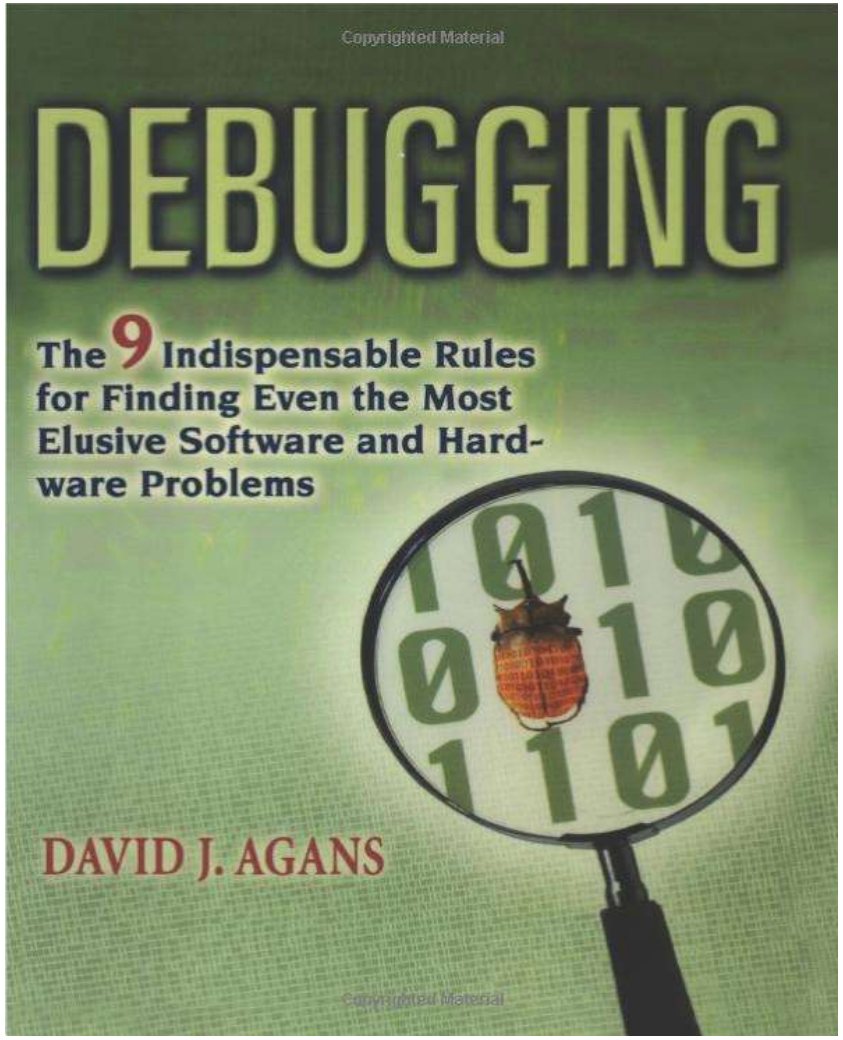
NC STATE UNIVERSITY

Debugging Embedded Systems

Highlights from Agans' **Debugging**



NC STATE UNIVERSITY

The Big Picture

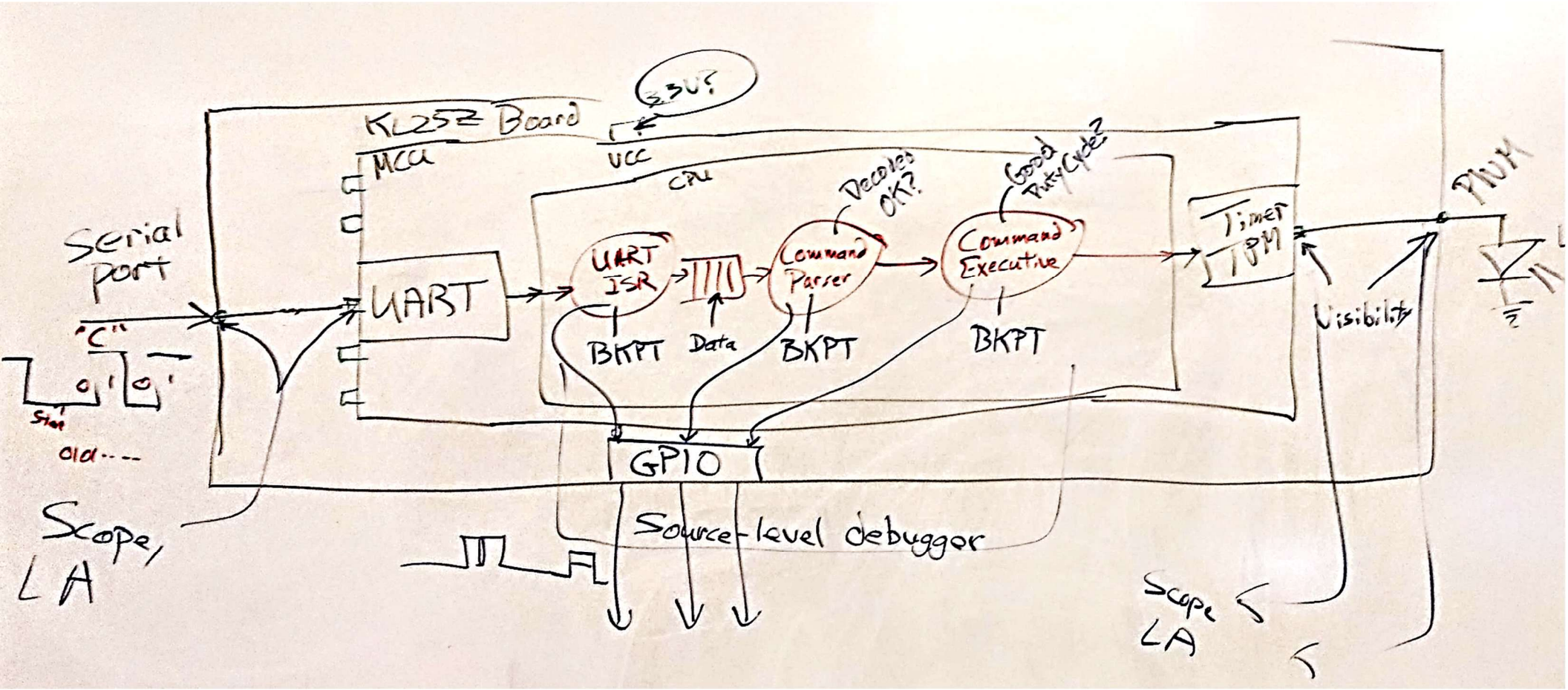
- Write code
- Run code and see if it works

NC STATE UNIVERSITY

Understand The System (1)

- Know the roadmap – need good representations
  - Hardware structure
  - Software structure – not just the code!
  - Hardware/software interactions
  - PCB
  - Module
  - System
- Where can you access the signals and data?
  - “Test points”

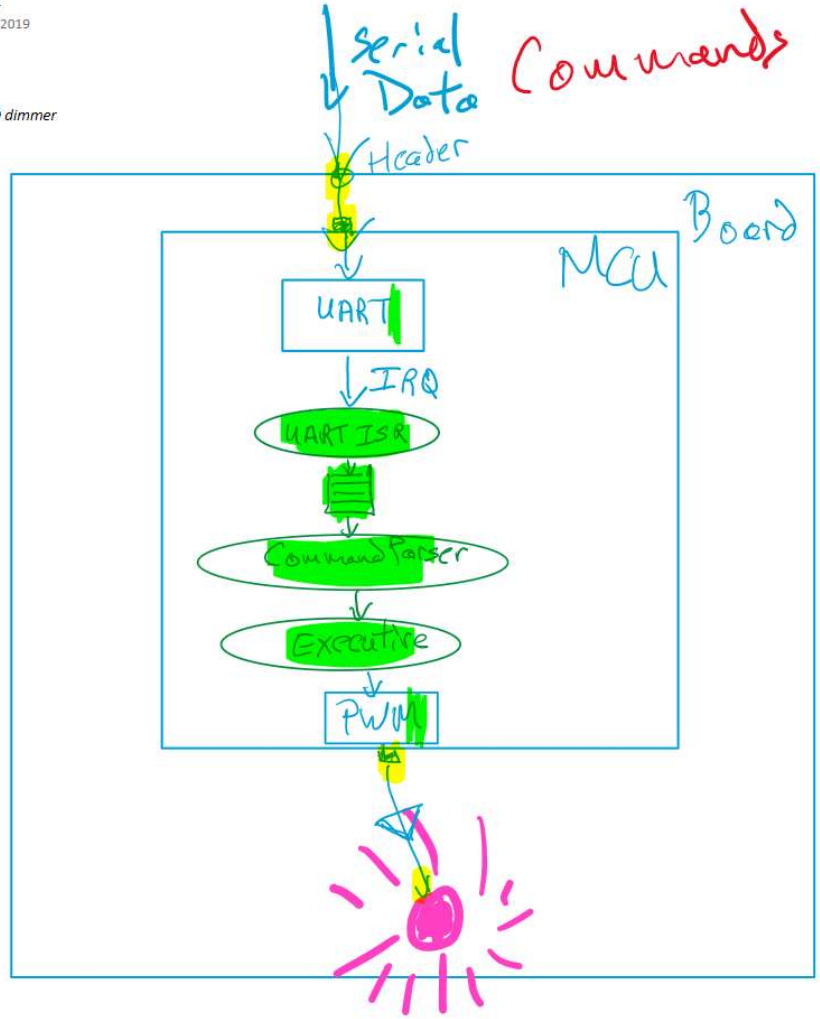
Build animated system diagram here

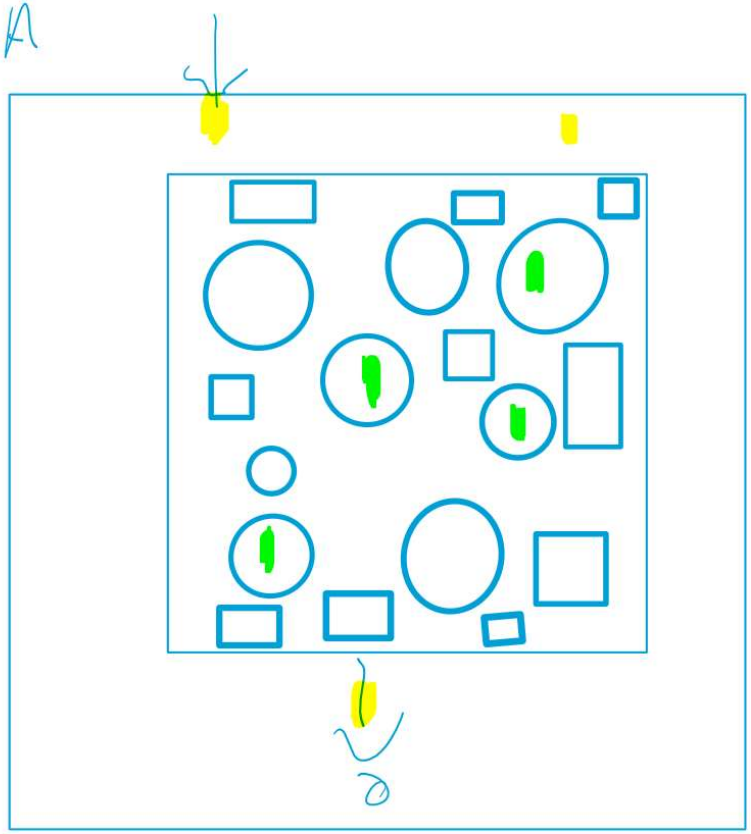

KL25Z Board
MCU
USB?
VCC
CPU
Serial port
"C"
UART
UART ISR
BKPT
Data
Command Parser
Decodes OK?
BKPT
Command Executive
Good Parse Cycle?
BKPT
Timer TPM
PWM
Visibility
GPIO
Source-level debugger
Scope, LA
Scope LA

Debugging 1

Thursday, September 5, 2019
10:26 AM

Know the Roadmap
Serially-controlled LED dimmer





Signals:-

Scope, LA

- SW Debug - Code Exec.
  - Variable Watches
- Peripheral Ctl/Status Regs using System Viewer

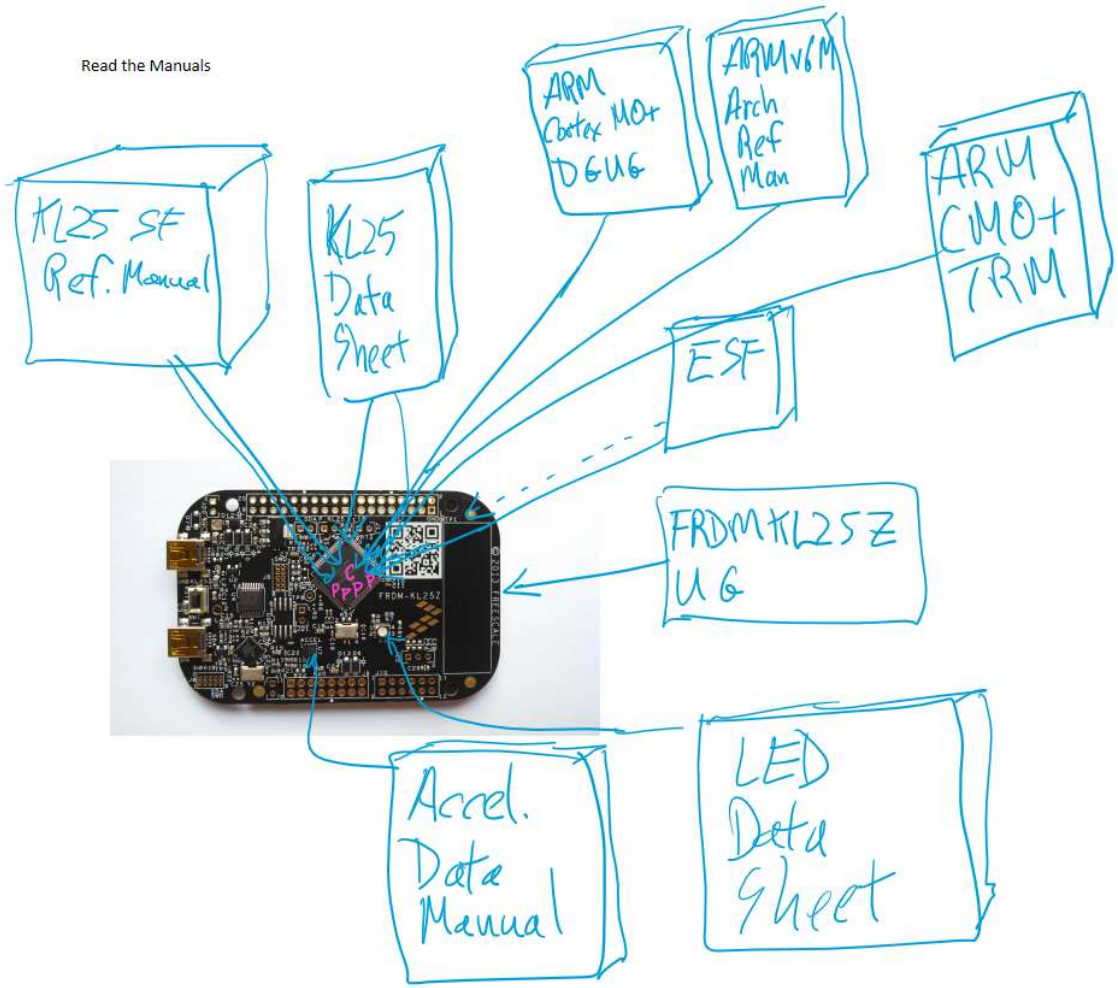
NC STATE UNIVERSITY

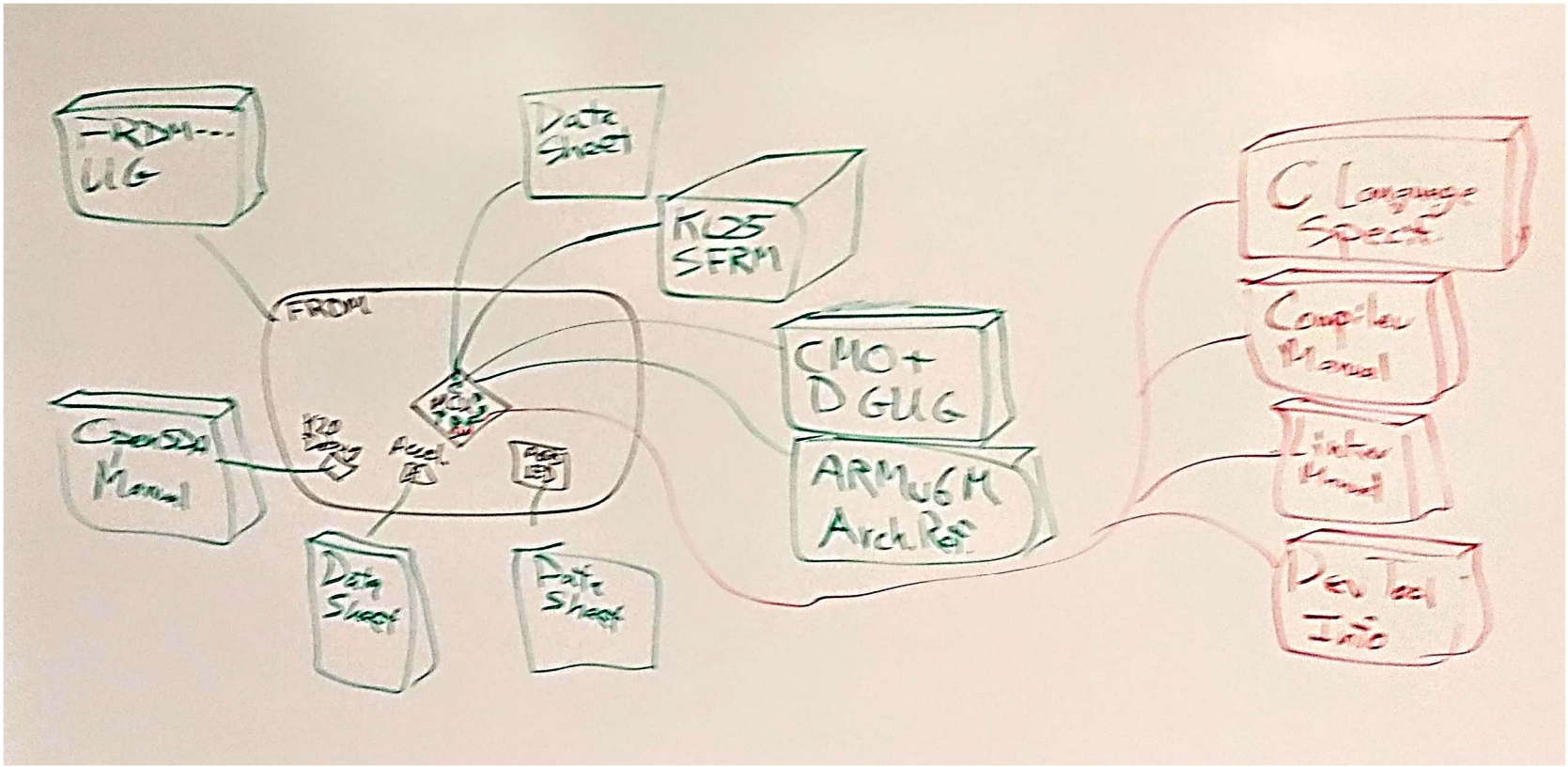
Understand The System (2)

- Read the manuals, data sheets and everything else which is relevant
  - Tough for systems due to multiple layers of abstraction (boxes within boxes)
  - Roadmap/structure is useful for managing complexity, identifying likely problem areas, ruling out others

Build animated documentation diagram here

Read the Manuals




FRDM...
UG
Data Sheet
KL05 SFRM
FRDM
CMO+ DGUG
ARMv6M Arch Ref
C Language Spec
Compiler Manual
Linker Manual
Dev Tool Info
Data Sheet
Data Sheet

NC STATE UNIVERSITY

Understand The System (2)

- Know what's reasonable
  - LED indicator: How short of a flash can you actually see?
  - Printf logging: How fast is the serial port?
- Know your tools
  - Understand your debugger's capabilities
    - Conditional breakpoints, trace buffer, peripheral viewer, real-time variable updates, kernel viewer, etc.
  - Understand your logic analyzer's and scope's capabilities
    - Triggering, decoding. More later.
  - Understand that object code ≠ source code
- Look it up
  - Don't guess – it can lead you astray
  - Use search features and create bookmarks in online documentation (otherwise can get lost in all the virtual material)

NC STATE UNIVERSITY

Make It Fail

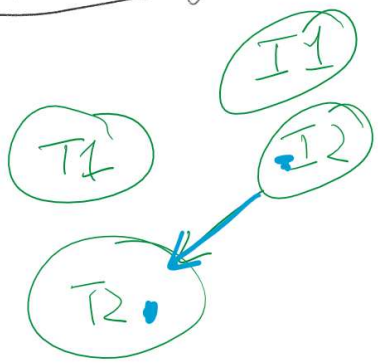
- Why?
  - So you can look at it
  - So you can focus on the cause
  - So you can tell if you've fixed it
- Be able to do it again
  - Document a process which makes it fail
    - Understand system's starting state for this process
  - For complex process, may be helpful to **stimulate** (trigger) the failure using automation
    - Can **simulate** conditions which **stimulate** the failure. E.g. periodic IRQ asserter
    - But don't **simulate** the failure itself – requires too many assumptions, some likely to be wrong
- Handling intermittent failures
  - Try to understand uncontrolled conditions
    - Uninitialized data, floating input signals, electric noise, etc.
    - Fix problem conditions
    - Tweak other conditions to evaluate sensitivity
  - Record everything from failures (e.g. log file) for analysis
  - Don't rely too much on statistics
- "That can't happen" is wrong
  - Your definition of "that" is wrong!
- Save every debugging tool

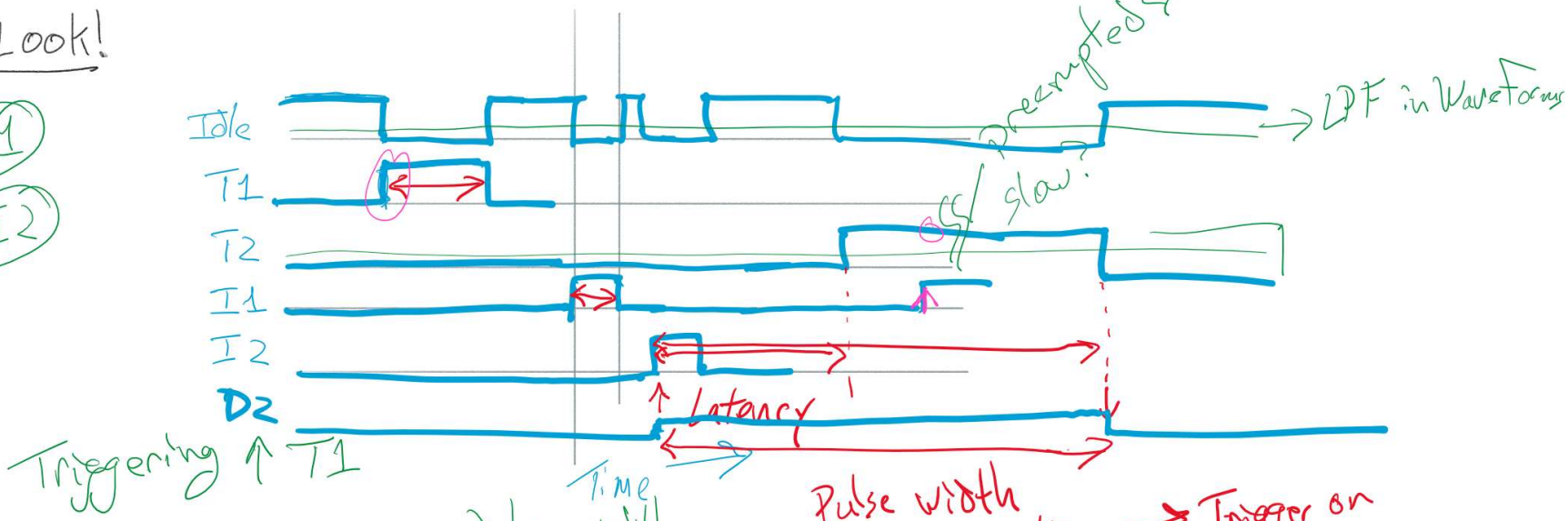
NC STATE UNIVERSITY

Quit Thinking and Look (1)

- See the failure
  - See what actually happened
  - Guessing about the failure may make you waste time "fixing" the wrong part
- See the details
  - Keep digging into the details until there are just a few possible causes (see pp. 52-55)
- Use instrumentation
  - Build instrumentation into the system
    - Use source-level debugger, debug logs, well-formatted status messages, trace buffer, lights, etc.
- Use external instrumentation
  - Use logic analyzers, scopes, meters, etc.
  - When helpful, add debug signals to reveal critical behavior
    - task/ISR/idle activity and durations
    - response latencies
    - event chains
  - See **Know Your Tools** earlier
    - Triggering: on edges or levels or combination, pulse widths, delays, message type, data on bus (parallel, UART, I^2C, SPI, SD card protocol, etc.)

Quit Thinking and Look!

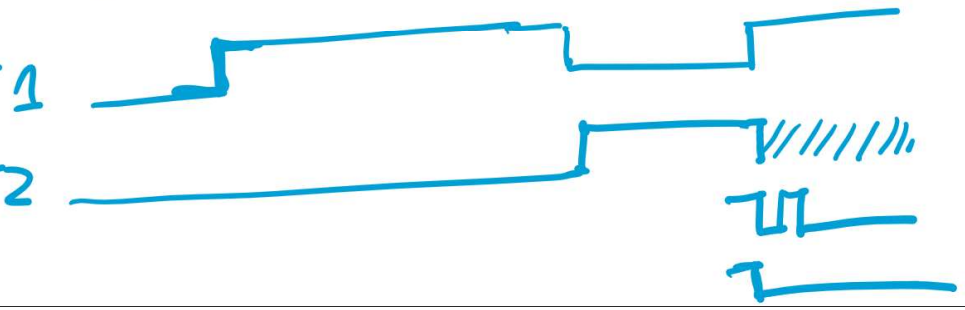


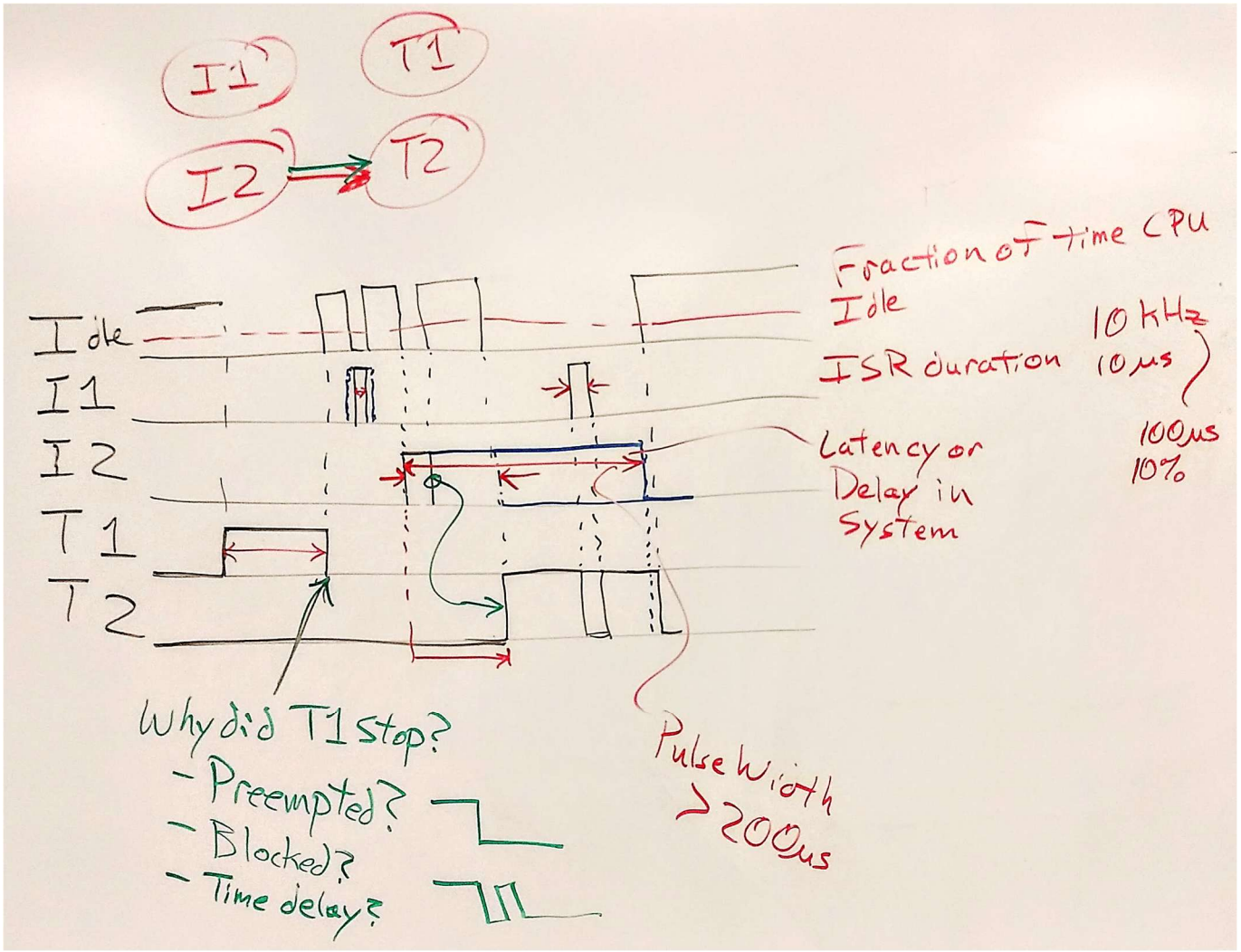


Conditional Triggering ↑+ —

I1↑ and T2==1

Preemptive System




I1
T1
I2
T2
Idle
I1
I2
T1
T2
Fraction of time CPU Idle
ISR duration
10 kHz
10 µs
100 µs
10%
Latency or Delay in System
Why did T1 stop?
- Preempted?
- Blocked?
- Time delay?
Pulse Width > 200 µs

NC STATE UNIVERSITY

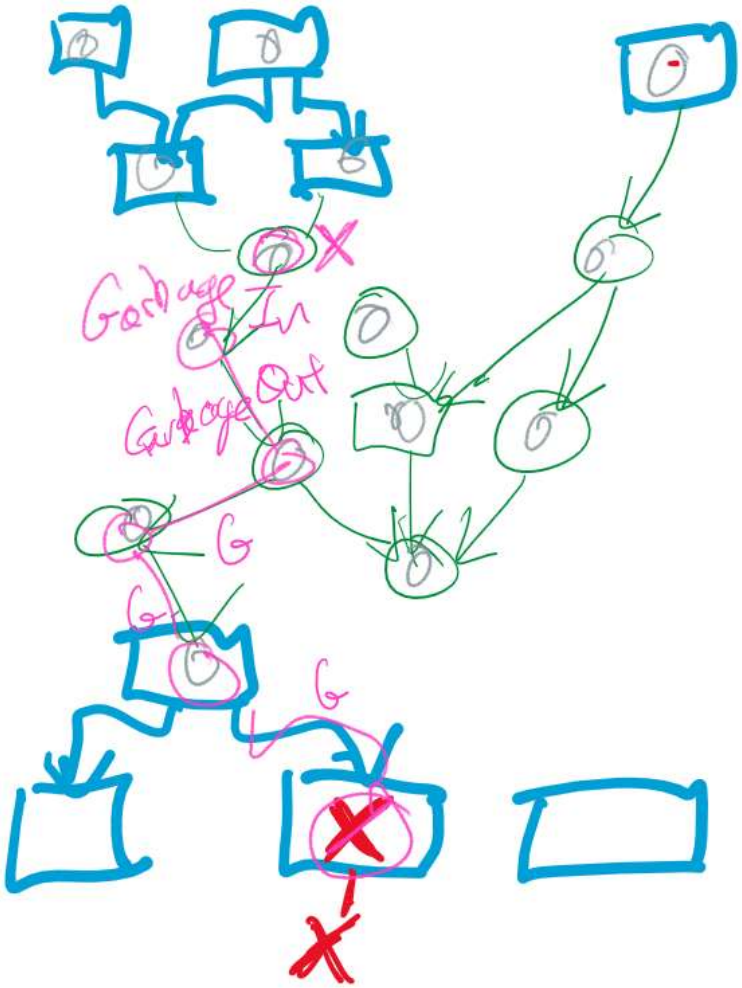
Quit Thinking and Look (2)

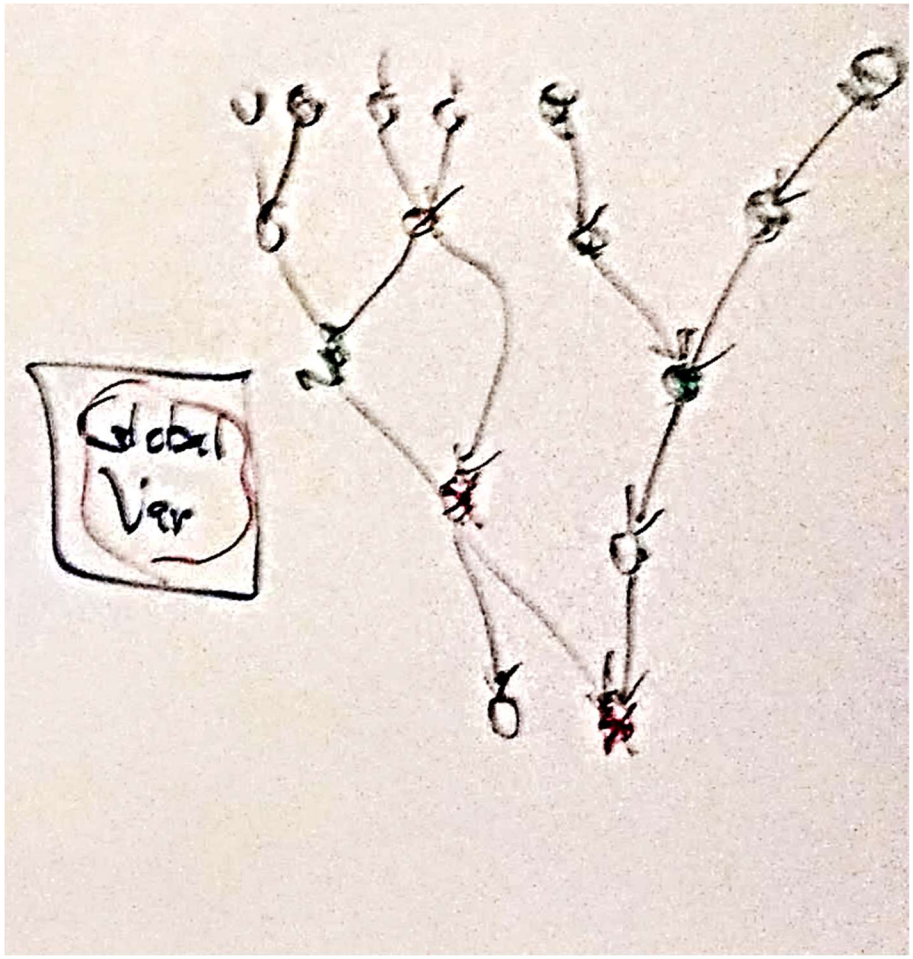
- Don’t be afraid to dive in
  - Create debug version of code, rebuild it
  - Minor code modifications can help dramatically (e.g. instrumentation)
- But watch out for Heisenbugs
  - Don’t mess up the system with your instrumentation (e.g. printf is slow)
- It’s OK to guess, but do it only to **focus the search**
  - Don’t overdo it

Divide and Conquer – One Rule to Rule Them All (1)

- Narrow the search with successive approximation
  - Define initial range as entire system (dangerous to rule anything out)
- Use easy-to-see test patterns
  - Make it easier to see the bugs
- Determine which side of the bug you're on
  - Essential to know how information flows through system's structures
- Start with the bad
  - Work upstream until you find the original failure
  - Working downstream covers too much area, takes too much time

Build animated
data flow diagram here


Garbage In
Garbage Out


Global
View

Divide and Conquer – One Rule to Rule Them All (2)

- Keep the system clean
  - Fix the other bugs you know about
    - One could cause the bug you're working on now, and others you haven't reached yet
    - Enable and pay attention to compiler warnings
  - Fix the noise first
    - Some bugs are likely to cause other bugs
  - But don't overdo it and waste time
    - Other bugs are superficial and relatively benign

NC STATE UNIVERSITY

Change One Thing at a Time

- Grab the brass bar with both hands.
  - Don’t change things without thinking through the likely impact
- Change one thing at a time
  - Keep all other variables constant (controlled)
  - If it doesn’t do anything, undo it!
- Compare it with a good one
  - Helps to have a debug log from each
- What changed since it last worked?
  - Helps to keep a log of your code changes and use version control

NC STATE UNIVERSITY

Keep an Audit Trail

- Write down what you did, in what order, and what happened
- Any detail may be the important one
- Correlate events (requires some thinking)
- Use design audit trails for testing clues
- Write it down!

NC STATE UNIVERSITY

Check the Plug

- Question your assumptions
- Start at the beginning
  - Don’t start at square three
- Test the tool
  - Learn your tool’s limitations

NC STATE UNIVERSITY

Get a Fresh View

- Ask for fresh insights
- Tap expertise
- Listen to the voice of experience
- Know that help is all around you
- Don’t be proud
- Report symptoms, not theories
- Realize you don’t have to be sure

NC STATE UNIVERSITY

If You Didn't Fix It, It Ain't Fixed

- Check that it's really fixed
- Check that it's really **your fix** that fixed it
- Know that it **never** just goes away by itself
- Fix the cause
- Fix the process
  - Get rid of bad habits