设备要调用读写器处理票卡前必须先执行此调用。设备在初始化之前应先调用‘Config\_Parameter’配置好设备必须的参数，这些参数包括：

**表4.10 参数列表**

|  |  |
| --- | --- |
| 编号 | 参数名称 |
| 1 | 设备控制参数 |
| 2 | 车站代码表 |
| 3 | SAM卡对照表 |
| 4 | 换乘车站代码表 |
| 5 | 票卡参数 |
| 7 | 票价参数 |
| 8 | 降级模式使用记录 |
| 9 | 地铁单个黑名单 |
| 10 | 地铁黑名单段 |
| 11 | 一卡通单个黑名单 |
| 12 | 一卡通黑名单段 |
| 13 | 互联网票单个黑名单 |
| 14 | 互联网票黑名单段 |
| 15 | 行政罚金参数 |
| 16 | 乘次票专用参数 |
| 17 | 闸机专用通道参数 |
| 18 | 二维码扫描器参数 |
| 19 | 蓝牙模块参数 |

1、初始化设备函数

void Api::Common\_Initialize\_Device(uint8\_t \* param\_stack, uint8\_t \* data\_to\_send, uint16\_t& len\_data)

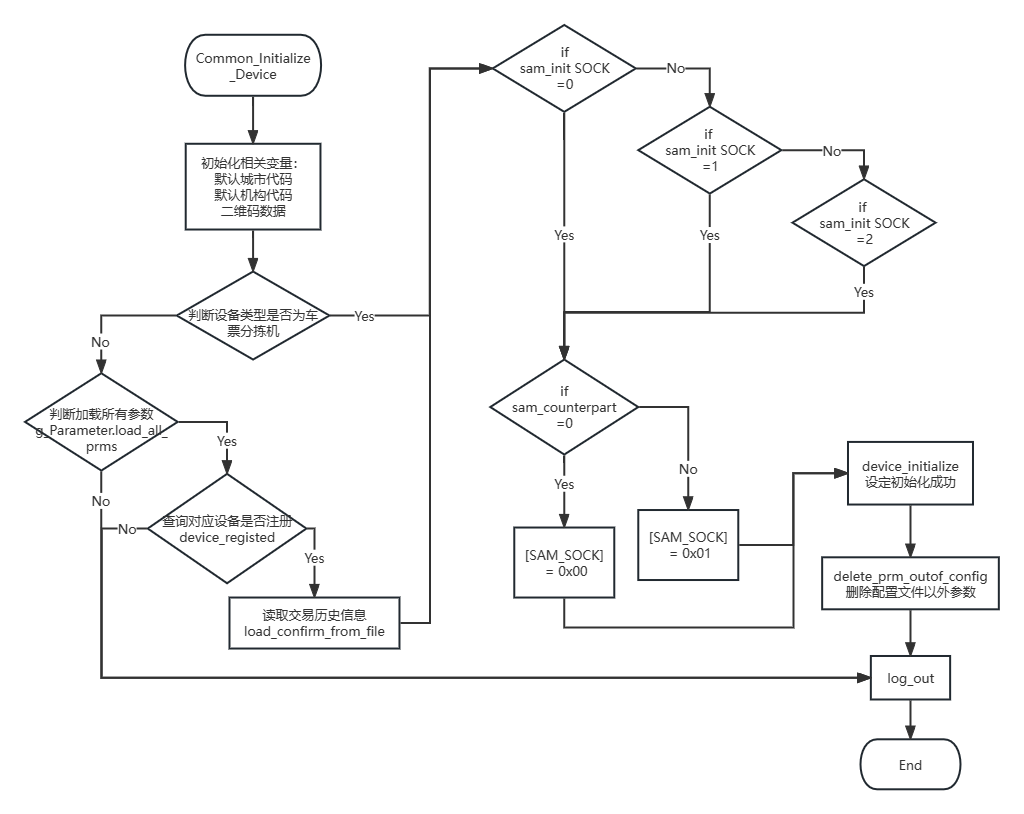


图4.10 初始化设备函数流程图

Common\_Initialize\_Device() 的流程如下：

（1）初始化相关变量，包括默认城市代码、默认机构代码和二维码数据等。

（2）根据设备类型进行不同的初始化操作。如果设备类型是“车票分拣机”，则直接进行 SAM 卡的初始化。否则，调用参数加载函数 g\_Parameter.load\_all\_prms 进行设备参数的加载。如果参数加载失败，则输出日志并结束设备初始化函数。。

（3）如果参数加载成功，则进行设备注册信息的检验操作，启动代码函数 device\_registed()。如果设备注册信息检验不通过，则进行日志输出并结束设备初始化函数；如果通过了检验，则继续下一步操作。

（4）下一步进行函数 load\_confirm\_from\_file() 用于读取交易历史信息。

（5）在完成相关参数的加载后，进行SAM卡的初始化。首先判断卡槽的槽号，包括（0/1/2）号卡槽，并进行相应的sam init函数，完成SAM卡的初始化。初始化完成之后，进行SAM卡与设备注册信息的匹配检查，函数sam\_counterpart作用于此。若匹配，则SAM卡状态返回值为0x00，如若不匹配，则sam卡槽返回值为0x01。

（6）判断完SAM卡槽状态后，进行函数device\_initialize设定参数为“初始化成功”，然后进行函数delete\_file\_overdue删除除配置文件外的参数配置，最后进行日志输出并结束初始化函数。

**表4.11 设备初始化函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| current\_station\_id | Uint16\_t | 当前车站代码 |  |
| current\_city\_id | Uint16\_t | 当前城市代码 |  |
| bLocalInstitutionCode | Uint8\_t | 受理机构代码 |  |
| device\_type | Uint8\_t | 设备类型码 | 09为车票分拣机  4为双通道闸机  5为入通道闸机  6为出通道闸机 |
| SAM\_SOCK\_i | #define | SAM卡卡槽号 | 0、1、2 |
| sam\_status | Uint8\_t | SAM卡与设备注册信息状态 | 0x00 为正确  0x01 为错误 |
| device\_init\_flag | Bool | 初始成功标志 |  |
| err\_code |  | 当前调用错误码 | err\_code查找到的级别高于系统日志级别，就写日志 |

2、SAM卡初始化函数

uint16\_t sam\_init(int sock\_id, char \* p\_sam\_id, char \* p\_tml\_id)

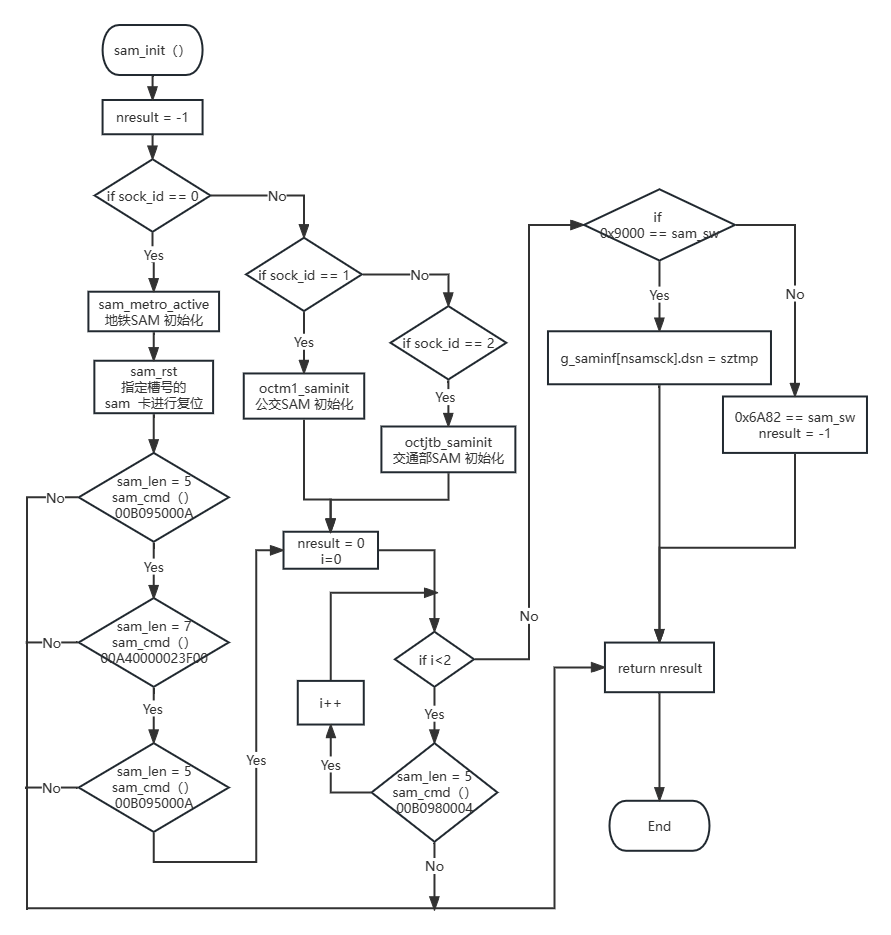


图4.11 SAM卡初始化函数流程图

进入初始化SAM卡函数sam\_init（），首先初始化变量nreuslt=-1，用于函数结束时判断是否初始化成功。然后判断此时的要初始化的SAM卡槽是哪种SAM卡，如果卡槽号为1，则是公交SAM卡的初始化函数。如果卡槽号为2，则是交通部SAM卡的初始化函数。本论文的设计实现为地铁AFC设备读卡器。因此着重于地铁SAM卡的初始化。当卡槽号为0时，进行地铁SAM卡的初始化函数sam\_metro\_active（）。

该函数首先进行sam卡相关信息的打印，接着启用函数sam\_cmd（），用于发送数据报文命令给SAM卡进行数据的校验。执行三种不同的数据校验，如果出错则直接返回nresult=-1，初始化失败，结束函数运行。如若正确则nresult赋值为0，进入for循环，再进行两次公共的SAM卡数据校验。检验失败则输出返回值，并结束函数进行。检验成功则判断流水号sam\_sw是否正确，当其值为0x9000时则是正确，输出SAM卡全局相关信息，包括：密钥索引、终端号长度、终端号逻辑卡号长度、取逻辑卡号的偏移、逻辑卡号等。若其的值不为0x9000，则返回错码6A84。结束判断sam\_sw的值后，返回nresult的值并结束SAM卡初始化函数的运行。

**表4.12 SAM卡初始化函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| current\_station\_id | Uint16\_t | 当前车站代码 |  |
| current\_city\_id | Uint16\_t | 当前城市代码 |  |
| bLocalInstitutionCode | Uint8\_t | 受理机构代码 |  |
| device\_type | Uint8\_t | 设备类型码 | 4为双通道闸机  5为入通道闸机  6为出通道闸机 |

4.2.3 版本信息获取

{1, Common\_GetVersion},//获取版本信息

void Api::Common\_GetVersion(uint8\_t \* param\_stack, uint8\_t \* data\_to\_send, uint16\_t& len\_data)

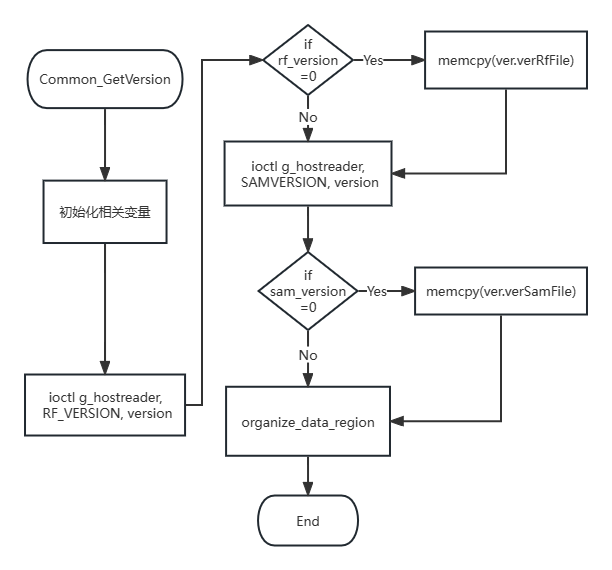


图4.12 版本信息获取函数流程图

进入版本获取函数Common\_GetVersion，首先初始化相关变量，然后用函数ioctl用于获取Linux开发板的信息。先是获取RF天线驱动信息。然后判断信息是否获取成功，如若是则存储获取的版本信息。然后获取SAM驱动信息。如若信息获取不成功直接跳转获取SAM驱动信息。同理，获取SAM信息成功则进行版本信息的存储。然后用函数organize\_data\_region返回相关的命令报文至上位机。结束函数运行。

4.2.4 获取SAM卡信息

{2, Common\_GetSamInfo},//获取SAM卡信息

void Api::Common\_GetSamInfo(uint8\_t \* param\_stack, uint8\_t \* data\_to\_send, uint16\_t& len\_data)

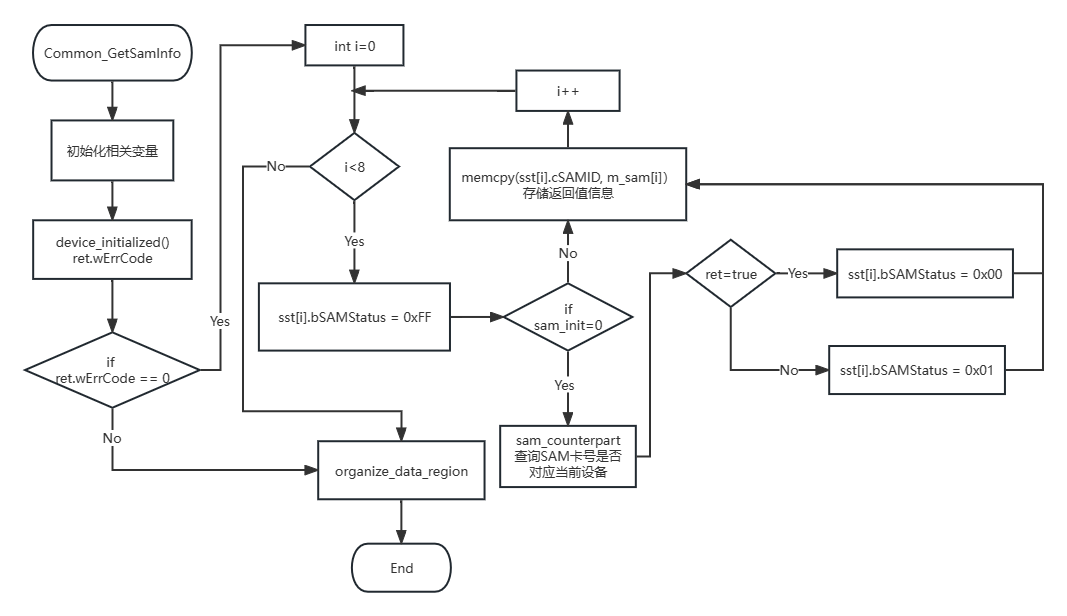


图4.13 获取SAM卡信息函数流程图

获取SAM卡信息函数启用，首先初始化相关变量，并获取Linux开发初始化设备的返回值，若返回值为0则是初始化失败，返回报文命令至AGM机，并结束函数运行。如果初始化成功。则进入for循环。

进入for循环，初始化i为0，因为本设计中的Linux开发一共有8个SAM卡槽，所以一共循环8次。当i<8时，进入循环体，将SAM卡信息初始值设为0xFF，即错误失败。然后判断SAM卡初始化是否成功，如果SAM卡初始失败，存储返回值信息并进行i++，判断下一个SAM卡槽状态。如果SAM卡初始化成功，就启用函数sam\_counterpart()查询SAM卡号是否对应当前设备。如果对应则将SAM卡信息值设为0x00，如果不对应则设为0x01。查询完SAM卡状态之后，存储返回值信息。继续判断下一个SAM卡槽状态。

当查询完8个SAM卡槽的状态之后，跳出循环体，返回报文至AGM机，并结束获取SAM卡信息函数运行。

4.2.5 获取票价

{4, GetFare},//获取到本站的票价

4.2.6 参数管理

{5, Common\_GetParamInfo},//获取参数信息

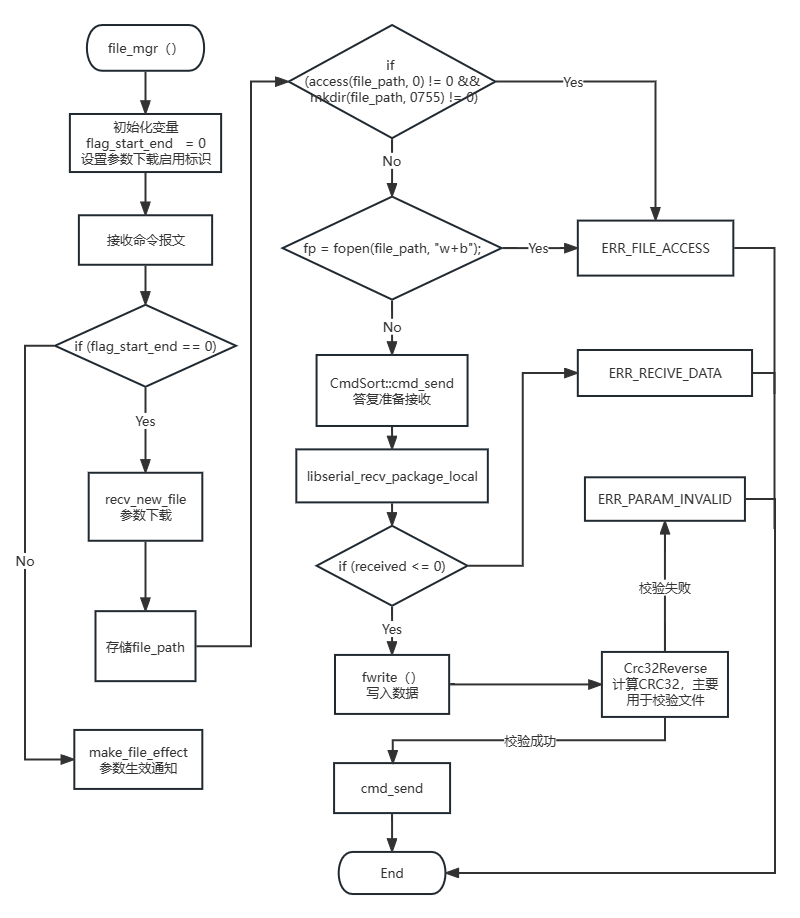
## 运营参数模块详细设计

从软件总模块功能函数操作可知运营参数操作为case COMMAND\_B:

Download dl; dl.file\_mgr(p\_cmd\_recved, p\_data\_to\_send, lentosend);

4.4.1参数下载

recv\_new\_file(p\_msg\_file, file\_name, file\_size, file\_crc, p\_data\_send, len\_send);



图参数下载函数流程图

dl.file\_mgr该函数用于参数管理，进入参数管理函数，首先先初始化参数下载标识为0，然后依照接收的上位机命令报文进入判断。如果加载报文命令后，标识仍为则启动参数下载函数recv\_new\_file()，否则启动参数生效函数make\_file\_effect()。

进入recv\_new\_file先一个指定的路径下创建一个临时文件夹。具体来说：running\_directory() 返回当前应用程序运行的目录（即当前路径）。表示临时文件夹的名称。使用 sprintf() 函数将其与宏定义NAME\_TEMP\_FOLDER结合起来，形成一个完整的路径字符串。存储到file\_path中。

然后判断指定的文件夹路径是否存在。：access(file\_path, 0)函数检查 file\_path 变量表示的路径是否存在，并且当前用户是否有读取权限。如果返回值为0，表示该路径存在且可读取；如果返回值不为0，说明路径不存在或者没有读取权限。如果路径不存在，则使用 mkdir() 函数来创建该路径。mkdir(file\_path, 0755) 中的 0755 表示权限控制，其中第一个数字7表示owner（文件所有者）的权限，第二个数字5表示group的权限，第三个数字5表示其他人的权限。其中的数字可以根据具体需求进行调整。该代码段整体的作用是：如果指定的路径不存在，就创建该路径。如果创建文件夹失败则返回错误码ERR\_FILE\_ACCESS。

接着使用 fopen() 函数打开file\_path 的文件，以 w+b 的模式打开，既可以写入，也可以读取二进制数据。如果打开文件失败，fopen() 函数将返回 NULL，程序检查文件指针是否为 NULL，如果是，则表示打开文件失败。返回错误码ERR\_FILE\_ACCESS。

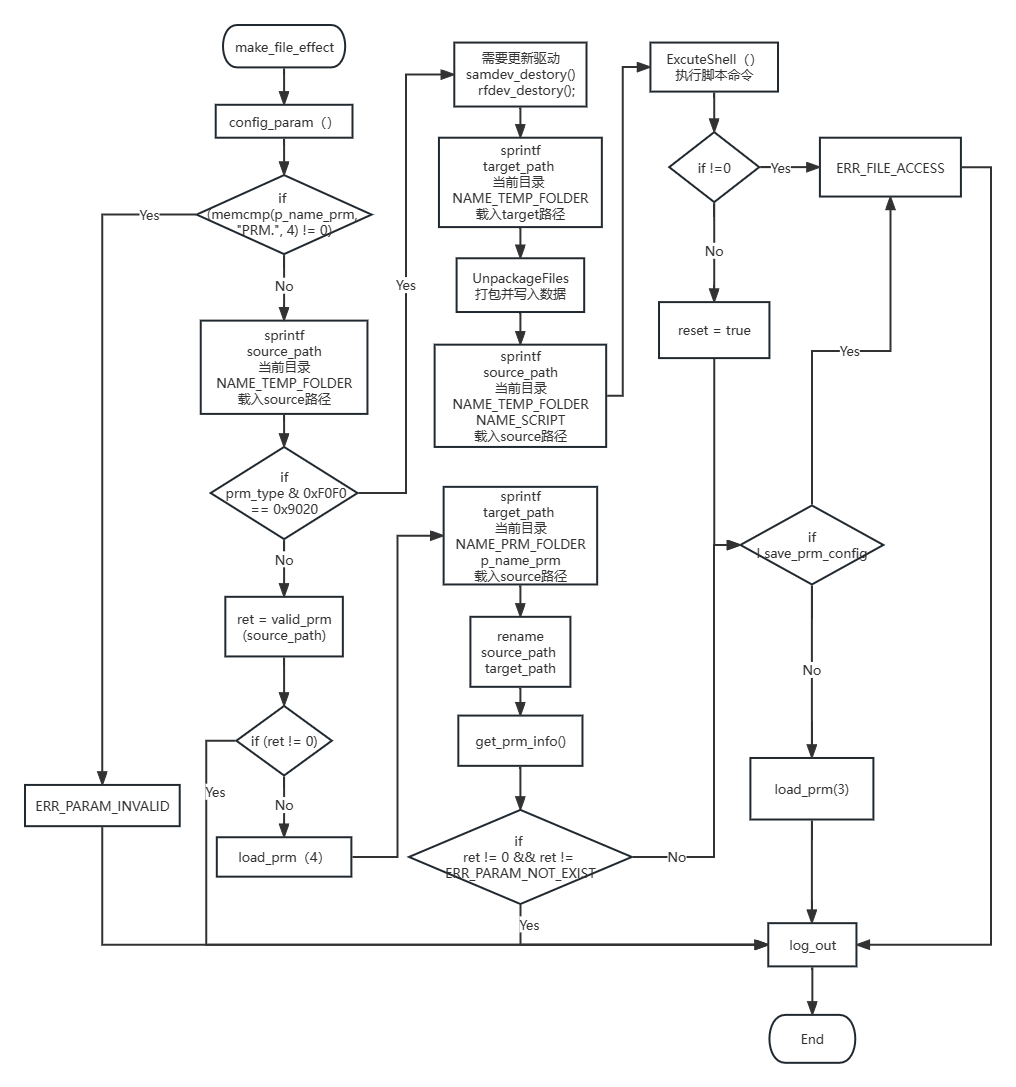
打开文件成功之后，读卡器返回应答报文，即 CmdSort::cmd\_send()函数用于答复准备接收。再之后进行数据的写入，写入完成之后会进行一个CRC校验。如若校验失败，则返回ERR\_PARAM\_INVALID。如果校验成功则返回写入成功的应答报文。结束参数下载函数，等待上位机的下一命令。

**表参数下载函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| flag\_start\_end | uint8\_t | 参数下载启用标识 | 0=参数下载通知  1=参数生效通知 |
| file\_path | char | 下载路径字符串 | 由当前目录与Tmp拼接 |
| NAME\_TEMP\_FOLDER | char \* | Tmp |  |
| ERR\_FILE\_ACCESS | #define |  | 文件错误 |
| fp | typedef struct | 文件打开标识 |  |
| total\_recv | uint32\_t | 下载文件大小 |  |
| crc\_calc | uint32\_t | CRC校验成功与否标识 | 初始值为：0xFFFFFFFF |

4.4.2 参数生效通知

make\_file\_effect(p\_msg\_file, file\_name, file\_size, file\_crc, p\_data\_send, len\_send);



图参数生效函数流程图

当参数下载启用标识为1时，进入make\_file\_effect（）参数生效函数，启用函数config\_param（）进行参数配置。首先判断下发的参数文件是否合法。若与字段"PRM."

不符合，则是返回结果ERR\_PARAM\_INVALID（参数文件不合法）。若是通过合法判断。进获取source的字符串路径，其由当“前路径+ NAME\_TEMP\_FOLDER+部分接收数据”组成。

接着用参数类型码和0xF0F0进行与运算，运算结果与0x9200若等于，则存在需要更新驱动的情况。进行函数samdev\_destory()和rfdev\_destory()用于相关驱动的更新。更新完驱动进行target字符串路径的更新载入，其由“当前路径+ NAME\_TEMP\_FOLDER”组成。然后启用函数:UnpackageFiles（）用于打包并写入参数数据。完成该函数后，获取source的字符串路径，由“当前路径+NAME\_TEMP\_FOLDER+ NAME\_SCRIPT”组成。再然后用函数ExcuteShell命令Linux开发板执行source路径下脚本命令。若执行失败，返回ERR\_FILE\_ACCESS，执行成功则结束该模块。

如若参数类型码和0xF0F0进行与运算，运算结果与0x9200若不等于，则是其他参数类型处理。启用函数valid\_prm（）用于检查参数的合法性。若参数不合法，则输出日志，结束参数生效函数运行。参数合法则进行函数load\_prm（）加载当个参数。加载失败同样输出日志并结束参数生效函数运行。加载成功则获取target的字符串类型路径，由“当前路径+NAME\_PRM\_FOLDER+数据接收部分”组成。接着用函数rename（），将target的字符串赋给source字符串。然后进行函数get\_prm\_info（）获取参数信息。获取失败返回ERR\_PARAM\_NOT\_EXIST，成功则结束该模块。

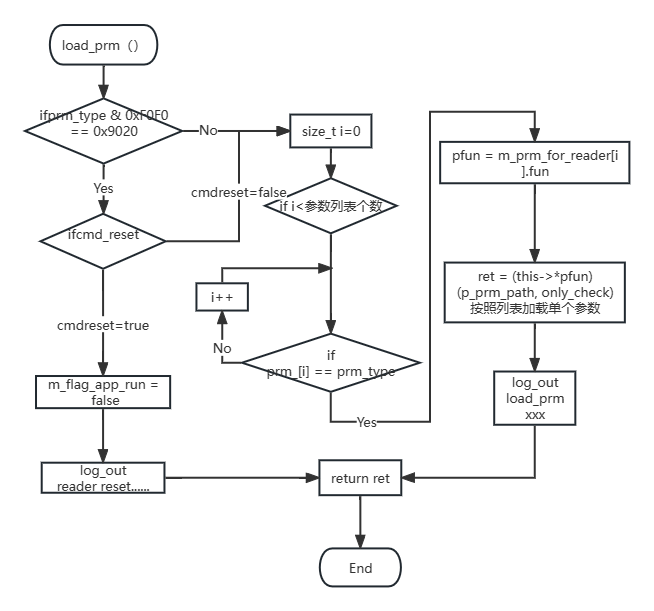
在结束以上任一功能模块后，进行参数配置的更改即函数save\_prm\_config（）。更改失败则返回ERR\_FILE\_ACCESS。更改成功则再次运行函数load\_prm（）加载单个参数。

**表参数生效函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| p\_name\_prm | char \* | 接收的部分报文 |  |
| ERR\_PARAM\_INVALID | #define | 参数文件不合法 | 0x0101 |
| ERR\_FILE\_ACCESS | #define | 文件错误 | 0x0102 |
| ERR\_PARAM\_NOT\_EXIST | #define | 参数文件不存在 | 0x0100 |
| NAME\_TEMP\_FOLDER | char \* | Tmp |  |
| NAME\_SCRIPT | char \* | api\_script.sh |  |
| NAME\_PRM\_FOLDER | char \* | ParamFolder |  |

4.4.3 参数加载

load\_prm(const uint16\_t prm\_type, const char \* p\_prm\_path, bool cmd\_reset/\* = false\*/, bool only\_check/\* = false\*/)



图加载单个参数函数

进入参数加载函数load\_prm，首先判断prm\_type & 0xF0F0的结果是否等于 0x9020，如果等于再进行判断cmd\_reset的值，如果其值为true，则将m\_flag\_app\_run赋值为false，直接移动到下载文件启用消息。输出日志"reader reset......"，并结束函数运行。

如果与的结果不等于0x9020或者cmd\_reset为false，则进入for循环，先初始化size\_t i的值为0，，判断参数列表中i的值是否与当前下载参数一致，如若不一致则进行i++，继续判断参数列表值。如若一样则启用对应参数列表中的加载单个参数函数。以此循环不断检索参数列表，直到成功加载对应的单个参数。加载完成后则进行日志输出，并结束函数运行。

**表加载单个函数字段说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 说明 | 详细内容 |
| prm\_type | uint16\_t \* | 当前要加载函数 |  |
| m\_flag\_app\_run | Bool | 重启标志位 |  |
| i | int | 参数列表编号 |  |
| m\_prm\_for\_reader[i] | typedef struct | 参数列表 |  |
| ret | uint16\_t | 加载的参数 |  |

**表AGM读卡器参数说明**

|  |  |  |
| --- | --- | --- |
| 列表编号 | 功能函数名称 | 说 |
| 0x0101 | load\_device\_control | PRM\_0101\_DEVICE\_CTRL设备控制参数 |
| 0x0202 | load\_station\_config | PRM\_0202\_STATION\_CFG车站表 |
| 0x0203 | load\_sam\_compare | PRM\_0203\_SAM\_CMP SAM卡对照表 |
| 0x0204 | load\_change\_station | PRM\_0204\_STATION\_EX换乘车站代码表 |
| 0x0301 | load\_ticket\_prm | PRM\_0301\_TICKET票卡参数 |
| 0x0302 | load\_special\_passageway | PRM\_0302\_ALLEY\_WAY闸机专用通道参数 |
| 0x0303 | load\_tct\_config | PRM\_0303\_TCT乘次票专用参数 |
| 0x0400 | load\_fare\_prm | PRM\_0400\_FARE票价文件 |
| 0x0501 |  |  |
| 0x0601 | load\_mtr\_single\_black | PRM\_0601\_SINGLE\_BLACK\_M地铁单个黑名单 |
| 0x0602 | load\_mtr\_section\_black | PRM\_0602\_SECTION\_BLACK\_M地铁黑名单段 |
| 0x0603 | load\_oct\_single\_black | PRM\_0603\_SINGLE\_BLACK\_O一卡通单个黑名单 |
| 0x0604 | load\_oct\_section\_black | PRM\_0604\_SECTION\_BLACK\_O一卡通黑名单段 |
| 0x0605 | load\_jtb\_single\_black | PRM\_0605\_JTB\_SINGLE\_BLACK\_O交通部一卡通单个黑名单 |
| 0x0606 | load\_jtb\_white | PRM\_0606\_JTB\_WHITE\_O交通部一卡通白名单 |
| 0x0801 | load\_policy\_penalty | PRM\_0801\_POLICE\_FINE行政罚金参数 |

## AGM机业务功能模块详细设计

4.3.1通道类型处理

1、通道类型函数

{13, Gate\_AisleModel}

void Api::Gate\_AisleModel(uint8\_t \* param\_stack, uint8\_t \* data\_to\_send, uint16\_t& len\_data)

（1）先调用uint16\_t device\_initialized();

判断设备是否完成初始化，即加载参数（读写器异常重启后，设备程序无法知晓，导致没有调用设备初始化）如果无错返回0，有错则返回相应错误码

（2）再调用uint16\_t Api::api\_match\_device(ETPDVC dvc\_type\_support)

传入dvc\_type\_support判断设备是否为AGM机

（3）再次调用uint16\_t Api::api\_match\_device(dvcEntryAGM)

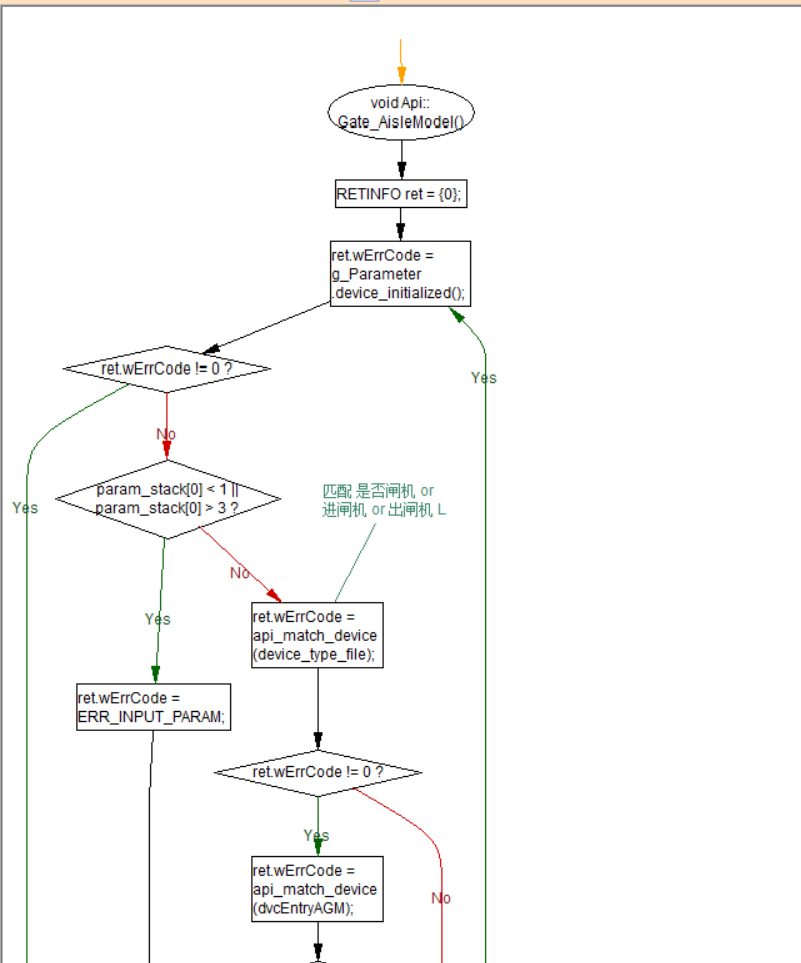
判断是否为入闸机

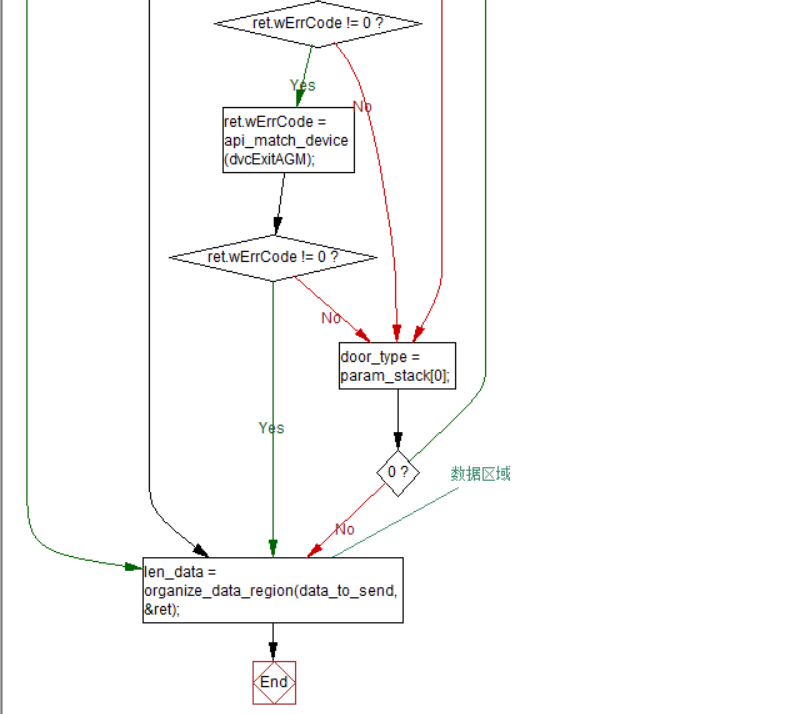
（4）再次调用uint16\_t Api::api\_match\_device(dvcExitAGM)

判断是否为出闸机

（5）最后设定闸机类型door\_type = param\_stack[0];

（6）len\_data = organize\_data\_region(data\_to\_send, &ret);





4.3.2入闸处理

{14, Gate\_EntryFlow},

void Api::Gate\_EntryFlow(uint8\_t \* param\_stack, uint8\_t \* data\_to\_send, uint16\_t& len\_data)

分析票卡开始

ret.wErrCode = g\_Parameter.query\_ticket\_prm(ticket\_data.logical\_type, ticket\_prm);

if (ret.wErrCode) break;

dbg\_formatvar("passageway\_allow\_pass");

检查专用通道

ret.wErrCode = g\_Parameter.passageway\_allow\_pass(ticket\_data.logical\_type, door\_type);

if (ret.wErrCode) break;

dbg\_formatvar("device\_support\_ticket");

设备支持

ret.wErrCode = ParamMgr::device\_support\_ticket(device\_type\_file, ticket\_prm.usable\_device);

if (ret.wErrCode != 0) break;

dbg\_formatvar("analyse\_common");

检查是否重复过闸 如果是电子票卡的话，需要检查重复使用二维吗

if(ticket\_data.qr\_flag == 0x01)

{

if(g\_LoopList.QueryList((char \*)ticket\_data.timeAndcenterCode,operEntry))

{

dbg\_formatvar("this is the same qr code");

ret.wErrCode = ERR\_CARD\_NONE;

memset(Api::qr\_readbuf,0x00,sizeof(Api::qr\_readbuf));

交易结束，不管是否成功，再次清掉二维码

memset(Api::clear\_qr\_readbuf,0x00,sizeof(Api::clear\_qr\_readbuf));//交易结束，不管是否成功，再次清掉二维码

ret = p\_ticket->analyse\_free\_zone(&md\_effect, &ticket\_prm, NULL, NULL);

// 交易确认

ret = p\_ticket->entry\_gate(&entry,&eticketdealinfo);//开门L

if(ret.wErrCode == ERR\_NEED\_CONTINUE\_LAST)//交易确认

memset(Api::qr\_readbuf,0x00,sizeof(Api::qr\_readbuf));//交易结束，不管是否成功，再次清掉二维码

memset(Api::clear\_qr\_readbuf,0x00,sizeof(Api::clear\_qr\_readbuf));//交易结束，不管是否成功，再次清掉二维码

//memset(Api::qr\_readbuf,0x00,sizeof(Api::qr\_readbuf));//交易结束，不管是否成功，再次清掉二维码

4.3.3出闸处理

{15, Gate\_ExitFlow},

void Api::Gate\_ExitFlow(uint8\_t \* param\_stack, uint8\_t \* data\_to\_send, uint16\_t& len\_data)

# 系统测试

## 测试环境

## 功能测试

## 测试结果

本项目中互联网票务处理读写器是一个软件硬件相结合的产品，本文档中所指读写器如未作特殊说明，均表示含带票务处理软件的硬件产品。读写器作为AFC设备的一个标准件，负责IC卡车票、新增互联网电子车票及SAM卡相关的业务处理功能，不参与主控设备相关的控制，从而在物理层及业务应用层上很好的解决了设备间的藕合性。

AFC设备通过调用读写器提供的标准协议接口完成IC卡车票、互联网电子车票的业务处理，整个业务处理由读写器独立完成，读写器处理完成后返回执行状态值和回传参数，AFC设备根据读写器返回的信息完成通行逻辑处理及行政处理业务。

AFC主控设备可通过RS232接口或TCP/IP与读写器进行数据交互，读写器以被动模式进行工作，，由AFC主控设备发起通讯，读写器收到指令后进行相应的处理并回应执行结果。

# 总结

# 参考文献

1. 陈宇，刘晶晶，黄曼全，张海峰.探讨地铁自动售检票系统(AFC)车站设备布置的原则[J].中国安全生产科学技术，2020，16(S1):82-85.
2. 顾珉睿，郝旭鹏，汤茂东，刘建新，王学进.闸机通行识别技术的研究[J].工业控制计算机,2018,31(03):55-58.
3. 李伯虎,柴旭东，张霖，卿杜政，施国强，林廷宇，郭丽琴，杨晨，谷牧，贾政轩，公慧，唐震.面向智慧物联网的新型嵌入式仿真技术研究[J].系统仿真学报，2022，34(3):419-441
4. 潘丽华,王莉莉.嵌入式软件开发环境的构建[J].信息与电脑，2019，0(1):117-118
5. 秦婷婷.交通领域近距离无线通信技术应用[J].运输经理世界，2022(05):61-63.
6. 吴镭，陈敬华.排查一起由NFC杂散信号引发的干扰[J].中国无线电，2022(3):59-60.
7. Sonwalkar Prakash K.,Kalmani Vijay. Design and implementation of enhanced security model for wireless sensor network on ARM processor[J]. Measurement: Sensors,2022,24.

# 致谢