



HardenedLinux

Firmware Freedom

coreboot for RISC-V

Xiang Wang & Shawn Chang





Self intro

HardenedLinux

- Xiang Wang
- Day job at TYA infotech
- Member of HardenedLinux





Self intro

HardenedLinux

- Shawn C[a.k.a "citypw"]
- Day job at TYA infotech
 - Open source security consulting
- GNU/Linux security engineer
- Free/libre SW/FW/HW enthusaists
- Member of EFF/FSF/FSFE/RISC-V
- Patient Zer0 at HardenedLinux community(<https://hardenedlinux.github.io/>)





HardenedLinux

Agenda

- What's wrong with x86 firmware
- Intro to RISC-V & coreboot
- Porting story
- Conclusion





HardenedLinux

Wait, which CORE exactly?

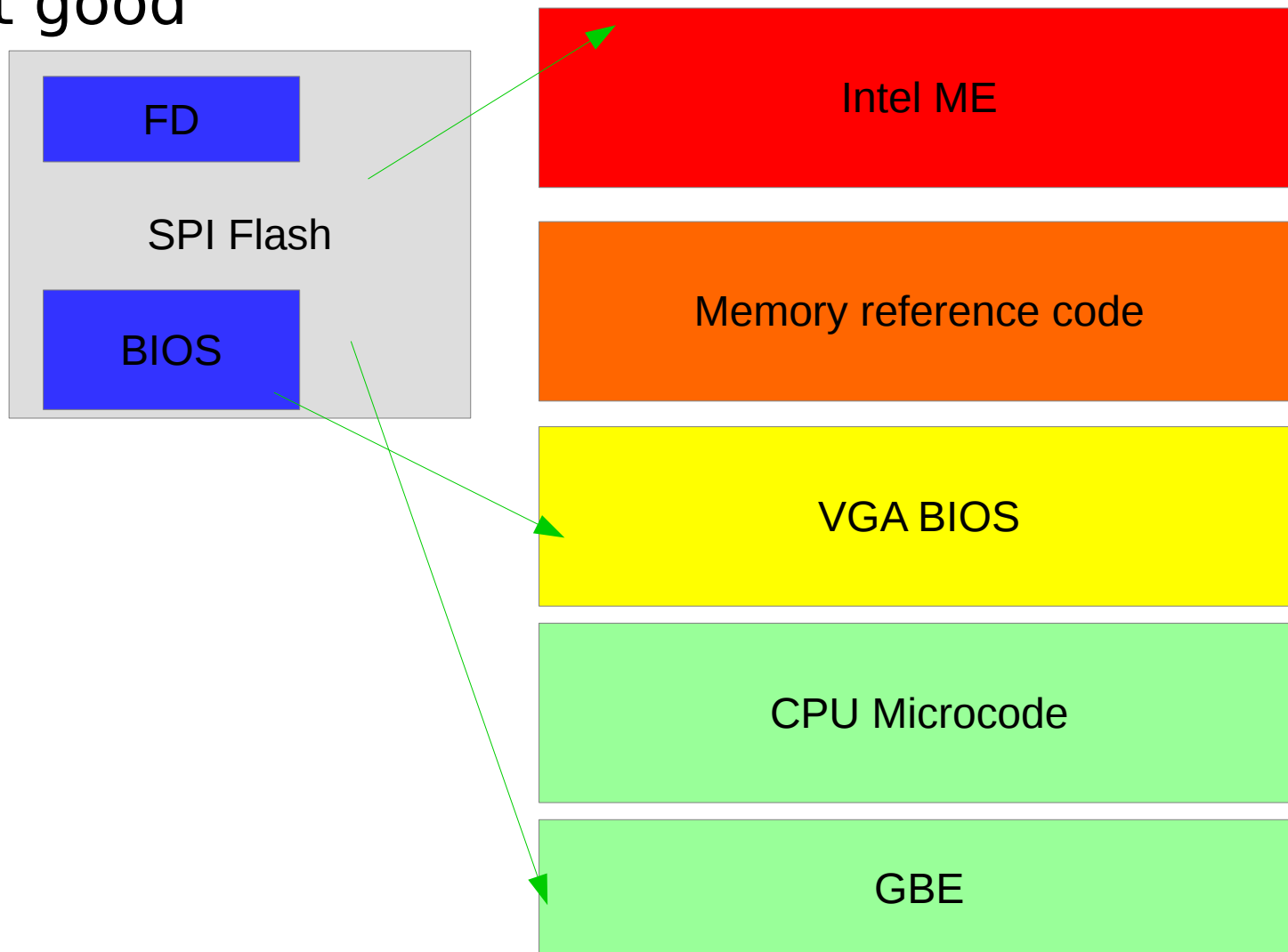
RING 0 is **not** the CORE anymore





From a libre FW's perspective

Not good





HardenedLinux

Summary

- [Hardening the COREs] solution has its limits in x86
- Too many things can not be audited





Intro to RISC-V

- Pronounced "risk-five".
- Design by University of California, Berkeley.
- Open ISA under BSD license.
- RISC architecture
- load-store type
- conditional jump without status register
- Variable length of instruction encoding
- Instruction set modular

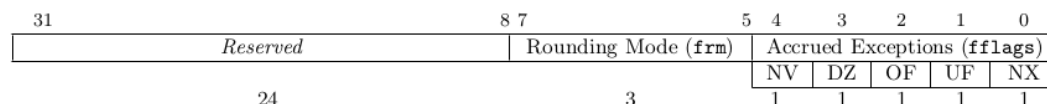




Intro to RISC-V

register

| | | | |
|-----------|---|--------|---|
| XLEN-1 | 0 | FLEN-1 | 0 |
| x0 / zero | | f0 | |
| x1 | | f1 | |
| x2 | | f2 | |
| x3 | | f3 | |
| x4 | | f4 | |
| x5 | | f5 | |
| x6 | | f6 | |
| x7 | | f7 | |
| x8 | | f8 | |
| x9 | | f9 | |
| x10 | | f10 | |
| x11 | | f11 | |
| x12 | | f12 | |
| x13 | | f13 | |
| x14 | | f14 | |
| x15 | | f15 | |
| x16 | | f16 | |
| x17 | | f17 | |
| x18 | | f18 | |
| x19 | | f19 | |
| x20 | | f20 | |
| x21 | | f21 | |
| x22 | | f22 | |
| x23 | | f23 | |
| x24 | | f24 | |
| x25 | | f25 | |
| x26 | | f26 | |
| x27 | | f27 | |
| x28 | | f28 | |
| x29 | | f29 | |
| x30 | | f30 | |
| x31 | | f31 | |
| XLEN | | FLEN | |
| XLEN-1 | 0 | 31 | 0 |
| pc | | fcsr | |
| XLEN | | 32 | |



| Rounding Mode | Mnemonic | Meaning |
|---------------|----------|--|
| 000 | RNE | Round to Nearest, ties to Even |
| 001 | RTZ | Round towards Zero |
| 010 | RDN | Round Down (towards $-\infty$) |
| 011 | RUP | Round Up (towards $+\infty$) |
| 100 | RMM | Round to Nearest, ties to Max Magnitude |
| 101 | | <i>Invalid. Reserved for future use.</i> |
| 110 | | <i>Invalid. Reserved for future use.</i> |
| 111 | | In instruction's <i>rm</i> field, selects dynamic rounding mode; In Rounding Mode register, <i>Invalid</i> . |

| Flag Mnemonic | Flag Meaning |
|---------------|-------------------|
| NV | Invalid Operation |
| DZ | Divide by Zero |
| OF | Overflow |
| UF | Underflow |
| NX | Inexact |



Intro to RISC-V

Variable length of instruction encoding

| | | | |
|--------------------|--------------------|---------------------------|------------------------------------|
| xxxxxxxxxxxxxxxxaa | | | 16-bit ($aa \neq 11$) |
| xxxxxxxxxxxxxxxxxx | xxxxxxxxxxxxbbb11 | 32-bit ($bbb \neq 111$) | |
| ···xxxx | xxxxxxxxxxxxxxxxxx | xxxxxxxxxx011111 | 48-bit |
| ···xxxx | xxxxxxxxxxxxxxxxxx | xxxxxxxxxx011111 | 64-bit |
| ···xxxx | xxxxxxxxxxxxxxxxxx | xnnnxxxxx111111 | $(80+16*nnn)$ -bit, $nnn \neq 111$ |
| ···xxxx | xxxxxxxxxxxxxxxxxx | x111xxxxx111111 | Reserved for ≥ 192 -bits |

Byte Address: base+4 base+2 base





Intro to RISC-V

Base Instruct Formats

| | | | | | | | | | | | | | | | | | | |
|------------|----|-----------|----|-----|-----|---------|-----|------------|--------|--------|----------|----------|--------|---------|--------|--------|--|--------|
| 31 | 30 | 25 | 24 | 21 | 20 | 19 | 15 | 14 | 12 | 11 | 8 | 7 | 6 | 0 | | | | |
| funct7 | | | | rs2 | | | rs1 | | funct3 | | rd | | | opcode | | R-type | | |
| imm[11:0] | | | | | | rs1 | | funct3 | | rd | | | opcode | | I-type | | | |
| imm[11:5] | | | | rs2 | | | rs1 | | funct3 | | imm[4:0] | | | opcode | | S-type | | |
| imm[12] | | imm[10:5] | | | rs2 | | | rs1 | | funct3 | | imm[4:1] | | imm[11] | | opcode | | B-type |
| imm[31:12] | | | | | | | | | | rd | | | opcode | | | U-type | | |
| imm[20] | | imm[10:1] | | | | imm[11] | | imm[19:12] | | | rd | | | opcode | | J-type | | |





HardenedLinux

Intro to RISC-V

Privilege levels & Privileged Software Stack

| Level | Encoding | Name | Abbreviation |
|-------|----------|------------------|--------------|
| 0 | 00 | User/Application | U |
| 1 | 01 | Supervisor | S |
| 2 | 10 | <i>Reserved</i> | |
| 3 | 11 | Machine | M |

| Number of levels | Supported Modes | Intended Usage |
|------------------|-----------------|---|
| 1 | M | Simple embedded systems |
| 2 | M, U | Secure embedded systems |
| 3 | M, S, U | Systems running Unix-like operating systems |

| |
|-------------|
| Application |
| ABI |
| AEE |

| | |
|-------------|-------------|
| Application | Application |
| ABI | ABI |
| OS | |
| SBI | |
| SEE | |

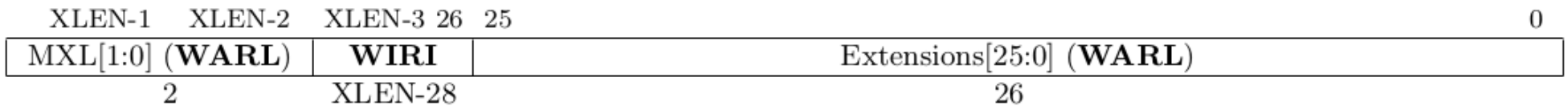
| | | | |
|-------------|-------------|-------------|-------------|
| Application | Application | Application | Application |
| ABI | ABI | ABI | ABI |
| OS | | OS | |
| SBI | | SBI | |
| Hypervisor | | | |
| HBI | | | |
| HEE | | | |





Intro to RISC-V

Instruction set modular



| MXL | XLEN |
|-----|------|
| 1 | 32 |
| 2 | 64 |
| 3 | 128 |

- RISC-V can be implemented as 32/64/128-bit architecture.
- Only Integer instruction set must be implemented.
- Instruction info can be read from **misa** CSR.

| Bit | Character | Description |
|-----|-----------|--|
| 0 | A | Atomic extension |
| 1 | B | <i>Tentatively reserved for Bit operations extension</i> |
| 2 | C | Compressed extension |
| 3 | D | Double-precision floating-point extension |
| 4 | E | RV32E base ISA |
| 5 | F | Single-precision floating-point extension |
| 6 | G | Additional standard extensions present |
| 7 | H | <i>Reserved</i> |
| 8 | I | RV32I/64I/128I base ISA |
| 9 | J | <i>Tentatively reserved for Dynamically Translated Languages extension</i> |
| 10 | K | <i>Reserved</i> |
| 11 | L | <i>Tentatively reserved for Decimal Floating-Point extension</i> |
| 12 | M | Integer Multiply/Divide extension |
| 13 | N | User-level interrupts supported |
| 14 | O | <i>Reserved</i> |
| 15 | P | <i>Tentatively reserved for Packed-SIMD extension</i> |
| 16 | Q | Quad-precision floating-point extension |
| 17 | R | <i>Reserved</i> |
| 18 | S | Supervisor mode implemented |
| 19 | T | <i>Tentatively reserved for Transactional Memory extension</i> |
| 20 | U | User mode implemented |
| 21 | V | <i>Tentatively reserved for Vector extension</i> |
| 22 | W | <i>Reserved</i> |
| 23 | X | Non-standard extensions present |
| 24 | Y | <i>Reserved</i> |
| 25 | Z | <i>Reserved</i> |



HardenedLinux

Intro to coreboot

- Began in winter of 1999 in LANL(Los Alamos National Laboratory)
- formerly known as LinuxBIOS
- Armed at replacing proprietary firmware
- Open source under GPLv2
- Platform support : x86,ARM,MIPS,RISC-V and so on
- Written in mostly C, and about 1% in assembly.





Design of coreboot for RISC-V

- The start up divided into multiple stage.
 - Bootblock stage : initialize flash memory
 - Rom stage : initialize memory and chipset
 - Ram stage : set environment for payload
 - Payload : It can be Linux kernel image (including Heads) or stand-alone EFI executable, such as iPXE, gPXE, GRUB, SeaBIOS, TianoCore and so on.
- Each stage is separate program
- bootblock is located at the reset entry of SoC.
- Other stage loaded by previous stage (which could be utilized by HardenedBoot).





Design of coreboot for RISC-V

bootblock stage

- First, there are a little assembly code to initialize stack point, exception and interrupt.
- Then jmp to frame code(src/lib/bootblock.c), the code export four interface.
 - void bootblock_soc_early_init(void);
 - void bootblock_mainboard_early_init(void);
 - void bootblock_soc_init(void);
 - void bootblock_mainboard_init(void);





HardenedLinux

Design of coreboot for RISC-V

rom stage

- Do nothing but to load and boot next stage.





Design of coreboot for RISC-V

ram stage

- Ram stage have a frame code(src/lib/hardwaremain.c).
- Ram stage divided into 12 steps, which is defined by **boot_state_t**.

```
typedef enum {  
    BS_PRE_DEVICE, BS_DEV_INIT_CHIPS, BS_DEV_ENUMERATE,  
    BS_DEV_RESOURCES, BS_DEV_ENABLE, BS_DEV_INIT,  
    BS_POST_DEVICE, BS_OS_RESUME_CHECK, BS_OS_RESUME,  
    BS_WRITE_TABLES, BS_PAYLOAD_LOAD, BS_PAYLOAD_BOOT,  
} boot_state_t;
```

- Each step is describe by **boot_state**.
- There are three sets of callback functions in boot_state, through which each step is divided into three phases.

Entry callbacks

State Actions

Exit callbacks

- [**State Actions**] defined by system, which mainly used to process device tree.



Design of coreboot for RISC-V

ram stage

- Callbacks can be describe by **boot_state_init_entry**. So we can insert the desired operation in a specific location.
- **boot_state_schedule_static_entries** function used to associate boot_state and callbacks.
- **bs_walk_state_machine** function used to iterate **boot_state** array and perform callbacks step by step.





Design of coreboot for RISC-V

struct of firmware file

- Top layout of firmware file is some continuous blocks.
- One of blocks records the layout of blocks.
- This block holds two structures that describe layout information

```
struct fmap {  
    /* "__FMAP__" (0x5F5F464D41505F5F) */  
    uint8_t  signature[8];  
    uint8_t  ver_major;  
    uint8_t  ver_minor;  
    /* address of the firmware binary */  
    uint64_t base;  
    /* size of firmware binary in bytes */  
    uint32_t size;  
    /* name of this firmware binary */  
    uint8_t  name[FMAP_STRLEN];  
    uint16_t nareas;  
    struct fmap_area areas[];  
} __attribute__((packed));
```

```
struct fmap_area {  
    /* offset relative to base */  
    uint32_t offset;  
    /* size in bytes */  
    uint32_t size;  
    /* descriptive name */  
    uint8_t  name[FMAP_STRLEN];  
    /* flags for this area */  
    uint16_t flags;  
} __attribute__((packed));
```

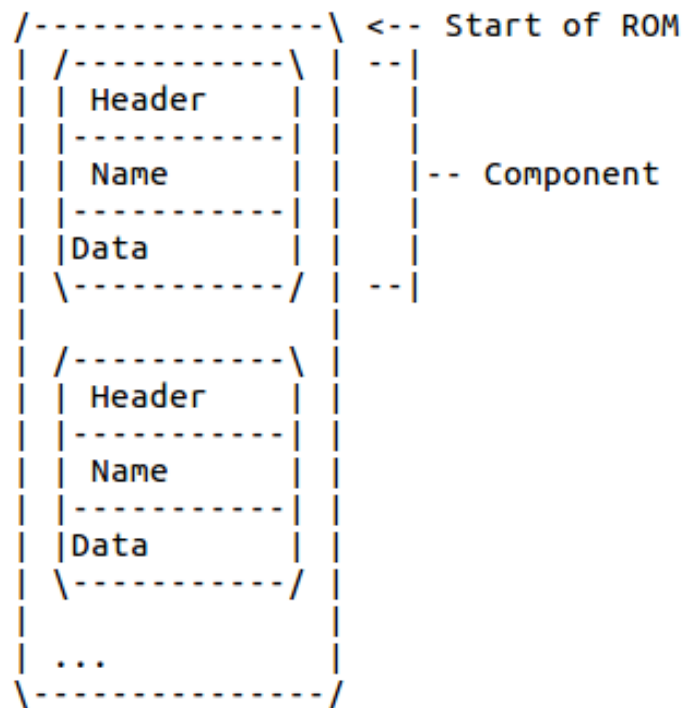




Design of coreboot for RISC-V

struct of firmware file

- A block named **COREBOOT** used to store cbfs.
- **cbfs** consists of components.





Design of coreboot for RISC-V

struct of firmware file

- Component structure as shown.

```
struct cbfs_file {  
    uint8_t magic[8];  
    uint32_t len;  
    uint32_t type;  
    uint32_t attributes_offset;  
    uint32_t offset;  
    char filename[];  
} __PACKED;
```

| | |
|---------|-----------------------------|
| /-----\ | <- start |
| Header | |
| ----- | <- sizeof(struct cbfs_file) |
| Name | |
| ----- | <- 'offset' |
| Data | |
| ... | |
| \-----/ | <- start + 'offset' + 'len' |



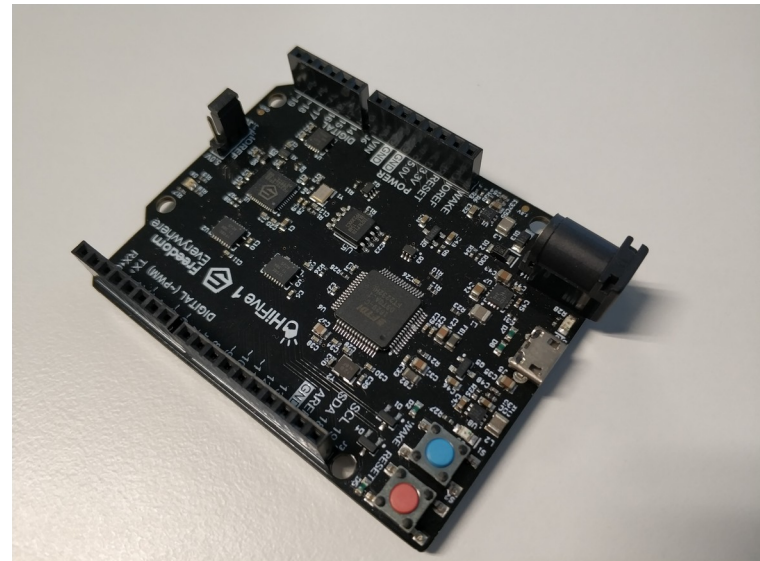


HardenedLinux

Porting story

board info

- Board: HiFive1
- SoC: SiFive Freedom E310 (FE310)
 - Architecture: RV32IMAC
 - Speed: 320MHz
 - Memory: 16KB instruction cache, 16KB data scratchpad
- Flash Memory: 16Mbit SPI Flash
- Host Interface(microUSB FTD2232HL): Program, Debug and Serial Communication
- SPI controller
- External Interrupt pin: 19
- External Wakeup pin: 1
- Dimensions: 68 mm x 51 mm
- Weight: 22g





HardenedLinux

Porting story

Problem & Scheme

- Problem
 - Source of coreboot base on RV64
 - Ram is too little, can't load firmware to memory
- Solution
 - Modify build option to support RV32.
 - Delete firmware file structure. Layout program directly in flash.





HardenedLinux

Porting story

build option

- Create directory :
 - src/soc/sifive
 - src/mainboard/sifive
- Create **Kconfig** in soc directory, add build option in it.
- Modify src/arch/riscv/Makefile.inc pass macro to toolchain.

```
config SOC_SIFIVE_E300
.....
bool
default n
if SOC_SIFIVE_E300
    config RISCV_ARCH
        string
        default "rv32imac"
    config RISCV_ABI
        string
        default "ilp32"
endif
```

```
riscv_flags = ... -march=$(CONFIG_RISCV_ARCH) -mabi=$(CONFIG_RISCV_ABI)
riscv_asm_flags = -march=$(CONFIG_RISCV_ARCH) -mabi=$(CONFIG_RISCV_ABI)
```





Porting story

memory layout

- Linker script (src/lib/program.ld) and memory layout (src/include/memlayout.h) limit code and data segments must be adjacent.
- FE310 program run in flash memory, not closely data segments.

```
#define RAM_START    0x80000000
#define FLASH_START  0x20400000
SECTIONS
{
    BOOTBLOCK(      FLASH_START, 64K)
    ROMSTAGE ( FLASH_START + 64K, 64K)
    RAMSTAGE (FLASH_START + 128K, 64K)

    SRAM_START      (RAM_START)
    STACK           (RAM_START + 4K, 4K)
    PAGETABLES      (RAM_START + 8K, 4K)
    PRERAM_CBMEM_CONSOLE (RAM_START + 12K, 4K)
    SRAM_END        (RAM_START + 16k)
}
```



HardenedLinux

Porting story

console support

- Reference BSP code and src/console/console.c
- Define macro HAVE_UART_SPECIAL
- Then implement five functions

```
void    uart_init(int idx);  
uint8_t uart_rx_byte(int idx);  
void    uart_tx_byte(int idx,unsigned char data);  
void    uart_tx_flush(int idx);  
void    uart_fill_lb(void *data);
```





HardenedLinux

Porting story

jump to next stage

- Because know where each stage is in memory, so can use jump instructions to jump directly to the next stage.
- Can use function pointer to do this in C language.
- Ex: `((void (*)(void))0x20410000)();`





Porting story

link

- Convert each stage to binary by **objcopy**.
- Pack binary to ELF

```
.global _start
.section .text
_start:
.incbin "bootblock.bin"
```

- link

```
ENTRY(_start)
SECTIONS {
    .text 0x20400000 : {
        . = 0x00000;
        bootblock-raw.o(.text)
        . = 0x10000;
        romstage-raw.o(.text)
        . = 0x20000;
        ramstage-raw.o(.text)
        . = 0x30000;
        hello-raw.o(.text)
    }
}
```





HardenedLinux

Porting story

Download

➤ Start **openocd**

```
openocd -f openocd.cfg >/dev/null 2>&1 &
```

➤ Download to board by **gdb**

```
riscv64-unknown-elf-gdb coreboot-raw.elf \  
--batch \  
-ex "set remotetimeout 500" \  
-ex "target extended-remote localhost:3333" \  
-ex "monitor reset halt" \  
-ex "monitor flash protect 0 64 last off" \  
-ex "load" \  
-ex "monitor resume" \  
-ex "monitor shutdown" \  
-ex "quit" && echo Successfully uploaded coreboot-raw.elf
```





HardenedLinux

Porting story

running screenshorts

```
minicom x

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

Press CTRL-A Z for help on special keys

core freq at 266420224 Hz
hart 0: HLS is 80001fc0
Time is 00000000 and timecmp is 00000000

coreboot-586246c Thu Aug  3 10:42:19 UTC 2017 bootblock starting...

coreboot-586246c Thu Aug  3 10:42:19 UTC 2017 romstage starting...

coreboot-586246c Thu Aug  3 10:42:19 UTC 2017 ramstage starting...

loading payload...

core freq at 266649600 Hz
hello world!

Progam has exited with code:0x00000000
```





HardenedLinux

Porting story

board info

- Board: HiFive-Unleadshed
- SoC: SiFive Freedom U540 (FU540)
 - World's fastest RISC-V Processor
 - World's only GNU/Linux-capable RISC-V SoC
 - Architecture: 4xRV64GC + 1xRV64IMAC
 - Speed: 1.5GHz
 - Cache: Coherent 2M L2 Cache
- Memory: 8G DDR4 with ECC
- Flash: 32MB Quad SPI flash
- MicroSD Card
- Gigabit Ethernet Port





Porting story original boot procrss

- SoC has 4 pins called **MSEL** to choose where boot loader is.
- ZSBL(Zeroth Stage Boot Loader) is stored in the mask ROM of SoC. Download FSBL from a partition with GUID type **5B193300-FC78-40CD-8002-E86C45580B47**.
- FSBL download BBL from a partition with GUID type **2E54B353-1271-4842-806F-E436D6AF69851**.

| MSEL | Reset address | Purpose |
|------|---------------|------------------------------------|
| 0000 | 0x0000_1004 | loops forever waiting for debugger |
| 0001 | 0x2000_0000 | memory-mapped QSPI0 |
| 0010 | 0x3000_0000 | memory-mapped QSPI1 |
| 0011 | 0x4000_0000 | uncached ChipLink |
| 0100 | 0x6000_0000 | cached ChipLink |
| 0101 | 0x0001_0000 | ZSBL |
| 0110 | 0x0001_0000 | ZSBL |
| 0111 | 0x0001_0000 | ZSBL |
| 1000 | 0x0001_0000 | ZSBL |
| 1001 | 0x0001_0000 | ZSBL |
| 1010 | 0x0001_0000 | ZSBL |
| 1011 | 0x0001_0000 | ZSBL |
| 1100 | 0x0001_0000 | ZSBL |
| 1101 | 0x0001_0000 | ZSBL |
| 1110 | 0x0001_0000 | ZSBL |
| 1111 | 0x0001_0000 | ZSBL |

| MSEL | FSBL | BBL | Purpose |
|------|---------|---------|---|
| 0000 | - | - | loops forever waiting for debugger |
| 0001 | - | - | jump directly to 0x2000_0000 (memory-mapped SPI0) |
| 0010 | - | - | jump directly to 0x3000_0000 (memory-mapped SPI1) |
| 0011 | - | - | jump directly to 0x4000_0000 (uncached ChipLink) |
| 0100 | - | - | jump directly to 0x6000_0000 (cached ChipLink) |
| 0101 | SPI0 x1 | SPI0 x1 | - |
| 0110 | SPI0 x4 | SPI0 x4 | Rescue image from flash (preprogrammed) |
| 0111 | SPI1 x4 | SPI1 x4 | - |
| 1000 | SPI1 SD | SPI1 SD | - |
| 1001 | SPI2 x1 | SPI2 x1 | - |
| 1010 | SPI0 x4 | SPI1 SD | - |
| 1011 | SPI2 SD | SPI2 SD | Rescue image from SD card |
| 1100 | SPI1 x1 | SPI2 SD | - |
| 1101 | SPI1 x4 | SPI2 SD | - |
| 1110 | SPI0 x1 | SPI2 SD | - |
| 1111 | SPI0 x4 | SPI2 SD | Default boot mode |



HardenedLinux

Porting story

our target

- The original boot process:

```
+-----+ +-----+ +-----+ +-----+  
| MSEL |--->| ZSBL |--->| FSBL |--->| BBL |  
+-----+ +-----+ +-----+ +-----+
```

- Our target should looks like this:

```
+-----+ +-----+ +-----+  
| MSEL |--->| ZSBL |--->| coreboot |  
+-----+ +-----+ +-----+
```





Porting story

Problem & Scheme

- Problem
 - Different hart instruction set are different
 - PPL, clock and DDR controler initialization code is not open source.
 - Not support SMP
- Solution
 - Modify compiler options to support the smallest instruct set.
 - Welcome to the party: Reverse-engineering the “evil” blob(FSBL).





HardenedLinux

Porting story

build option

- Management Core RV64IMAC
- Application Cores RV64GC → RV64IMAFDC
- To support the smallest instruct set.

```
ifeq ($(CONFIG_SOC_SIFIVE_FU540),y)
    riscv_flags = -I$(src)/arch/riscv/ -mmodel=medany -march=rv64imac -mabi=lp64
    riscv_asm_flags = -march=rv64imac -mabi=lp64
    COMPILER_RT_bootblock = $(top)/util/crossgcc/xgcc/lib/gcc/riscv64-elf/6.3.0/rv64imac/lp64/libgcc.a
    COMPILER_RT_romstage = $(top)/util/crossgcc/xgcc/lib/gcc/riscv64-elf/6.3.0/rv64imac/lp64/libgcc.a
    COMPILER_RT_ramstage = $(top)/util/crossgcc/xgcc/lib/gcc/riscv64-elf/6.3.0/rv64imac/lp64/libgcc.a
endif
```





Porting story

SMP support

- Initialize stack point for each hart.

```
# initialize stack point for each hart
la    t0, _stack
slli  t1, a0, RISCVP_GSHIF
add   t0, t0, t1
li    t1, 0xDEADBEEF
sd    t1, 0(t0)
li    t1, RISCVP_GSIZE - HLS_SIZE
add   sp, t0, t1
```

0xDEADBEEF it used to check stack overflow.

- Stack is continuous page-aligned memory.
- Hart-local storage(HLS) is a block memory at top of stack.
- HLS can be access by follow code.

```
#define MACHINE_STACK_TOP() ({ \
    register uintptr_t sp asm ("sp"); \
    (void*)((sp + RISCVP_GSIZE) & -RISCVP_GSIZE); })
```

```
#define HLS() ((hls_t*)(MACHINE_STACK_TOP() - HLS_SIZE))
#define OTHER_HLS(id) ((hls_t*)((void*)HLS() + RISCVP_GSIZE * ((id) - HLS()->hart_id)))
```





HardenedLinux

Porting story

SMP support

- Add a field to HLS used this memory to communicate between harts.

```
void select_bp(void) {  
    static int cnt;  
    const void* entry;  
    int hartid = read_csr(mhartid);  
    const void *fdt = (const void *)read_csr(mscratch);
```

```
    HLS()->entry = NULL;
```

```
    if(atomic_add(&cnt, 1) == 0)  
        return;
```

```
    do {  
        entry = HLS()->entry;  
        mb();  
    }while(!entry);
```

```
    printk(BIOS_SPEW, "resume %d hart, fdt at %p, entry at %p\n",hartid,fdt,entry);  
    HLS()->entry(hartid,fdt);
```

```
}
```

```
void wake_other_hart(void (*entry)(int,const void*)) {  
    for(int i = 0; i < 5; i++) {  
        OTHER_HLS(i)->entry = entry;  
        mb();  
    }  
}
```





HardenedLinux

Porting story

use coreboot instead of BBL

- Original code of coreboot adds FSBL GPT partition head for to bootblock. Remove this (src/soc/sifive/fu540/Makefile.inc).
- Partition SD card in advance. Retain partition for coreboot, which partition type must be **2E54B353-1271-4842-806F-E436D6AF69851**.

```
sgdisk \
    --clear \
    --new=1:2048:67583 --change-name=1:bootloader \
        --typecode=1:2E54B353-1271-4842-806F-E436D6AF6985 \
    --new=2:264192:      --change-name=2:root \
        --typecode=2:0FC63DAF-8483-4772-8E79-3D69D8477DE4 \
    $(DISK)
```
- Then used **bb1** compiled by BSP as payload.
- Finally write coreboot.rom to BBL partition through dd.





HardenedLinux

What need firmware to do

- Hardware-related initialization
- Initialize SBI(supervisor binary interface)





Interface of SBI

- Trigger SBI via **ecall** instruction
- a0,a1 and a2 registers is used to parameter passing
- a7 register is used to select function
- a0 used to pass back the result





Functions of SBI

- SBI function used by current linux

```
#define SBI_SET_TIMER 0
#define SBI_CONSOLE_PUTCHAR 1
#define SBI_CONSOLE_GETCHAR 2
#define SBI_CLEAR_IPI 3
#define SBI_SEND_IPI 4
#define SBI_REMOTE_FENCE_I 5
#define SBI_REMOTE_SFENCE_VMA 6
#define SBI_REMOTE_SFENCE_VMA_ASID 7
#define SBI_SHUTDOWN 8
```





HardenedLinux

SBI_SET_TIMER

- It used to set timed events.

- Appear in

`drivers/clocksource/riscv_timer.c`

```
static int next_event(unsigned long delta, struct clock_event_device *ce);
```





HardenedLinux

SBI_CONSOLE_PUTCAHR SBI_CONSOLE_GETCHAR

- It used to early printk. Appear in

```
arch/riscv/kernel/setup.c
```

```
static void sbi_console_write(struct console *co, const char *buf, unsigned int n);  
struct console riscv_sbi_early_console_dev;  
void __init setup_arch(char **cmdline_p);
```

```
kernel/printk/printk.c
```

```
struct console *early_console;  
asmlinkage __visible void early_printk(const char *fmt, ...);
```

- It used for tty driver. Appear in

```
drivers/tty/hvc/hvc_riscv_sbi.c
```





HardenedLinux

SBI_SEND_IPI

- It used to scheduling hart. Appear in

`arch/riscv/kernel/smp.c`





HardenedLinux

SBI_SEND_IPI

- It used to scheduling hart. Appear in

`arch/riscv/kernel/smp.c`





HardenedLinux

sbi_remote_fence_i
sbi_remote_sfence_vma
sbi_remote_sfence_vma_asid

- It used to operate cache. Appear in

`arch/riscv/include/asm/cacheflush.h`

`arch/riscv/include/asm/tlbflush.h`





Conclusion

- From firmware freedom's perspective, x86 is pretty much dead unless Intel open everything which is unlikely.
- Firmware freedom can be benefits from the nature of RISC-V.
- We should support free/libre firmware projects like coreboot in the 1st place.





References

- <https://riscv.org/>
- <https://coreboot.org/>
- <https://www.sifive.com/products/hifive1/>
- <https://www.sifive.com/products/hifive-unleashed/>
- Try Harder 2 Hardening the COREs
- Fuck the binary of FSBL
-





Q&A

HardenedLinux



HardenedLinux

