Lab 7: lab notes:

Stanford Winter 2026

UART high (8n1) bits

Medium complexity device with the usual patterns:

- bunch of magic addresses, bunch of magic constants
- passive definitional voice when you just want an example.
- errata.
- a couple hardware-is-not-software gotchas.
- but, also: you can always bit bang it (and you have/will)
- bonus: very hard to debug. broken: no debug, no bootload.

Solve the above, get the big prize: no more magic.

- delete last low level bit of code that isn't yours.

UART bugs = a bit different than a blinky light

If have a bug can at least can print?

- Oh, right.

Inverting no-output to root cause is hard.

- Not a biijection
- F(Bad context switch) = no output.
- F(Bad interrupt handler) = no output.
- F(Inf loop) = no output.
- …

So: be careful. Like really careful.

- Last year: Joe was helping multiple at 3am.

Law: No easy bug at lowest level of the system

This is the real real.

Previous low-level example: Bootloader:

- flip a bit?
- one instruction in your code becomes a different one.
- mostly runs fine, certainly does initially.
- and then: does something "impossible"
- Stare at program doesn't help: bug not there.
- You will never find this (within one lab).
- (VM will be the ultimate.)

Some Mitigations

1. Be really (really (like really)) careful.
2. Cross-check. Sing along comrade:
   - "if you have the same as everyone else;
   - if one is right;
   - all are right."
3. Plan B: Bit-banged UART to print
   - (IMO: bit-bang is a under-utilized plan B)
   - Over even more old skool: blink LED.
4. Common device mistake: device already enabled!
   - Q: what is UART state after bootloader? So?

Our standard i/o device Qs

1. How to turn off? (why?)
2. How to configure? (today: 8n1, 115200)
   - How to clear old data? (Why?)
   - What can we ignore? (Everyone's favorite!)
   - How to sanity check? (initial values, readback, xcheck)
   - What other devices do we need? (Speed \implies usually > 1)
3. How to turn on?
4. RX: Has data? How to get?
5. TX: Has space? How to send?

Now: take a partner and find the page numbers

Algorithm:

1. Go through each register in order (so don't miss).
2. Get the page number, bit position
3. Get ordering rules.

UART cheatsheet: fill in.

How to turn UART: on/off?

- other devices?

TX: on/off?

- can send?
- clear TX fifo?
- how to know done? (why care?)

RX: on/off?

- has data?
- how to clear RX fifo?

Hygeine:

- how to disable interrupts?

How to set 8n1?

- 8 bits, no parity, stop bit

How to set 115,200 baud (bits/sec)

- what: system clock?
- what: baud rate reg value?

UART rules

1. Make sure on, but not enabled when changing
   - not on: register writes ignored.
   - enabled: will puke garbage on the wire.
2. Ordering? (AUX, GPIO, UART?)
3. Device barriers?
4. Read/Write entire 32-bits
   - Common: device not guaranteed to handle sub-word operations
   - Especially: never write a byte in a 32-bit register.
   - Often works. Until.
5. For cross-checking:
   - write the registers in ascending address so traces match

Dumb: "It's correct b/c it worked when I ran it"

LOL. OMFG NO.

Running some tests != verification

Plenty of broke driver code runs "fine", until:

- Configure device faster?
- Run with other bus traffic?
- Turn on caches so the timing is faster?
- Shift code changes timing?
- Go from `-Og` to `-O3` so code faster?
- Go from `-O3` to `-Og` to debug something else and code slower?

Real 2025 CS340LX example: PL011 ordering rule.

BCM2835 omits rule: must set IBRD and FBRD before LCRH reg

- Half the class can't get bluetooth working.
- But (confusing): sometimes it does.

Timing issue:

- If broke code sets LCRH first *and* code fits in a prefetch block: works.
- Until: Code shifts so split into two prefetch buffer fetches.
- What causes a shift-split?
  - Add one instruction shifts, breaks (or fixes)
  - Add a printk to debug, shifts, breaks (or fixes)
  - Rinse and repeat.
- Nasty: looks just like if you omitted a device barrier.