文本

描述已自动生成

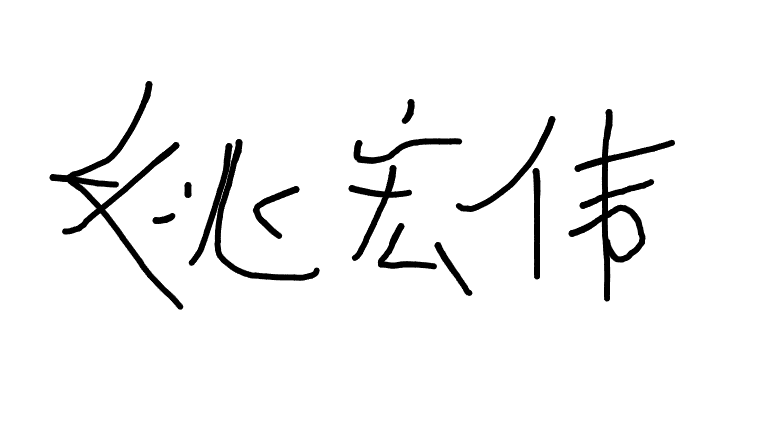
**本 科 毕 业 论 文（设 计）**

|  |  |
| --- | --- |
| 题 目 | 基于Rust的跨平台USB驱动 |
|  | 子系统的设计与实现 |
| 学生姓名 | 姚宏伟 |
| 学 号 | 2311729 |
| 学 院 | 信息技术学院 |
| 专业班级 | 计科B21-10(专升本) |
| 指导教师 | 丁菊 |
| 交稿日期 | 2025年3月20日 |

教务处制

**上海建桥学院毕业论文（设计）学术诚信声明**

本人郑重声明：所呈交的毕业论文（设计），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本毕业论文（设计）不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。



作者签名： 日期： 2025年 3 月 20日

**上海建桥学院毕业论文（设计）版权使用授权书**

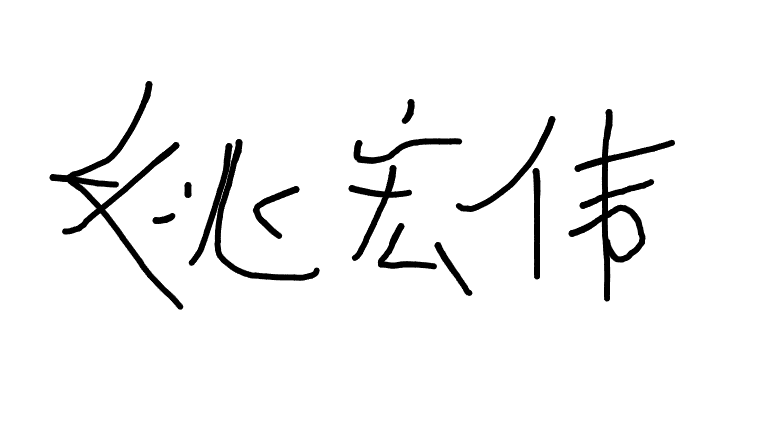
本毕业论文（设计）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海建桥学院可以将本毕业论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业论文（设计）。

**保 密** □，在 年解密后适用本授权书。

本论文属于

**不保密** ☑。

（请在以上方框内打“**√**”，如作者未做出选择的情况下，按不保密处理。）



作者签名： 指导教师签名：

日期：2025年 3 月20 日 日期： 年 月 日

基于Rust的跨平台USB驱动子系统的设计与实现

摘 要

随着计算机技术与硬件的不断进步，USB（通用串行总线）已经成为连接计算机与各类电子设备的通用通信接口。USB驱动程序在这些设备间的通信过程中扮演着至关重要的角色。然而，随着国内操作系统开发的兴起，众多国产操作系统应运而生。由于USB接口的复杂性——甚至可以明确的说，它是目前最复杂的总线之一——其驱动程序的开发颇具挑战性。此外，传统的USB驱动程序往往针对特定操作系统进行定制开发，这使得将现有的USB驱动程序迁移到其他操作系统框架下变得异常困难。

鉴于此，本文提出了一种跨架构、跨操作系统的USB主机端驱动框架。该框架兼容当前主流的USB主机控制器版本，并为上层驱动提供了一个统一的抽象接口。它支持在单线程或多线程环境下异步运行，并采用了一种基于中断、主动让出、轮询进行任务切换的异步编程模型。

此框架采用Rust编程语言实现，旨在充分利用Rust语言的安全性和可移植性优势。

**关键词：通用串行总线（USB）；驱动程序；操作系统；Rust编程语言**

A Cross-Platform USB Host Driver Subsystem Based On Rust

Abstract

With the continuous advancement of computer technology and hardware, USB (Universal Serial Bus) has become a universal communication interface for connecting computers with various electronic devices. USB drivers play a crucial role in the communication process between these devices. However, with the rise of domestic operating system development, numerous locally produced operating systems have emerged. Due to the complexity of USB interfaces—which can even be unequivocally stated as one of the most complex buses currently—the development of USB drivers is quite challenging. In addition, traditional USB drivers are often custom-developed for specific operating systems, making it extremely difficult to migrate existing USB drivers to other operating system frameworks.

In view of this, this paper proposes a cross-architecture and cross-operating system USB host driver framework. This framework is compatible with current mainstream USB host controller versions and provides a unified abstract interface for upper-layer drivers. It supports asynchronous operation in single-threaded or multi-threaded environments and adopts an asynchronous programming model based on interrupts, active yielding, and polling for task switching.

The framework is implemented using the Rust programming language, aiming to fully leverage Rust's advantages of safety and portability.

**Key Words：Universal Serial Bus (USB); Driver System; Operating System; Rust Programming Language**

目 录

[目 录 III](#_Toc12393)

[1 绪论 1](#_Toc27905)

[1.1 课题背景 1](#_Toc32401)

[1.2 相关领域国内外发展现状与趋势 2](#_Toc22795)

[1.2.1 国外发展现状 2](#_Toc11408)

[1.2.2 国内相关技术的发展趋势 2](#_Toc6102)

[1.3 论文研究内容及主要工作 3](#_Toc14454)

[1.3.1 论文研究内容 3](#_Toc13165)

[1.3.2 论文主要工作 5](#_Toc24437)

[2 跨平台USB主机端驱动子系统的整体设计方案及原理概述 7](#_Toc10820)

[2.1 系统整体方案设计 7](#_Toc11284)

[2.2 下文所涉及概念与名词解释 8](#_Toc21282)

[2.3 系统各模块具体的技术研究 11](#_Toc7913)

[2.3.1 主机驱动层的技术细节 11](#_Toc25963)

[2.3.2 USB驱动层的技术细节 13](#_Toc3668)

[2.4 本章小结 16](#_Toc28796)

[3 硬件与仿真环境 17](#_Toc31946)

[3.1 开发所用平台环境 17](#_Toc9024)

[3.1.1 对于Arm架构的测试与验证 17](#_Toc25260)

[3.1.2 对于RISC-V架构的测试与验证 19](#_Toc3374)

[3.2 测试所用硬件设备 19](#_Toc4042)

[3.3 本章小结 21](#_Toc3134)

[4 软件程序设计与实现 22](#_Toc10747)

[4.1 程序设计语言及开发环境 22](#_Toc30630)

[4.1.1 编程语言：Rust程序设计语言 22](#_Toc14705)

[4.1.2 工具：Makefile 22](#_Toc21849)

[4.1.3 集成开发环境：VSCode 22](#_Toc30900)

[4.2 模块化设计与项目依赖 23](#_Toc6301)

[4.3 USB驱动子系统框架的实现 24](#_Toc3430)

[4.3.1 USB驱动子系统模块 24](#_Toc6708)

[4.3.2 主机控制器接口 25](#_Toc14399)

[4.3.3 USB系统驱动层模块 26](#_Toc16108)

[4.4 USB描述符解析系统的实现 26](#_Toc15850)

[4.4.1 USB描述符的数据流 26](#_Toc10131)

[4.4.2 USB描述符解析系统的设计 28](#_Toc6044)

[4.5 本章小结 29](#_Toc23969)

[5 系统测试 30](#_Toc12709)

[5.1 USB描述符解析器的软件测试 30](#_Toc18963)

[5.2 USB驱动子系统软硬件协同测试 31](#_Toc28490)

[5.2.1 USB驱动子系统的初始化流程 32](#_Toc28754)

[5.2.2 基于驱动无关设备中各描述符所对应USB功能的驱动指派 33](#_Toc9893)

[5.2.3 驱动模块是否正常工作的测试 34](#_Toc29977)

[5.3 USB驱动子系统在进一步测试过程中所暴露出的问题与解决方案 35](#_Toc7271)

[结 论 38](#_Toc31153)

[参 考 文 献 39](#_Toc1700)

[致 谢 40](#_Toc22024)

# 绪论

## 课题背景

随着计算机技术和硬件的发展，USB（Universal Serial Bus）已成为目前计算机和其他电子设备之间通信和连接所使用的最广泛的接口。与此同时，随着国产化的兴起，无数国产操作系统开始涌现，USB作为一种较为复杂的接口，其驱动开发起来较为困难，而且传统的USB驱动程序通常针对特定的操作系统进行开发，这导致了众多非linux发行版的国产操作系统总是缺少USB驱动支持的问题。事实上，在整个驱动开发领域，包括嵌入式领域，都有着此类缺少驱动生态的共性问题。不同平台的软硬件底层架构和驱动模型不同。更进一步的，特定的驱动往往只在特定的平台上开发，根本无法通用，从而产生了诸多重复劳动。更甚者，由于开发人员的精力分散，难以形成统一的编写规范，从而还对所编写出程序的安全性产生了影响。事实上，驱动程序所隐含的漏洞已成为现代操作系统生态中最主要的威胁。由H. Peng和M. Payer在29th USENIX Security Symposium上提出的USBFuzz[1]测试框架揭示了linux中现有USB驱动中可能存在的安全漏洞及其测试方法，凸显了对新开发驱动进行严格安全性测试的重要性。

造成第三方操作系统USB生态难以构建的原因还有另一方面。随着USB规范的更新迭代，USB总线的吞吐容量越来越高，其规范也越来越复杂，这也对相应的设备驱动编写人员的技术，乃至于操作系统管理IO的能力提出了更高的要求，J. Huang和Y. Wang的研究[2]专注于基于USB 3.0外设控制器的高速同步数据传输系统，这表明未来USB驱动将更加注重性能优化。且可以看到的是，IO设备，尤其是USB设备，有着将应用层协议逐渐下沉到协议层的趋势。这令使用传统C语言进行驱动开发的开发者们愈发难以应付高层次协议的高抽象程度的要求。Rust语言是一种内存安全，提供零成本抽象的高级编程语言，其独特的设 计理念与极高的性能优势使得其在面向计算机底层的开发的环境下大展身手。而N. Borgsmüller的研究[3]也指出，Rust在嵌入式开发上比起传统的C语言开发在多个方面上更具优势。

2019年，图灵奖获得者John Hennessy和David Patterson在他们合著的《计算机体系结构：量化研究方法》中，提出了“计算机体系结构即将迎来新黄金年代”的观点。而且也能看到，不同架构，设计理念的操作系统正如雨后春笋一般涌现出来。韩乃平和李蕾的文章[4]分析了国产操作系统的生态建设现状。对于一个成熟操作系统，USB系统的支持是不可或缺的，然而由于USB系统本身的复杂性，现有的USB驱动程序普遍都是为特定的操作系统设计的，难以在不同操作系统之间移植。文章也指出，构建跨平台兼容的驱动程序对于推动技术自主可控具有重要意义。

## 相关领域国内外发展现状与趋势

### 国外发展现状

在国外，特别是欧美地区，这些地区的科技公司与研究机构成立已久，往往掌握着业界事实上的工业标准。如USB-IF（USB Implementers Forum）和PCI-SIG（Peripheral Component Interconnect Special Interest Group）这类国际标准组织，往往能在背后看到各家大型企业的身影。这些组织所定下的标准的指定不但促进了不同厂商的产品之间的兼容性，亦是驱动程序开发的依据。但现实的情况是，对于驱动程序生态而言，如Intel、NVIDIA、AMD这类大型跨国企业往往自行开发其设备的驱动以推广自家产品。虽然也积极的参与到开源社区中，分享其成果，但也反过来产生了强者恒强的羊群效应。对于各类全新的平台，之所以难以构建其驱动生态，正是因为业界往往只会给当前已经存在的，已有了大量基础工作的unix-like与windows系统进行驱动开发。而可预见的，随着时代的发展，此类技术债问题会越积越多，若是不早以想办法解决，最终会导致计算机体系结构的基础设施创新的难度越来越高。

### 国内相关技术的发展趋势

近年来，中国在信息技术领域取得了显著进步，尤其是在移动互联网、大数据及人工智能等方面展现出了强劲的增长势头。得益于政府对科技创新的支持力度加大以及市场环境的改善，本土企业在追赶国际先进水平的同时也逐渐形成了自己的特色。

以华为、中兴通讯为代表的通信设备制造商通过自主研发掌握了大量核心技术，其中包括适用于特定应用场景下的定制化驱动开发。但也依然存在与国外类似的问题，即对于非主流平台的生态支持程度不高。事实上，由于国内IT行业起步较晚，并未形成类似于国外那般繁荣的开源社区与文化，使得计算机体系结构的生态构建的问题更加严峻，呈现出了“头重脚轻”的问题。一些专注于嵌入式系统研发的小型创业团队正努力填补国内市场空白，他们往往聚焦于某一垂直领域深入挖掘，力求提供差异化的产品和服务，但小型企业更容易面临人手不足，从而导致社区支持力度不足的问题。除了大型企业之外，各家研究机构也加大了对于整个计算机体系结构方向的投入。而计算机体系结构是一个整体，软件与硬件之间有着联动的关系。相信随着国内硬件供应链的逐步完善，大量的硬件驱动开发需求也会随之涌现，与此同时产生的效应是大量的开源社区成果的诞生。

## 论文研究内容及主要工作

### 论文研究内容

本次设计主要研究对象与目标为设计并实现一套高效，可靠且安全的USB主机端驱动子系统，其包含了若干组成部分及要求。

#### 主机控制器驱动层及其抽象层

对于当前的USB协议栈，USB主机端的工作模式与USB设备端的工作模式存在着显著的差异。具体的，在USB设备端的视角下，USB接口就仅仅只是一个IO接口而已，所需要在意的也仅仅只是数据的收发格式。而USB主机端的复杂度则明显高了很多。USB主机端往往通过特定的控制器来统一管理整个USB总线的资源。随着时代的变迁，USB主机控制器也演化出了诸多版本。因此，处于历史包袱与现实的需要，USB主机端的驱动开发通常不仅限于针对特定USB设备类型的驱动编写与管理，而是首先需要构建一个高效的USB主机控制器驱动，在此控制器的基础上去再去写具体的主机端协议驱动。更糟糕的是，由于历史原因，不同版本的USB主机控制器规范并不保持向下兼容性。而实践中厂商所生产的设备平台，其外设组合也千变万化。鉴于此，为了确保系统的兼容性和通用性，有必要将不同版本的USB主机控制器驱动抽象化为统一的接口层，以便上层主机端驱动程序能够通过统一的接口层，消除不同版本控制器规范之间的巨大差异，与外设进行交互。

#### USB功能驱动层

USB协议是分层设计的，其如互联网架构一般，从物理层，到链路层，再到传输层与协议层，各层的信息基本单位不同。不同层之间有一套特定的规范，规定了信息的分解与组合的方式。如用户所看见的U盘（USB块协议），USB鼠标（USB HID协议）,这类协议实际上是处于协议层的应用协议。更甚者，单个物理设备可以包含着多个应用协议。对于同一个应用协议，甚至还有着不同版本之间的差异。但总而言之，在USB规范的视角下，应用协议的内部版本由协议驱动自身管理，而特定的应用协议会被抽象为一个USB Function（USB功能）。通常，一个特定的USB功能可由一个三元组唯一标识，该三元组包括Class（类别）、SubClass（子类别）和Protocol（协议）。一个设备可以存在多个USB功能，在传输层，有一套用于管理单个设备上多个USB功能的机制。鉴于存在众多不同的USB协议，并且可预见到的，未来这些协议的数量必将持续增长，因此，构建一套高效且可扩展、可灵活管理的驱动模块管理系统显得尤为必要。

该系统应能实现驱动模块的热拔插。当有着新连接的USB设备时，其应加载相应的驱动程序，实现即插即用的功能。同时，系统还应能妥善处理设备的移除，确保在设备断开连接时，能够安全的终止通信，避免数据丢失或系统稳定性。其与下层的主机驱动层应当规定一套高效的通信机制，使得应用层驱动与主机控制器协议之间不至于拥有过高的耦合度。

#### 可单线程运行的异步工作

USB作为输入/输出（IO）总线，其面临的主要问题在于不同层级设备间数据交换的延迟差异极大。 对于USB而言，通常会在同一时刻并发的对多个USB功能进行读写操作。这一现象要求系统必须采用非阻塞的工作模式，以避免任务饥饿现象的发生，进而影响用户的体验与系统的稳定。但是在嵌入式系统环境中，往往缺少多任务管理机制，而是通常采用定时轮询结合中断的编程范式来完成IO任务。进一步的，在实时操作系统（RTOS）场景中，其对实时性要求更为严苛，传统的延迟-轮询工作模式也并不适用，中断机制也会受到严格的限制。Rust语言提供了一套独特的异步编程范式，称为无栈协程。其允许IO任务在即将发生阻塞时主动让出控制权（Yield），从而实现不同于以往线程模型中，由线程控制块（PCB）为基础的栈上下文切换来实现的异步工作模式，而是基于Future机制的控制流切换，故此被称为无栈协程。这一理念使得即便在单线程环境下，甚至在缺乏中断支持的情况下，也能实现异步程序的运行。这种编程范式不仅提升了程序的运行效率，还增强了系统的响应能力，确保系统在高并发和高实时性要求的场景下能够稳定且高效的运作。

#### USB设备报文解析模块

正如前文所说，USB协议栈在传输层具有一套特定的设备控制机制，其定义了一组称为USB描述符（USB Descriptor）的控制信息格式。USB设备的协议栈的详细信息通过这些描述符得以表达，这些描述符通常以树状结构组织，具体结构可见第2.2章。目前，市场上尚未出现通用的描述符解析库。因此，本驱动子系统还须包含一套通用的解析系统以供应用层使用。同时，鉴于不同的应用层往往会添加特定的描述符结构，描述符解析系统也应同USB功能驱动层一样，为不同的应用层协议提供描述符解析模块的热拔插机制。

#### 事件总线及其订阅机制

在工业领域，一个重要的研究课题针对各系统组件的动态调试。研究者们通常期望在组件正常运行期间，对其性能进行深入的评估。如eBPF，XDP等技术，为了实现这一目标，这类技术通常会在代码中插入一种特殊的注入点。当程序运行到该注入点时，会触发挂载于该注入点上的外部代码的调用，从而探查进程的上下文变量，甚至改变原始运行逻辑。这种注入的代码逻辑通常被称为“钩子（Hook）”。通过钩子技术，开发者能够在不修改原有代码的基础上，对系统的运行行为进行干预和控制，实现更灵活的功能扩展和性能优化。

对于USB协议栈及其对应的驱动程序而言，此类技术是必要的。对于典型的unix类的系统，一类常见的需求就是将外部IO设备挂载至文件上。这要求系统能够监控对应块设备驱动程序的工作状态，并根据需要发起请求。因此，USB系统必须向外部提供相应的操作接口。在直觉上，往往会倾向于直接向外暴露出一个驱动设备的索引表。问题在于，对于驱动来说，各个驱动及各个驱动的各个驱动设备实例往往是互不相关且状态大小均可变的。因此，直接向外暴露出索引表，但索引表内的指针类型却实际上不同，从而导致了大量的问题。并且，这样的设计还不是进程安全的。这便产生了更进一步探索的价值。

为了满足这一类需求，本系统采用事件驱动下的外部闭包注入的方式来实现外部逻辑注入的目的。事件驱动模型是一种编程范式，它允许系统在特定事件发生时，如设备连接、数据传输请求等，自动触发相应的处理函数或方法。在本系统中，存在着一条逻辑上的事件总线。各驱动设备实例及系统本身的各部分模块都会向该事件总线发布带有上下文引用的事件以供订阅了该事件的外部逻辑进行操作。通过这种方式，USB系统可以更加高效和灵活的响应外部设备的变化，确保数据的正确传输和设备的稳定运行。通过事件系统将平台的耦合逻辑编写事件处理器，亦可以保证系统本身与平台代码上的解耦与系统本身的可扩展性。

### 论文主要工作

本文围绕跨平台USB主机端驱动子系统的设计与实现展开研究，主要工作涵盖理论分析、架构设计、模块实现以及实验验证等方面，主要的内容如下：

第一章，绪论。本章系统阐述了USB驱动开发的背景与挑战，分析了国内外技术发展现状，明确了跨平台兼容性、安全性及异步执行效率等核心问题。通过文献调研，提出了基于Rust语言构建跨平台驱动框架的可行性，并规划了论文的研究方向与目标。

第二章，跨平台USB主机端驱动子系统的整体设计方案及原理概述。本章提出了分层架构设计，包含主机驱动层、USB驱动层和系统抽象层。重点设计了驱动无关的设备抽象层与统一的主机控制器接口（Trait），实现了对不同版本USB主机控制器的兼容。此外，结合Rust的异步编程模型，提出了一种基于Future的无栈协程调度机制，支持单线程环境下的异步任务主动让出与轮询切换，降低了多线程资源竞争风险。

第三章，硬件与仿真环境。本章构建了跨平台测试验证体系，在Arm架构（飞腾派硬件平台与Qemu仿真）和RISC-V架构（Qemu仿真）上完成系统部署。设计了多种测试场景，包括USB设备热插拔、高并发数据传输以及异常状态恢复，并通过USB协议分析仪对数据传输的完整性与实时性进行验证，确保系统在不同硬件环境下的稳定运行。

第四章，软件程序设计与实现。本章基于Rust语言实现了驱动框架的核心模块，包括：

1. 主机驱动层：开发了XHCI控制器驱动模块，支持事件环、控制环与传输环的动态管理；
2. USB驱动层：设计了可热插拔的驱动模块管理系统，通过统一的行为规范（Trait）实现驱动实例的动态加载与卸载；
3. 描述符解析系统：自主开发了支持动态解析的USB描述符树处理模块，兼容厂商自定义描述符与复杂拓扑结构；
4. 事件总线机制：构建了基于订阅-发布模式的事件系统，实现驱动层与外部组件的高效交互。

第五章，系统测试。本章对USB系统的各模块进行了详细的单元测试，并逐步剖析了代码的主要逻辑结构，其中包括：

1. USB描述符解析器的单元测试：对所编写的USB描述符解析器模块在描述符树的多个层次上进行了测试，以验证其解析结果的正确性。
2. USB驱动子系统的软硬件协同测试：基于第三章所引入的跨平台测试验证体系，在实际环境与虚拟环境下展开了对USB驱动子系统的功能测试，验证了主机驱动层在实际硬件与虚拟硬件上的兼容性。并编写了实验性的HID设备驱动以验证热拔插驱动模块的正确性及对应接口设计的合理性。
3. USB驱动子系统在进一步测试过程中所暴露出的问题与解决方案：对在测试过程中所发现的问题进行了全面分析，找出了问题发生的原因，并通过所提出的解决方案修复了问题。

# 跨平台USB主机端驱动子系统的整体设计方案及原理概述

## 系统整体方案设计

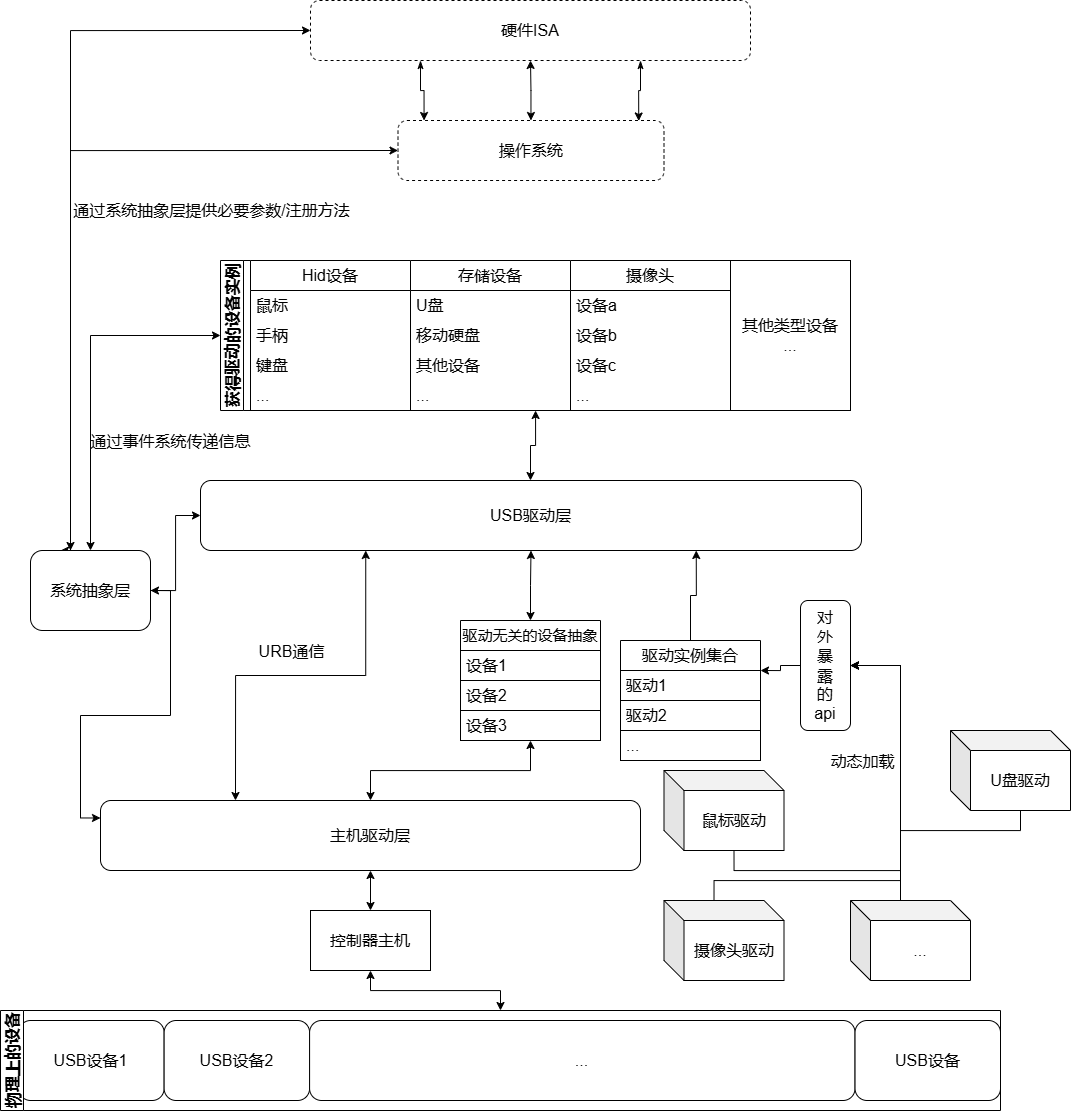
如图2.1所示，该系统架构图揭示了所设计的系统由三个主要层次构成：主机驱动层、USB驱动层以及系统抽象层。在主机驱动层与USB驱动层之间，存在一个驱动无关的设备抽象层，该层内存储了所有特定USB设备的基本信息，包括共享的USB描述符、请求缓冲区以及控制传输的状态锁。这些信息对于USB驱动层中的各个具体驱动实例而言是通用的，它们通过系统抽象层所提供的事件总线进行外部交互。主机驱动层与USB驱动层之间以异步的方式运行，确保了系统的高效性和响应性。在USB驱动层内部，每个具体的驱动实例都持有一份对驱动无关设备的引用，并将发起的请求组织成USB请求块（URB）的形式。主机驱动层负责不断的从各个设备的请求缓冲区中获取这些USB请求块，并将它们批量的执行。采用USB请求块来传递请求，而不是采用更为直接的方法调用，是为了避免在多层调用中频繁加锁，从而减少访问竞争，并确保任务响应的公平性。

图 2.1 系统总体设计方案

## 下文所涉及概念与名词解释

1. USB描述符，即一组用于描述USB硬件设备信息的报文段，其拓扑结构如图2.2所示，其中主要部分包括：
   1. USB设备描述符，此描述符内主要包括该设备的制造厂商，设备型号，USB协议物理层相关参数，及设备的不同配置模式数量。
   2. USB配置描述符，是USB设备描述符的下级节点，不同的配置标志着设备的不同工作模式，一个配置下面会一组不同的USB功能，即一个设备可能同时有多个功能。
   3. USB接口描述符，是USB配置描述符的下级节点，每个接口描述符都是一个独项工作的USB功能(USB Function)，包含着描述功能本身的三元组。每个USB功能都可能存在不同的配置，因此其还额外的包含了表示当前处于何种配置模式的二元组 (Interface Number, Alternative Interface Numver)。 其中，前者为每个配置下全局唯一的接口编号，后者为每个接口的不同工作模式的编号。
   4. USB端点描述符，是USB接口描述符的下级节点，每个USB功能都需要占用USB设备上的USB端点，及物理信道。该描述符包含所用端点的编号，类型，数据包大小等物理信息。
2. USB端点，即设备上的物理端点，每个端点都是独立的通信信道。端点有四种类型，分别对应一种特定的数据传输模式，适应于不同的任务。

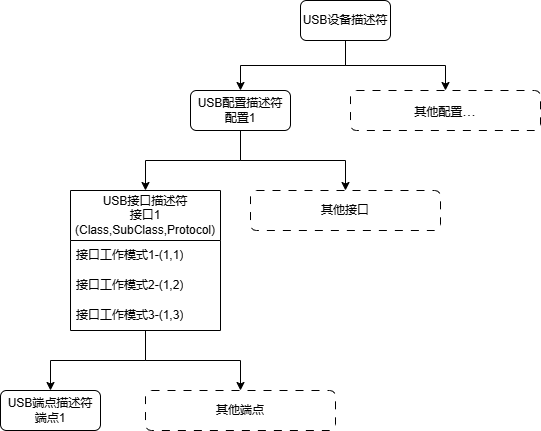


图 2.2 USB描述符拓扑结构

1. Future，在Rust异步编程中，Future是指一组被封装为状态机的异步代码块。Future作为状态机，有三种状态：
   1. Pending，该状态表明Future内的异步代码块正在执行阻塞IO操作。
   2. Waiting，该状态表面Future内的异步代码块已经完成了先前的阻塞IO操作，正在等待被再次调度以继续执行。
   3. Ready，该状态表面Future的任务已经彻底完成，此时Future指向了其内部异步代码块的最终返回值，等待被调用者回收。

Future引用一片内存区域，Future内的异步代码块正是对该片内存区域进行操 作的逻辑，而Future本身并非存放数据的结构体。

1. Cargo.toml配置文件，Rust拥有一套完善的项目管理机制，单个完整的Rust项目被称为Crate，大多数Rust开发者们使用 Cargo来管理他们的Rust项目。Cargo可以处理很多任务，比如构建代码、下载依赖库，以及编译这些库。Cargo.toml是Cargo所用的，描述一个Crate的各类属性的文件，其内包含该Crate所引用的库，该Crate的可选功能（feature），及作者等信息。
2. Rust异步编程，其思想类似于传统的多线程/多进程编程，其主要目的是解决IO所带来的高额延迟开销，在当前代码片段因IO而进入阻塞时，主动将资源让出，转而去执行其他与当前IO资源不直接相关的可执行代码段，大大提升了编程灵活性。Rust的异步编程采用的是无栈协程的思想，其将异步代码段组织为可嵌套的Future，Future并不会主动执行自己，而是要由外部去执行。Future本身封装为状态机，如图2.3所示，当Future由于IO而进入阻塞时，便将自身的状态转为Pending，外部的调用者便会将该Future暂存起来，转而去执行其他Future，当时机来临时，回来执行可继续执行的Future。
   1. 调用者可以是一个封装好的运行时——类似于操作系统中的任务调度器，或者也可以通过良好的设计，使得外部调用者以阻塞的方式执行最上层的Future。

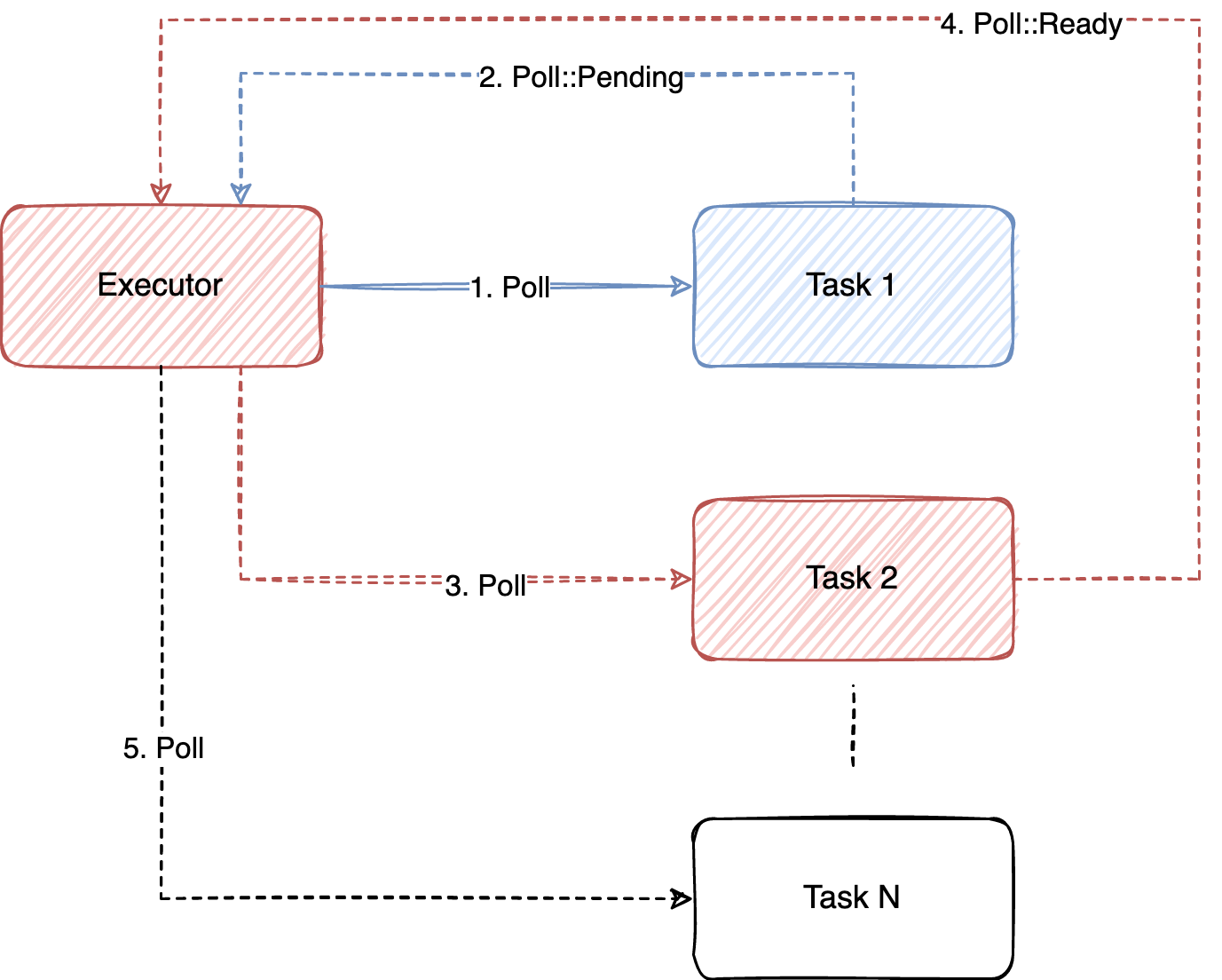


图 2.3 Rust的异步执行模型

## 系统各模块具体的技术研究

### 主机驱动层的技术细节

#### XHCI主机控制器

XHCI（eXtensible Host Controller Interface）是目前主流的USB HOST控制器标准，其作为第三代USB主机控制器被设计出来，并向下兼容USB1.0/USB2.0规范。其设计使得上层USB驱动的部分与XHCI控制器本身的驱动解耦，提高了灵活性。同时XHCI从逻辑上是事件驱动的，对高IO，高并发的情况更为友好。杨志荣等人[5]介绍了FPGA的xHCI协议软核IP设计及其应用，这为理解USB主机控制器接口提供了基础。C. Wang探讨了通过多线程编程分析硬件和软件并行性的局限性[6]，这对优化USB数据传输机制具有指导作用。张帅林在其博士论文中详细描述了基于嵌入式处理器的高性能中间件软件的设计与实现[7]，这对于提升USB驱动的整体性能至关重要。刘巍与黄智勇研究了DDS中间件的跨平台实现与优化技术[8]，为构建高效稳定的通信组件提供了参考。王硕与胡飞的工作则集中在基于数据发布订阅服务的嵌入式通信中间件设计上[9]，有助于探索事件驱动模型在USB驱动中的应用。

XHCI控制器设计的核心理念就是事件驱动与状态机。其工作是围绕三种环形缓冲区与本身的状态展开的，即事件环，控制环与传输环。在这三种环中，传输的信息被封装为一个个的TRB(Transfer Request Block)，其工作逻辑可简要的概括为：

1. 控制环内容纳改变主机控制器本身状态的TRB请求。
2. 传输环容纳主机端所发起的USB传输请求，每个设备的每个端点都有一个传输环。
3. 当向环内填充好了请求之后，置位门铃寄存器（Doorbell Register）以通知主机控制器开始处理这一轮请求。
4. 当某个请求完成时，主机向事件环填充一个标志着请求处理结果的TRB，如果配置了中断寄存器且请求时要求以中断的形式进行通知，则通过中断唤醒对应的事件处理程序，否则，由程序负责轮询并处理事件环中的新事件。

#### 驱动无关的设备实例

如前文所述，驱动无关的设备实例位于USB驱动层与主机驱动层之间。对于USB驱动层而言，USB主机所做的工作是透明的。所有请求均被导入设备实例的缓冲区。在本系统架构中，每个设备实例均维护一个环形缓冲区。设备实例在此扮演生产者（Producer）的角色，而主机驱动层则承担消费者（Consumer）的角色。主机驱动层定期的从各个设备实例中获取设备请求，并集中处理。设备实例也保存USB设备的描述符信息，各个USB驱动正是基于这些描述符信息开发的，并且每个驱动仅引用描述符中与自己功能相关的接口部分，这些接口部分彼此之间独立，互不干扰。这种设计确保了即使有若干特定的驱动模块发生崩溃，也不会对整个USB子系统的稳定运行造成影响。之所以仅保存一份描述符树，其原因在于描述符树往往会很大，复制会产生很多额外的开销，并且描述符树本身不会被变动，在逻辑上属于常量。

#### 需要不断发起请求的特殊事务

在USB（通用串行总线）技术的应用范畴内，众多上层功能协议常常表现出一些相同的需求。以HID协议为例，无论是何种HID设备，其总是频繁的重复发起相同的报文请求，并总是会收到大量的空回复。HID协议使用的是中断传输（Interrupt Transfer）这一传输层所规定的传输类型，这是一种一次只传输一份数据的传输类型，其特点在于，中断传输并不要求立即返回响应，而是当外设发生了对应的事件后，再返回报文。这种类似于中断的工作模式正是中断传输这一名称的由来。然而，在实际应用中，外设在没有事件，或事件结束后的第一次请求时，通常会选择直接返回一条空报文。而HID驱动在接收到报文后会再度发起请求，这就导致了大量无意义的带宽被浪费。用户通常对HID设备的响应速度有一定的期望和要求，因此这类损耗是难以接受且不必要的。类似于这一类的情况还有很多，其特征都如同HID一样，主要的开销来自于反复的重复请求。也就是说，需要让主机驱动层承担起一部分责任，由主机驱动层来不断的发起请求，并且负责过滤掉那些无意义的消息。实际上，主机驱动层会维护一个请求完成后的收尾工作集合，每个请求块都携带有一个唯一的编号，与集合中的收尾工作一一对应。这样的设计模式带来的好处在于，其统一了各类请求的处理流程。对于那些需要不断发起请求的特殊事务，主机驱动层同样会按照这种方式进行管理。

#### 主机驱动层的异步运作模式

从宏观视角审视主机驱动层，其功能可抽象化为图2.4所描绘的模型。在此模型中，主机驱动层负责异步的执行请求的发送以及收尾工作的处理。收尾工作的处理策略因USB子系统驱动框架所处的工作环境差异而异。当请求完成时，若已配置中断机制，则中断处理函数将被激活以执行处理。若未配置中断，则系统将通过轮询机制持续检查请求的完成状态。根据实时性要求的不同，系统将采取定时轮询或主动让出轮询的策略进行资源的合理分配，以确保系统的高效运作。

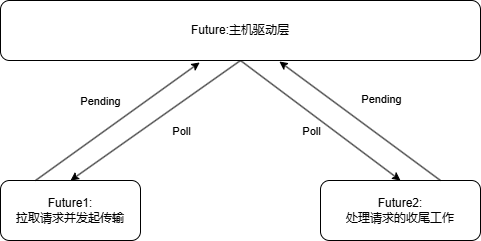


图 2.4 主机驱动层的工作模型

### USB驱动层的技术细节

#### 可热拔插的USB驱动模块

在嵌入式系统领域，鉴于资源的有限性和特定应用需求，通常不具备完整的文件系统。因此，在此类环境中，传统的基于文件系统的动态链接库加载机制并不适宜。实际上，为了满足嵌入式环境的特殊要求，驱动子系统的设计必须保证其始终在内存中常驻，并由操作系统内核直接进行管理。由此，依赖文件系统来加载和管理驱动模块的需求得以免除。核心在于，必须构建一套机制，使得系统能够在运行时动态的添加或移除USB驱动模块。换言之，应当明确一套USB驱动模块必须遵循的行为规范（Trait），确保所有遵循该规范的驱动模块在内存中的实例均具有统一的内存布局。这样，操作系统内核便能够无缝的管理这些驱动模块，而无需关注它们的加载时机及方式。

#### USB驱动模块的工作模式

USB驱动模块的工作过程可以被划分为两个核心组成部分：首先是USB驱动模块实例的创建，其次是USB功能实例的构建。USB驱动模块实例本身并不直接执行任何实际的驱动代码，它的主要职责和功能在于对USB描述符进行筛选和分析，以确定目标设备是否与该驱动模块相兼容。一旦通过了兼容性检查，确认目标设备与驱动模块匹配，USB驱动模块实例就会承担起一个重要的任务，那就是负责创建和初始化相应的USB功能实例。这个实例化过程是USB驱动模块能够正常工作和与设备通信的关键步骤，确保了驱动模块能够根据设备的特定需求来调整其行为，从而实现高效的数据传输和设备管理。

#### USB功能实例的工作模式

USB功能实例由两个核心组成部分构成：USB功能实例结构体以及对应的封装为Future的运行逻辑。USB功能实例结构体的布局和设计会根据不同的USB驱动而有所差异。在USB驱动层，它会负责拉取所有的USB功能实例结构体中定义的逻辑Future，并对这些Future进行执行。这一过程在图2.5中得到了清晰的展示，图中详细描绘了USB功能实例结构体与Future运行逻辑之间的关系以及USB驱动层如何处理这些结构体。

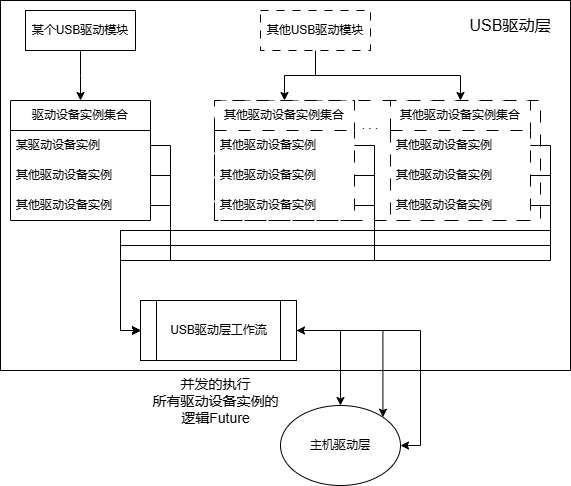


图 2.5 USB驱动层工作流

#### 事件系统

如图2.6所示，事件系统架构的设计允许USB驱动子系统通过事件总线与外部环境进行交互。事件本身携带有上下文信息，这些信息对于事件处理函数的执行至关重要。以USB移动存储设备的创建为例，当一个新的USB Block Device驱动实例被创建时，系统会触发一个名为USB\_NEW\_BLOCK\_DEVICE\_EVENT的事件。该事件中包含了丰富的上下文信息，包括对新创建的驱动实例的引用。操作系统在接收到此事件后，能够利用上下文信息中的驱动设备实例引用，执行挂载操作，将该驱动设备实例挂载至文件系统，从而实现对USB移动存储设备的识别与管理。

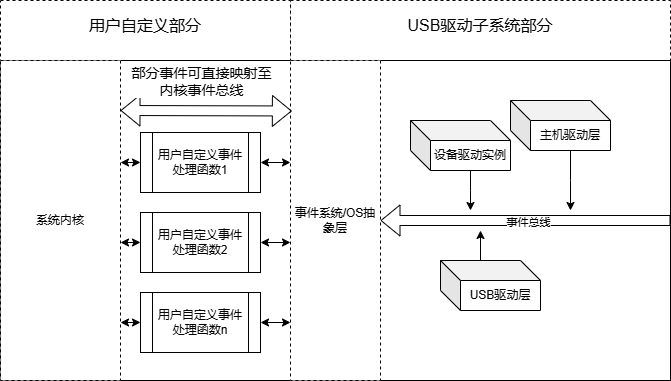


图 2.6 事件系统架构

## 本章小结

本章提出了分层架构的跨平台USB驱动子系统设计方案，包含主机驱动层、USB驱动层与系统抽象层。通过定义驱动无关的设备抽象层与统一的主机控制器接口（Trait），实现了对不同版本USB主机控制器的兼容支持。系统采用Rust语言的异步编程模型，结合Future无栈协程调度机制，支持单线程环境下的任务主动让出与轮询切换，有效降低多线程资源竞争风险。核心模块设计中，主机驱动层通过事件环、控制环与传输环的协同管理实现高效请求处理；USB驱动层则通过热拔插模块化设计，确保驱动实例的动态加载与安全隔离。此外，事件总线机制的设计强化了系统与外部组件的交互能力，为即插即用功能提供了技术保障。

# 硬件与仿真环境

## 开发所用平台环境

本研究旨在确保系统的跨平台及跨架构兼容性，因此必须在多种硬件架构和操作系统上进行广泛的测试验证。目前，该系统已在基于ARM架构和RISC-V架构的平台上完成了测试。

### 对于Arm架构的测试与验证

对于Arm架构，本系统主要在飞腾派上进行测试，辅以Qemu进行虚拟设备仿真，飞腾派由中电港萤火工场与飞腾公司合作推出，采用飞腾自主研发的高能效异构多核处理器技术，具有模块化、易上手、全开源、低成本的特点，大大降低了国产处理器的开发使用门槛，解决了国产处理器难以普及推广的问题。其具体技术规格如下表3.1所示

表 3.1 飞腾派参数列表

| 参数名 | 参数描述 |
| --- | --- |
| CPU | 飞腾四核处理器，兼容ARMv8指令集，2×FTC664@1.8GHz+2×FTC310@1.5GHz |
| 内存 | 64位DDR4，分2G版本和4G版本 |
| 硬盘 | 支持microSD 和EMMC启动 |
| 网络 | 2x千兆以太网（RJ45） |
| USB | 1×USB 3.0 host，3xUSB 2.0 host（键鼠） |
| PCIe | 1×Mini-PCle，支持AI、5G/4G、SATA转换等模组 |
| 蓝牙 | 蓝牙BT 4.2/BLE 4.2 |
| WiFi | 2.4G + 5G 双频WIFI |
| 4G/5G | 可通过miniPCIE扩展实现 |
| AI加速 | 可通过miniPCIE扩展实现 |
| 显示 | 1x HDMI，支持分辨率1920\*1080 |
| 视频解码 | 支持VPU硬解，格式：H.265\264 |
| 音频 | 3.5mm耳机口输出 |
| UART | 1x调试串口+2xMIO (可配置多功能IO口为UART模式) |
| 12C | 2+2xMIO (可配置多功能IO口为I2C模式) |
| 12S | 1路 |
| SPI | 2路 |
| CAN | 1路CANFD |
| GPIO | 最多29个 |
| SIM卡 | 支持1路SIM 卡 |
| SD卡 | 支持1路SD卡 |
| LED灯 | 电源指示灯和状态指示灯 |
| 供电要求 | 12V 3A直流电源 |
| 产品尺寸 | 119mmx93.1mm |
| 重量 | 72±2g |
| 工作 (运行) 温度 | 0~50°C |

Qemu仿真所使用的环境为qemu-system-aarch64，其余设置如表3.2所示

表 3.2 Arm架构的Qemu仿真参数

| 参数 | 参数值 |
| --- | --- |
| -m | 2G |
| -smp | 1 |
| -cpu | Cortex-a72 |
| -machine | virt,highmem=off |
| -device | qemu-xhci,id=xhci |
| -device | usb-mouse,bus=xhci.0 |

### 对于RISC-V架构的测试与验证

对于RISC-V架构，当前仅在Qemu上进行了仿真测试，环境为qemu-system-riscv64,其余设置如表3.3所示：

表 3.3 RISC-V架构的Qemu仿真参数

|  |  |
| --- | --- |
| 参数 | 参数值 |
| -m | 2G |
| -bios | default |
| -machine | virt,highmem=off |
| -device | qemu-xhci,id=xhci |
| -device | usb-mouse,bus=xhci.0 |

## 测试所用硬件设备

对于USB系统的测试，所使用的是USB总线分析仪进行协议栈测试，如图3.1，图3.2所示，此外，还使用实物设备进行功能测试。

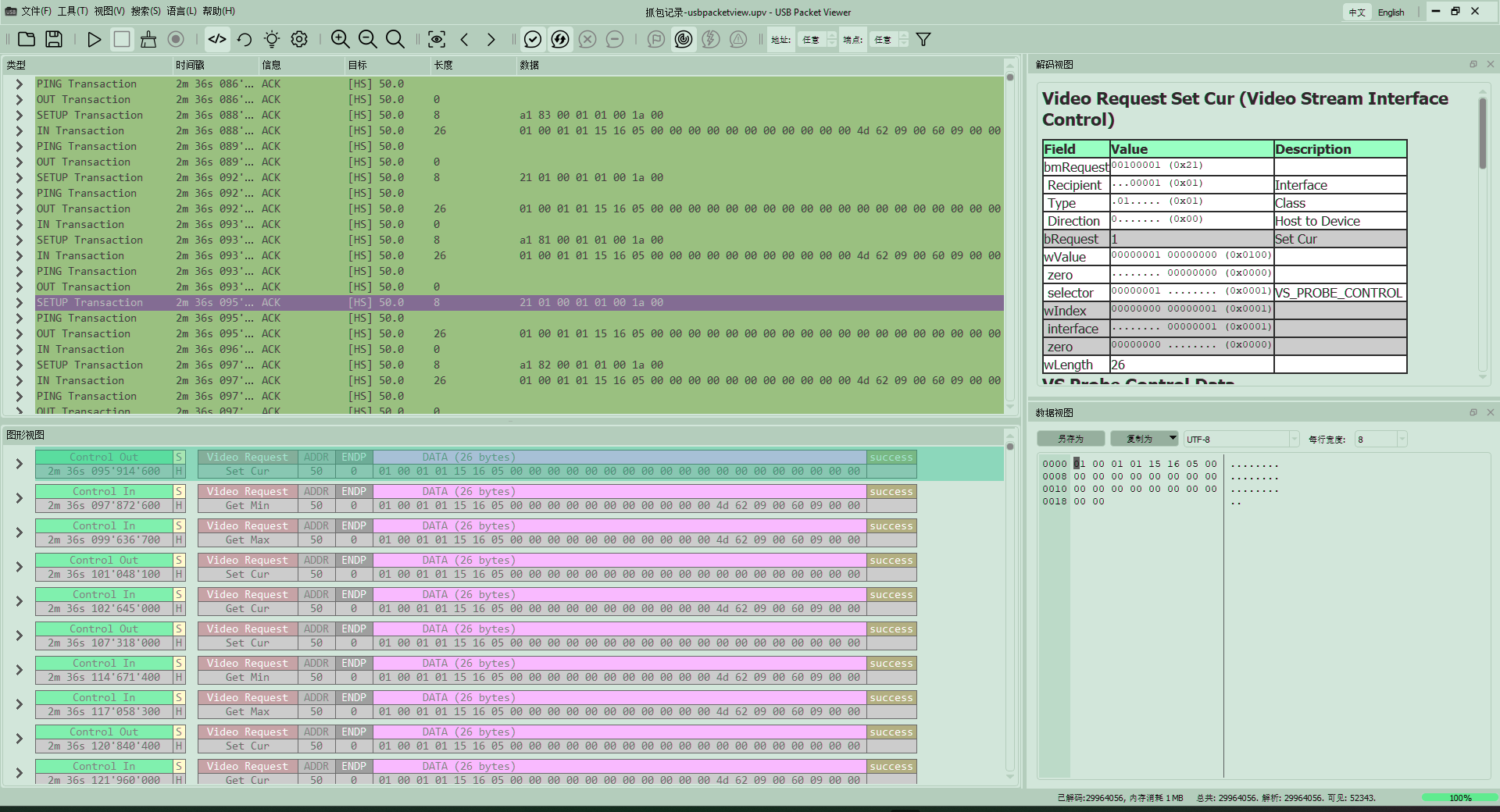


图 3.1 USB协议分析仪软件界面



图 3.2 USB总线分析仪实物

## 本章小结

本章构建了跨架构的硬件与仿真测试体系，以验证系统的兼容性与稳定性。测试环境覆盖Arm架构（飞腾派硬件平台与Qemu仿真）和RISC-V架构（Qemu仿真）以验证系统的可移植性。

# 软件程序设计与实现

## 程序设计语言及开发环境

### 编程语言：Rust程序设计语言

本系统选择通过Rust程序设计语言进行项目开发，Rust是一种现代的系统编程语言，由 Mozilla 基金会开发并维护。它的核心设计理念是“安全、并发、高效”，旨在解决传统系统编程语言（如 C++ 和 C）中存在的内存安全问题和并发编程挑战。Rust 通过所有权系统（Ownership System）、借用检查器（Borrow Checker）和生命周期（Lifetime）等独特机制，在编译时确保内存安全，从而避免了空指针、悬垂指针和数据竞争等问题。这些机制使得 Rust 在编写高并发、高可靠性和高性能的软件时表现出色。此外，Rust 的生态系统非常活跃，拥有丰富的库和工具（如 Cargo 包管理器），由于其编译器基于LLVM进行开发，这使得Rust能够轻松的编译到不同的硬件平台上。Rust 被广泛应用于系统编程、嵌入式开发、高性能计算等领域。

### 工具：Makefile

Makefile是一种用于自动化构建软件项目的工具，广泛应用于 C/C++、Rust 等编程语言的项目中。它是基于make工具的脚本文件，通过定义一系列规则和命令，帮助开发者自动化编译、构建、测试和部署等流程。在Rust项目中，虽Cargo是官方推荐的构建工具，但Makefile仍然在一些复杂场景中发挥重要作用。当需要自定义构建逻辑、管理多个构建目标或与其他工具链集成时，其能提供更高的灵活性和控制力。

### 集成开发环境：VSCode

Visual Studio Code（简称VSCode）是一款由微软开发的轻量级、开源且高度可定制的代码编辑器，广泛应用于多种编程语言和开发场景。它以其卓越的性能、丰富的插件生态和跨平台支持而著称，能够满足从个人开发者到大型团队的多样化需求。

#### VSCode插件：Rust Analyzer

Rust Analyzer是一个专为Rust编程语言设计的先进语言服务器协议 (LSP) 实现，它通过提供智能感知、代码补全、错误检测、重构建议和文档查找等功能，显著提升了开发者的编码效率和体验；基于Rust的强大类型系统和所有权模型，Rust Analyzer能够深入理解代码结构，从而实现精准的上下文感知和高效的代码分析；此外，它还支持与多种主流IDE和编辑器（如VS Code、IntelliJ IDEA和 Neovim）无缝集成，并通过持续优化性能和扩展插件生态系统，已成为Rust开发者不可或缺的工具之一。

## 模块化设计与项目依赖

基于功能性和可复用性的综合评估，USB子系统在开发过程中被划分为两个独立的程序库（Crate），具体包括负责USB描述符解析的usb-descriptor-decoder，以及作为USB子系统核心架构的axusb-host。axusb-host的Cargo.toml配置文件如图4.1所示。

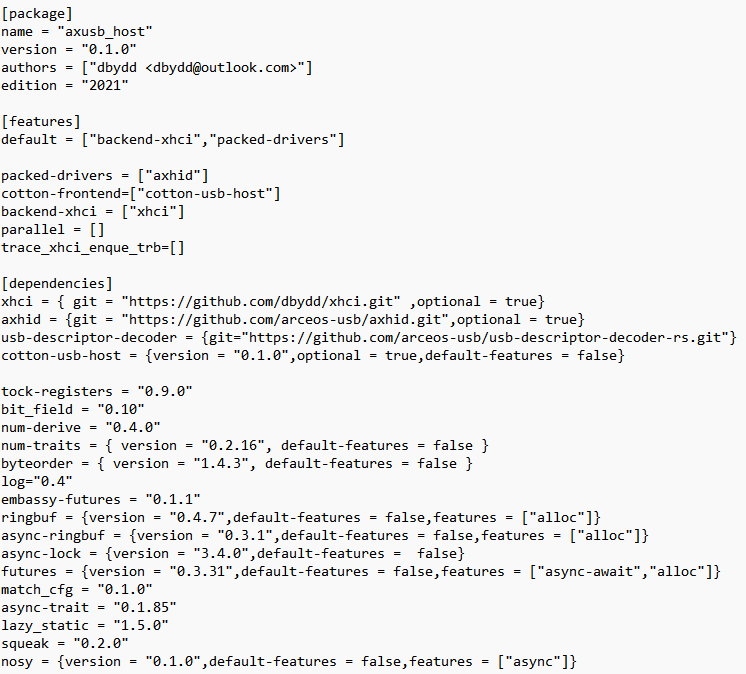


图 4.1 axusb-host Crate的Cargo.toml配置文件

注1） xhci Crate为修改版本，添加了诸多辅助性的函数，并修复了若干潜在的BUG

注2） axhid Crate为额外编写的为USB-HID设备提供报文解析的工具库

usb-descriptor-decoder的Cargo.toml配置文件如图4.2所示

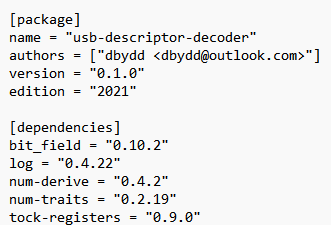


图 4.2 usb-descriptor-decoder Crate的Cargo.toml配置文件

## USB驱动子系统框架的实现

作为驱动子系统的主体框架，axusb-host的模块组织如图4.3所示：

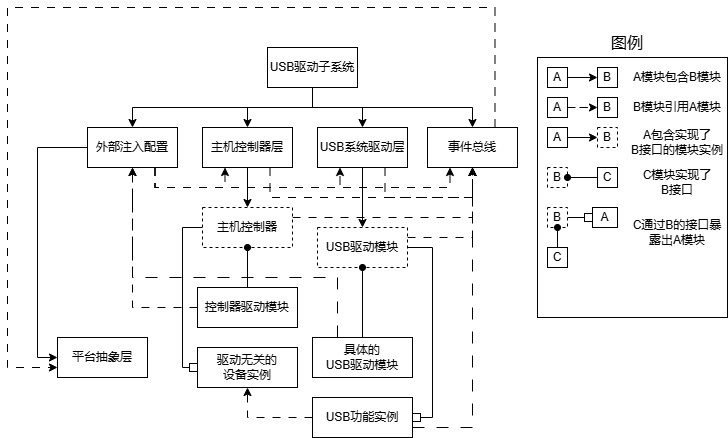


图 4.3 USB驱动子系统的模块组织

### USB驱动子系统模块

本模块作为子系统架构的最外层封装，通过事件总线机制实现对主机控制器层与USB系统驱动层交互的管理。其对应的结构体定义代码如下：

pub struct USBSystem<'a, O, const RING\_BUFFER\_SIZE: usize>

where

O: PlatformAbstractions + 'static,

'a: 'static,

{

config: Arc<USBSystemConfig<O, RING\_BUFFER\_SIZE>>,

controller: Box<dyn Controller<'a, O, RING\_BUFFER\_SIZE>>,

usb\_layer: USBLayer<'a, O, RING\_BUFFER\_SIZE>,

event\_bus: Arc<EventBus<'a, O, RING\_BUFFER\_SIZE>>,

}

### 主机控制器接口

如图4.3所示主机控制器层并不直接持有特定的主机控制器驱动，而是通过定义了一组通用的行为（Trait），而达到可以使用任何实现了该行为的主机控制器驱动的目的。这组行为的定义如下：

pub trait Controller<'a, O, const RING\_BUFFER\_SIZE: usize>: Send + Sync

where

O: PlatformAbstractions,

{

fn new(

config: Arc<USBSystemConfig<O, RING\_BUFFER\_SIZE>>,

event\_bus: Arc<EventBus<'a, O, RING\_BUFFER\_SIZE>>,

) -> Self

where

Self: Sized;

fn init(&self);

fn device\_accesses(&self) ->

&Vec<Arc<USBDevice<O, RING\_BUFFER\_SIZE>>>;

fn workaround(&'a self) -> BoxFuture<'a, ()>;

}

### USB系统驱动层模块

不同于主机控制器层，USB系统驱动层需要管理诸多USB系统驱动模块，因此需要封装一层，USB系统驱动层的结构体定义如下：

pub struct USBLayer<'a, O, const RING\_BUFFER\_SIZE: usize>

where

O: PlatformAbstractions,

{

config: Arc<USBSystemConfig<O, RING\_BUFFER\_SIZE>>,

eventbus: Arc<EventBus<'a, O, RING\_BUFFER\_SIZE>>,

pub driver\_modules: BTreeMap<

String,

Box<dyn USBSystemDriverModule<'a, O, RING\_BUFFER\_SIZE>>

>,

pub functional\_interfaces: RwLock<

BTreeMap<String,

Vec<Arc<RwLock<

dyn USBSystemDriverModuleInstanceFunctionalInterface< 'a,O>>>

>

>,

>,

}

## USB描述符解析系统的实现

针对USB描述符解析系统，当前市场上尚未出现一套标准化的通用解析库，鉴于此，为此，需要自主开发一套解析程序。

### USB描述符的数据流

在逻辑上，USB描述符被组织成具有特定层次拓扑结构的树状结构，即所谓的描述符树。在实际的数据传输过程中，这些描述符以特定顺序的字节流形式进行传递。字节流的起始部分是通过前序遍历描述符树得到的描述符序列，随后是厂商自定义的特殊描述符。每个USB描述符都包含两个定长字段，位于描述符的起始部位，分别用于描述该描述符片段的长度和类型，如图4.7所示。

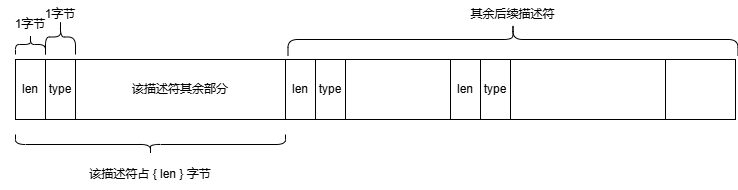


图 4.4 USB描述符字节流

特别需要指出的是，在USB接口描述符的后面，除了包含端点信息之外，通常还会附加一些特定于该USB接口描述符所对应的USB协议栈的特殊信息。在一些更为复杂的情形中，接口描述符可能会进行嵌套，从而形成一个更为庞大的USB功能实例。一个典型的例子就是UVC协议栈，即USB Video Class（USB视频类设备）。其扩展后的描述符树结构如图4.5所示。

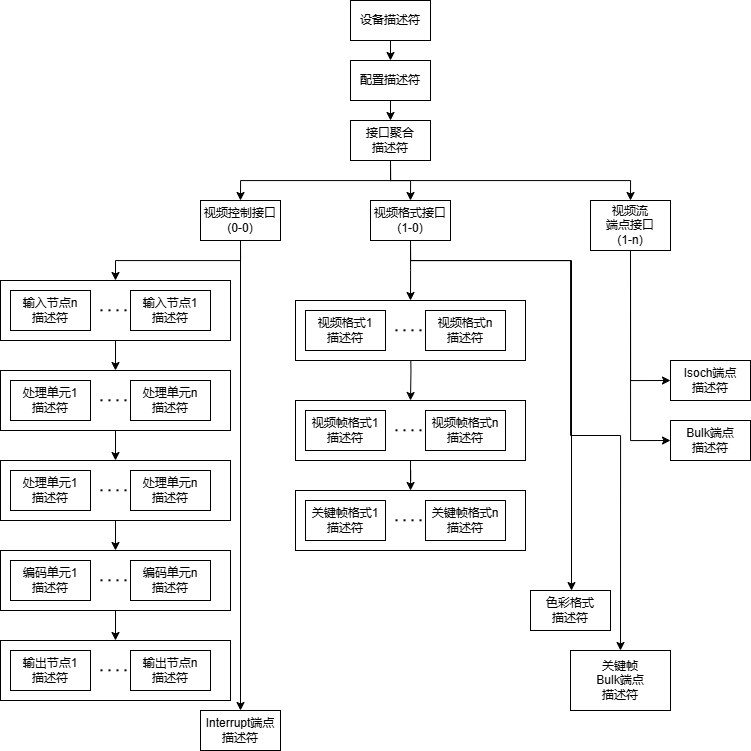


图 4.5 UVC协议栈的描述符拓扑

其中，接口聚合描述符负责将下层接口聚合为单个USB Function类。

### USB描述符解析系统的设计

鉴于USB描述符树在接口层面上的复杂性，以及不同厂商和协议栈之间私有描述符的类型编号可能存在冲突，这就意味着只有在成功解析出接口描述符之后，方能明确接下来将会遇到何种格式的描述符子树。因此，为了处理这些复杂性并确保系统的灵活性和可扩展性，有必要提出一套合理，高效且可扩展的解析流程。USB描述符解析系统的解析流程如图4.6所示：

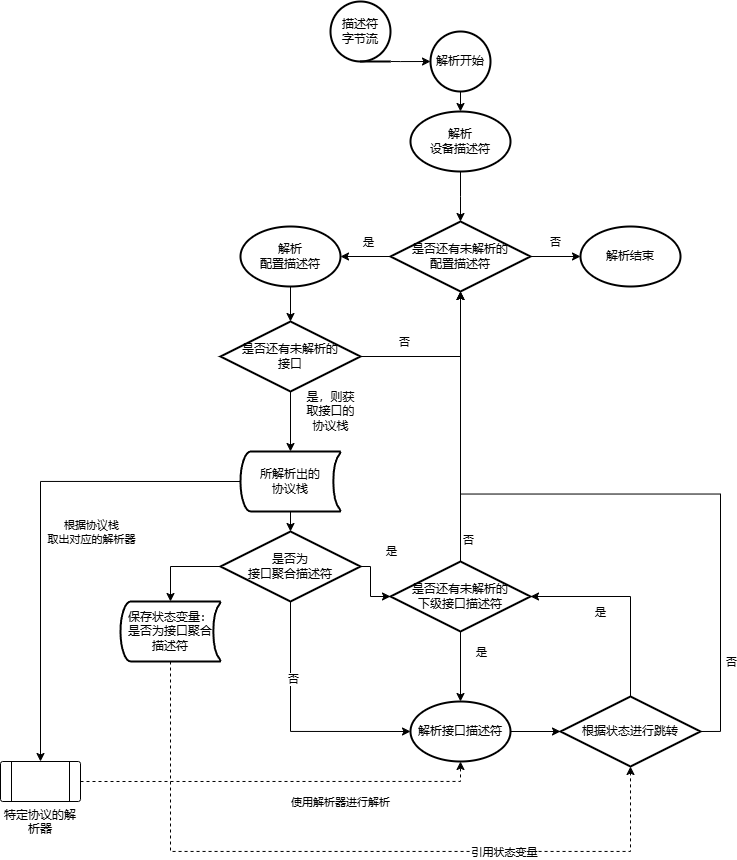


图 4.6 描述符解析系统的流程

## 本章小结

本章基于Rust语言实现了驱动子系统的核心模块，涵盖主机驱动层、USB驱动层、描述符解析系统与事件总线机制。通过模块化设计，系统被拆分为axusb-host与usb-descriptor-decoder两个程序库，前者负责框架主体，后者负责解析USB描述符树。主机驱动层通过抽象接口来实现不同版本规范主机的管理；USB驱动层采用热拔插模块化设计，依托统一行为规范（Trait）实现驱动实例的动态加载。

# 系统测试

## USB描述符解析器的软件测试

USB描述符解析器的单元测试使用Rust编程语言自身的测试框架，部分单元测试代码如下：

#[cfg(test)]

mod test {

#[test]

fn test\_parse\_config() {

let input = [

9u8, 2, 34, 0, 1, 1, 6, 160, 50, 9, 4, 0, 0, 1, 3, 1, 2, 0, 9, 33, 1, 0, 0, 1, 34, 52,

0, 7, 5, 129, 3, 4, 0, 7, 0, 0,

];

let descriptor\_decoder = DescriptorDecoder::new();

let (result, \_) = descriptor\_decoder.parse\_config(&input[..]).unwrap();

let formatted = format!("{:?}", result);

assert\_eq!(formatted, "TopologyConfigDesc { desc: Configuration { length: 9, ty: 2, total\_length: 34, num\_interfaces: 1, config\_val: 1, config\_string: 6, attributes: 160, max\_power: 50 }, functions: [Interface([USBInterface { interface: Interface { len: 9, descriptor\_type: 4, interface\_number: 0, alternate\_setting: 0, num\_endpoints: 1, interface\_class: 3, interface\_subclass: 1, interface\_protocol: 2, interface: 0 }, endpoints: [Endpoint { len: 7, descriptor\_type: 5, endpoint\_address: 129, attributes: 3, max\_packet\_size: 4, interval: 7, ssc: None }], flag: \"hid\", extra: [Hid { len: 9, descriptor\_type: 33, hid\_bcd: 1, country\_code: 0, num\_descriptions: 1, report\_descriptor\_type: 34, report\_descriptor\_len: 52 }] }])] }")

}

}

测试结果如下图5.1所示。

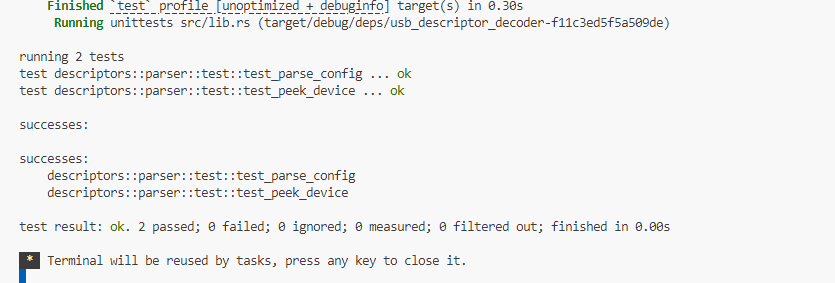


图 5.1 USB描述符解析器单元测试结果

## USB驱动子系统软硬件协同测试

USB驱动子系统的各模块完成对应的单元测试后，即可进行软硬件协同测试。软硬件协同测试所使用的硬件平台为飞腾派，其各参数如前文3.1.1章节中所示。测试环境如下：



图 5.2 软硬件协同测试

飞腾派使用U-Boot作为开机引导，通过串口与主机进行交互及下载操作系统内核文件，测试所用的操作系统为ArceOS,ArceOS是一个架构为Unikernel的操作系统，在其上进行驱动开发，可以在不更改所编写的代码的前提下进行多种环境下的测试（如裸机，开启虚拟内存，开启分页等环境状态下的测试）。

### USB驱动子系统的初始化流程

USB驱动子系统的初始化流程如图5.3所示:

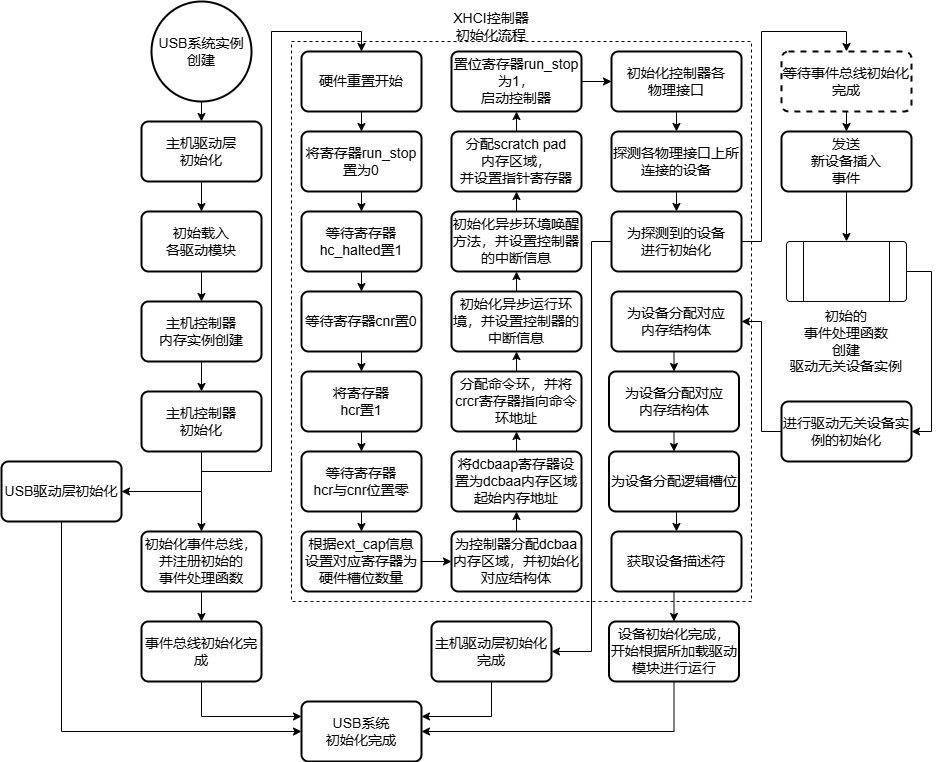


图 5.3 USB驱动子系统的初始化流程

其中，由于采用异步的编程范式，在主机控制器初始化后会同步的进行USB驱动层，事件总线及主机驱动层的初始化。由于飞腾派所使用的主机控制器为XHCI控制器，因此主机驱动层实际上是在运行XHCI控制器的初始化及工作流程。由于硬件工作的特殊性，不妨从是否成功获取到所插入设备的描述符来判断整个系统是否正常工作，进而判断该系统是否成功初始化，图5.4所展示的为串口中的部分输出结果。



图 5.4 所获取到的鼠标设备描述符树

### 基于驱动无关设备中各描述符所对应USB功能的驱动指派

在驱动无关设备被创建后，其经过初始化后将会获得所对应物理设备的描述符树。基于描述符树，驱动无关设备会向事件总线发送 NEW\_DEVICE\_PLUGIN事件以表示有新设备等待被分配驱动。当USB驱动层收到该事件后，将会取出事件上下文中驱动无关设备的引用，并根据其中不同的接口描述符所对应的USB功能进行驱动模块的指派，这部分代码如下：

pub fn new\_device\_initialized(

&self,

device: &Arc<USBDevice<O, RING\_BUFFER\_SIZE>>

) {

self.driver\_modules

.values()

.filter\_map(|module| {

module

.should\_active(&device, &self.config)

.map(|a| (a, module.name()))

})

.for\_each(|(function, name)| {

embassy\_futures::block\_on(self.functional\_interfaces.write())

.entry(name)

.or\_insert(Vec::new())

.push(function);

});

debug!("initialized new device!");

}

### 驱动模块是否正常工作的测试

对于当前测试的设备，即遵循USB-HID协议的鼠标，编写了一个实验性的HID驱动模块以验证整个USB子系统的工作状态。该HID驱动的逻辑如图5.5所示：

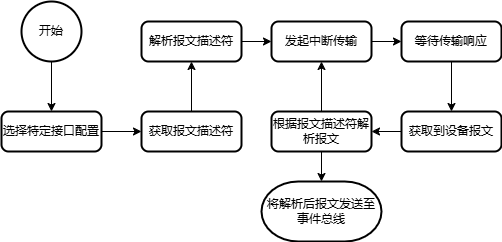


图 5.5 HID驱动的工作流程

对于HID驱动模块，测试方法为通过从事件总线读出设备报文来判断对应驱动是否正常工作，从事件总线读出的报文如下图5.6所示：

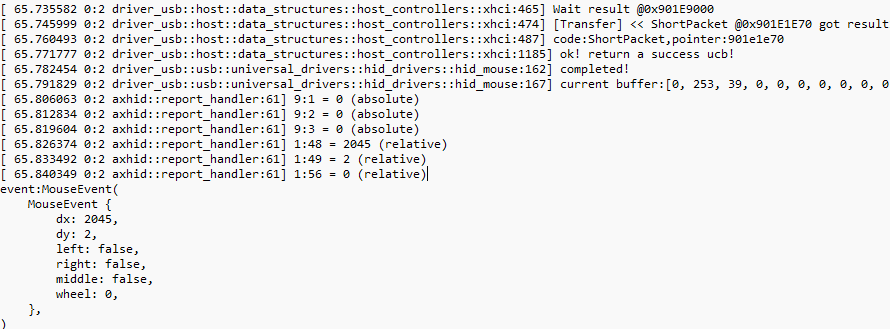


图 5.6 HID驱动的设备报文

可以看到，HID驱动成功获取到了设备的回报报文，并正确的解析了报文。图中，该鼠标事件描述了一次鼠标的平面滑动。该鼠标的x，y轴分别被施加了对应的移动速度。

## USB驱动子系统在进一步测试过程中所暴露出的问题与解决方案

为了测试USB驱动子系统在不同架构下是否能够正常工作，使用Qemu进行环境仿真测试，测试环境的参数如3.1中所列举的一致。测试过程中，发现了USB驱动子系统对于虚拟内存的适配存在问题：

对于现代计算机系统而言，对虚拟内存的支持已经成为了基本要求，更进一步的，虚拟内存在实践中往往是以内存分页的方式进行实现的。要进行内存的分页管理，对于硬件上有特殊的要求。即引入了一种特殊的硬件，称为内存管理单元（MMU）。其中，有一类特殊的MMU，其为DMA访问及设备间的IO提供了增强。使得各IO部件可以根据虚拟内存进行寻址，此类MMU称为IOMMU。对于现代的大多数计算机系统，甚至是嵌入式的系统而言，IOMMU都会存在。在当前背景下，这就使得向XHCI的特定几个指针寄存器写入内存地址时，可以直接写入虚拟地址。但是在QEMU仿真环境中，默认不使用此类MMU。因此在测试时，往往会发生驱动程序在特定步骤卡死的情况。通过对Qemu进行断点调试，并追踪到最底层的代码得知，DMA寻址在访问虚拟内存块时发生了失败的情况，如图5.7所示。

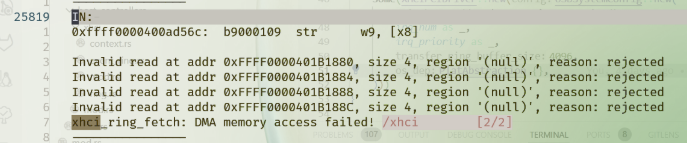


图 5.7 Qemu下DMA访问失败

这暴露出了目前的USB子系统尚未针对此类情况做适配。于是对此的修复方案为，在USB系统初始化时传入的平台配置中添加了针对是否使用了IOMMU相关功能的开关，并进一步的为平台抽象层添加了虚拟地址与物理地址之间相互转换的Trait方法，相关代码如下：

pub trait PlatformAbstractions: Clone + Send + Sync + Sized {

type VirtAddr:

From<Self::PhysAddr> + From<usize> + Into<usize> + Clone + Send + Sync;

type PhysAddr:

From<Self::VirtAddr> + From<usize> + Into<usize> + Clone + Send + Sync;

type DMA: Allocator + Send + Sync + Clone;

const PAGE\_SIZE: usize;

const RING\_BUFFER\_SIZE: usize;

const WORD: SystemWordWide;

fn dma\_alloc(&self) -> Self::DMA;

}

此处，为VirtAddr类型与PhysAddr类型添加了约束，要求彼此之间可以通过From Trait进行互相的转换。VirtAddr与PhysAddr是虚拟地址与物理地址的封装类型，其并未改变对应地址在内存上的布局（即内存地址以64位无符号整数表示），并为其附加了额外的编译期属性。用户可根据所用平台的实际情况，为虚拟与物理地址实现不同的转换方法。在主机驱动层中，所有对寄存器的内存读写都会使用PhysAddr这个物理地址包装类。修复后的运行结果如图5.8所示。图中能看到qemu的debug输出，显示虚拟XHCI控制器设备正确的处理了对特定内存地址的读写

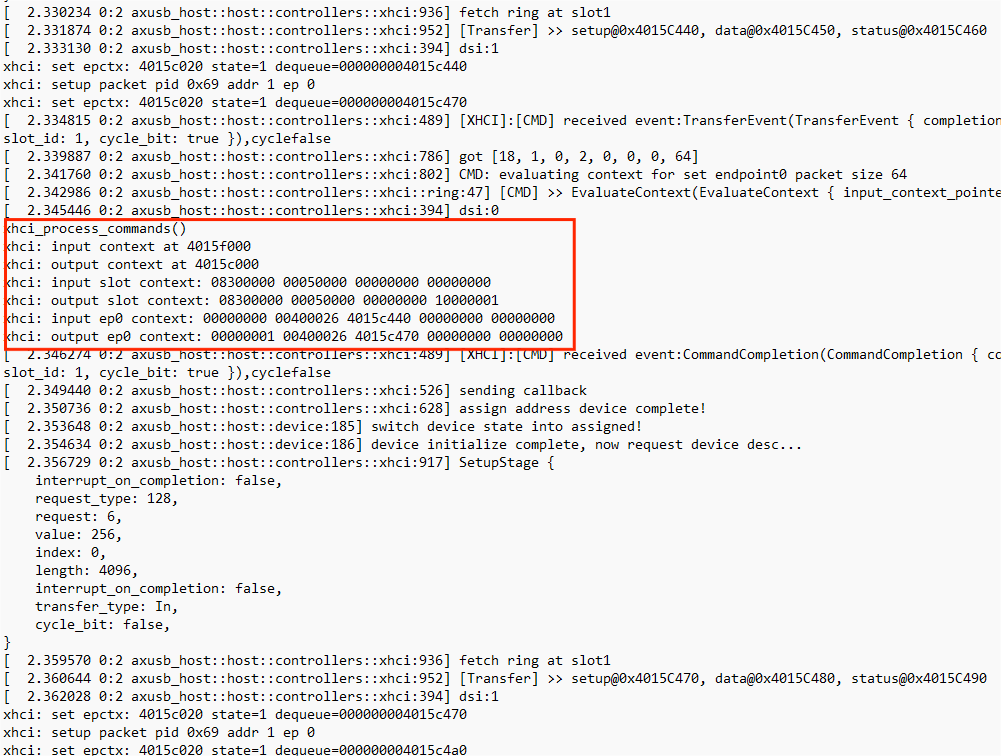


图 5.8 修复后的qemu运行结果

结 论

随着国产操作系统的快速发展与多样化硬件架构的广泛应用[10]，USB驱动程序的跨平台兼容性已成为制约技术自主可控的关键问题。传统USB驱动开发高度依赖特定操作系统框架，导致迁移成本高昂且难以适应新兴硬件生态。本文针对这一挑战，提出并实现了一种基于Rust的跨平台USB主机驱动子系统，通过理论分析与实验验证，探索了通用性、安全性与高效性并重的驱动开发新范式。

本研究首先从USB主机控制器驱动层与功能驱动层的解耦设计入手，构建了一套分层的抽象架构。通过引入驱动无关的设备实例与统一的主机控制器接口（Trait），实现了对不同版本USB主机控制器（如XHCI）的兼容支持。在此基础上，结合Rust语言的异步编程模型，设计了一种基于Future的无栈协程调度机制，支持单线程环境下异步任务的主动让出与轮询切换。这一设计不仅降低了多线程环境下的资源竞争风险，还通过零成本抽象保障了系统性能。此外，自主研发的USB描述符解析模块采用动态解析策略，有效应对了复杂拓扑结构与厂商自定义描述符的兼容性问题，为上层驱动的即插即用提供了可靠基础。

实验验证方面，本系统在Arm架构（飞腾派，Qemu仿真）与RISC-V架构（Qemu仿真）平台上完成了功能与性能测试。测试结果表明，系统能够稳定识别并驱动多种USB设备（如HID设备、存储设备），并通过事件总线机制实现与外部组件的信息交互。通过协议分析仪验证了USB传输的完整性与低延迟特性，而异步任务调度模型在单线程环境下展现出与多线程相当的吞吐效率。

本文的主要贡献在于：其一，提出了一种面向跨平台场景的USB驱动框架设计范式，通过分层抽象与异步模型解决了操作系统异构性带来的兼容难题；其二，首次将Rust语言系统级安全特性深度融入USB驱动开发，利用所有权机制与借用检查器规避了内存泄漏与数据竞争风险；其三，设计的事件总线与热拔插驱动管理系统，为国产操作系统的外设生态建设提供了可扩展的技术支撑。

尽管当前系统已实现预期目标，未来仍可在以下方向进一步优化：一是扩展对USB4与Thunderbolt协议的支持，适应更高带宽设备的接入需求；二是探索Rust与现有C/C++驱动生态的互操作性，降低迁移成本；三是结合形式化验证工具，对关键模块（如异步调度器）进行数学证明，提升系统可靠性。本研究为构建自主可控的跨平台驱动生态提供了理论参考与实践基础，对推动国产操作系统技术演进具有重要意义。

参 考 文 献

[1] H. Peng, M. Payer, USBFuzz: A Framework for Fuzzing USB Drivers by Device Emulation, in: 29th USENIX Security Symposium (USENIX Security 20), USENIX Association, 2020: pp. 2559–2575.

[2] J. Huang, Y. Wang, Research and Application of High-Speed and Adjustable Synchronous Data Transfer System Based on USB 3.0 Peripheral Controller, Journal of Circuits, Systems and Computers. 30 (2021) 2150118.

[3] N. Borgsmüller, The Rust programming language for embedded software development, phdthesis, Technische Hochschule Ingolstadt, 2021.

[4] 韩乃平,李蕾.国产操作系统生态体系建设现状分析[J].信息安全研究,2020,6(10):887-891.

[5] 杨志荣,胡茂海,董理,李守业.FPGA的xHCI协议软核IP设计及应用[J].单片机与嵌入式系统应用,2022,22(12):7-11.

[6] C. Wang, Analyzing the limitations of parallelism in hardware and software through threaded programming, in: Proceedings of 8th International Conference on Computer-Aided Design, Manufacturing, Modeling and Simulation (CDMMS 2023), 2023: pp. 24–29.

[7] 张帅林. 基于嵌入式处理器的高性能中间件软件设计与实现[D].北京理工大学,2021.DOI:10.26948/d.cnki.gbjlu.2021.001581.

[8] 刘巍,黄智勇.国产化实时通信中间件DDS的跨平台实现和优化技术[J].现代雷达,2019,41(07):39-43.DOI:10.16592/j.cnki.1004-7859.2019.07.007.

[9] 王硕,胡飞. 基于数据发布订阅服务的嵌入式通信中间件设计[C]//中国航空学会.第九届中国航空学会青年科技论坛论文集.中航出版传媒有限责任公司（China Aviation Publishing & Media CO.,2020:8.DOI:10.26914/c.cnkihy.2020.052267.

[10] 李荣.2022年物联网应用趋势[J].计算机与网络,2021,47(21):42-44.

致 谢

一同工作的同志对本研究所做的贡献应在论文中做明确的说明并表示谢意。

感想丁菊老师，丁菊老师作为我的毕业设计指导老师，在我设计并完善该系统的过程中提供了很多的帮助。并为我的论文的撰写提供了非常多宝贵的建议与指导。

感想rCore-OS社区的诸多同志，USB驱动子系统最初编写的目的是作为ArceOS的USB驱动子系统模块而存在的，在我编写该模块的过程中，我逐渐意识到了该模块作为跨平台/跨操作系统的通用库存在的可行性与必要性，于是便有了如今的成果。在我设计并完善该库的过程中，为我提供了很多的帮助。

感谢我在项目中所使用的第三方库的诸多作者。他/她们为许多先进的理论与设计模式提供了良好且安全高效的封装，使得我可以以更安全高效的方式来进行该系统的编写。

感谢在我开发该系统的过程中提出了诸多宝贵建议，批评，指导及其他各种形式帮助的各位老师，同学！