# ECE 471 – Embedded Systems
# Lecture 15

Vince Weaver

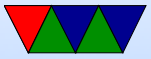
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

6 October 2025

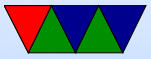
# Announcements

- Next week Fall Break already
- Midterm on Friday 17th, more details as we get closer
- Don't forget HW#5 (get display if needed)
- Don't rush to hand back in the i2c displays, you'll need them for HW#9
- Raspberry Pi prices going up, especially 4GB and 8GB Upton claims due to HBM RAM from AI demand affecting all DRAM prices
- Will post project document soon



# Last Note on HW#3 – Code Comments

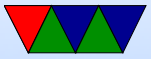
- Was mostly looking that you had them at all
- A lot of people the comment was something like "changed value" which is what you did, but that's not what the code is actually doing and if you come back to the code in 5 years not very helpful
- In theory it might almost be OK if it's in git and you have a better commit message about why it changed, but it's best to have good comments too so you don't have to poke around commit messages



# Real Time Constraints

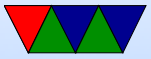
What are real time constraints?

- Time deadlines that hardware needs to respond in.
- Goal not performance, but response time
- Deadlines are often (but not always) short (order of milliseconds or microseconds)



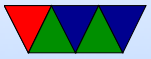
# Real Time Definition Confusion

- Real time can also mean other things with computers
- Real time, as in actual (not virtual) time of day. "Real Time Clock" (RTC) or `real` value from `time` command
- Real time, as in happening live, like real-time rendering in video games (as opposed to being recorded)
- In embedded systems it means something a bit different



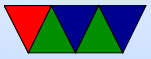
# Real-time example

- Self-driving car driving 65mph
  - That's roughly 95 feet/s
  - That's roughly 10 feet / 100ms
  - Stopping distance is 180 feet
- Equipped with image processor: GPU and camera.
  - Camera 60fps, 16ms.
  - GPU code recognize a deer within 100ms.
- Specification: if something jumps out, can stop within 200 feet.



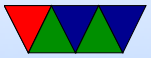
Can we meet that deadline?

- What if something goes wrong? Interrupt happens taking 100ms? Garbage collection? What if we miss deadline?
- Another example, turn in the road. How long does it take to notice, make turn? What if there's a delay?
- What if wiggly road, and you consistently miss by 100ms? Over-compensate?



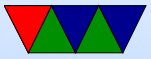
# Real Time Summary

- Real time is all about writing code to try to avoid missing timing deadlines
- Also about analyzing what parts of code are timing critical and which aren't
- Writing perfect code to guarantee deadlines is extremely difficult / expensive so it's generally only done when it's absolutely necessary
- Tools don't make this easy. There's no way to specify maximum time in a C function.



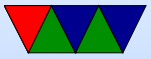
# Types of Real Time Constraints

- **Hard** – miss deadline, total failure of system. Minor or major disaster (people may die?)
  Antilock brakes?
- **Firm** – result no longer useful after deadline missed
  small number of misses might be acceptable
  lost frames in video, missed frames in video game
- **Soft** – results still wanted but gradually less useful as deadline passes.
  Caps lock LED coming on?

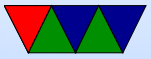


# Note on Soft Realtime

- Some people worry, what if it is something like “press a button, but takes minutes to run”
- If that’s unacceptable, does that mean essentially all things are hard-real time?
- Part of it comes down to how likely the failure is. If with minimal programming effort the response is reasonable 99.9% of the time, and any consequence of missing is minor, than soft real time
- If something consistently goes wrong and it takes minutes



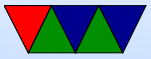
for each button press and that's unacceptable, then yes, maybe what you have is more than soft... but maybe also you need to rethink your hardware/software design



# Uses of Real Time

Who uses realtime?

- Timing critical situations. Cars, medical equipment, space probes, etc.
- Industrial automation. SCADA. Stuxnet.
- Musicians, important to have low-latency when recording
- High-speed trading

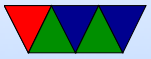


# Constraints depend on the Application

Try not to over-think things.

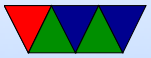
Can almost always come up with a scenario where a soft constraint could become hard.

For example: Unlocking a car door taking an extra second? Not hard real-time, except maybe if your car is about to crash and you need to escape quickly.



# Why isn't everything Real-time?

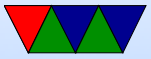
- It's hard to do right
- It's expensive to do right
- It might take a lot of testing
- It's usually not necessary



# Deviations from Real Time

Sometimes referred to as “Jitter”

- On an ideal system the same code would take the same, predictable amount of time to run each time
- In real life (and moreso on high-end systems) the hardware and operating systems cannot do this
- Deviation (often some sort of random distribution) is jitter

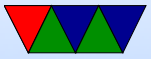


# Hardware Sources of Jitter – Historical

- Typically Less jitter on older and simple hardware
- Old chips like 6502 – fixed clock, each instruction takes an exact number of cycles. Deterministic. With interrupts disabled you can perfectly predict how long code will take.

  Steve Wozniak famously wrote disk firmware on 6502 that more or less cycle-accurate bit-banged stepper motors.

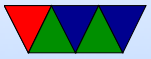
  Also video games, racing the beam.



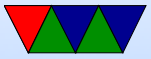
# Hardware Sources of Jitter – Modern

Modern hardware more complex.
Tradeoff: systems faster on average but with hard to predict jitter

- Branch prediction / Speculative Execution
- Memory accesses unpredictable with caches – may take 2 cycles or 1000+ cycles (Memory Wall)
- Virtual Memory / Page faults
- Interrupts can take unknown amount of time
- Power-save may change clock frequency

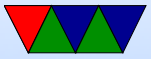


- Even in manuals instructions can take a range of cycles
- Slow/unpredictable hardware (hard disks, network access)
- Memory refresh (LPDDR burst refresh can avoid this a bit)



# Ways to Avoid Hardware Jitter

- Some newer embedded boards have simple helper processors that can run more deterministically
- The big cores run an OS and the small ones programmed separately
- Beaglebone – PRU (Programmable Real-Time Unit)
- Pi5, Pi Pico – PIO (Programmable I/O)



# Software Sources of Jitter

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