# ECE 471 – Embedded Systems
# Lecture 19

Vince Weaver

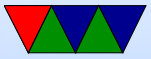
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

18 October 2021

# Announcements

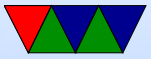
- HW#6 will probably be skipped
- Midterms not quite graded yet
- No class on Wednesday due to Engineering Career Fair
- I'll be giving a talk on Code Density and How it can be Fun on Friday (the 22nd) at 2pm in the Hill Auditorium



# Real Time Constraints

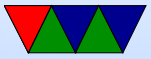
What are real time constraints?

- Time deadlines that hardware needs to respond in.
- Goal not performance, but response time



# Quick Example

- Self driving car, something falls into its path.
  - How quickly can the radar notice this and report it to CPU?
  - How quickly can the CPU activate emergency braking?
  - You probably want a time limit on the acceptable length of time each of these tasks is allowed
  - You don't want the CPU to be busy doing other things causing the deadline to be missed

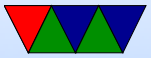


# Types of Real Time Constraints

- Hard – miss deadline, total failure of system. Minor or major disaster (people may die?)
  Antilock brakes?

- Firm – result no longer useful after deadline missed
  lost frames in video, missed frames in video game

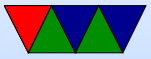
- Soft – results gradually less useful as deadline passes.
  Caps lock LED coming on?



# Uses of Real Time

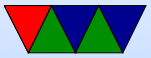
Who uses realtime?

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



# 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

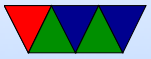


# Constraints depend on the Application

Try not to over-think things.

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

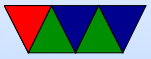
For example: Unlocking a car door taking an extra second? Not hard real-time, except maybe if your car is about to crash and you need to escape quickly.



# 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 due this
- Deviation (often some sort of random distribution) is jitter

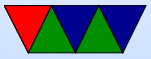


# Hardware Sources of Jitter – Historical

- Typicall 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.

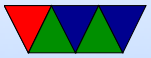
  Also video games, racing the beam.



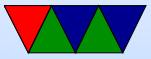
# Hardware Sources of Jitter – Modern

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

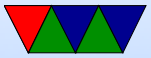


- Even in manuals instructions can take a range of cycles
- Slow/unpredictable hardware (hard disks, network access)
- Memory refresh (LPDDR burst refresh can avoid this a bit)



# Software Sources of Jitter

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



# 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