ECE 471 – Embedded Systems
Lecture 15

Vince Weaver

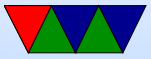
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

2 October 2023

Announcements

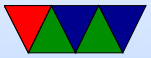
- HW#5 was posted
  If you need an i2c display let me know
- Midterm is a week from Friday, the 13th
- The Pi5 will reportedly have PIO (parallel I/O) and cortex-M3 chip you can program



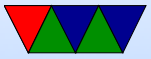
i2c followup – Reserved Addresses

Address	R/W Bit	Description
000 0000	0	General call address
000 0000	1	START byte (helps make polling cheaper)
000 0001	X	CBUS address
000 0010	X	Reserved for different bus format
000 0011	X	Reserved for future purposes
000 01XX	X	Hs-mode master code
111 10XX	X	10-bit slave addressing
111 11XX	X	Reserved for future purposes

10-bit addresses work by using special address above with first 2 bits + R/W, then sending an additional byte with the lower 8 bits.



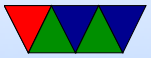
System Booting



Boot Firmware

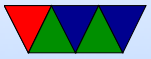
Provides booting, configuration/setup, sometimes provides rudimentary hardware access routines.
Kernel developers like to complain about firmware authors.
Often mysterious bugs, only tested under Windows, etc.

- 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



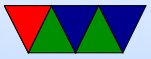
Bootloaders

- 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



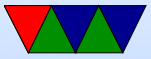
Raspberry Pi Booting

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



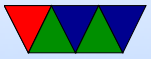
Raspberry Pi Booting (pre pi4)

- 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)
- This runs on videocard, enables SDRAM, then loads `start.elf` (3M)
- This initializes things, the loads and boots Linux onto ARM chip `kernel/kernel7/kernel7l/kernel8.img`. (also reads some config files there first) (4M)



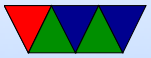
Pi4 booting

- 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



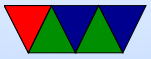
Typical ARM booting

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

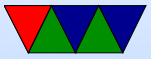


Disk Partitions

- 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?

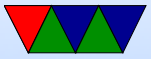


- Dual/Triple boot (multiple operating systems)
- Old: filesystems can't handle disk size



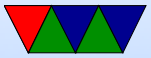
Why a FAT Partition?

- /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?

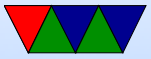


Device Detection

- x86, well-known standardized platform. What windows needs to boot. Can auto-discover things like PCI bus, USB. Linux kernel on x86 can boot on most.

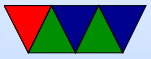
- Old ARM, hard-coded. So a rasp-pi kernel only could boot on Rasp-pi. Lots of pound-defined and hard-coded hw info.

- New way, device tree. A blob that describes the hardware. Pass it in with boot loader, and kernel can use



it to determine what hardware is available. So instead of Debian needing to provide 100 kernels, instead just 1 kernel and 100 device tree files that one is chosen at install time.

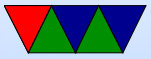
- Does mean that updating to a new kernel can be a pain.



Detecting Devices

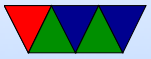
There are many ways to detect devices

- 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



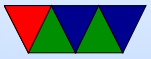
it is located

- 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