# ECE 471 – Embedded Systems
# Lecture 17

Vince Weaver

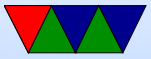
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

10 October 2025

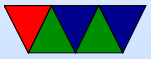
# Announcements

- Midterm on Friday, the 17th, in class
- Am trying to grade all homeworks by then
- HW#6 will be posted, have two weeks
- Hang on to displays, needed for HW#9



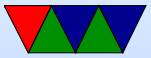
# Midterm Notes

- The midterm will be in-person during class time
- Closed book/notes but you are allowed one page (8.5"x11") full of notes if you want

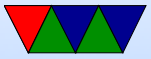


# Midterm Content

- Be sure you know the characteristics of an embedded system, and can make an argument about whether a system is one or not.
  - Inside of something (embedded)
  - Fixed-purpose
  - Resource constrained
  - Sensor I/O
  - Real time constraints (if you use this, be sure you can explain)

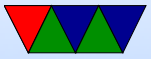


- Benefits/downsides of using an operating system on an embedded device
  - Benefits: "Layer of Abstraction"
  - Downsides: overhead, timing
- C code
  - Have you look at some code and know what it is doing
  - Fill in missing comments
  - Look at code and find bugs
  - Mostly know what file I/O, loops, usleep, open/ioctl (things we've done in the homeworks)
- Code Density



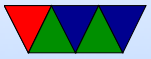
- Why is dense code good in embedded systems?
- Know why ARM introduced THUMB/THUMB2

- GPIO & i2c
  - Know some of its limitations (speeds, length of wires, number of wires, etc)
  - Don't need to know the raw protocol
  - Know the Linux interface (open, ioctl, write) and be familiar with how those system calls work
- Realtime won't be on this midterm

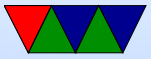


# Project Preview

- Posted a PDF with full details to the website
- Can work in groups
- Embedded system (any type, not just Pi)
  Pi Pico, Beagleboard, Orange Pi, 271 STM boards, TS-7600, etc.
- Written in any language (asm, C, python, C++, Java, Rust, etc.)
- Do some manner of input and some manner of output using the various capabilities we discussed

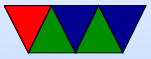


- I have a large amount of i2c, spi, and other devices that you can borrow if you want to try anything interesting.
- Past projects: games, robots, weather stations, motor controllers, music visualization, etc.
- Note: this will have to be distinct in some way from senior project
- Will be a final writeup, and then a short minute presentation and demo in front of the class during last week of classes.
- Deadlines:
  - November 7: pick topic (send e-mail with group

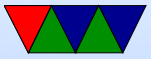


members and preliminary topic)

- November 24: progress report (e-mail with summary, how it's going, and which day you'd prefer to present)
- Last 4 days of classes: project presenations
- December 19: Writeup due



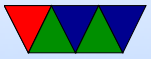
# Operating System Scheduling



# Real Time without an O/S

Often an event loop. All parts have to be written so deadlines can be met. This means all tasks must carefully be written to not take too long, this can be extra work if one of the tasks is low-priority/not important

```
main() {
    while(1) {
        do_task1(); // read sensor
        do_task2(); // react to sensor
        do_task3(); // update GUI (low priority)
    }
}
```



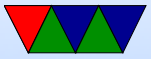
# Real Time with an O/S

What if instead you ran all three at once, and let OS switch between them

```
while(1) {
  do_task1();
}
```

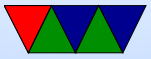
```
while(1) {
  do_task_2();
}
```

```
while(1) {
   do_task3();
}
```



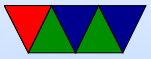
# Bare Metal

- What if want priorities?
- Have GUI always run, have the other things happen in timer interrupt handler?
- What if you have multiple hardware all trying to use interrupts (network, serial port, etc)
- At some point it's easier to let an OS handle the hard stuff



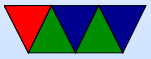
# Common OS scheduling strategies

- Event driven – have priorities, highest priority pre-empts lower
  Usually can "yield" rest of your timeslice
- Time sharing – only switch at regular clock time, round-robin



# Scheduler Types

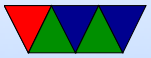
- There is a large body of work on scheduling algorithms.
- Assume you tell it to run tasks, they are put into queue
- How should they be run? A few (not exhaustive) possibilities:
  - Simple: In order the jobs arrive
  - Static: (RMS) Rate Monotonic Scheduling – shortest first
  - Dynamic: (EDF) Earliest deadline first



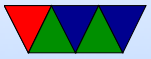
# Deadline Scheduler Example

- Three tasks come in
  - A: deadline: finish by 10s, takes 4s to run
  - B: deadline: finish by 3s, takes 2s to run
  - C: deadline: finish by 5s, takes 1s to run
- Can they meet the deadline?

| In-order | A | A | A | A | B | B | C | - | - | - |
|---|---|---|---|---|---|---|---|---|---|---|
| RMS | C | B | B | A | A | A | A | - | - | - |
| EDF | B | B | C | A | A | A | A | - | - | - |

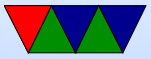


# Bonus Material – Real time on 8-bit Gaming Platforms



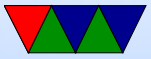
# Real Time on an Atari 2600

- 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
  - Atari 2600 video
  - Apple II disk accesses

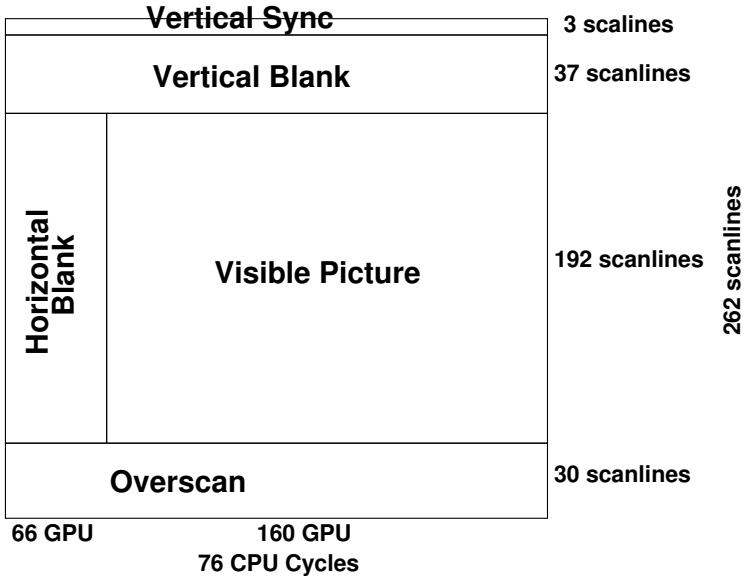


# Atari 2600 Background

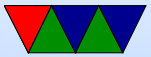
- Video game system from 1977
- 6507 processor 1.19MHz – 6502 but only 12 address pins (8k address range)
- 128 bytes of RAM. Total. That includes stack
- Memory mapped I/O for audio/video
- No firmware, jumps directly to reset vector



# Atari 2600 Graphics – CRT

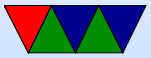



- NTSC gives you 262 lines at 60Hz (PAL, in europe, more lines 50Hz)



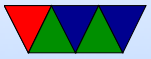
# Atari 2600 Graphics

- 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



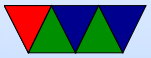
- No height, only width. Can have own color
- Can't specify location. X location you have to write a register just as beam is where you want it

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



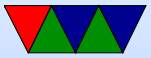
# Atari 2600 Graphics – Racing the Beam

- 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



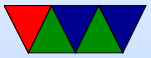
# Atari 2600 Graphics – Possibilities

- 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