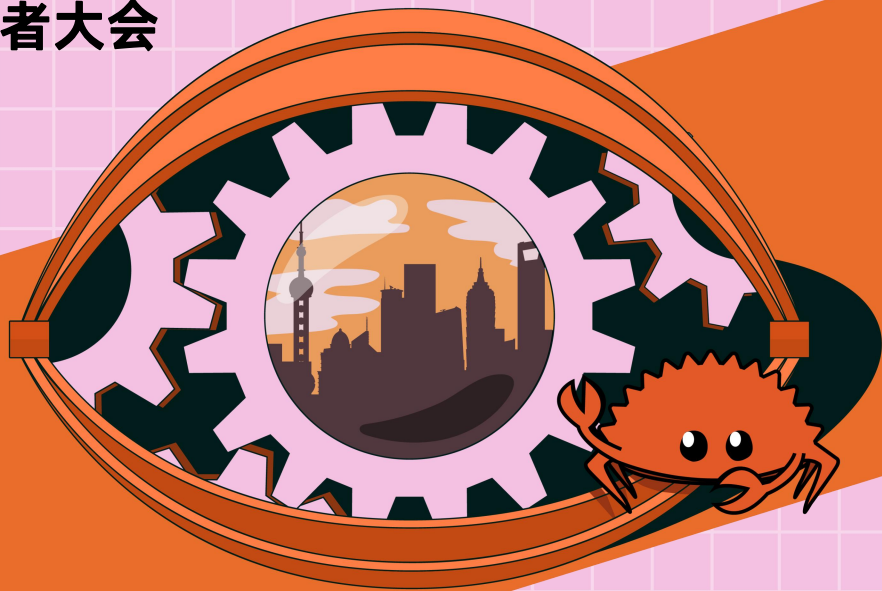


RUST CHINA CONF 2023

第三届中国Rust开发者大会



6.17-6.18 @Shanghai

■ 本次演讲.....

组件化驱动、ROM运行环境与RustSBI

洛佳

华中科技大学 网络空间安全学院

2023年6月



关于我.....

- 笔名洛佳
- 华中科技大学网络安全学院研一在读（导师：周威老师）
- 研究方向：物联网安全、系统安全
- 热爱开源，乐于尝试新技术
- RustSBI项目维护者
- 致力于向科研、教学和产业界推广Rust语言



目录



组件化驱动

汲取Rust嵌入式和操作系统生态经验，总结而成的新一代驱动开发方法。动、静态基地址结合，高灵活性；一次开发，同时复用于嵌入式、固件和内核中。



ROM运行环境

合理利用嵌入式、桌面和服务端芯片片内ROM代码，构造零开销的运行环境。进一步地，可完成安全引导、安全镜像分发和通常的嵌入式开发等功能。

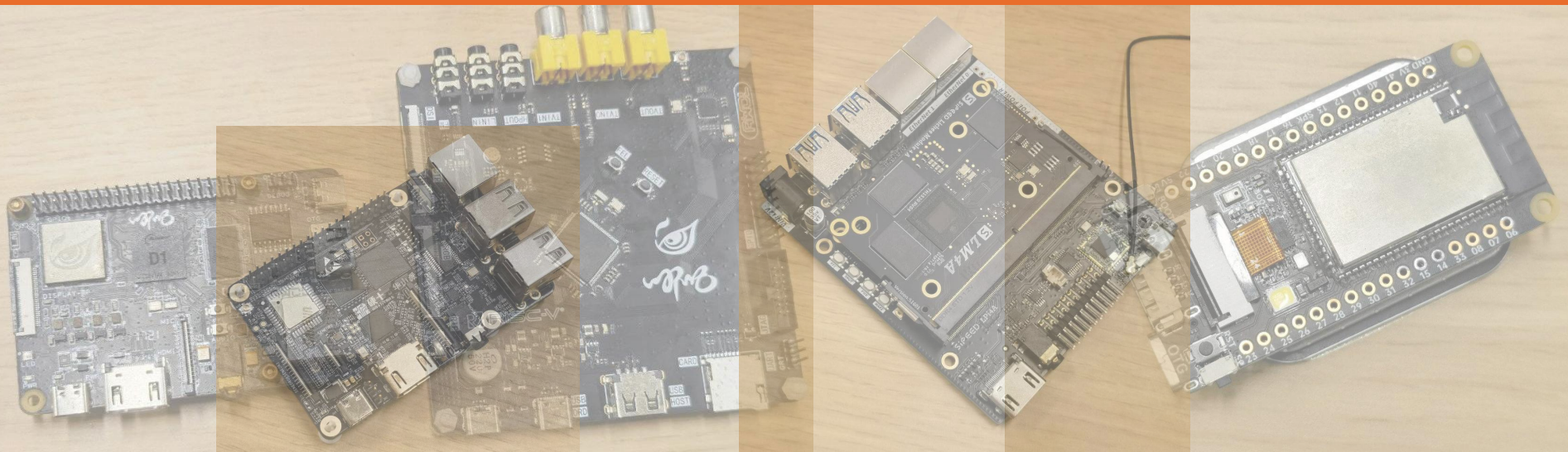


2023年的RustSBI

作为RISC-V SBI固件的RustSBI，2023年将与UEFI、LinuxBoot擦出火花。在驱动、环境和SBI接口的基础上，提供快速实现具体引导流程的解决方案。

第 01 部分

组件化驱动



什么是组件化驱动？

21 世纪的驱动程序

运用生命周期、可变性等最新的编程语言理论成果，构造适应开发需求的驱动程序。可结合过程宏等工程设计，提高开发效率。

高可复用、生态融合

同系列芯片可共用驱动，同系列外设驱动可复用。对接业界及开源成熟标准，新芯片系统开箱即用，与成熟组件自由组合。



灵活、高效、低成本

动、静态基地址结合，零开销抽象。只开发一次，同时运用于嵌入式、固件和操作系统生态中。轻松构造测试框架，快速验证组件。

系统软件开发新模式

从基础算法到文件、网络，操作系统的各个部分可拆为组件。灵活组合组件，构成符合应用需求的组件化操作系统。

■ 组件化驱动的组成方法



分享性外设：以GPIO为例

- 从前级环境获取所有权，如从ROM运行环境的#[entry]获得；
- 配置GPIO状态后，只有对应外设类型允许的操作函数能通过编译，否则拒绝编译，避免不安全行为；
- 开源标准抽象的功能，使用抽象规定的调用方法。本芯片外设专属的功能也可通过专有函数使用；
- 用户代码简短易懂，容易编写和调试，降低开发成本。

```
254 /// Available GPIO pins.
255 pub struct Pins<A: BaseAddress> {
256     // GPIO I/O 0.
257     pub io0: Pin<A, 0, Disabled>,

10 /// Individual GPIO pin.
11 pub struct Pin<A: BaseAddress, const N: usize, M: Alternate>

15 #[entry]
16 fn main(p: Peripherals) -> ! {
23     let mut led = gpio.io8.into_floating_output();
24     let mut button_1 = gpio.io22.into_pull_up_input();
25     let mut button_2 = gpio.io23.into_pull_up_input();
26     button_1.enable_schmitt();
27     button_2.enable_schmitt();
28     let mut led_state = PinState::High;
29     loop {
30         let button_1_pressed = button_1.is_low().unwrap();
31         let button_2_pressed = button_2.is_low().unwrap();
32         if button_1_pressed && button_2_pressed {
33             led.set_state(led_state).ok();
34             led_state = !led_state;
35             for _ in 0..100_000 {
36                 unsafe { core::arch::asm!("nop") }
37             }
38         } else if button_1_pressed {
39             led.set_low().ok();
40         } else if button_2_pressed {
41             led.set_high().ok();
42         }
43     }
44 }
```

*BL808组件化驱动操作GPIO按钮和灯

组件化外设的分类和设计方法

通用连接外设

UART、SPI和I2C等，通过互斥IO引脚划分资源，抽象接口，使用片外外设支持库

多媒体外设

MIPI、HDMI、DisplayPort和音频连接等，与对应的功能、电源外设共同设计

AI加速外设

包括自研AI核、核显和向量扩展等，编写专用驱动后，对接常用软件框架

无线连接外设

Wi-Fi基带、蓝牙、UWB等，合理编写频域、功率等软件限制，结合开源协议栈

高速板级通信

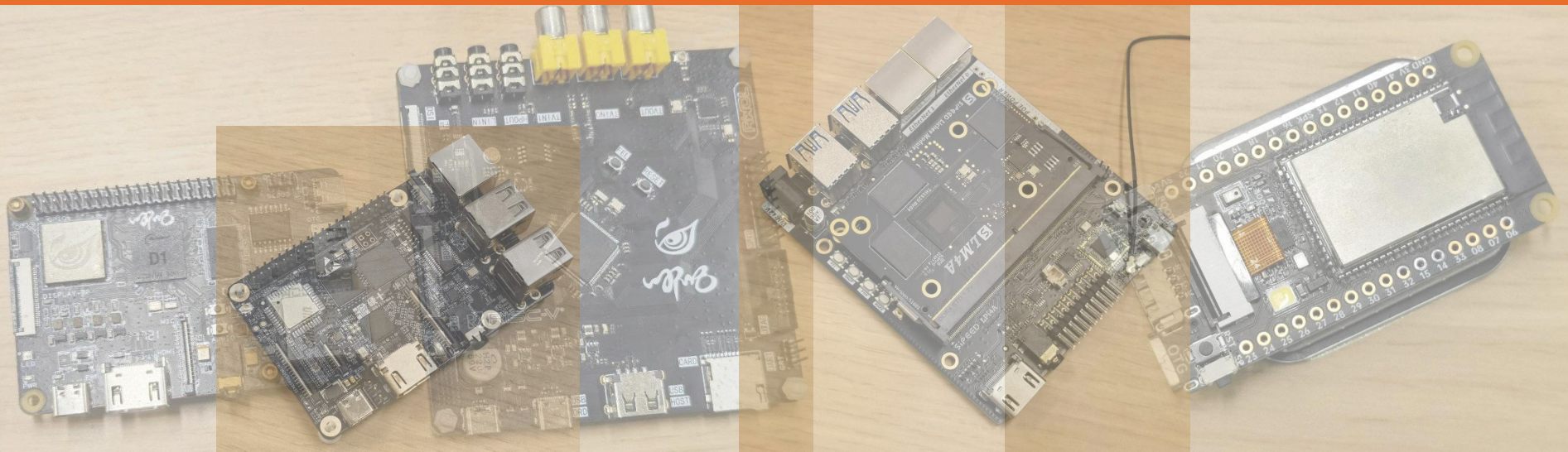
DDR、PSRAM等，结合控制器选择和输入合适的参数，用于准备后续启动过程

中断控制器

统一编写同一IP核或SoC设计的控制器支持，填入常量泛型，即可用于运行环境

第 02 部分

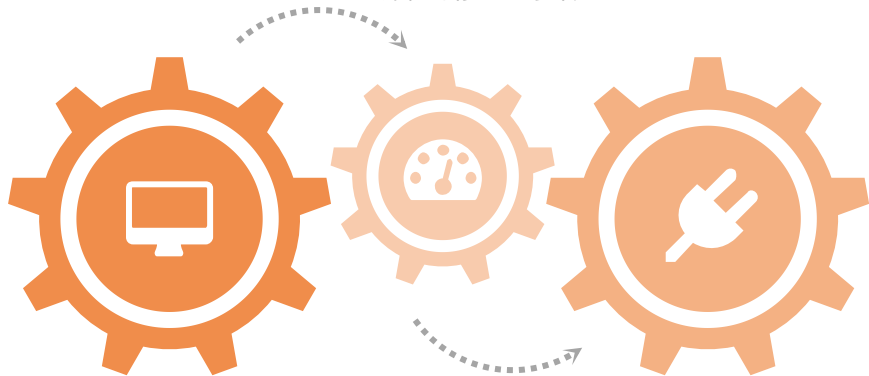
ROM运行环境



裸机和引导程序的ROM阶段

生成镜像结构

编译时生成镜像头，通常包含处理器配置、时钟和闪存配置等部分



高级语言环境

初始化bss段、data段，加载栈寄存器，构成高级语言运行的最小环境

外设和时钟

提供具备所有权的外设列表，提供ROM初始化完成的时钟配置

#[entry] 过程宏

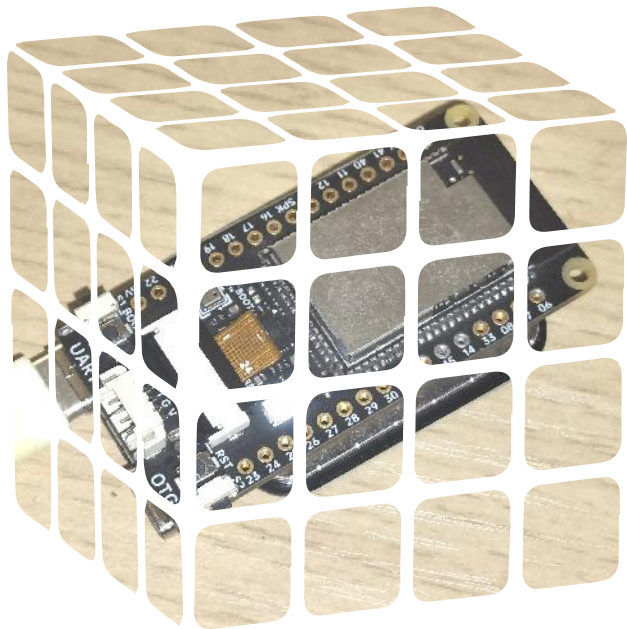
- 过程宏是卫生宏，完成语法树间的转换，此处用于将main函数转换为固件需要的入口函数。
- 包含ABI转换、检查参数等步骤。
- 使用过程宏时，同时使用对应包中的start初始化代码。start代码无需由用户编写，而是包含在宏生成的输出代码中。
- 编译即可获得包含镜像头的固件包，这是传统开发方法不具备的功能。

```
7 /// ROM runtime function entry.
8 #[proc_macro_attribute]
9 pub fn entry(args: TokenStream, input: TokenStream) -> TokenStream {
10     if !args.is_empty() {
11         return parse::Error::new(Span::call_site(), "#[entry] attribute accepts no arguments")
12             .to_compile_error()
13             .into();
14     }
15 }

1 // Build this example with:
2 // m0:
3 // rustup target install riscv32imac-unknown-none-elf
4 // cargo build --example blinky-bl808 --features bl808-m
5 // d0:
6 // rustup target install riscv64imac-unknown-none-elf
7 // cargo build --example blinky-bl808 --features bl808-d
8
9 #![no_std]
10 #![no_main]
11
12 use bl_rom_rt::{entry, Peripherals};
13 use embedded_hal::digital::OutputPin;
14
15 #[entry]
16 fn main(p: Peripherals) -> ! {
17     let mut led = p.gpio.io8.into_floating_output();
18     loop {
19         led.set_low().ok();
20         for _ in 0..100_000 {
21             unsafe { core::arch::asm!("nop") }
22         }
23         led.set_high().ok();
24         for _ in 0..100_000 {
25             unsafe { core::arch::asm!("nop") }
26         }
27     }
28 }
29
30 #[panic]
31 fn pani
32     loo
33 }

34 #[cfg(feature = "bl808-m0")]
35 #[naked]
36 #[link_section = ".text.entry"]
37 #[export_name = "_start"]
38 unsafe extern "C" fn start() -> ! {
39     #[link_section = ".bss.uninit"]
40     static mut STACK: Stack<LEN_STACK_M0> = Stac
41     asm!(
42         " la      sp, {stack}
43         li      t0, {hart_stack_size}
44         add     sp, sp, t0",
45         " la      t1, sbss
46         la      t2, ebss
47         1: bgeu  t1, t2, 1f
48         sw      zero, 0(t1)
49         addi    t1, t1, 4
50         j       1b
51     1:",
52         " la      t3, sidata
53         la      t4, sdata
54         la      t5, edata
55         1: bgeu  t4, t5, 1f
56         lw      t6, 0(t3)
57         sw      t6, 0(t4)
58         addi    t3, t3, 4
59         addi    t4, t4, 4
60         j       1b
61     1:",
62         " call {main}",
63         stack = sym STACK,
64         hart_stack_size = const LEN_STACK_M0,
65         main = sym main,
66         options(noreturn)
67     )
68 }
```

多核异构芯片的镜像融合（以BL808为例）



ROM多核启动

直接使用ROM机制开启所有三个核，并加载相关的固件。相比额外引导程序而言，节省引导链级数，增加安全性和效率。

部分固件调试

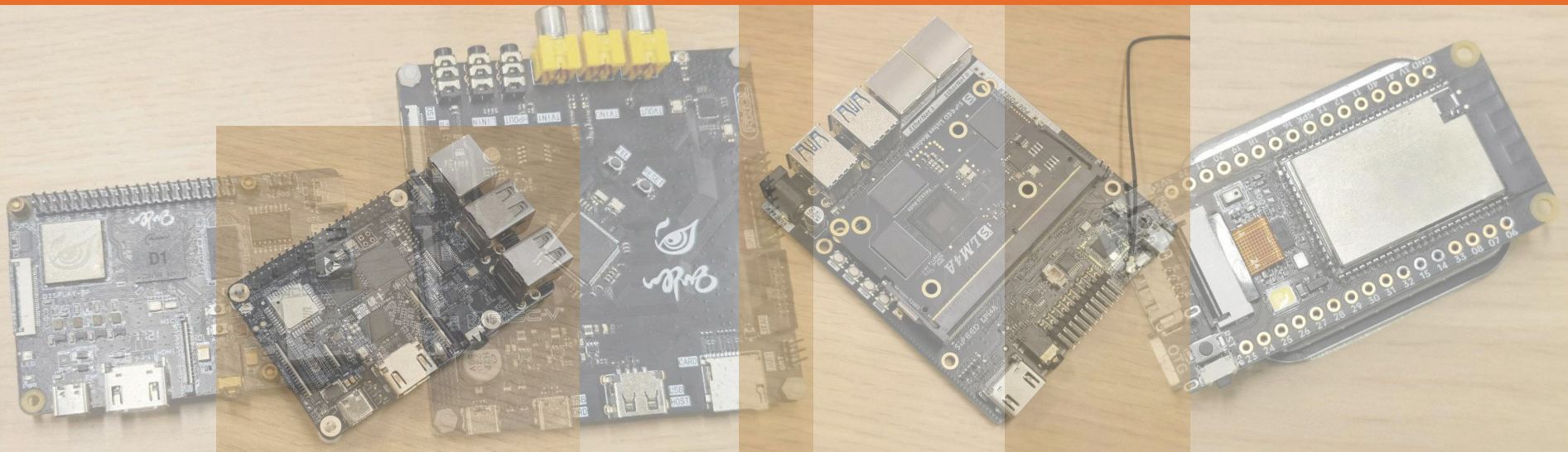
首先编译程序为单核固件，再融合三个固件为多核。单核固件可独立运行，易于按处理器核单独划分和调试。

融合规则

若三个不同固件中镜头头的闪存配置、时钟配置不同，或使用的CPU核有交叉，则拒绝合并。否则，融合为新的镜头头，再导出镜像。

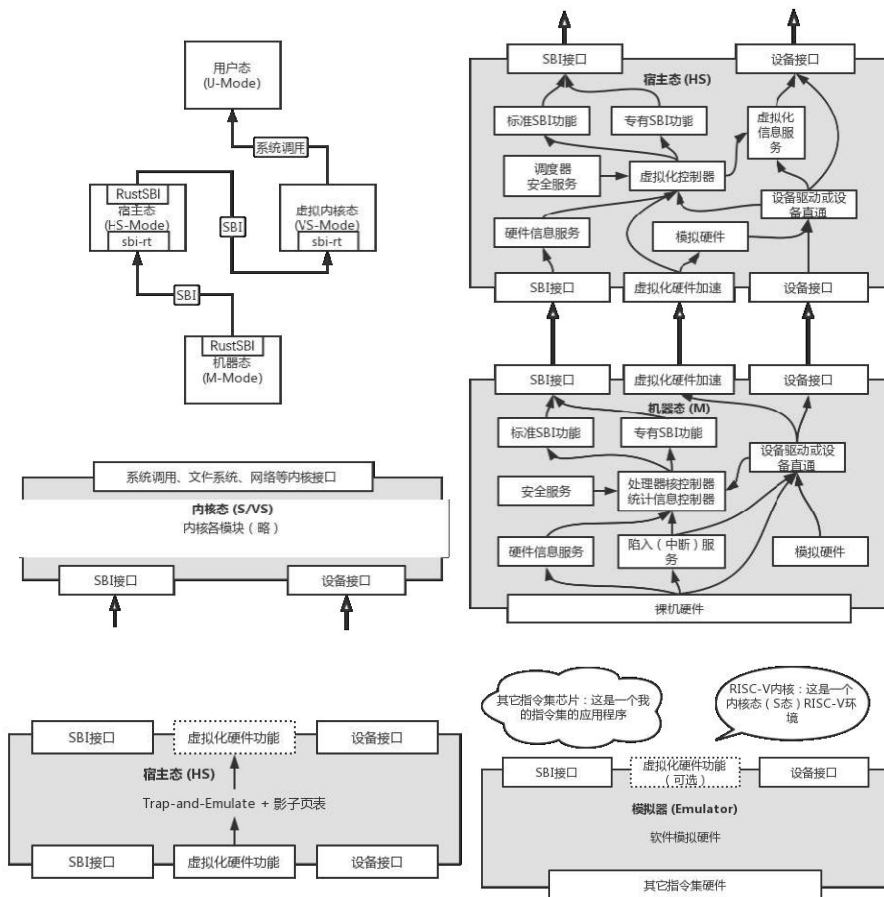
第 03 部分

2023年的RustSBI



RustSBI软件架构更新

- RISC-V SBI可运用于机器态和虚拟化的宿主态，此时RustSBI实现应为虚拟机提供电源、核管理等功能。
- 嵌套虚拟化存在时，RustSBI实现应当为内部虚拟机软件模拟H指令集。在这方面，Dramforever的项目¹提供了很好的例子。
- LARVa²项目是固件充当模拟器的例子，这里RustSBI被编译到RISC-V之外的指令集。
- YdrMaster设计的sbi-testing³测试框架可轻松检查SBI实现的正确性。



¹ <https://github.com/dramforever/opensbi-h>

² LARVa: <https://github.com/xen0n/larva>

³ sbi-testing: <http://github.com/rustsbi/sbi-testing>

RustSBI原型设计系统

构建原型

在组件化驱动、ROM运行环境基础上，快速构建完整的原型引导程序产品

快速选型

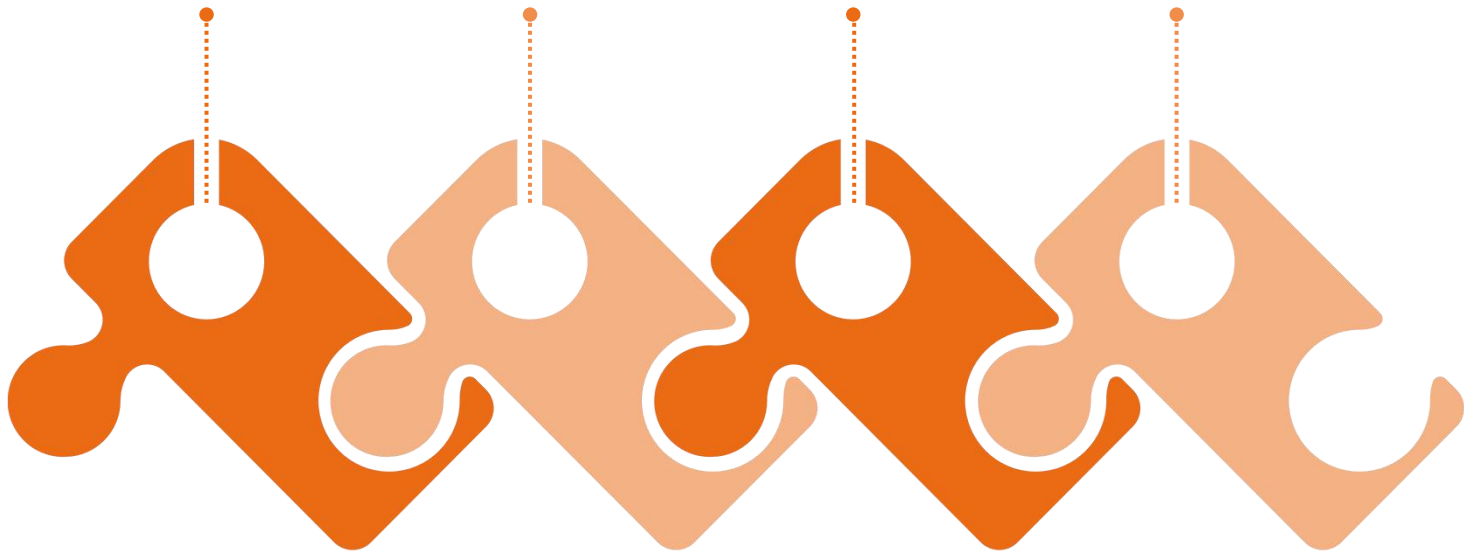
图形化界面选取功能，配置可保存，使用轻便。多语言支持，利于国际化

SBI功能

快速添加需要的机器态SBI功能，包括标准RISC-V SBI和厂商专有的SBI扩展

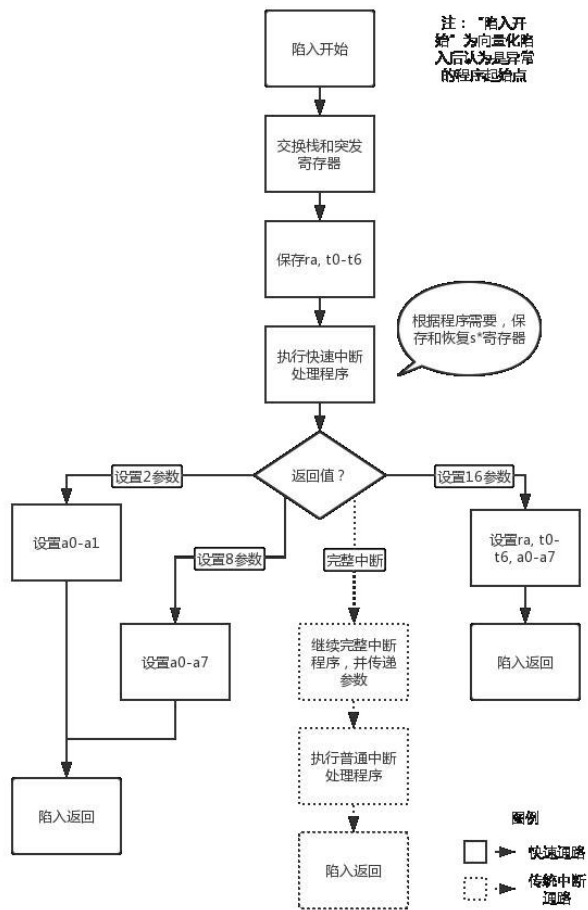
引导启动

可选内核态接口包括UEFI或LinuxBoot，生态丰富完善，快速对比解决方案



RISC-V上的快速陷入通道

- RISC-V并未强制规定陷入栈的内容，它的上下文切换过程可定制，若给予上下文切换更多的信息，它的性能就可得到进一步提升
- 上下文调用时先保存部分寄存器，让高级语言判断是否进入完整流程，或给定需要设置的寄存器数量
- 尽量减少上下文切换对空间局部性的破坏
- 向量化陷入：硬件取向量，分流mtime、msoft等中断过程和异常过程，进一步细化通路，明确上下文保存需求
- 不同等级的上下文保存到不同结构体中，地址存于突发寄存器，快速处理程序可为完整处理程序提供参数
- 项目地址：<https://github.com/YdrMaster/fast-trap>



示例：RustSBI原型设计系统选型界面

RustSBI 原型设计系统 - 主界面

| 编号 | 选项 | 简述 |
|-----------------|-------|------------------|
| >> Language | 配置语言 | 简体中文 (中国) |
| Bootstrap | 启动程序 | Hello World 示例程序 |
| MachineMode | 机器态功能 | 启动程序不支持机器态功能 |
| SupervisorMode | 内核态功能 | |
| PlatformSupport | 平台支持 | 全志® D1-H 系列平台 |
| BootloadMedia | 引导介质 | |
| CompileFlags | 编译配置 | |
| HelpVerAbout | 帮助关于 | |
| QuitAndSave | 退出并保存 | |

RustSBI 原型设计系统 - 标准 SBI 功能

| 编号 | 选项 | 简述 |
|-----------------|--------|-----|
| TimerExtension | 时钟扩展 | 已启用 |
| IpiExtension | 核间中断扩展 | 已启用 |
| RfenceExtension | 远程栅栏扩展 | 已启用 |
| HsmExtension | 核状态扩展 | 已启用 |
| SrstExtension | 系统复位扩展 | 已启用 |
| PmuExtension | 性能监视扩展 | 已启用 |
| >> Back | 返回 | |

RustSBI 原型设计系统 - 语言

| 编号 | 语言 |
|----------|-----------|
| >> zh-CN | 简体中文 (中国) |
| en-US | 英文 (美国) |

RustSBI 原型设计系统 - 机器态功能

| 编号 | 选项 | 简述 |
|-----------------|-----------|---------------|
| StandardSbiFeat | 标准 SBI 功能 | 标准 SBI 1.0 实现 |
| >> Back | 返回 | |

RustSBI 原型设计系统 - 启动程序

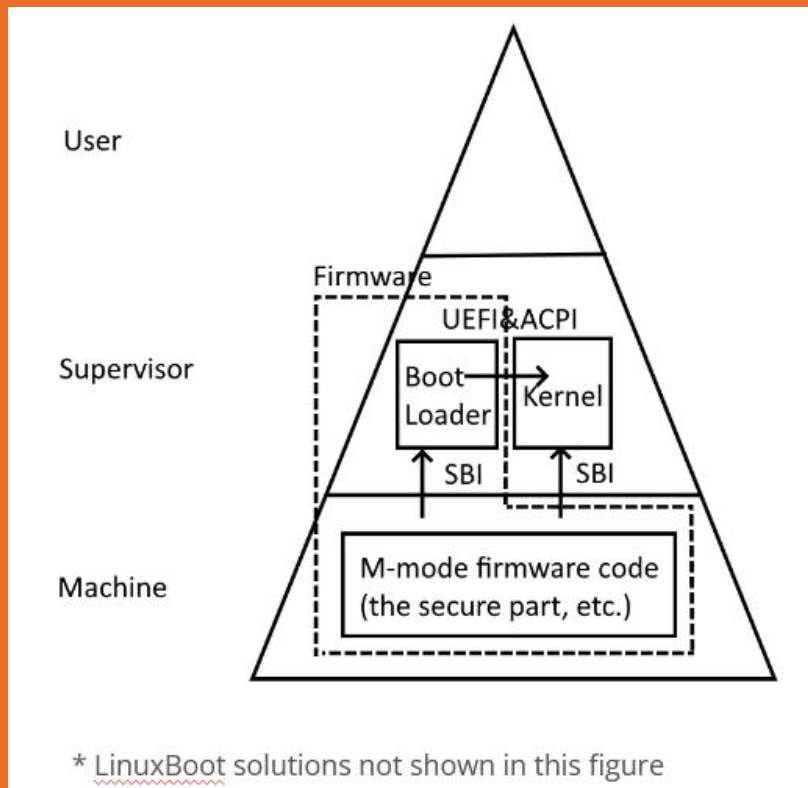
| 编号 | 选项 | 简述 |
|---------------|----------|---------|
| >> JumpToDram | 跳转至 DRAM | 已选中 |
| SampleProgram | 仅启动示例程序 | 不使用示例程序 |
| Back | 返回 | |

RustSBI 原型设计系统 - 全志® D1-H 系列平台

| 编号 | 选项 | 简述 |
|-------------------|-------|--------|
| >> ChoosePlatform | 选择此平台 | 已选中此平台 |
| Back | 返回 | |

RustSBI与生态后续引导链

- 对RISC-V UEFI, RustSBI准备好SBI环境。
 - RustSBI充当至关重要的安全层, 并准备好S态软件的环境
 - UEFI部分运行在S态
- 对LinuxBoot, RustSBI参与准备好rootfs和最小Linux环境的代码中。
 - 一个优秀的例子是Oreboot¹
 - RustSBI原型设计系统将会提供此类环境准备软件
- 后续生态的固件也可复用RustSBI编写的静态检查等相关工具。



¹ <https://github.com/oreboot/oreboot>

致谢

- 感谢Rust语言让我拥有重新认识嵌入式、固件开发的机会。
- 感谢Rustcc嵌入式社区、TUNA嵌入式社区和华科网安的开源团队在关键问题上的答疑解惑，社区的良好氛围对生态有非常大的帮助。
- 感谢@YdrMaster、@duskmoon314、@OrangeCMS和更多直接参与RustSBI开发的贡献者，感谢@双倍多多冰、@dramforever、@大佬鼠的小粉丝和更多熟悉RISC-V的朋友答疑解惑。
- 感谢我在华科的本科生和研究生同学们贡献到bl-rom-rt和bl-soc中，感谢六次开源工坊活动的参与者。
- 感谢博流、Sipeed公司的设备支持和沟通交流机会！



Thank you!

