

# oreboot 2022 status report

on to RISC-V

Daniel Maslowski



Hey Again OSFC!



Remember oreboot?



# Remember oreboot?

*oreboot is a fork of coreboot...*



**coreboot**



**OREBOOT**



# Remember oreboot?

*oreboot is a fork of coreboot...*



# coreboot



# OREBOOT

From aa246f71de7f9900a30d938ab618c46839436616 [...]

From: "Ronald G. Minnich" <rminnich@gmail.com>

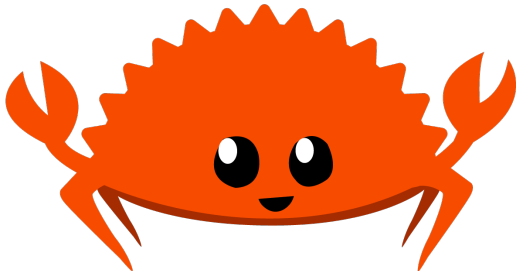
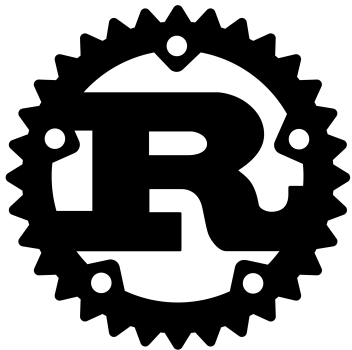
Date: Mon, 1 Apr 2019 23:48:56 +0000

Subject: [PATCH] Initial removal of C code



# oreboot is firmware in Rust!

*oreboot is a fork of coreboot, with C removed, written in Rust.*



<https://github.com/oreboot>



# State of Development in 2022 / Q4



# State of Development in 2022 / Q4



reworked to Cargo Workspace, thanks to 洛佳 (Luo Jia)

- ▶ got rid of a lot of boilerplate
- ▶ build speed increased with shared cache
- ▶ can still be built from top level or board dir running `make`



# State of Development in 2022 / Q4



reworked to Cargo Workspace, thanks to 洛佳 (Luo Jia)

- ▶ got rid of a lot of boilerplate
- ▶ build speed increased with shared cache
- ▶ can still be built from top level or board dir running `make`



removed FSP bits and everything but Allwinner D1 (sunxi/nezha)

- ▶ references kept in a `graveyard.md` for revival



# State of Development in 2022 / Q4

-  reworked to Cargo Workspace, thanks to 洛佳 (Luo Jia)
  - ▶ got rid of a lot of boilerplate
  - ▶ build speed increased with shared cache
  - ▶ can still be built from top level or board dir running `make`
-  removed FSP bits and everything but Allwinner D1 (sunxi/nezha)
  - ▶ references kept in a `graveyard.md` for revival
-  main focus is on RISC-V now
-  build setup based on `xtask`
-  driver model discarded in favor of Rust `embedded-hal`



# State of Development in 2022 / Q4

-  reworked to Cargo Workspace, thanks to 洛佳 (Luo Jia)
  - ▶ got rid of a lot of boilerplate
  - ▶ build speed increased with shared cache
  - ▶ can still be built from top level or board dir running `make`
-  removed FSP bits and everything but Allwinner D1 (`sunxi/nezha`)
  - ▶ references kept in a `graveyard.md` for revival
-  main focus is on RISC-V now
-  build setup based on `xtask`
-  driver model discarded in favor of Rust `embedded-hal`

Website and documentation are being worked on.



# Cleaning up



## Cleaning up

```
$ git diff --stat 9b1614cc..ee205123 | tail -n1  
23 files changed, 2 insertions(+), 13005 deletions(-)
```



## Cleaning up

```
$ git diff --stat 9b1614cc..ee205123 | tail -n1  
23 files changed, 2 insertions(+), 13005 deletions(-)  
  
$ git show --stat c0061370 | tail -n1  
139 files changed, 1 insertion(+), 37399 deletions(-)
```



# Cleaning up

```
$ git diff --stat 9b1614cc..ee205123 | tail -n1  
23 files changed, 2 insertions(+), 13005 deletions(-)
```

```
$ git show --stat c0061370 | tail -n1  
139 files changed, 1 insertion(+), 37399 deletions(-)
```

```
$ git diff --stat 85f8dc9b..04d750f2 | tail -n1  
69 files changed, 5 insertions(+), 7062 deletions(-)
```



# Cleaning up

```
$ git diff --stat 9b1614cc..ee205123 | tail -n1  
23 files changed, 2 insertions(+), 13005 deletions(-)
```

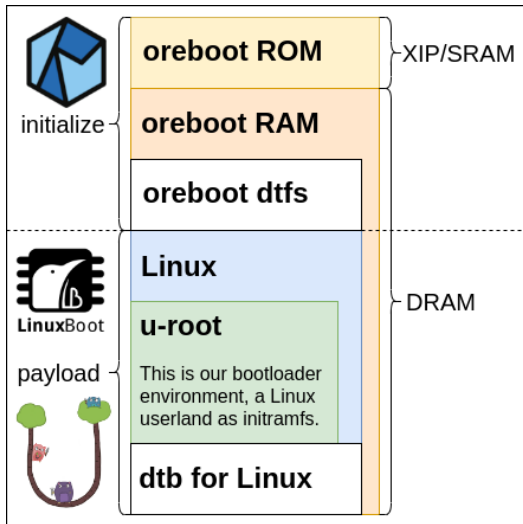
```
$ git show --stat c0061370 | tail -n1  
139 files changed, 1 insertion(+), 37399 deletions(-)
```

```
$ git diff --stat 85f8dc9b..04d750f2 | tail -n1  
69 files changed, 5 insertions(+), 7062 deletions(-)
```

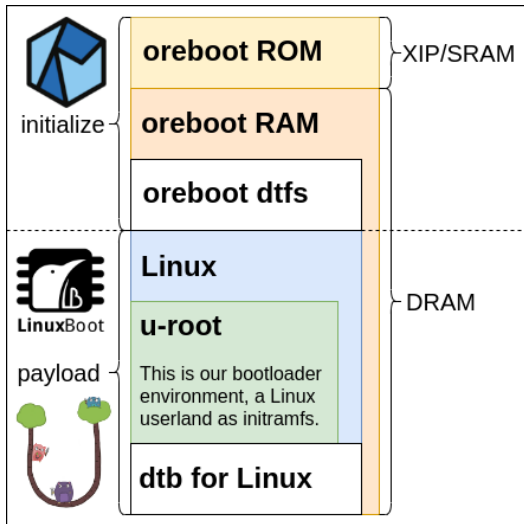
Removals are documented in `graveyard.md`.



# Firmware Matryoshka



# Firmware Matryoshka



## Stages



### XIP/SRAM

- ▶ early initialization
- ▶ PLLs, clocks, GPIOs
- ▶ UART, say hello
- ▶ SPI flash MMIO
- ▶ DRAM controller



### DRAM

- ▶ what didn't fit in SRAM
- ▶ extract payload
- ▶ set up handlers
- ▶ run payload (done)



## Firmware Runtime Services



# Benefits



# Benefits

## Vulnerability REsearch



| Vulnerability Category | Count | Average Impact         |
|------------------------|-------|------------------------|
| PEI Memory Corruption  | 3     | CVSS: 8.0 (High)       |
| SMM Memory Corruption  | 49    | CVSS: 8.0 (High)       |
| DXE Memory Corruption  | 7     | CVSS: 7.7 (High)       |
| Mitigation Failures    | 2     | CVSS: 6.0 (HighMedium) |

#DXE #Firmware #Fujitsu #HP #Lenovo #PEI #SMI #SMM #UEFI



# Benefits

| Vulnerability REsearch |       |                        |
|------------------------|-------|------------------------|
| Vulnerability Category | Count | Average Impact         |
| PEI Memory Corruption  | 3     | CVSS: 8.0 (High)       |
| SMM Memory Corruption  | 49    | CVSS: 8.0 (High)       |
| DXE Memory Corruption  | 7     | CVSS: 7.7 (High)       |
| Mitigation Failures    | 2     | CVSS: 6.0 (HighMedium) |

#DXE #Firmware #Fujitsu #HP #Lenovo #PEI #SMI #SMM #UEFI

... for attackers - whoops! Closed source means we can hardly fix issues on our own, while OEMs are slow to push out updates.



# RISC-V Runtime Services: SBI



# RISC-V Runtime Services: SBI

RISC-V *Runtime Services* are listed in the **platform spec**, referencing the **SBI spec**.



# RISC-V Runtime Services: SBI

RISC-V *Runtime Services* are listed in the **platform spec**, referencing the **SBI spec**.

The SBI (*Supervisor Binary Interface*) spec is a living document:

<https://github.com/riscv-non-isa/riscv-sbi-doc>



# RISC-V Runtime Services: SBI

RISC-V *Runtime Services* are listed in the **platform spec**, referencing the **SBI spec**.

The SBI (*Supervisor Binary Interface*) spec is a living document:

<https://github.com/riscv-non-isa/riscv-sbi-doc>

It defines extensions and functions similar to system calls.



# RISC-V Runtime Services: SBI

RISC-V *Runtime Services* are listed in the **platform spec**, referencing the **SBI spec**.

The SBI (*Supervisor Binary Interface*) spec is a living document:

<https://github.com/riscv-non-isa/riscv-sbi-doc>

It defines extensions and functions similar to system calls.

Arguments and extension/function are passed through *A* (argument) registers.



# RISC-V Runtime Services: SBI

RISC-V *Runtime Services* are listed in the **platform spec**, referencing the **SBI spec**.

The SBI (*Supervisor Binary Interface*) spec is a living document:

<https://github.com/riscv-non-isa/riscv-sbi-doc>

It defines extensions and functions similar to system calls.

Arguments and extension/function are passed through *A* (argument) registers.

Then the call is performed via the ECALL instruction.



# RISC-V Runtime Services: SBI

RISC-V *Runtime Services* are listed in the **platform spec**, referencing the **SBI spec**.

The SBI (*Supervisor Binary Interface*) spec is a living document:

<https://github.com/riscv-non-isa/riscv-sbi-doc>

It defines extensions and functions similar to system calls.

Arguments and extension/function are passed through *A* (argument) registers.

Then the call is performed via the ECALL instruction.

Example, writing a **B** character to the serial console:

```
li a0, 'B'    # argument
li a7, 0x01   # extension "console putchar"
ecall
```



# RISC-V Runtime Services: SBI

RISC-V *Runtime Services* are listed in the **platform spec**, referencing the **SBI spec**.

The SBI (*Supervisor Binary Interface*) spec is a living document:

<https://github.com/riscv-non-isa/riscv-sbi-doc>

It defines extensions and functions similar to system calls.

Arguments and extension/function are passed through *A* (argument) registers.

Then the call is performed via the ECALL instruction.

Example, writing a **B** character to the serial console:

```
li a0, 'B'    # argument
li a7, 0x01   # extension "console putchar"
ecall
```

Ports need to be written per platform (core/SoC/board).



# OpenSBI



# OpenSBI

An SBI written in C: <https://github.com/riscv-software-src/opensbi>



# OpenSBI

An SBI written in C: <https://github.com/riscv-software-src/opensbi>

 **riscv-software-src / opensbi**

Public

Watch 51 Fork 253 Star 437

[Code](#) [Issues 32](#) [Pull requests 7](#) [Actions](#) [Projects](#) [Wiki](#) [...](#)

master opensbi / platform / [Go to file](#) [Add file](#) [...](#)

 **Jan Remes** and **avpatel** platform: generic: check if CPU node is enabled ... 12 days ago [History](#)

[..](#)

|               |                                                               |              |
|---------------|---------------------------------------------------------------|--------------|
| andes/ae350   | lib: utils/serial: support 'reg-offset' property              | last month   |
| fpga          | lib: utils/serial: support 'reg-offset' property              | last month   |
| generic       | platform: generic: check if CPU node is enabled               | 12 days ago  |
| kendryte/k210 | include: sbi_utils: Introduce an helper to get fdt base ad... | 7 months ago |
| nuclei/ux600  | include: sbi_utils: Introduce an helper to get fdt base ad... | 7 months ago |
| template      | lib: utils/serial: support 'reg-offset' property              | last month   |



# RustSBI

*Rust*  
*SBI*



# RustSBI

*Rust*  
*SBI*



library for implementing a specific platform SBI



solves the abstraction issue based on Rust `embedded-hal`



# RustSBI

*Rust*  
*SBI*



library for implementing a specific platform SBI



solves the abstraction issue based on Rust `embedded-hal`

*This library adapts to embedded Rust's `embedded-hal` crate to provide basic SBI features. When building for own platform, implement traits in this library and pass them to the functions begin with `init`.*



# RustSBI

*Rust*  
*SBI*



library for implementing a specific platform SBI



solves the abstraction issue based on Rust `embedded-hal`

*This library adapts to embedded Rust's `embedded-hal` crate to provide basic SBI features. When building for own platform, implement traits in this library and pass them to the functions begin with `init`.*

When handlers are set up, call `enter_privileged` to enter the OS in S-Mode.



# RustSBI

*Rust*  
*SBI*



library for implementing a specific platform SBI



solves the abstraction issue based on Rust `embedded-hal`

*This library adapts to embedded Rust's `embedded-hal` crate to provide basic SBI features. When building for own platform, implement traits in this library and pass them to the functions begin with `init`.*

When handlers are set up, call `enter_privileged` to enter the OS in S-Mode.

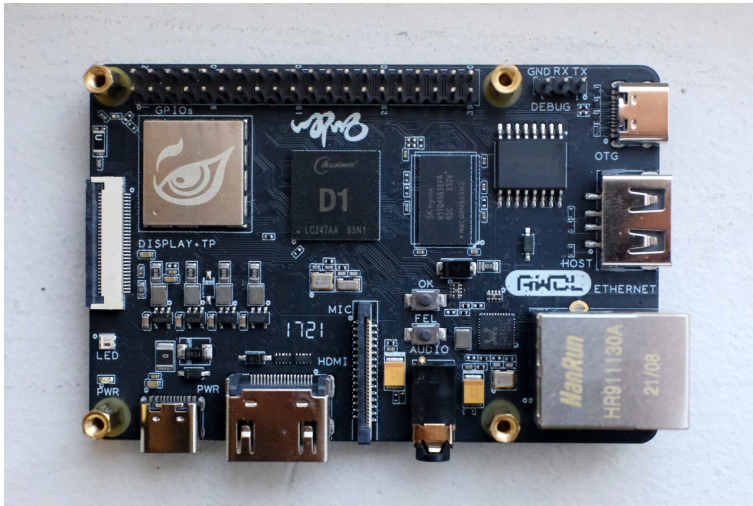
<https://docs.rs/rustsbi/latest/rustsbi/>



Porting sunxi/nezha



# Allwinner Nezha Board



[https://linux-sunxi.org/Allwinner\\_Nezha](https://linux-sunxi.org/Allwinner_Nezha)



Allwinner D1



# Allwinner D1

The mask ROM loads a blob from SPI NOR/NAND, eMMC or SD card into SRAM (32K). It has to start with a specific header.



# Allwinner D1

The mask ROM loads a blob from SPI NOR/NAND, eMMC or SD card into SRAM (32K). It has to start with a specific header.

## SRAM Stage

Boot from flash and DRAM init were prototyped separately:

<https://github.com/luojia65/test-d1-flash-bare/>



# Allwinner D1

The mask ROM loads a blob from SPI NOR/NAND, eMMC or SD card into SRAM (32K). It has to start with a specific header.

## SRAM Stage

Boot from flash and DRAM init were prototyped separately:

<https://github.com/luojia65/test-d1-flash-bare/>

It has been copied and developed further in oreboot:

`src/mainboard/sunxi/nezha/bt0`



# Allwinner D1

The mask ROM loads a blob from SPI NOR/NAND, eMMC or SD card into SRAM (32K). It has to start with a specific header.

## SRAM Stage

Boot from flash and DRAM init were prototyped separately:

<https://github.com/luojia65/test-d1-flash-bare/>

It has been copied and developed further in oreboot:

`src/mainboard/sunxi/nezha/bt0`

## Payload Stage



# Allwinner D1

The mask ROM loads a blob from SPI NOR/NAND, eMMC or SD card into SRAM (32K). It has to start with a specific header.

## SRAM Stage

Boot from flash and DRAM init were prototyped separately:

<https://github.com/luojia65/test-d1-flash-bare/>

It has been copied and developed further in oreboot:

`src/mainboard/sunxi/nezha/bt0`

## Payload Stage

First RustSBI implementation in oreboot is for Allwinner Nezha (D1):

`src/mainboard/sunxi/nezha/main`



# Allwinner D1

The mask ROM loads a blob from SPI NOR/NAND, eMMC or SD card into SRAM (32K). It has to start with a specific header.

## SRAM Stage

Boot from flash and DRAM init were prototyped separately:

<https://github.com/luojia65/test-d1-flash-bare/>

It has been copied and developed further in oreboot:

```
src/mainboard/sunxi/nezha/bt0
```

## Payloader Stage

First RustSBI implementation in oreboot is for Allwinner Nezha (D1):

```
src/mainboard/sunxi/nezha/main
```

## Status

We can boot Linux. \o/



# Allwinner D1

The mask ROM loads a blob from SPI NOR/NAND, eMMC or SD card into SRAM (32K). It has to start with a specific header.

## SRAM Stage

Boot from flash and DRAM init were prototyped separately:

<https://github.com/luojia65/test-d1-flash-bare/>

It has been copied and developed further in oreboot:

```
src/mainboard/sunxi/nezha/bt0
```

## Payloader Stage

First RustSBI implementation in oreboot is for Allwinner Nezha (D1):

```
src/mainboard/sunxi/nezha/main
```

## Status

We can boot Linux. \o/

And xv6, MnemOS, r9 - with even more to come.



# RustSBI in oreboot: example

```
pub fn sbi_exec(payload_offset: usize, dtb_offset: usize) -> ! {  
    let hartid = riscv::register::mhartid::read();  
    init_pmp();  
    init_csrs();  
    runtime::init();  
    if hartid == 0 {  
        init_plic();  
        peripheral::init_peripheral();  
    }  
    delegate_interrupt_exception();  
    // NOTE: This sets up handlers for SBI calls and traps  
    execute_supervisor(payload_offset, hartid, dtb_offset)  
}
```



# Credits



## Credits

We are very grateful for **RustSBI**. It makes our life a lot easier. :)  
Mr Yang Derui, Vivian Wang and Luo Jia helped us out to get going.



# Credits

We are very grateful for **RustSBI**. It makes our life a lot easier. :)  
Mr Yang Derui, Vivian Wang and Luo Jia helped us out to get going.

Paul Ruizendaal translated the DRAM init from an assembly dump to C, which we in turn based our Rust implementation on. Mimoja wrote and ran a full DRAM test ranging over the entire address space.



## Credits

We are very grateful for **RustSBI**. It makes our life a lot easier. :)  
Mr Yang Derui, Vivian Wang and Luo Jia helped us out to get going.

Paul Ruizendaal translated the DRAM init from an assembly dump to C, which we in turn based our Rust implementation on. Mimoja wrote and ran a full DRAM test ranging over the entire address space.

Samuel Holland took the effort to pick up Allwinner's BSP, work out changes for mainline, and upstream patches to U-Boot, OpenSBI and Linux, joined our Nezha online community meetup, and continuously helped us out when we had questions or ran into issues.



# Credits

We are very grateful for **RustSBI**. It makes our life a lot easier. :)  
Mr Yang Derui, Vivian Wang and Luo Jia helped us out to get going.

Paul Ruizendaal translated the DRAM init from an assembly dump to C, which we in turn based our Rust implementation on. Mimoja wrote and ran a full DRAM test ranging over the entire address space.

Samuel Holland took the effort to pick up Allwinner's BSP, work out changes for mainline, and upstream patches to U-Boot, OpenSBI and Linux, joined our Nezha online community meetup, and continuously helped us out when we had questions or ran into issues.

Michael Engel ported xv6 to the D1, which made a nice and small test payload.



# Credits

We are very grateful for **RustSBI**. It makes our life a lot easier. :)  
Mr Yang Derui, Vivian Wang and Luo Jia helped us out to get going.

Paul Ruizendaal translated the DRAM init from an assembly dump to C, which we in turn based our Rust implementation on. Mimoja wrote and ran a full DRAM test ranging over the entire address space.

Samuel Holland took the effort to pick up Allwinner's BSP, work out changes for mainline, and upstream patches to U-Boot, OpenSBI and Linux, joined our Nezha online community meetup, and continuously helped us out when we had questions or ran into issues.

Michael Engel ported xv6 to the D1, which made a nice and small test payload.

The D1 Mainline Telegram group with all its members had always been supportive.

Special shoutout to Pierce who kept believing in our success!



# Things We Messed Up: Off-by-one **at scale**



## Things We Messed Up: Off-by-one **at scale**

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 256, o as u8 % 256]);
```



## Things We Messed Up: Off-by-one **at scale**

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 256, o as u8 % 256]);
```

Rust analyzer says something about 256 being too much for u8.



## Things We Messed Up: Off-by-one **at scale**

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 256, o as u8 % 256]);
```

Rust analyzer says something about 256 being too much for u8.

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 255, o as u8 % 255]);
```



## Things We Messed Up: Off-by-one **at scale**

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 256, o as u8 % 256]);
```

Rust analyzer says something about 256 being too much for u8.

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 255, o as u8 % 255]);
```

Kernel won't come up. Some long hours at OSF hackathon reveal:

We are off by ff00.



## Things We Messed Up: Off-by-one **at scale**

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 256, o as u8 % 256]);
```

Rust analyzer says something about 256 being too much for u8.

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8 % 255, o as u8 % 255]);
```

Kernel won't come up. Some long hours at OSF hackathon reveal:

We are off by ff00.

```
let b = f.copy_into([(o >> 16) as u8, (o >> 8) as u8, o as u8]);
```

After another hour of looking again, we managed to figure it out and learned that Rust already drops the higher bits for foo as u8.



# Things We Messed Up: Cache coherence



# Things We Messed Up: Cache coherence

```
Unable to handle kernel paging request at virtual address ffffffff92f
Oops [#1]
Modules linked in:
CPU: 0 PID: 103 Comm: kworker/0:0 Not tainted 5.19.0-rc1-14138-g4e54
Hardware name: Sipeed Lichee RV Dock (DT)
epc : wq_worker_running+0xa/0x56
   ra : wq_worker_running+0xa/0x56
epc : ffffffff80018b68 ra : ffffffff80018b68 sp : fffffffc804113e40
```



# Things We Messed Up: Cache coherence

```
Unable to handle kernel paging request at virtual address ffffffff92f
Oops [#1]
Modules linked in:
CPU: 0 PID: 103 Comm: kworker/0:0 Not tainted 5.19.0-rc1-14138-g4e54
Hardware name: Sipeed Lichee RV Dock (DT)
epc : wq_worker_running+0xa/0x56
   ra : wq_worker_running+0xa/0x56
epc : ffffffff80018b68 ra : ffffffff80018b68 sp : fffffffc804113e40
```

Well, that wasn't us, but it took a long time to find it.



# Things We Messed Up: Cache coherence

```
Unable to handle kernel paging request at virtual address ffffffff92f
Oops [#1]
Modules linked in:
CPU: 0 PID: 103 Comm: kworker/0:0 Not tainted 5.19.0-rc1-14138-g4e54
Hardware name: Sipeed Lichee RV Dock (DT)
epc : wq_worker_running+0xa/0x56
  ra : wq_worker_running+0xa/0x56
epc : ffffffff80018b68 ra : ffffffff80018b68 sp : fffffffc804113e40
```

Well, that wasn't us, but it took a long time to find it.

<https://github.com/rust-embedded/riscv/pull/107>

*Fix reading marchid and mimpid*

*It seems to have been a copy-paste error. We ran into a very nasty bug in oreboot on the Allwinner D1 (C906) where an errata patch in Linux relies on those two being zero.*



Awesome Demo



Thank you!

