coreboot on RISCV

Ron Minnich, Google

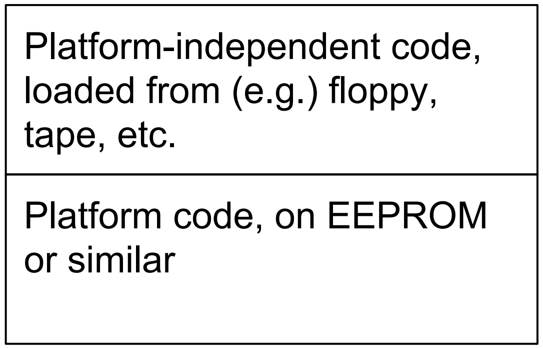
Thanks to Stefan Reinauer, Duncan Laurie, Patrick Georgi, ...

Overview

- What firmware is
- What coreboot is
- Why we want it on RISCV
- History of the port
- Structure of the port
- Status
- Lessons learned

Firmware, 1974-present, always-on

- Bottom half of the operating system
- Provided an abstract interface (Basic Input Output System, or BIOS) to top half
- Supported DOS, CP/M, etc.
- Sucked
  - Slow
  - No easy bugfix path
  - Not SMP capable

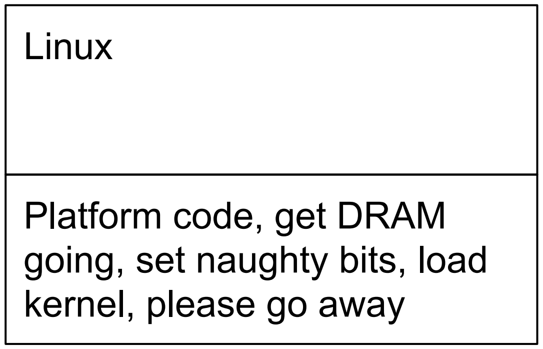


Platform-independent code, loaded from (e.g.) floppy, tape, etc.

Platform code, on EEPROM or similar

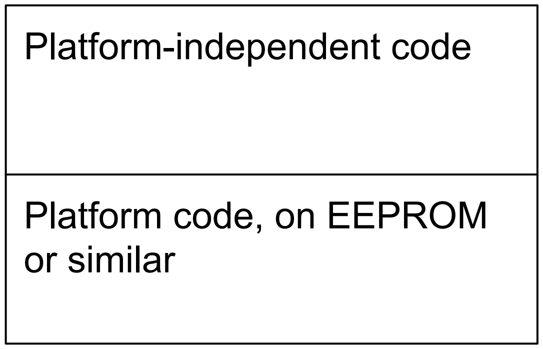
Firmware, 1990-2005, “Fire and Forget”

- Just set up bootloader and get out of the way
- Set all the stuff kernels can’t do
  - Magic configuration, etc.
  - Even now, Linux can not do most of what this code does
- LinuxBIOS is one example
- 2000: boot complex server node to Linux in 3 seconds
- 2015: EFI can do the same in 300 seconds


Linux
Platform code, get DRAM going, set naughty bits, load kernel, please go away

Firmware, 2005-present, “The Empire Strikes Back”

- Kernel is Ring 0
- Hypervisor is Ring -1
- Firmware is Ring -2
- Firmware gets hardware going
- But never goes away
- Sucks
    - Slow
    - No easy bugfix path
    - Not SMP capable on x86
- This model is even being pushed for ARM V8
    - :-(


Platform-independent code
Platform code, on EEPROM or similar

Why don’t we (ok, I) like persistent firmware?

- It’s just another attack vector
  - Indistinguishable from persistent embedded threat
  - Is the code an exploit or …
  - Not necessary in an open source world
  - Main function is to preserve top secret “core IP”
- So [my] preference is that platform management code run as a kernel thread
- Minion cores are ideal for this
- Again, from the point of view of an end user, “fire and forget” is an ideal mode
  - Note: end user interests != vendor interests

coreboot

- GPLv2 BIOS replacement
    - Started as LinuxBIOS in 1999 by Ron Minnich
    - Renamed to coreboot in 2007
- Mostly C, small amounts of Assembly, and ASL
- Kconfig and modified Kbuild
- High-level organization around block diagram
    - Modular CPU, Chipset, Device support
- NOT a bootloader

coreboot Stages

- Boot Block
  - Prepare Cache-as-RAM and Flash access
- ROM stage
  - Memory and early chipset initialization
- RAM stage
  - Device enumeration and resource assignment
  - Start other CPU cores
  - ACPI table creation, SMM handler [x86]

coreboot stages: Payload

- Bootloader (grub2, etc.)
- In the early days, a Linux kernel
- Flash got *smaller*, Linux got bigger
  - From 2001 on, payload was “an ELF binary”
  - etherboot, lilo, grub, plan 9, linux, …

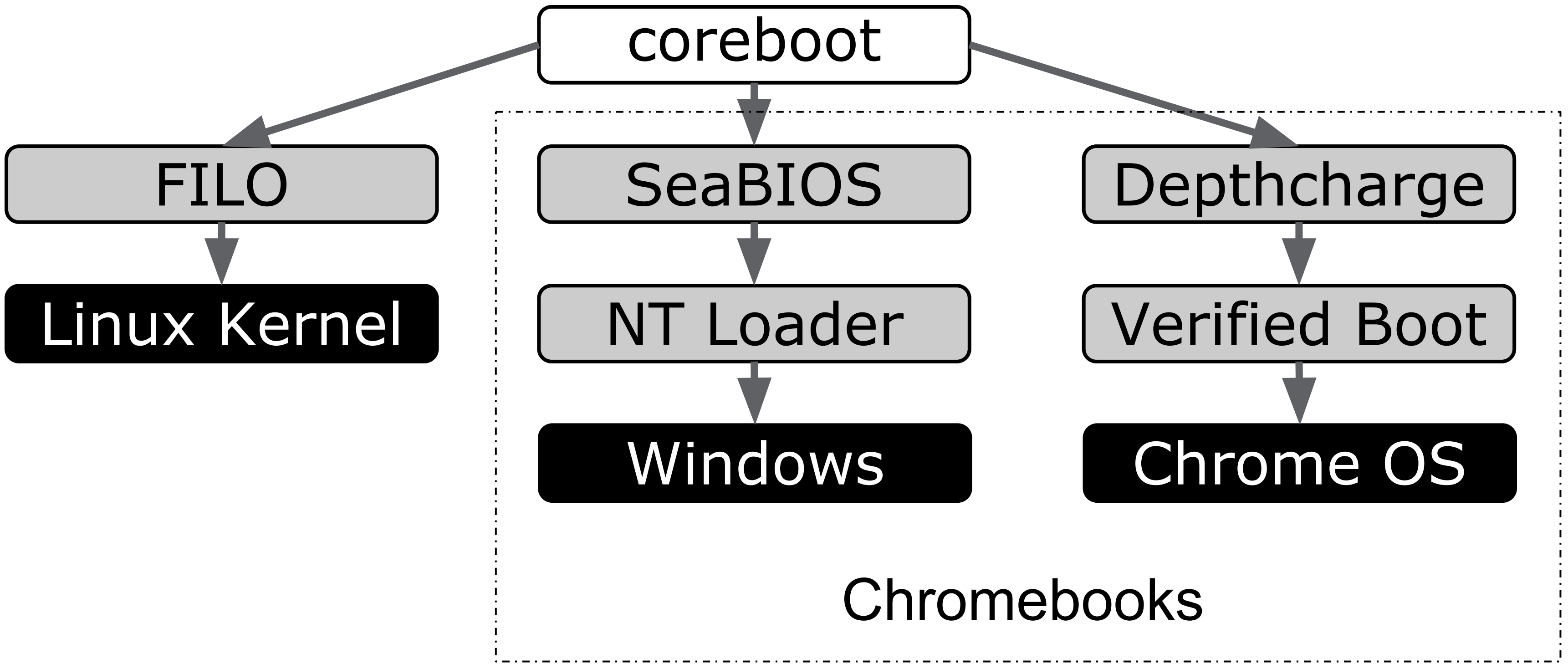
coreboot and UEFI

coreboot	UEFI
Boot Block	SEC
ROM Stage	PEI
Reference Code	
RAM Stage	DXE
Option ROMs	
Payload	BDS
Operating System	

libpayload

- Library of common payload functions
  - Subset of libc functions
  - Tiny ncurses implementation
  - Various hardware drivers
  - Read and parse coreboot table
- BSD License
- www.coreboot.org/Libpayload

coreboot Payloads



Why coreboot on RISCV?

- It’s becoming a standard for consumer hardware
- All chrome hardware built since Jan 2012 runs coreboot
    - i.e., 50% of all educational devices
- Has verified boot solution
- Well worked out recovery/update model
    - Roll out firmware bug fixes to millions of devices
- RISCV: Open source firmware for open source CPU

RISCV port history

- First port: October 7, 2014
  - mainly toolchain and utilities
- First qemu boot about 6 weeks later
  - But a full-time effort would have been about a week
- Effort resumed July 2015 with protection mode
- Working again in September
- Changed toolchain, boot, code layout, … everything

port history

- First port runs on qemu
- Most recent runs on spike
    - qemu still not ready?
- Must use 5.2 gcc
- since 11/2014, for all commits to coreboot repo, riscv build has to work
    - or commit is blocked
- riscv is a first class citizen

coreboot structure on riscv

$$\begin{tikzcd}
\text{src/mainboard/emulation/spike-riscv} \arrow[dr] & & & \\
& \text{src/soc/ucb/riscv} \arrow[dr] & & \\
& & \text{src/arch/riscv} \arrow[dr] & \\
& & & \text{common code}
\end{tikzcd}$$

src/mainboard/emulation/spikev-riscv 501 lines

```
 61 Kconfig
  2 Kconfig.name
 26 Makefile.inc
  2 board_info.txt
 30 bootblock.c
 20 devicetree.cb
 34 mainboard.c
 28 memlayout.ld
 23 romstage.c
218 spike_util.c
 57 uart.c
```

src/soc/ucb/riscv 38 lines

```
12 Kconfig
 6 Makefile.inc
20 cbmem.c
```

src/arch/riscv 2685 lines

```
  6 ./include/arch/byteorder.h
 48 ./include/arch/cpu.h
 27 ./include/arch/early_variables.h
960 ./include/arch/encoding.h
 66 ./include/arch/errno.h
 61 ./include/arch/exception.h
 28 ./include/arch/header.ld
  4 ./include/arch/hlt.h
 35 ./include/arch/io.h
 26 ./include/arch/memlayout.h
 25 ./include/arch/stages.h
 72 ./include/atomic.h
 57 ./include/bits.h
 86 ./include/spike_util.h
 63 ./include/stdint.h
 74 ./include/vm.h
 21 ./Kconfig
113 ./Makefile.inc
  9 ./misc.c
 17 ./prologue.inc
 26 ./rom_media.c
 48 ./stages.c
 64 ./tables.c
218 ./trap_handler.c
132 ./trap_util.S
138 ./virtual_memory.c
```

Totals

- Total set of .c is roughly 10KLOC (just .c files)
- All riscv source is about 900 lines (just .c)
- Port effort is in the files you saw
- Rest is unchanged

QA

- Federal Office for Information Security (BSI) in Germany runs https://testlab.coreboot.org/BSI/
- Hardware test station, automated flash, reboot, check serial outputs, etc. up to Linux boot and user mode tests
- As soon as real hardware is running they’ve offered to hook it up to their station for us

For more info on test stand

- https://secure.raptorengineeringinc.com/content/REACTS/intro.html
- If you’re going to do a system, it needs firmware
- If you do firmware, it might as well be coreboot
- And you’ll get a free hardware test stand and maintenance of same from BSI.
    - Which many companies spend lotsa $$$ on doing

Status

- Did two ports with a few weeks work
- First port booted to qemu
- second port boots on spike to linux
- Correctly sets up page structs and enters linux at correct priv mode
- RISCV build success is required to pass for *ALL* commits to coreboot

Lessons learned

- Provide a boot time SRAM
  - Make sure address is fixed and not aliased by DRAM once DRAM is up
- Provide a serial port
  - It’s one pin and cheap to implement
- [me] Runtime functions belong in the kernel, not persistent firmware

Lessons learned

- firmware tables always need translation by kernel
    - So make them text, not binary -- avoids version issues
    - Open Firmware tree is not bad
    - But JSON could work too
    - They should be self-describing
- Mask ROM: KISS
    - May not need a full USB stack
- Need network hardware for test stand

Lessons learned

- Don’t cheap out on SPI or flash part size
  - just plan for 64 MiB flash part; you’ll thank me later
- Don’t reset chipset on IO device not present
  - Seen in real life: chipsets that die if you do outb to port 80
  - What’s port 80? Just the POST port for PCs for the last 35 years

Conclusions

- RISCV needs firmware
- We have open source firmware ported today
- Same firmware used in millions of consumer/embedded systems: laptops, tablets, routers, …
- Has verified boot and update models

Questions?