项目申请书

项目 ID：255660173

基于 RISC-V 的 Linux 操作系统原生无人机飞控开发

申请学生：巴晨翰
申请日期：2025 年 6 月 7 日
联系邮箱：hanchen7219@163.com

目录：

1. 项目介绍
   (1) 项目背景
   (2) 项目主要目标
2. 项目方案
   (1) 驱动适配 RISC-V 平台
   (2) 驱动适配 OOB 上下文，支持 EVL 线程使用驱动
   (3) 驱动测试
3. 时间安排

1. 项目介绍

1.1 项目背景

随着低空经济的发展，无人机的需求越来越负责。运行在 RTOS 上的飞控难以满足复杂的需求，所以 Linux 原生的飞控逐渐被大家认可，呈现出飞控主控二合一的趋势，具有开发简单，兼容性好，生态丰富的特点。但是 Linux 实时性可能无法满足飞控的要求，RROS 作为 Linux 硬实时的解决方案，在这个场景中具备明显的优势。

RROS 是一个双内核实时操作系统，基于 Rust-For-Linux（RFL）重构了 Xenomai 的实时内核。具体来说，使用 RFL 提供的在 Linux 中使用 Rust 编写驱动的框架，与 Xenomai 社区的 dovetail 中断虚拟化接口，基于 Rust 语言实现了一个实时内核。实时内核将 Linux 内核作为 idle 任务进行调度，优先调度实时内核的任务，同时 RROS 兼容了实时核心的用户态接口库 libevl。

以往无人机飞控和主控都以 ARM 生态为主，本题目想要基于国产化的 RISC-V 开发板控制无人机，适配 RustPilot 飞控。

1.2 项目主要目标

本项目的主要目标是，在 RISCV 开发板（Visionfive2）上，实现针对无人机的实时驱动，包括的驱动内容可能有：DMA，SPI，IMU。

首先，参考现有基于 ARM 板卡的驱动，以此为基础进行修改，充分了解架构与内存访问差异与硬件平台差异，将驱动移植到国产 RISC-V 结构的开发板；或者针对现有的 ARM 板卡，使用板卡供应商提供的软件内核及驱动。

其次，对移植后的驱动按照 EVL 的要求进行改造，将其修改为 OOB-safe 版本，能安全地在 OOB 上下文中运行，被 EVL 线程进行使用。

2. 项目方案

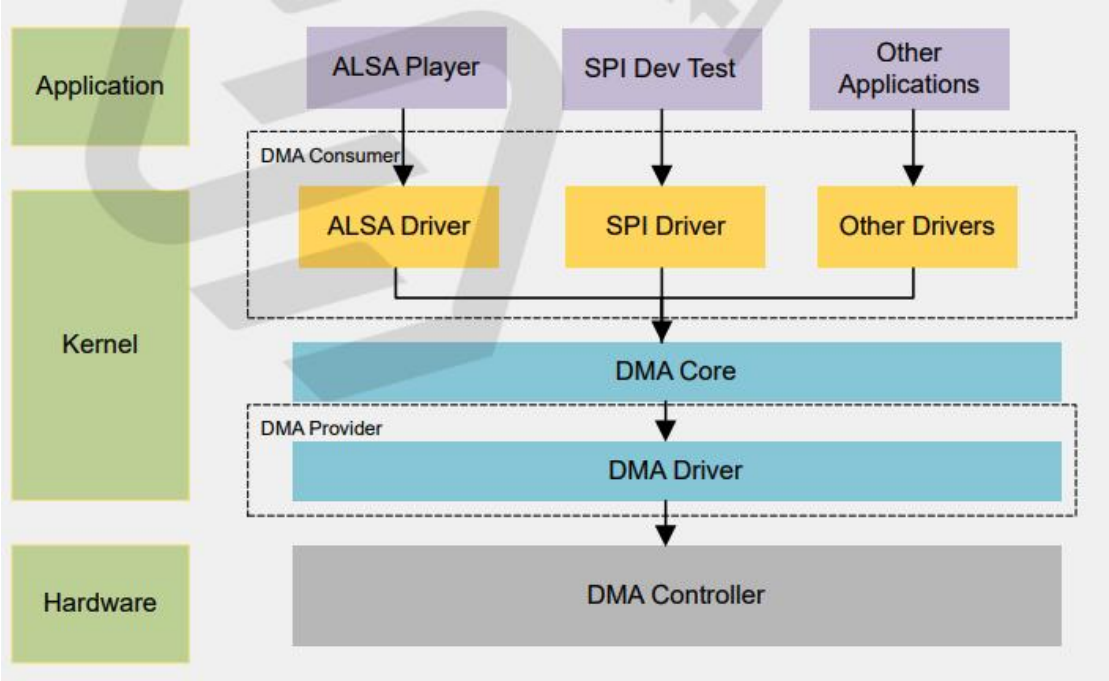
2.1 驱动适配 RISC-V 平台

这里以 DMA 驱动举例, 在 VisionFive 2 使用的 JH7110 中, 查看设备树中, 得知 JH7110 已经自带 AXI-DMA 控制器，StarFive 已经提供了支持它的 DMA 驱动，对于 DMA 这种及其依赖硬件 IP 的驱动，应该可以选择直接选择使用 StarFive 提供的 DMA 驱动作为基础。

类似地，VisionFive 2 板卡的 SPI 驱动同样提供了驱动框架。对于提供了实现的驱动，选择将已有驱动作为基础，进行第二步的驱动改造，使其支持 EVL 实时线程。VisionFive 2 板卡中 DMA 与 SPI 的设备树描述如下：

```
dma: dma-controller@16050000 {
        compatible = "starfive,jh7110-axi-dma";
        reg = <0x0 0x16050000 0x0 0x10000>;
        clocks = <&stgcrg JH7110_STGCLK_DMA1P_AXI>,
                 <&stgcrg JH7110_STGCLK_DMA1P_AHB>;
        clock-names = "core-clk", "cfgr-clk";
        resets = <&stgcrg JH7110_STGRST_DMA1P_AXI>,
                 <&stgcrg JH7110_STGRST_DMA1P_AHB>;
        interrupts = <73>;
        #dma-cells = <1>;
        dma-channels = <4>;
        snps,dma-masters = <1>;
        snps,data-width = <3>;
        snps,block-size = <65536 65536 65536 65536>;
        snps,priority = <0 1 2 3>;
        snps,axi-max-burst-len = <16>;
};
spi0: spi@10060000 {
        compatible = "arm,pl022", "arm,primecell";
        reg = <0x0 0x10060000 0x0 0x10000>;
        clocks = <&syscrg JH7110_SYSCLK_SPI0_APB>,
                 <&syscrg JH7110_SYSCLK_SPI0_APB>;
        clock-names = "sspclk", "apb_pclk";
        resets = <&syscrg JH7110_SYSRST_SPI0_APB>;
        interrupts = <38>;
        arm,primecell-periphid = <0x00041022>;
        num-cs = <1>;
        #address-cells = <1>;
        #size-cells = <0>;
        status = "disabled";
};
```

对于目标板卡 starfive 2，现有的 DMA 引擎主要分为两部分：DMA Core 与 DMA Driver。DMA Core 向上层的驱动提供一系列调用接口，来访问 DMA 通道，使用 DMA 功能。DMA Core 则通过 DMA Driver 对 DMAC 进行访问。板卡提供的源代码结构如下图，针对 DMA 的后续工作将针对这一部分展开。



DMA Core 为上层驱动提供的接口如下，在实现驱动的过程中，可能涉及到下列接口 OOB 版本的实现。

```
struct dma_channel *dma_request_channel(struct device *dev, const char *name);
void dma_release_channel(struct dma_chan *chan);
int dmaengine_slave_config(struct dma_chan *chan, struct dma_slave_config *config);
dma_cookie_t dmaengine_submit(struct dma_async_tx_descriptor *desc);
void dma_async_issue_pending(struct dma_chan *chan);
...
```

2.2 驱动适配 OOB 上下文，支持 EVL 线程使用驱动

将原生 Linux 驱动修改为 EVL 的实时驱动，主要涉及下列关键部分需要修改：

修改中断注册 API 及中断处理函数

原驱动中可能使用的中断处理函数如下：

```
request_irq(irq, bcm2835_dma_irq_handler, IRQF_SHARED, ..., dev);
```

需要使用 EVL 内核提供的 API 进行中断函数注册，如下：

```
evl_request_irq(irq, bcm2835_dma_irq_handler, IRQF_OOB | IRQF_SHARED, ..., dev);
```

除此之外，在这一代码例子中，需要将注册的中断处理函数进行修改，使其能够安全地在 OOB 上下文中执行，为 EVL 线程提供服务。

涉及 oob 上下文的函数的改造

在驱动 probe 阶段进行注册的一系列中断处理函数，以及其他涉及到在 oob 上下文中执行的代码段，都需要进行改造，使其能安全地在 oob 上下文运行。改造的过程主要包括下列内容：

- 在中断处理函数中保持 OOB-safe

在中断处理函数中，不能调用 schedule(),mutex_lock,printk()等阻塞/抢占操作。

- 自旋锁与互斥锁的替换

普通的 spinlock_t,需要替换为 EVL 提供的实时版本：struct evl_spinlock

普通的互斥锁 mutex，需要替换为 struct evl_kmutex，保证在 OOB 上下文中对共享资源进行串行访问。

- 上下文判断与跳转

在 DMA 配置，DMA 内存分配等情况下，这部分工作适合在 in-band 上下文中进行，在代码中可能会遇到上下文跳转的情况，需要在代码中添加如下的上下文跳转代码。

```
if (evl_in_oob()) {
    evl_switch_oob();   // 切回 in-band 上下文
}
```

- 在 oob 上下文中，涉及阻塞延时，等待事件等相关操作，需要将其替换为 EVL 内核提供的 API。

基于上述改造计划，计划对 DMA，SPI 等驱动先后进行改造，支持 EVL 线程使用。

2.3 驱动测试

对于已经实现的驱动成果，使用 EVL 线程对 DMA，SPI 等驱动进行测试。EVL 线程的主要工作可能包含下列步骤：

1. 打开目标设备文件，对目标设备进行基本配置与初始化（SPI 模式，时钟频率等）
2. 使用 ioctl 请求启用对应设备的带外模式，启用 oob 模式。
3. 将内存进行映射，将驱动缓冲区映射到用户空间，以访问内存中用于传输的数据区域。
4. 将当前线程绑定到实时内核，执行 SPI 传输操作，同时使用 EVL 时钟测量一次完整传输所花费的时间。
5. 清除使用资源，进行收尾操作，打印测试结果。

3. 时间安排

时间	阶段	规划任务
7.1-7.31	项目开发预热	阅读现有的 ARM 架构下的 DMA，SPI 相关的驱动代码，理清 oob 驱动代码逻辑
8.1-8.15	项目开发第一阶段	在目标开发板卡中搭建起驱动的设备框架，对驱动进行初步改造，使得驱动能够成功在内核中初始化与退出。
8.16-8.31	项目开发第二阶段	完成驱动的 oob 上下文改造
9.1-9.30	项目开发第三阶段	进行代码调试，测试修改后的驱动能否正常运行，支持 EVL 线程使用，并且对存在问题进行修复
10.1-10.15	总结	完成项目，进行总结，整理项目代码，文档与结果。