ECE 471 – Embedded Systems
Lecture 21

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

28 October 2026

Announcements

- Don't leave HW#7 to the last minute!
- Leaked info on yet more new Raspberry Pi models: Pi 4A and Pi Zero-3
- Still working on grading Midterms. Soon.

Bootloaders

- Firmware doesn't always directly load Operating System
- Bootloader (relatively simple code, just smart enough to load OS and jump to it) is loaded first
- Bootloader is often on a very simple filesystem (such as FAT) as the code has to be simple (possibly even written in assembly language)
- Bootloader is often just complex enough to load OS kernel from disk/network/etc and jump to it

Raspberry Pi Booting

- Unusual – GPU handles it
- Small amount of permanent firmware on SoC
- ARM chip brought up inactive (in reset)
- Videocore GPU loads first stage from ROM

Pi GPU Notes

- Large GPU chip, originally small helper ARM chip
- VPU – dual code Videocore IV (better on later models) SIMD, Parallel Processing. Runs ThreadX. Power management, video
- ISP – image sensor pipeline, handles camera
- QPU – quad processor unit. 24GFLOPs.

Raspberry Pi Early Booting (pre pi4)

- Boot ROM on GPU runs enough to get started
- Videocore reads `bootcode.bin` from FAT partition on SD card into L2 cache.
  It's actually a RTOS (real time OS) in own right "ThreadX" (50k)
  GPU in control, draws rainbow to screen
- bootcode.bin is **binary blob**, in GPU assembly language, 1 million lines of code, rumored to be maintained by one guy at Broadcom
- Inits and parks ARM chips
- Enables SDRAM, then loads `start.elf` (3M) which is the bootloader

Aside – Can You Replace Early Firmware?

- The asseumble for the Pi GPU has been reverse engineered
- There are tools for dealing with it
- While a lot of people would love a “fully free” software stack for the Pi, no one has written replacement early firmware yet

Pi4 booting

- https://www.raspberrypi.org/documentation/hardware/raspberrypi/booteeprom.md
- SPI EEPROM holds equivalent of `bootcode.bin`, no longer read from SD card partition
- Why? SDRAM, PCIe USB, etc are more complex
- Supports network and USB booting which is much more complex than just loading a file off of SD card

Pi5 booting

- The Pi5 is even more different than Pi4 (software abstraction hides most of this)
- Still boots off the eeprom, blob brings up LPDDR4 RAM
- bootmain.elf loaded from eeprom
- bootconf.txt lets you configure how the boot goes
- Can possibly load kernels directly without going through the bootloader

Pi bootloader

- `start.elf`, `start4.elf`
- This initializes things, then loads and boots Linux onto ARM chip
- Reads/parses config files
- Sets up device tree
- Kernel files are in `/boot`
  - kernel – original ARM1176 systems
  - kernel7 – ARMv7
  - kernel7l – pi4 (32-bit compat)
  - kernel8 – 64-bit kernel for pi3/4/5

Raspberry Pi Boot Firmware

- The kernels now live in `/boot`
- The firmware now lives in `/boot/firmware`
- Device tree (DTB) files are there too
- config.txt
- What is the 3, 3cd, 4db, 4x stuff
  cd=cut down (low GPU RAM), db=debug, x=extra
- Overlays directory for dt overlays?
- fixup.dat and similar describe memory layout

More Typical ARM booting

- Pi is unusual, most other boards not use GPU
- The UBoot bootloader is common
  - ARM chip runs first-stage boot loader (often MLO)
  - Then loads second-stage (uboot)
- Although UBoot is common, many boards still have own custom boot methods

Disk Partitions

- Way to virtually split up disk.
- DOS GPT – old partition type, in MBR. Start/stop sectors, type
- Types: Linux, swap, DOS, etc
- GPT had 4 primary and then more secondary
- Lots of different schemes (each OS has own, Linux supports many). UEFI more flexible, greater than 2TB

Why partition disks?

- Different filesystems; bootloader can only read FAT?
- Dual/Triple boot (multiple operating systems)
- Old: filesystems can't handle disk size

Why a FAT Partition?

- /boot on Pi is a legacy (40+ years old) File-Allocation Table (FAT) filesystem
- Why FAT? (Simple, Low-memory, Works on most machines, In theory no patents despite MS's best attempts (see exfat))
- The boot firmware (burned into the CPU) is smart enough to mount a FAT partition
- Opening SD card via SPI and reading/parsing FAT partition relatively straightforward (writing FAT much more difficult)

Boot Methods

- Floppy
- Hard-drive (PATA/SATA/SCSI/RAID)
- CD/DVD
- USB
- Network (PXE/tftp)
- Flash, SD card
- Tape
- Networked tape
- Paper tape? Front-panel switches?

Detecting Devices

- Guessing/Probing – can be bad if you guess wrong and the hardware reacts poorly to having unexpected data sent to it
- Standards – always knowing that, say, VGA is at address 0xa0000. PCs get by with defacto standards
- Enumerable hardware – busses like USB and PCI allow you to query hardware to find out what it is and where it is located
- Hard-coding – have a separate kernel for each possible board, with the locations of devices hard-coded in. Not very maintainable in the long run.
- Device Trees – see next slide

Devicetree

- Traditional Linux ARM support a bit of a copy-paste and #ifdef mess
- Each new platform was a compile option. No common code; kernel for pandaboard not run on beagleboard not run on gumstix, etc.
- Work underway to be more like x86 (where until recently due to PC standards a kernel would boot on any x86)
- A “devicetree” passes in enough config info to the kernel to describe all the hardware available. Thus kernel much more generic

Devicetree Implementation

- Text file describing the hardware in a system, .dts
- Tools you can install that compile this in a binary .dtb file
- Pass this file to the bootloader or kernel and it uses it to find hardware capabilities
- Doesn't change much, so much easier to maintain a device tree file for a SBC and use a generic ARM kernel than it is to maintain a custom kernel.

ACPI

- ARM servers often use ACPI for same thing (from x86) for device discovery
- Mostly because Microsoft wants to run Windows on ARM but doesn't want to have to code up device tree support
- ACPI is a huge mess, originally from intel. Big files. Custom interpeted language. Not well documented.