ECE 471 – Embedded Systems
Lecture 17

Vince Weaver

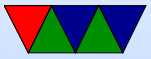
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

6 October 2023

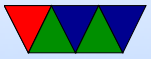
Announcements

- HW#6 won't be posted until next week. Enjoy your break



Midterm Info

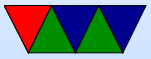
- In class
- Closed books, closed notes
  Can have one page of notes, 8.5"x11", one-sided
- Mostly short answer, questions like homework
- Won't make you code from scratch, but it might give you some code similar to that on homework and ask you what it's doing or what's wrong with it
- Rough outline of things covered
  - Characteristics of embedded system



- Benefits of having OS
- Be able to read C code and know what it's doing
- GPIO and i2c

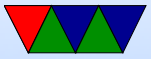
  know the limitations

  be familiar with Linux C code for accessing
- Code density, but at high level (no assembly coding)



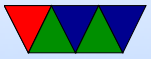
Real Time on an Atari 2600

- Older 8-bit systems would have real time constraints
- Hardware needed to be updated, sometime to cycle-exact (1us) deadlines or it wouldn't work
- Example: Atari 2600 video or Apple II disk accesses

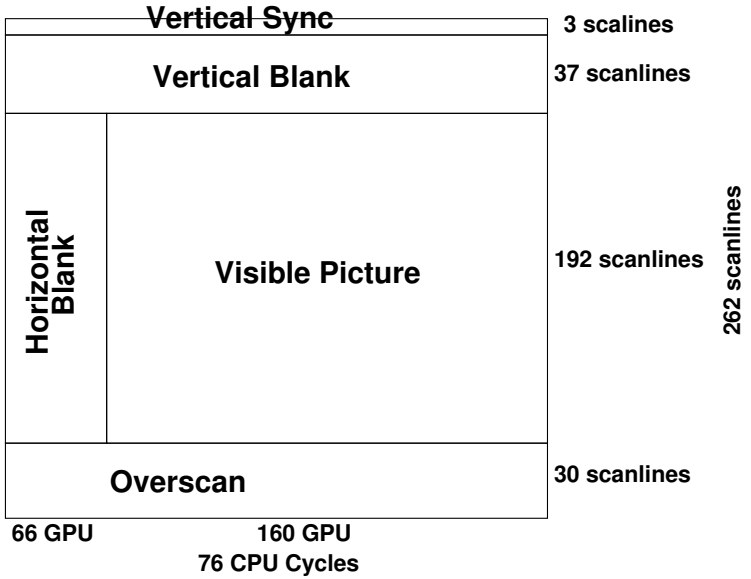


Atari 2600 Background

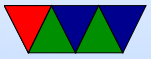
- Video game system from 1977
- 6507 processor – 6502 but only 12 address pins (8k address range)
- 128 bytes of RAM. Total. That includes stack
- Memory mapped I/O for audio/video
- No firmware, jumps directly to reset vector



Atari 2600 Graphics – CRT

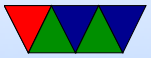


- NTSC gives you 262 lines at 60Hz (PAL, in europe, more lines 50Hz)



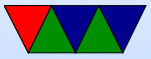
Atari 2600 Graphics

- Playfield
  - One foreground, one background color (out of 128 palette)
  - Have 20 bits of framebuffer. Each block 4 pixels wide.
    Right half screen mirror or dupe of right
    All you get is 40 columns, no rows
- Sprites
  - Two sprites. 8 pixels wide. You can scale them or duplicate them



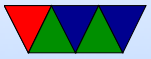
- No height, only width. Can have own color
- Can't specify location. X location you have to write a register just as beam is where you want it

- Missile
  - One pixel wide.
  - Same color as playfield



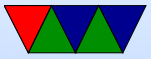
Atari 2600 Graphics – Racing the Beam

- How can you possibly make a game from this?
- You need to "race the beam"
- Your code needs to redraw the screen just ahead of the beam
- Real-time, you often only have a few 6502 assembly instructions to do this and if you miss your deadline graphics can be corrupt or the whole screen loses sync



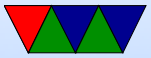
Atari 2600 Graphics – Possibilities

- Asynchronous playfield – rewrite 20-bit framebuffer each line before it is drawn
- Change colors mid screen
- Turn on/off sprites, missiles, re-use later in screen



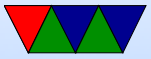
Atari 2600 – Other Features

- Collision detection in hardware
- Two channel sound, not designed for music. Some notes not physically possible to play due to clock divider



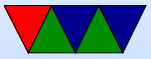
Atari 2600 – Examples

- Some examples in class if the projector cooperates
- (Spoiler, it didn't)



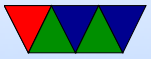
Software Sources of Jitter

- Interrupts. Taking too long to run; being disabled (cli)
- Operating system. Scheduler. Context-switching.
- Dynamic memory allocation, garbage collection.



Latency in Modern Systems

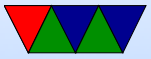
- Modern software stack has sources of latency



Video game keyboard latency example

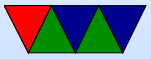
See Dan Luu's Paper "Computer Latency: 1977-2017"
`https://danluu.com/input-lag/`

- 1977 computers can have less latency to getting keypress on screen than fastest 2010s computers
- Having a fast processor only helps so much
- Slow hardware (keyboards, LCD displays), layers of abstraction in the way
- Apple II (1977) 30ms, modern machines 60-100+ms

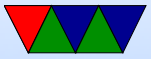


Latency of Apple II

- CPU running code reading memory access
  Each CPU instruction handful of cycles at 1MHz (few usec)
- Press happens, high bit set along with ASCII code, CPU reads in
- CPU writes out ASCII value to memory
- Video gen code running in parallel at 60 frames per second
- Electron beam scanning, reads out RAM, runs through

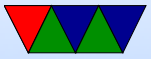


decode ROM to get 7-bit pattern, writes to screen within one frame worst case



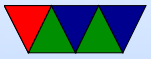
Latency of Modern System

- Press key, keyboard is own embedded system with CPU, scans keyboard, gets value, encodes it up as USB packet
- Sends out over USB bus (complex and with latency)
- USB controller gets packet, sends interrupt to CPU
- CPU gets interrupt, takes packet, notes it, returned from interrupt
- Later bottom half runs, decodes, to input subsystem,
- Operating system sees if anything is waiting for the input, if so it wakes it up (may take a bit if anything



else running)

- If it's a GUI, might have to run and see which window has focus, etc
- Program itself finally gets notified of keypress. scanf(). Immediately printf()
- Terminal emulator, update the graphics for the window (colors, font processing)
- GUI compositor puts together screen, tells OS
- OS sends out over PCIe bus to GPU
- GPU runs shaders/whatever outputs to display via HDMI
- LCD display gets the data, decodes it to display it



- Display might buffer a few frames to do extra processing (turn this off with “gaming” mode)