ECE 471 – Embedded Systems
Lecture 22

Vince Weaver

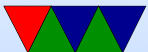
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

29 October 2025

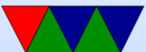
Announcements

- Don't leave HW#7 to the last minute!



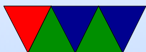
Firmware

- Software – code
- Hardware – the physical device (can throw it out the window)
- Firmware – low level code closely tied to hardware, often difficult (or impossible) to change



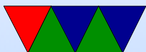
Firmware Location

- In old days in ROM (read-only memory)
- ROMs got more programmable over the years, EPROM, EEPROM, flash
- These days probably in flash of some sort
- Usually is harder to update than software



System Booting

One prominent place you'll find firmware is during the boot process



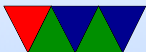
Boot Firmware

Provides booting, configuration/setup, sometimes provides rudimentary hardware access routines.

Kernel developers like to complain about firmware authors.

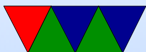
Often mysterious bugs, only tested under Windows, etc.

- BIOS – legacy 16-bit interface on x86 machines
- UEFI – Unified Extensible Firmware Interface
  ia64, x86, ARM. From Intel. Replaces BIOS
- OpenFirmware – old macs, SPARC
- LinuxBIOS



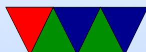
Boot Firmware Aside

- Tell story of doing spread spectrum on Progear webpad but early in BIOS so doing i2c in assembly language with no RAM configured yet



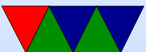
Bootloaders

- Firmware doesn't usually 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



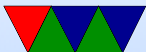
Raspberry Pi Booting

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



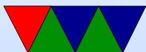
Pi Hardware Aside

- 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.

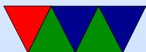


Raspberry Pi Early Booting (pre pi4)

- 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

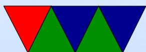


- Inits and parks ARM chips
- Enables SDRAM, then loads `start.elf` (3M) which is the bootloader



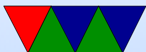
Pi4 booting

- https://www.raspberrypi.org/documentation/hardware/raspberrypi/booteeprom.md
- SPI EEPROM holds equivalent of `bootcode.bin`, no longer read from 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

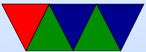


Pi4 booloader

- `start.elf`
- This initializes things, the loads and boots Linux onto ARM chip
- Reads/parses config files
- Sets up device tree
- Kernel files are
  - kernel – original ARM1176 systems
  - kernel7 – ARMv7
  - kernel7l – pi4 (32-bit compat)

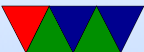


◦ kernel8 – 64-bit kernel for pi3/4/5



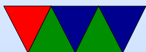
Raspberry Pi Boot Firmware

- 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?



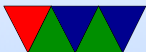
More Typical ARM booting

- The UBoot bootloader is common
- ARM chip runs first-stage boot loader (often MLO)
- Then loads second-stage (uboot)



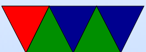
Disk Partitions

- 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?



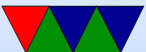
◦ Dual/Triple boot (multiple operating systems)

◦ Old: filesystems can't handle disk size



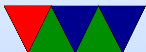
Why a FAT Partition?

- /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



Boot Methods

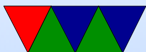
- Floppy
- Hard-drive (PATA/SATA/SCSI/RAID)
- CD/DVD
- USB
- Network (PXE/tftp)
- Flash, SD card
- Tape
- Networked tape
- Paper tape? Front-panel switches?



Detecting Devices

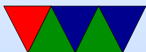
There are many ways to detect devices

- 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



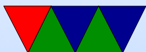
it is located

- 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



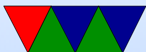
Devicetree

- 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



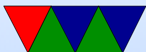
to describe all the hardware available. Thus kernel much more generic

- ARM servers use ACPI for same thing (from x86) mostly because of Microsoft



Quick Review of System Startup

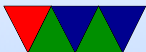
- On bare-metal machine, embedded device jumps to entry point and immediately runs your code
- When running an OS this is a bit more complicated



Booting Linux

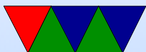
- Bootloader jumps into OS entry point
- Set Up Virtual Memory
- Setup Interrupts
- Detect Hardware / Install Device Drivers
- Mount filesystems
- Pass control to userspace / call init (systemd?)
- Run init scripts
- rc boot scripts, /etc/rc.local

  Start servers, or “daemons” as they’re called under



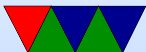
Linux.

- fork()/exec(), run login, run shell

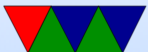


How a Program is Loaded on Linux

- Kernel Boots
- `init` started
- `init` calls `fork()`
- child calls `exec()`
- Kernel checks if valid ELF. Passes to loader (ld.so)
- Loader loads it. Clears out BSS. Sets up stack. Jumps to entry address (specified by executable)
- Program runs until complete.
- Parent process returned to if waiting. (`wait()`)

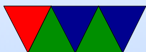


Otherwise, init.



Aside about Shared vs Static Libraries

- Shared libraries, only need one copy of code on disk and in memory
  - Good for embedded system (less room needed)
  - Good for security updates (only need to update lib, not every program using it
- Static libraries, all libraries included
  - No dependencies
- These days maybe containers, docker, kubertenes
- Also Flatpack, Snap. Why? Stability, Know package



will work on all distributions, Not have to install dependencies

- Can use `ldd` to view library usage