

动态设备的生成宏SBI实现

—— RustSBI 0.4.0 特性先览

洛佳 / 华中科技大学 / 2024.5

本次演讲将包含.....

- 动态设备和静态设备
 - 回顾#[derive(RustSBI)]的使用方法
- 为什么要引入#[rustsbi(dynamic)]
- RustSBI的项目生态

快速回顾.....

- RISC-V SBI是RISC-V下位于M和S特权态之间的调用接口
 - 在S和U特权态边界，存在的类似调用接口是系统调用
- RISC-V架构下的主板固件提供两大功能：引导和支持环境
 - “引导”由两部分组成，第一部分由ROM代码启动，第二部分启动具体的操作系统（支持LinuxBoot、UEFI、U-Boot或裸SBI的操作系统等）
 - “支持环境”负责提供RISC-V SBI实现，包括基础SBI功能、安全功能和其它的软件寄存器功能等
- 主流的支持环境（不考虑虚拟化）
 - RISC-V SBI标准里提到了RustSBI和OpenSBI，分别在Rust和C语言的RISC-V生态占主导地位
- RISC-V SBI国际标准已经发布2.0正式版，包含若干个基本的功能扩展
 - 支持环境提供商根据RISC-V协会的标准文件提供SBI实现，操作系统供应商也依据相同的技术标准使用固件提供的SBI功能

动态设备和静态设备

- RISC-V SBI各个扩展由芯片外设（或主板上有关的外设）实现
- 静态设备
 - 对相同的主板，外设的数量和逻辑地址（内存地址、总线基地址、总线位置等）固定，代码编译时已知设备的地址，这些设备都是静态设备
- 动态设备
 - 有些情况下，具有SBI实现的固件需要检测主板上具有哪些硬件设备
 - 其它数据结构（前级引导程序如U-Boot SPL和SyterKit等，或由mconfigptr提供）也可能动态地指定使用哪些设备完成SBI功能
 - 编译时设备的逻辑位置未知，这些都是动态设备
- 目前的RustSBI（0.3.2版本）以处理静态设备为主，但OpenSBI的fw_dynamic固件处理的是动态设备

#[derive(RustSBI)]

- 是RustSBI主要的生成宏
 - 生成宏是Rust语言的特性，它允许开发者提供宏代码，完成从语法树到语法树的变换过程
- #[derive(RustSBI)]可以直接从各个扩展分别的实现中*生成*总的SBI实现
 - 隐藏了具体的分发代码，开发者无需考虑具体的EID、FID等常数和概念，显著地减少代码量，提高开发效率
 - 如右图，可以从所有的SBI扩展实现中生成，这样就能直接调用FullyImplemented的handle_ecall函数了
- 目前（0.4.0-alpha.1版）的#[derive(RustSBI)]已经良好地提供了静态设备的支持

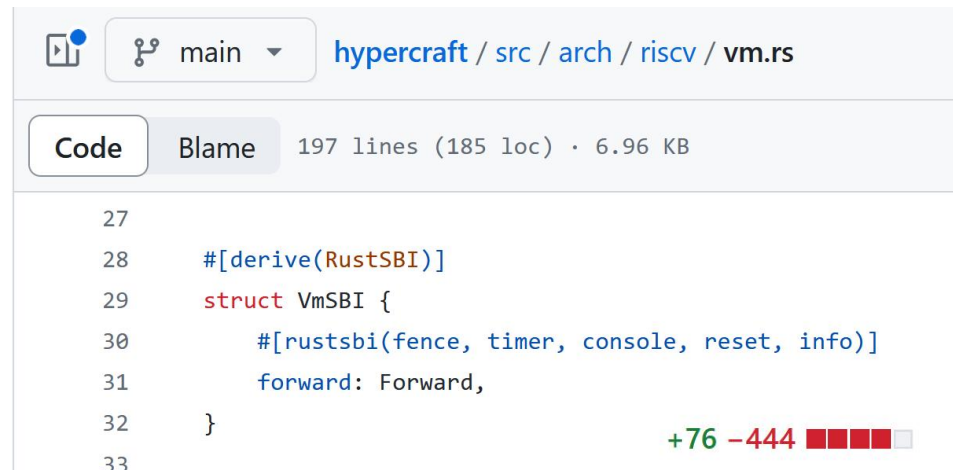
```
8     #[derive(RustSBI)]
9     struct FullyImplemented {
10         console: DummyConsole,
11         cppc: DummyCppc,
12         hsm: DummyHsm,
13         ipi: DummyIpi,
14         nacl: DummyNacl,
15         pmu: DummyPmu,
16         reset: DummyReset,
17         fence: DummyFence,
18         sta: DummySta,
19         susp: DummySusp,
20         timer: DummyTimer,
21         info: DummyEnvInfo,
22     }
```

#[derive(RustSBI)]的例子

- KuangjuX的[hypercraft](https://github.com/KuangjuX/hypercraft), 可支持Linux系统启动运行
 - <https://github.com/KuangjuX/hypercraft/pull/3>
- [rustyvisor](https://github.com/stemnic/rustyvisor)
 - <https://github.com/stemnic/rustyvisor/pull/5>
- RustSBI的若干独立支持包项目

```
10  #[derive(RustSBI)]
11  pub struct FdtBoard<'a> {
12      #[rustsbi(dbcn)]
13      serial: uart16550::Uart16550Handle<'a>,
14      #[rustsbi(time, ipi)]
15      clint: clint::ClintHandle<'a>,
16      #[rustsbi(reset)]
17      sifive_test: sifive_test::SifiveTestHandle<'a>,
18  }
```

```
#[derive(RustSBI)]
struct FixedRustSBI<'a> {
    #[rustsbi(ipi, timer)]
    clint: &'a clint::Clint,
    hsm: Hsm,
    reset: &'a qemu_test::QemuTest,
    dbcn: &'a dbcn::DBCN,
}
```

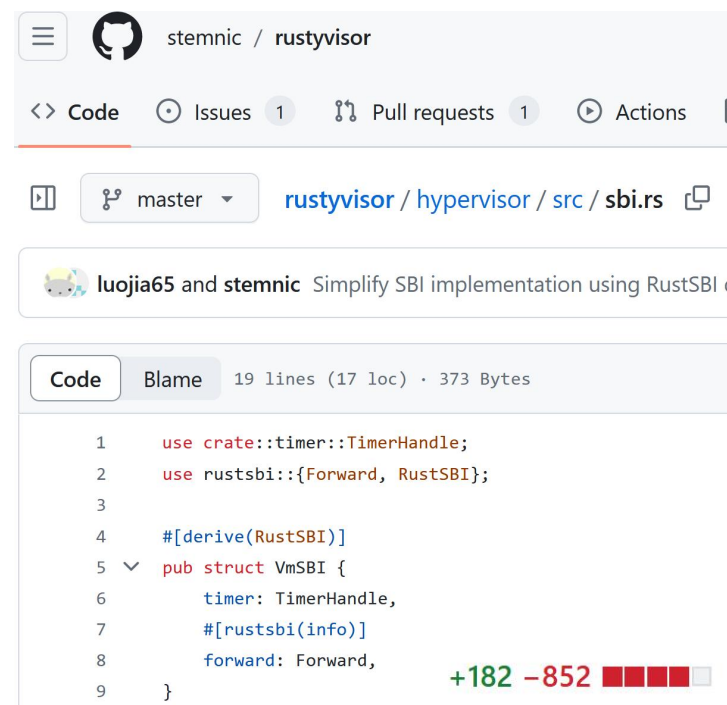


hypercraft / src / arch / riscv / vm.rs

Code Blame 197 lines (185 loc) · 6.96 KB

```
27
28  #[derive(RustSBI)]
29  struct VmSBI {
30      #[rustsbi(fence, timer, console, reset, info)]
31      forward: Forward,
32  }
33
```

+76 -444



stemnic / rustyvisor

<> Code Issues 1 Pull requests 1 Actions

rustyvisor / hypervisor / src / sbi.rs

luojia65 and stemnic Simplify SBI implementation using RustSBI c

Code Blame 19 lines (17 loc) · 373 Bytes

```
1  use crate::timer::TimerHandle;
2  use rustsbi::{Forward, RustSBI};
3
4  #[derive(RustSBI)]
5  pub struct VmSBI {
6      timer: TimerHandle,
7      #[rustsbi(info)]
8      forward: Forward,
9  }
```

+182 -852

为什么要引入#[rustsbi(dynamic)]?

- SBI固件（不含虚拟化）的市场需求包含动态设备的部分
 - 静态设备的#[derive(RustSBI)], 如独立包和Oreboot, 适用于为特定主板专门编写RustSBI实现
 - RustSBI + U-Boot需要识别U-Boot SPL中的FDT结构体
 - SyterKit + RustSBI、RustSBI + EDK II等生态适配
- 动态的RustSBI固件实现方法, 需要从支持的设备列表出发, 查找并选择设备
 - 适用于OpenSBI fw_dynamic生态的RustSBI固件, 免费获得安全性和性能提升
 - 便于对比测试, 科研学术固件的实现与验证平台
- 产业界对RustSBI固件的市场需求正在增加
 - 操作系统厂商为了保证安全性, 选择与特定安全固件捆绑发行
 - 芯片、主板厂商需要与C语言生态结合良好的RustSBI固件, 来实现安全功能和提高开发效率

实现#[rustsbi(dynamic)]

- 初步实现 ([#67](#)) , bug修复 ([#68](#), [#69](#))
- 剩余部分: 宏代码单元测试和文档、生态软件实现
- 0.4.0 roadmap组成部分
- 目前的难度在于需要为有生命周期、类型泛型和常量泛型的结构体都生成RustSBI实现, 不过同学们和社区的实现进度相当快!

```
532     let define_prober = quote! {
533 +         struct _Prober #impl_generics (&'lt #name #origin_ty_generics)
           #where_clause;
534 +         impl #impl_generics ::rustsbi::_ExtensionProbe for _Prober
           #ty_generics #where_clause {
535             #[inline(always)]
536             fn probe_extension(&self, extension: usize) -> usize {
537                 match extension {
```

```
25 + #[derive(RustSBI)]
26 + #[rustsbi(dynamic)]
27 + struct WithConstantGenerics<const N: usize> {
28 +     info: DummyEnvInfo,
29 +     _dummy: [u8; N],
30 + }
31 +
32 + #[derive(RustSBI)]
33 + #[rustsbi(dynamic)]
34 + struct WithLifetime<'a> {
35 +     info: &'a DummyEnvInfo,
36 + }
37 +
38 + #[derive(RustSBI)]
39 + #[rustsbi(dynamic)]
40 + struct WithEverythingCombined<'a, T: rustsbi::Timer, U, const N: usize>
41 + where
42 +     U: rustsbi::Reset,
43 + {
44 +     timer: T,
45 +     reset: U,
46 +     info: &'a DummyEnvInfo,
47 +     _dummy: [u8; N],
48 + }
```


与RustSBI适配的生态项目

- SyterKit项目：全志芯片的综合引导程序 (<https://github.com/YuzukiHD/SyterKit>)
 - 是良好的ROM阶段引导程序，可以用来启动RustSBI固件
- Oreboot项目：Rust的Oreboot, LinuxBoot引导程序 (<https://github.com/oreboot/oreboot>)
 - 综合三个阶段的引导程序，具有Oreboot生态的rootfs，可有效启动Linux系统
- DragonOS操作系统：使用Rust从0自研内核，具有Linux兼容性的操作系统 (<https://github.com/DragonOS-Community/DragonOS>)
 - 在RISC-V平台支持中使用了RustSBI的sbi-rt包完成SBI功能
- HermitOS操作系统和Hermit引导程序 (<https://github.com/hermit-os>)
 - 在系统启动的各个阶段都使用RustSBI生态的库，显著降低代码量，减少开发成本

未来的RustSBI应用.....

- TEE和科研项目
- RustSBI Prototyper（添加更多厂商主板的支持，欢迎厂商合作！）
- RISC-V生态基础软件（与华中科技大学校内团队合作）

感谢各位!

Used by 105



Contributors 15



Languages

