oreboot on RISC-V
Comparing Implementations on Two Platforms
Daniel Maslowski
Agenda

- Introduction
- Allwinner D1
- StarFive JH7100
- Approaches and Future Work
What is oreboot again?
What is oreboot again?

*oreboot is a fork of coreboot...*
What is oreboot again?

oreboot is a fork of coreboot...

From aa246f71df9900a30d938ab618c46839436616
From: "Ronald G. Minnich" <rminnich@gmail.com>
Date: Mon, 1 Apr 2019 23:48:56 +0000
Subject: [PATCH] Initial removal of C code

https://github.com/oreboot/oreboot
Firmware in Rust

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

Rust logo under CC BY 4.0, https://github.com/rust-lang/rust-artwork

Ferris the crab from https://rustacean.net/
Firmware Development

Feels like an RPG. You need to figure out how things work. Point on map = program counter (PC). Sometimes you have no clue where you are. World map = memory map. The islands or worlds are the peripherals. Player's guide = processor/SoC manual. It may be incomplete or not at all available (at least to you). Internet of Things = MMORPG. Yes, it can get very dangerous.
Firmware Development

feels like an RPG
You need to figure out how things work.
Firmware Development

feels like an RPG
You need to figure out how things work.

point on map = program counter (PC)
Sometimes you have no clue where you are.

Internet of Things = MMORPG
Yes, it can get very dangerous.
Firmware Development

feels like an RPG
You need to figure out how things work.

point on map = program counter (PC)
Sometimes you have no clue where you are.

world map = memory map
The islands or worlds are the peripherals.
Firmware Development

**feels like an RPG**
You need to figure out how things work.

**point on map = program counter (PC)**
Sometimes you have no clue where you are.

**world map = memory map**
The islands or worlds are the peripherals.

**player’s guide = processor/SoC manual**
It may be incomplete or not at all available (at least to you).
Firmware Development

feels like an RPG
You need to figure out how things work.

point on map = program counter (PC)
Sometimes you have no clue where you are.

world map = memory map
The islands or worlds are the peripherals.

player’s guide = processor/SoC manual
It may be incomplete or not at all available (at least to you).

Internet of Things = MMORPG
Yes, it can get very dangerous.
Single Board Computers

Many are marketed as open-source. Are they though?

Documentation, schematics, and board design manuals and instructions open license Source Code open tools for flashing, debugging, and image composition firmware, from the start, documented (U-Boot, orboot, …) Linux or other OS, mainline friendly (git fork, not sourcedump) all code usable with upstream toolchains, or provide toolchains in a reproducible form (not only binaries for a specific architecture/OS)

OSHWACertification: https://certification.oshwa.org/
Single Board Computers

Many are *marketed* as open source. Are they though?
Single Board Computers

Many are *marketed* as open source. Are they though?

**Documentation**
- *schematics* and board design
- manuals and instructions
- open *license*

[OSHWACertification: https://certification.oshwa.org/]
Single Board Computers

Many are marketed as open source. Are they though?

**Documentation**
- schematics and board design
- manuals and instructions
- open license

**Source Code**
- open tools for flashing, debugging and image composition
- firmware, from the start, documented (U-Boot, oreboot, …)
- Linux or other OS, mainline friendly (git fork, not source dump)
- all code usable with upstream toolchains, or provide toolchains in a reproducible form (not only binaries for a specific architecture/OS)

OSHWACertification: https://certification.oshwa.org/
Single Board Computers

Many are *marketed* as open source. Are they though?

**Documentation**
- *schematics* and board design
- manuals and instructions
- open *license*

**Source Code**
- *open* tools for flashing, debugging and image composition
- firmware, *from the start*, documented (U-Boot, oreboot, …)
- Linux or other OS, *mainline friendly* (git fork, *not* source dump)
- all code usable with upstream toolchains, or provide toolchains in a *reproducible* form (not only binaries for a specific architecture/OS)

OSHWA Certification: https://certification.oshwa.org/
Let’s look at a manual and a memory map!
Allwinner D1
D1 SoC

Production widely produced and easily available on many different boards from various vendors.

- **1x C906 core, 1GHz**
  - [https://github.com/T-head-Semi/openc906](https://github.com/T-head-Semi/openc906)

- **1x low-power core, Xtensa HiFi4**

Documentation:
- Largemanual provided
  - About 1400 pages

DRAM controller and HDMi missing.
D1 SoC

Production
- widely produced and easily available
- many different boards from various vendors

Cores
- 1xC906 core, 1GHz
- 1x low-power core, Xtensa HiFi4

Documentation
- large manual provided
- about 1400 pages

https://github.com/T-head-Semi/openc906

DRAM controller and HDMI missing
D1 SoC

Production
- widely produced and easily available
- many different boards from various vendors

Cores
- 1x C906 core, 1GHz
  - https://github.com/T-head-Semi/openc906
- 1x low-power core, Xtensa HiFi4
D1 SoC

Production
- widely produced and easily available
- many different boards from various vendors

Cores
- 1x C906 core, 1GHz
  - https://github.com/T-head-Semi/openc906
- 1x low-power core, Xtensa HiFi4

Documentation
- larger manual provided
  - about 1400 pages
- DRAM controller and HDMI missing
D1 Boards and SoMs

Allwinner Nezha
first board; Raspberry Pi form factor

DongshanPi Nezha STU
- SoM in custom form factor
- carrier board with many pins

ClockworkPi R01
- SoM in RPi CM 3 form factor
- DevTerm carrier board + case

MangoPi MQ-Pro
Raspberry Pi Zero form factor, drop-in replacement

https://linux-sunxi.org/Category:D1_Boards
D1 Boards: Lichee RV + Dock

TheregularDockhassolderjointsforaSPIflash,soIaddedone.
TheDockProalreadyhasa16MiBSPIflash,plusaUSBserialconverter.
The regular Dock has solder joints for a SPI flash, so I added one.
The regular Dock has solder joints for a SPI flash, so I added one.
The Dock Pro already has a 16 MiB SPI flash, plus a USB serial converter.
D1 Boot Behavior

Mask ROM

It loads a blob from SPI flash, eMMC or SD card into SRAM (32K).

The blob must start with a specific eGON header:

Disassembly of section .head:

0000000000020000 <head_jump>:
  20000: a5 a0 j 0x20068 <start+0x8>
  20002: 00 00 unimp

0000000000020004 <_ZN17oreboot_nezha_BT09EGON_HEAD17h5aa4b41b712905f2E>:
  20004: 65 47 4f 4e 2e 42 54 30 eGON.BT0
  2000c: 39 6c 0a 5f 00 00 00 00 9l._....

Note: The header is not documented in the manual.

SPI flash

We need to actively read from SPI flash and have no MMIO access to it.
D1 Boot Behavior

**Mask ROM**

It loads a blob from SPI flash, eMMC or SD card into SRAM (32K).

Note: The header is not documented in the manual.

SPI flash

We need to actively read from SPI flash and have no MMIO access to it.
D1 Boot Behavior

**Mask ROM**

It loads a blob from SPI flash, eMMC or SD card into SRAM (32K). The blob has to start with a specific eGON header:

```
00000000 00020000 <head_jump>:
20000: a5 a0 j 0x20068 <start+0x8>
20002: 00 00 unimp

00000000 00020004 <_ZN17oreboot_nezha_bt09EGON_HEAD17h5aa4b41b712905f2E>:
20004: 65 47 4f 4e 2e 42 54 30 eGON.BT0
2000c: 39 6c 0a 5f 00 00 00 00 9l._....
```

Note: The header is not documented in the manual.
D1 Boot Behavior

**Mask ROM**

It loads a blob from SPI flash, eMMC or SD card into SRAM (32K). The blob has to start with a specific eGON header:

Disassembly of section .head:

```
0000000000020000 <head_jump>:
  20000: a5 a0 j 0x20068 <start+0x8>
  20002: 00 00 unimp

0000000000020004 <ZN17oreboot_nezha_bt09EGON_HEAD17h5aa4b41b712905f:
  20004: 65 47 4f 4e 2e 42 54 30 eGON.BT0
  2000c: 39 6c 0a 5f 00 00 00 00 9l....
```

Note: The header is not documented in the manual.
D1 Boot Behavior

**Mask ROM**
It loads a blob from SPI flash, eMMC or SD card into SRAM (32K). The blob has to start with a specific eGON header:

Disassembly of section .head:

```
0000000000020000 <head_jump>:
  20000: a5 a0 j 0x20068 <start+0x8>
  20002: 00 00 unimp
```

```
0000000000020004 <_ZN17oreboot_nezha_bt09EGON_HEAD17h5aa4b41b712905f2E>:
  20004: 65 47 4f 4e 2e 42 54 30 eGON.BT0
  2000c: 39 6c 0a 5f 00 00 00 00 9l._....
```

Note: The header is not documented in the manual.

**SPI flash**
We need to actively read from SPI flash and have no MMIO access to it.
D1 oreboot Flow

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 Rust SBI implementation in oreboot is for Allwinner Nezha (D1):
src/mainboard/sunxi/nezha/main
SRAM Stage

Boot from flash and DRAM init were prototyped separately: https://github.com/luojia65/test-d1-flash-bare/
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
D1 oreboot Flow

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
D1 oreboot Flow

**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
D1 DRAM init

Remember: The DRAM controller is not documented in the manual, only its existence.

A bit less than 2000 lines of code

▶ Translated from C, which was translated from assembly dump

There are configurations per board. They could technically be determined at runtime.

Community has knowledge on how it works

▶ Some documentation on https://linux-sunxi.org

I tried my best to name registers; reviews and help wanted!

We may be able to reuse at least parts and apply them to other Allwinner SoCs.
D1 DRAM init

Remember: The DRAM controller is not documented in the manual, only its existence.
D1 DRAM init

Remember: The DRAM controller is not documented in the manual, only its existence.

- a bit less than 2000 lines of code
  - translated from C, which was translated from assembly dump

There are configurations per board. They could technically be determined at runtime. Some documentation is available at https://linux-sunxi.org. I tried my best to name the registers; reviews and help wanted! We may be able to reuse at least parts and apply them to other Allwinner SoCs.
Remember: The DRAM controller is not documented in the manual, only its existence.

- a bit less than 2000 lines of code
  - translated from C, which was translated from assembly dump

There are configurations per board. They could technically be determined at runtime.
D1 DRAM init

Remember: The DRAM controller is not documented in the manual, only its existence.

• a bit less than 2000 lines of code
  ▶ translated from C, which was translated from assembly dump

There are configurations per board. They could technically be determined at runtime.

• community has knowledge on how it works
  ▶ some documentation on https://linux-sunxi.org
D1 DRAM init

Remember: The DRAM controller is not documented in the manual, only its existence.

- a bit less than 2000 lines of code
  - translated from C, which was translated from assembly dump

There are configurations per board. They could technically be determined at runtime.

- community has knowledge on how it works
  - some documentation on https://linux-sunxi.org

I tried my best to name registers; reviews and help wanted!

We may be able to reuse at least parts, and apply them to other Allwinner SoCs.
D1 Development Status

- Available in main branch
- Boot from SPI flash
- SD card work in progress
- SBI is optional
- Multiple boards supported
- We can boot Linux or xv6, MnemOS r9
- More to come: FreeBSD, Illumos…
- Baremetal testing app
- https://github.com/adamgreig/d1rgb
D1 Development Status

- available in main branch
- boot from SPI flash
- SD card work in progress
- SBI is optional
- multiple boards supported

Wecan boot Linux/xv6/MnemOS
r9
more to come: FreeBSD, Illumos…
Baremetal testing app
https://github.com/adamgreig/d1rgb
D1 Development Status

- available in main branch
- boot from SPI flash
- SD card work in progress
- SBI is optional
- multiple boards supported

We can boot
- Linux \o/
- xv6
- MnemOS
- r9

more to come: FreeBSD, Illumos…

Baremetal testing app
https://github.com/adamgreig/d1rgb
D1 Development Status

- available in main branch
- boot from SPI flash
- SD card work in progress
- SBI is optional
- multiple boards supported

We can boot
- Linux \o/
- xv6
- MnemOS
- r9

more to come: FreeBSD, Illumos…

Bare metal testing app
https://github.com/adamgreig/d1rgb
StarFive JH7100
JH7100 SoC

Production no longer produced, but the successor JH7110 SoC appears to be using the same DRAM controller and parts, judging from very similar vendor code cores 2xU74 core, >1GHz

▶ https://sifive.cdn.prismic.io/sifive/ad5577a0-9a00-45c9-a5d0-424a3d586060_u74_core_complex_manual_21G3.pdf

1xVP6 Documentation

No full manual publicly available instructions for firmware recovery with open tools sparse datasheet with list of peripheral blocks and suppliers (p23)

▶ https://github.com/starfive-tech/JH7100_docs
JH7100 SoC

Production

no longer produced, *but* the successor JH7110 SoC appears to be using the same DRAM controller and parts, judging from very similar vendor code.
JH7100 SoC

Production
no longer produced, but the successor JH7110 SoC appears to be using the same DRAM controller and parts, judging from very similar vendor code

Cores

- 2x U74 core, >1GHz
  - [https://sifive.cdn.prismic.io/sifive/ad5577a0-9a00-45c9-a5d0-424a3d586060_u74_core_complex_manual_21G3.pdf](https://sifive.cdn.prismic.io/sifive/ad5577a0-9a00-45c9-a5d0-424a3d586060_u74_core_complex_manual_21G3.pdf)

- 1x VP6

Documentation

- no full manual publicly available
- instructions for firmware recovery with open tools
- sparse datasheet with list of peripheral blocks and suppliers (p23)

- [https://github.com/starfive-tech/JH7100_docs](https://github.com/starfive-tech/JH7100_docs)
JH7100 SoC

Production
no longer produced, *but* the successor JH7110 SoC appears to be using the same DRAM controller and parts, judging from very similar vendor code

Cores

- 2x U74 core, >1GHz
- 1x VP6

Documentation

- no full manual publicly available
- instructions for firmware recovery with open tools
- sparse datasheet with list of peripheral blocks and suppliers (p23)
- less than 140 pages

https://github.com/starfive-tech/JH7100_docs
only a select few people received the board as a prototype
obtained via RISC-V International developer program
https://riscv.org/risc-v-developer-boards/details/
JH7100 Boards

BeagleV

only a select few people received the board as a prototype
JH7100 Boards

BeagleV

only a select few people received the board as a prototype

StarFive VisionFive 1

obtained via RISC-V International developer program
https://riscv.org/risc-v-developer-boards/details/
JH7100 Development Stream
Live: https://twitch.tv/cyrevolt

Archive: https://www.youtube.com/playlist?list=PLenOHeTI_A9PSGshDnEc4dYK-GSnCshk6
Porting oreboot to the VisionFive1 board / JH7100 SoC

- MMIO mapped
  - mask ROM
  - second boot
  - DDR init
  - OpenSBI
  - U-Boot proper

- 128K SRAM 1
  - mask ROM
  - oreboot b0
  - DDR init
  - OpenSBI
  - U-Boot proper

- 128K SRAM 2
  - mask ROM
  - oreboot b0
  - DDR init
  - OpenSBI
  - U-Boot proper

- 8G DRAM
  - mask ROM
  - oreboot b0
  - DDR init
  - OpenSBI
  - LinuxBoot

JH7100 Vendor Code and Transition Plan
JH7100 Boot Behavior

Mask ROM not documented; can be dumped via preflashed U-Boot. It loads a blob to SRAM, which has to be prefixed with a 4-byte value for its size.

SPI flash MMIO access to the SPI flash is available, requiring little initialization. This means that loading from flash is just like copying from one area in memory to another.

Multicore Note: Multiple cores allows for accessing peripherals in parallel. Strategy: loop second hart; when done with peripherals, jump to OS.
JH7100 Boot Behavior

**Mask ROM**

not documented; can be dumped via prefished U-Boot
JH7100 Boot Behavior

**Mask ROM**
not documented; can be dumped via prefushed U-Boot
It loads a blob to SRAM, which has to be prefixed with a 4-byte value for its size.
JH7100 Boot Behavior

**Mask ROM**
not documented; can be dumped via prefetched U-Boot
It loads a blob to SRAM, which has to be prefixed with a 4-byte value for its size.

**SPI flash**
JH7100 Boot Behavior

Mask ROM
not documented; can be dumped via prefetched U-Boot
It loads a blob to SRAM, which has to be prefixed with a 4-byte value for its size.

SPI flash
MMIO access to the SPI flash is available, requiring little initialization.
JH7100 Boot Behavior

**Mask ROM**
not documented; can be dumped via preflected U-Boot
It loads a blob to SRAM, which has to be prefixed with a 4-byte value for its size.

**SPI flash**
MMIO access to the SPI flash is available, requiring little initialization. This means that loading from flash is just like copying from one area in memory to another.
JH7100 Boot Behavior

**Mask ROM**
not documented; can be dumped via prefleshed U-Boot
It loads a blob to SRAM, which has to be prefixed with a 4-byte value for its size.

**SPI flash**
MMIO access to the SPI flash is available, requiring little initialization. This means that loading from flash is just like copying from one area in memory to another.

**Multicore**
Note: Multiple cores allow for accessing peripherals in parallel.
Mask ROM
not documented; can be dumped via preflecteded U-Boot
It loads a blob to SRAM, which has to be prefixed with a 4-byte value for its size.

SPI flash
MMIO access to the SPI flash is available, requiring little initialization.
This means that loading from flash is just like copying from one area in memory to another.

Multicore
Note: Multiple cores allows for accessing peripherals in parallel.
Strategy: loop second hart; when done with peripherals, jump to OS.
DRAM init

We started with a case study, analyzing how the vendor code works. https://github.com/starfive-tech/JH7100_ddrinit

More than 4000 lines. Quite some registers have comments. Lots of magic values, little logic. More than 50% is just writing 0, probably unnecessary.

We found and fixed a double bug in the vendor code: writing back to the wrong register. We reported the issue, with no reply so far. https://github.com/starfive-tech/JH7100_ddrinit/issues/14
DRAM init

We started with a case study, analyzing how the vendor code works.

https://github.com/starfive-tech/JH7100_ddrinit
We started with a case study, analyzing how the vendor code works.

https://github.com/starfive-tech/JH7100_ddrinit

- more than 4000 lines
- quite some registers have comments
- lots of magic values, little logic
- more than 50% is just writing 0, probably unnecessary
We started with a case study, analyzing how the vendor code works.

https://github.com/starfive-tech/JH7100_ddrinit

- more than 4000 lines
- quite some registers have comments
- lots of magic values, little logic
- more than 50% is just writing 0, probably unnecessary

We found and fixed a double bug in the vendor code:

writing back to the wrong register
We started with a case study, analyzing how the vendor code works.

https://github.com/starfive-tech/JH7100_ddrinit

- more than 4000 lines
- quite some registers have comments
- lots of magic values, little logic
- more than 50% is just writing 0, probably unnecessary

We found and fixed a double bug in the vendor code:

writing back to the wrong register

We reported the issue, with no reply so far.

https://github.com/starfive-tech/JH7100_ddrinit/issues/14
JH7100 Development Status

Pull request open with DRAM init and jump to next stage
https://github.com/oreboot/oreboot/pull/606

We currently load and jump to the U-Boot + OpenSBI blob
► which can then load Linux, e.g., via network
Approaches and Future Work
Talking to peripherals

A register block is a set of registers that maps to a block in the SoC. The start is called the base register.

```rust
const CCU_BASE: usize = 0x0200_1000;
const CCU_PLL_PERI0_CTRL: usize = CCU_BASE + 0x0020;
```

MMIO (memory-mapped input/output) Writing to a peripheral register happens through a memory write instruction. To change a value, read first, apply a mask, and write back. Example:

```rust
unsafe {
    let peri0_ctrl = read_volatile(CCU_PLL_PERI0_CTRL as *mut u32);
    let new_val = peri0_ctrl | 1 << 29; // set bit `29`
    write_volatile(CCU_PLL_PERI0_CTRL as *mut u32, new_val);
}
```
Talking to peripherals

Register Blocks

A register block is a set of registers that maps to a block in the SoC. The start is called the base register.

```rust
const CCU_BASE: usize = 0x0200_1000;
const CCU_PLL_PERI0_CTRL: usize = CCU_BASE + 0x0020;

MMIO (memory-mapped input/output) writing to a peripheral register happens through a memory write instruction. To change a value, read first, apply a mask, and write back. Example:

```rust
unsafe {
    let peri0_ctrl = read_volatile(CCU_PLL_PERI0_CTRL as *mut u32);
    let new_val = peri0_ctrl | 1 << 29; // set bit `29`
    write_volatile(CCU_PLL_PERI0_CTRL as *mut u32, new_val);
}
```
Talking to peripherals

**Register Blocks**

A register block is a set of register that maps to a block in the SoC.
Talking to peripherals

Register Blocks

A register block is a set of register that maps to a block in the SoC. The start is called the *base register*. 

```rust
const CCU_BASE: usize = 0x0200_1000;
const CCU_PLL_PERI0_CTRL: usize = CCU_BASE + 0x0020;

MMIO (memory-mapped input/output) Writing to a peripheral register happens through a memory write instruction. To change a value, read first, apply a mask, and write back. Example:

```unsafe {
    let peri0_ctrl = read_volatile(CCU_PLL_PERI0_CTRL as *mut u32);
    let new_val = peri0_ctrl | 1 << 29;
    // set bit `29`
    write_volatile(CCU_PLL_PERI0_CTRL as *mut u32, new_val);
}
Talking to peripherals

Register Blocks

A register block is a set of register that maps to a block in the SoC. The start is called the *base register*.

```rust
const CCU_BASE: usize = 0x0200_1000;
const CCU_PLL_PERIO_CTRL: usize = CCU_BASE + 0x0020;
```
Talking to peripherals

Register Blocks
A register block is a set of register that maps to a block in the SoC. The start is called the \textit{base register}.

\begin{verbatim}
const CCU_BASE: usize = 0x0200_1000;
const CCU_PLL_PERI0_CTRL: usize = CCU_BASE + 0x0020;
\end{verbatim}

MMIO (memory-mapped input/output)
Writing to a peripheral register happens through a memory write instruction.
Talking to peripherals

**Register Blocks**

A register block is a set of registers that maps to a block in the SoC. The start is called the *base register*.

```rust
const CCU_BASE: usize = 0x0200_1000;
const CCU_PLL_PERI0_CTRL: usize = CCU_BASE + 0x0020;
```

**MMIO (memory-mapped input/output)**

Writing to a peripheral register happens through a memory write instruction. To change a value, read first, apply a mask, and write back.
Talking to peripherals

Register Blocks
A register block is a set of register that maps to a block in the SoC. The start is called the base register.

```rust
const CCU_BASE: usize = 0x0200_1000;
const CCU_PLL_PERI0_CTRL: usize = CCU_BASE + 0x0020;
```

MMIO (memory-mapped input/output)
Writing to a peripheral register happens through a memory write instruction. To change a value, read first, apply a mask, and write back.

Example:
```rust
unsafe {
    let peri0_ctrl = read_volatile(CCU_PLL_PERI0_CTRL as *mut u32);
    let new_val = peri0_ctrl | 1 << 29; // set bit `29`
    write_volatile(CCU_PLL_PERI0_CTRL as *mut u32, new_val);
}
```
SVD -> PAC -> HAL

SVD -> PAC -> HAL


https://docs.rs/d1-pac/latest/d1_pac
SVD -> PAC -> HAL

https://docs.rs/d1-pac/latest/d1_pac

Image under CC BY 4.0
SVD -> PAC -> HAL


https://docs.rs/d1-pac/latest/d1_pac

Layers in oreboot
1. App (mainboard)
2. HAL (“drivers”)
3. PAC (if available)
SVD -> PAC -> HAL

https://docs.rs/d1-pac/latest/d1_pac

Layers in oreboot
1. App (mainboard)
2. HAL (“drivers”)
3. PAC (if available)
PACs are commonly generated from SVD files.

Using a peripheral access crate (PAC)

Instead of using `write_volatile` directly, we call a semantic function from a library:

```rust
// light up led
let mut pb5 = gpio.portb.pb5.into_output();
pb5.set_high().unwrap();

Depending on the API, we may need to use a writer interface and pass a function:

```ccu.smhc0_clk.write(|w| w.clk_src_sel().pll_peri_1x());```
Using a peripheral access crate (PAC)

Instead of using `write_volatile` directly, we call a *semantic function* from a library:
Instead of using `write_volatile` directly, we call a *semantic function* from a library:

```rust
// light up led
let mut pb5 = gpio.portb.pb5.into_output();
pb5.set_high().unwrap();
```
Using a peripheral access crate (PAC)

Instead of using `write_volatile` directly, we call a semantic function from a library:

```rust
// light up led
let mut pb5 = gpio.portb.pb5.into_output();
pb5.set_high().unwrap();
```

Depending on the API, we may need to use a writer interface and pass a function:
Using a peripheral access crate (PAC)

Instead of using `write_volatile` directly, we call a *semantic function* from a library:

```rust
// light up led
let mut pb5 = gpio.portb.pb5.into_output();
pb5.set_high().unwrap();
```

Depending on the API, we may need to use a writer interface and pass a function:

```rust
ccu.smhc0_clk.write(distance | w.clk_src_sel().pll_peri_1x());
```
RISC-V Runtime Services

Runtime Services are listed in platform specs, referencing the SBI spec.

https://github.com/riscv/riscv-platform-specs

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.

Ports need to be written per platform (core/SoC/board). We have one for the D1.

In a reboot, we use Rust SBI.

https://github.com/rustsbi/rustsbi
https://docs.rs/rustsbi/latest/rustsbi/
RISC-V Runtime Services

*Runtime Services* are listed in **platform specs**, referencing the **SBI spec**. https://github.com/riscv/riscv-platform-specs
RISC-V Runtime Services

Runtime Services are listed in platform specs, referencing the SBI spec. https://github.com/riscv/riscv-platform-specs

The SBI (Supervisor Binary Interface) spec is a living document: https://github.com/riscv-non-isa/riscv-sbi-doc
RISC-V Runtime Services

*Runtime Services* are listed in [platform specs](https://github.com/riscv/riscv-platform-specs), referencing the [SBI spec](https://github.com/riscv/riscv-platform-specs). The SBI (*Supervisor Binary Interface*) spec is a living document: [https://github.com/riscv-non-isa/riscv-sbi-doc](https://github.com/riscv-non-isa/riscv-sbi-doc)

It defines extensions and functions similar to system calls.
RISC-V Runtime Services

*Runtime Services* are listed in **platform specs**, referencing the **SBI spec**.
https://github.com/riscv/riscv-platform-specs

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.

Ports need to be written per platform (core/SoC/board). We have one for the D1.
RISC-V Runtime Services

Runtime Services are listed in platform specs, referencing the SBI spec.
https://github.com/riscv/riscv-platform-specs

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.

Ports need to be written per platform (core/SoC/board). We have one for the D1.

In ooreboot, we use RustSBI.
https://github.com/rustsbi/rustsbi
https://docs.rs/rustsbi/latest/rustsbi/
A note on RISC-V customizability

The platform specs define sets of instructions necessary in order to run an OS. There are Control and Status Registers (CSRs), similar to x86 MSRs. They allow for vendor-specific custom extensions. They may be required for full usage of the SoC. Vendors may also implement custom instructions. Both CSRs and custom instructions may be neglected, and a subset of the SoC’s capabilities be used.
A note on RISC-V customizability

The platform specs define sets of instructions necessary in order to run an OS.
A note on RISC-V customizability

The platform specs define sets of instructions necessary in order to run an OS.

There are *Control and Status Registers (CSRs)*, similar to x86 MSRs.
A note on RISC-V customizability

The platform specs define sets of instructions necessary in order to run an OS.

There are *Control and Status Registers (CSRs)*, similar to x86 MSRs.

They allow for vendor specific custom extensions.
A note on RISC-V customizability

The platform specs define sets of instructions necessary in order to run an OS.

There are *Control and Status Registers (CSRs)*, similar to x86 MSRs.

They allow for vendor specific custom extensions.

They *may* be required for full usage of the SoC.
A note on RISC-V customizability

The platform specs define sets of instructions necessary in order to run an OS.

There are *Control and Status Registers (CSRs)*, similar to x86 MSRs. They allow for vendor specific custom extensions. They *may* be required for full usage of the SoC. Vendors may also implement custom instructions.
A note on RISC-V customizability

The platform specs define sets of instructions necessary in order to run an OS.

There are *Control and Status Registers (CSRs)*, similar to x86 MSRs.
They allow for vendor specific custom extensions.

They *may* be required for full usage of the SoC.

Vendors may also implement custom instructions.

Both CSRs and custom instructions may be neglected, and a subset of the SoC’s capabilities be used.
Potential RISC-V SoCs for oreboot

- BL808 already available (I have multiple boards)
- 1xC906 (512MHz), 1xE907, 1x low-power core JH7110 already available; some people received theirs from crowdfunding campaign marketed as open source
- No manual available
- Initial U-Boot and Linux sources available
  - https://github.com/starfive-tech/Tools (is closed source)

TH1520 SoM with SPI flash placeholder coming soon
- Multiple boards offered
  - https://sipeed.com/licheepi4a
  - LicheePi4A
  - LicheeCluster4A
  - LicheeRouter4A
  - LicheePad/Phone4A
Potential RISC-V SoCs for oreboot

**BL808**
- already available (I have multiple boards)
- 1x C906 (512MHz), 1x E907, 1x low-power core
Potential RISC-V SoCs for oreboot

**BL808**
- already available (I have multiple boards)
- 1x C906 (512MHz), 1x E907, 1x low-power core

**JH7110**
- already available; some people received theirs from crowdfunding campaign
- *marketed as open source*
- no manual available
- initial U-Boot and Linux sources available
- https://github.com/starfive-tech/Tools is closed source
Potential RISC-V SoCs for reboot

**BL808**
- already available (I have multiple boards)
- 1x C906 (512MHz), 1x E907, 1x low-power core

**JH7110**
- already available; some people received theirs from crowdfunding campaign
- marketed as open source
- no manual available
- initial U-Boot and Linux sources available
- https://github.com/starfive-tech/Tools

**TH1520**
- SoM with SPI flash placeholder coming soon
- multiple boards offered
  - https://sipeed.com/licheepi4a
    - Lichee Pi 4A
    - Lichee Cluster 4A
    - Lichee Router 4A
    - Lichee Pad/Phone 4A
Further Work

Other ideas: add SBoM, xtask.

Rust build framework used in oreboot needs extension with more boards and common functions factored out for ARM and other ISAs.

We had some ARM and x86 boards and discarded them in favor of getting on.

However, there are issues tracking the status with starting points.

https://github.com/oreboot/oreboot/issues
Further Work

**layoutflash**

- library within oreboot
- idea: DTS for flash partitioning
- other ideas: add SBoM

[xtask](https://github.com/oreboot/oreboot/issues)

Rust build framework used in oreboot needs extension with more boards and common functions factored out for ARM and other ISAs. We had some ARM and x86 boards and discarded them in favor of getting on. However, there are issues tracking their status with starting points.
Further Work

**layoutflash**
- library within oreboot
- idea: DTS for flash partitioning
- other ideas: add SBoM

**xtask**
Rust build framework used in oreboot
needs extension with more boards and common functions factored out
Further Work

**layoutflash**
- library within oreboot
- idea: DTS for flash partitioning
- other ideas: add SBoM

**xtask**
Rust build framework used in oreboot
needs extension with more boards and common functions factored out

**ARM and other ISAs**
We had some ARM and x86 boards and discarded them in favor of getting on.
However, there are issues tracking their status with starting points.
https://github.com/oreboot/oreboot/issues
Follow Me

https://github.com/orangecms
https://twitter.com/orangecms
https://twitch.tv/cyrevolt
https://youtube.com/@cyrevolt

Daniel Maslowski

https://github.com/oreboot/oreboot

https://metaspora.org/oreboot-comparison-riscv-d1-jh7100.pdf