

9/9/26

COMP 435:

Computer Security Concepts



Lecture 9: Hardware Bugs, Errata and CWEs



The University
of North Carolina
at Chapel Hill

<https://tinyurl.com/comp435-fa26>

Logistics Update



- First lab will go out Friday!
- We will go over the material needed to get started, then 😊

Plan for Class Today



- 10:10-10:40 – Hardware Bugs!
- 10:40-10:50 – Hardware CWEs

Guiding Question 😊

Where do hardware bugs come from?

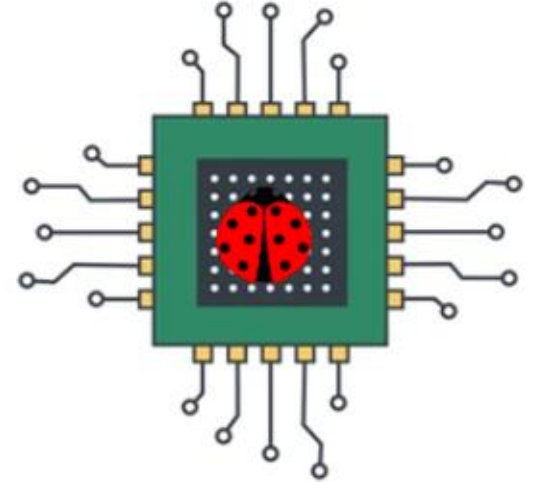
UNIT 1: TRUSTING THE MACHINE

Hardware Bugs

What is a bug in hardware?



- Hardware **Errata**: design defects or errors.
 - Errata may cause the processor's behavior to **deviate** from published **specifications**.
 - Companies and organizations **document** errata and **changes to designs** to help hardware (and software) engineers.



Intel Errata Table Example



Intel® Processor and Intel® Core™ i3 N-series

Intel® Processor and Intel® Core™ i3 N-series Specification Update

BIOS (Basic Input/Output System): is firmware that runs when a computer first powers on, before the operating system starts

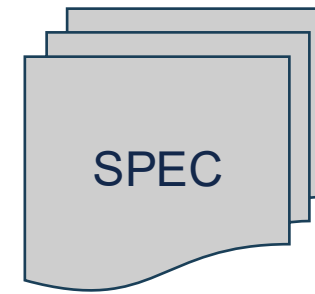
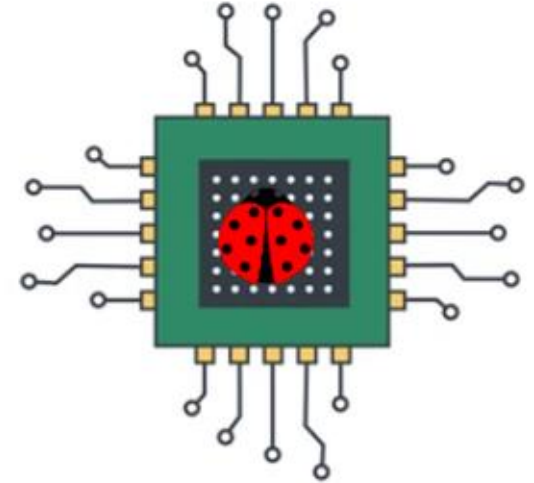
018	CPU May Not Load The Most Recent Data
Problem	Under complex microarchitectural conditions, a read on one logical processor may not receive the most recently stored data by another logical processor.
Implication	Due to this erratum, unpredictable system behavior or a system hang may occur. Intel has only observed this behavior in a synthetic test environment. Intel has not observed this erratum with any commercially available system.
Workaround	It is possible for the BIOS to contain a workaround for this erratum.
Status	For the steppings affected, refer to Summary Tables of Changes .

<https://edc.intel.com/content/www/us/en/design/products/platforms/processor-and-core-i3-n-series-specification-update/002/errata/>

Additional Situations W/ Buggy Behavior



- Hardware **Errata**: design defects or errors.
 - Errata may cause the processor's behavior to **deviate** from published **specifications**.
 - Companies and organizations **document** errata and **changes to designs** to help hardware (and software) engineers.
- We have an **incorrect specification**
 - Bug in the spec leads to bug in the HW
- Or an **imprecise specification**
 - Vagueness in the spec leads to buggy/unwanted behavior



Recall: Increasing Design Complexity Means...



Number of Transistors in CPUs over Time



An Increased Attack Surface!

Hardware Bug Example: Floating Point Division (FDIV)

- The problem: implementation of FDIV on Intel's Pentium chips did not match the specifications.... 😞

The specification:

$$(1 + f) \times 2^e, 0 \leq f < 1, e \in \mathbb{Z}$$

Q: What is the problem with implementing this with hardware / digital logic?

We only have a finite number of bits to work with!!

Finding the bug... in the wild!



This bug was reproduced in the wild in multiple different settings!

- Thomas Nicely, a math professor, was trying to find the reciprocals of large primes in 1994, starting seeing lots of errors.
- He tested on other processors... not able to see the bug...
- Other users able to reproduce it!
- The issue: **a few missing entries from a look-up table!**

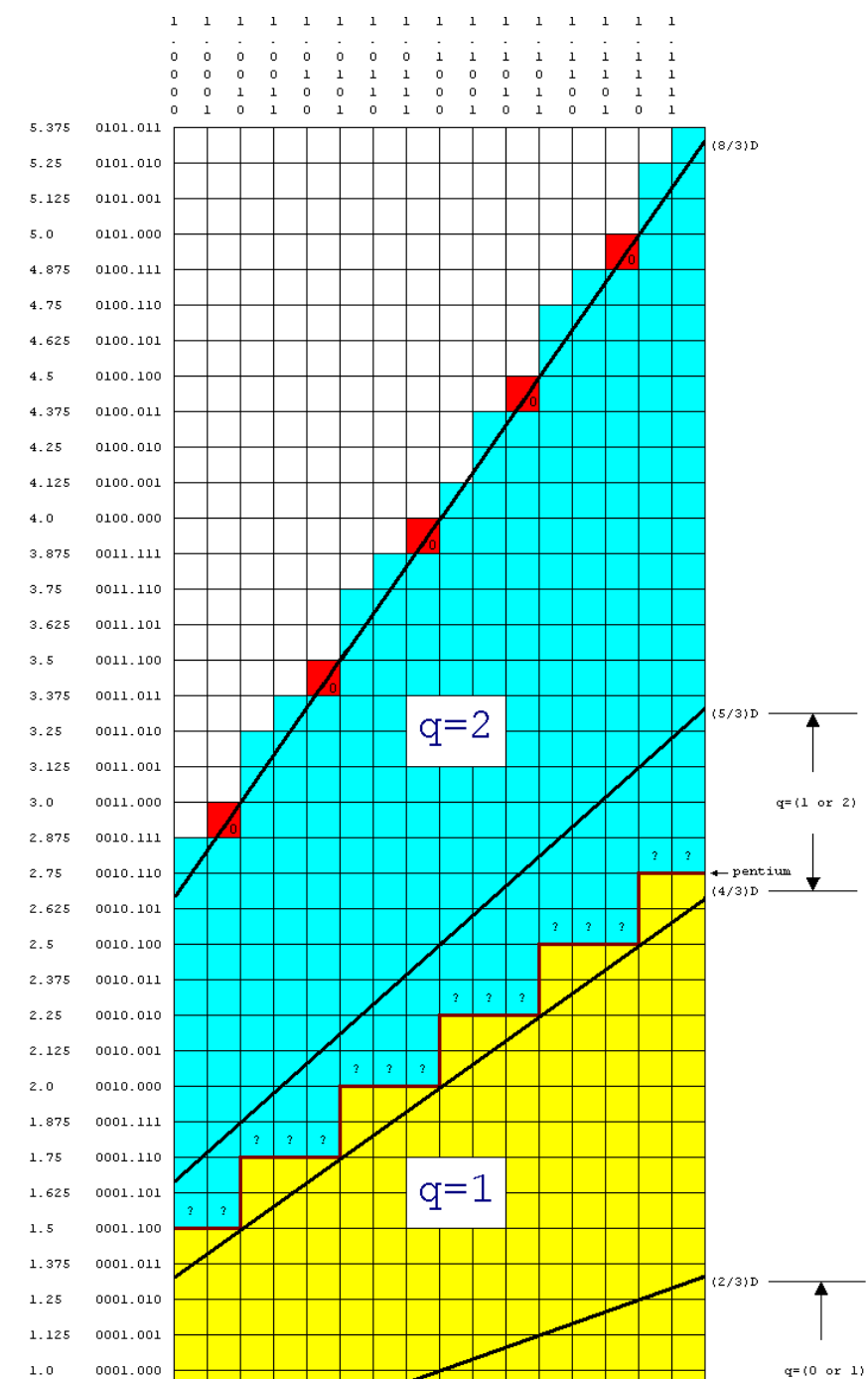
Famous Example



- $4195835 \div 3145727$
- Correct $\approx 1.333820449\dots$
- Buggy Pentium ≈ 1.333739068

What was going wrong?

- Hardware repeatedly chooses the next quotient digit
- Choice is based on a small lookup table
- A few bad entries rare but incorrect divisions



Impact of the Pentium FDIV Bug



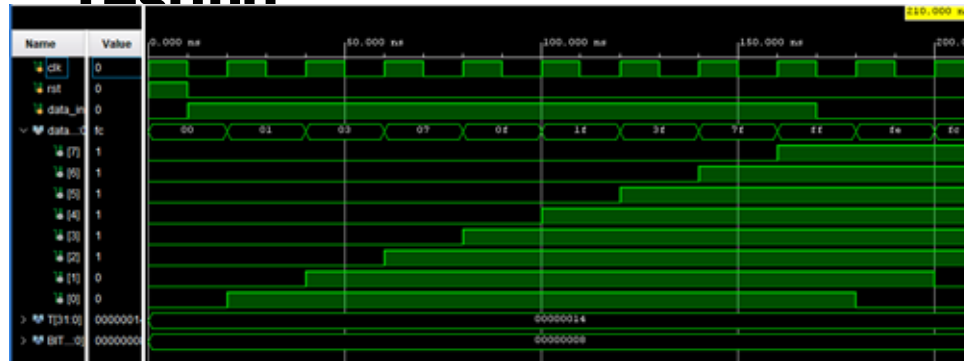
- Costs to Intel to Intel were steep.
 - Much bad press!
 - First, would only replace if you could prove it was impacting you/reproduce...
 - Then they changed their tune, no-questions-asked replacement. \$475M HW replacement cost in 1994!
- A growth in attention, funding and research in formal methods! 😊
 - This was a turning point!
 - We started to realize we needed better solutions than just having experts stare at the code...

Hardware Bug Finding Approaches

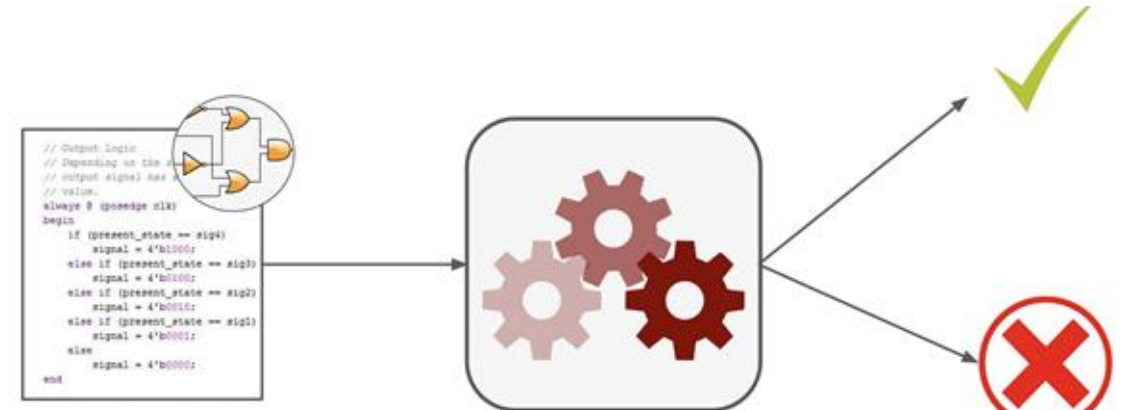
Recall: State of the Art in Hardware Validation



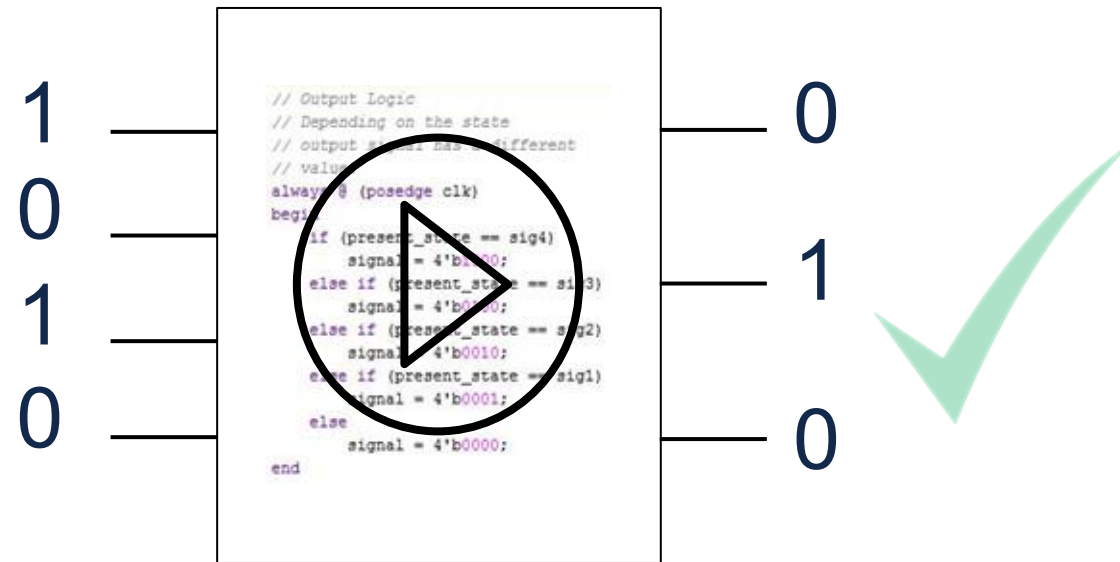
Simulation Based Testing



Formal Methods



Simulation-Based Testing



Simulation-Based Testing: Limitations



```
if (x == 13):  
    ERROR  
if (x > 0):  
    count := count + 1  
else:  
    count := count - 1
```

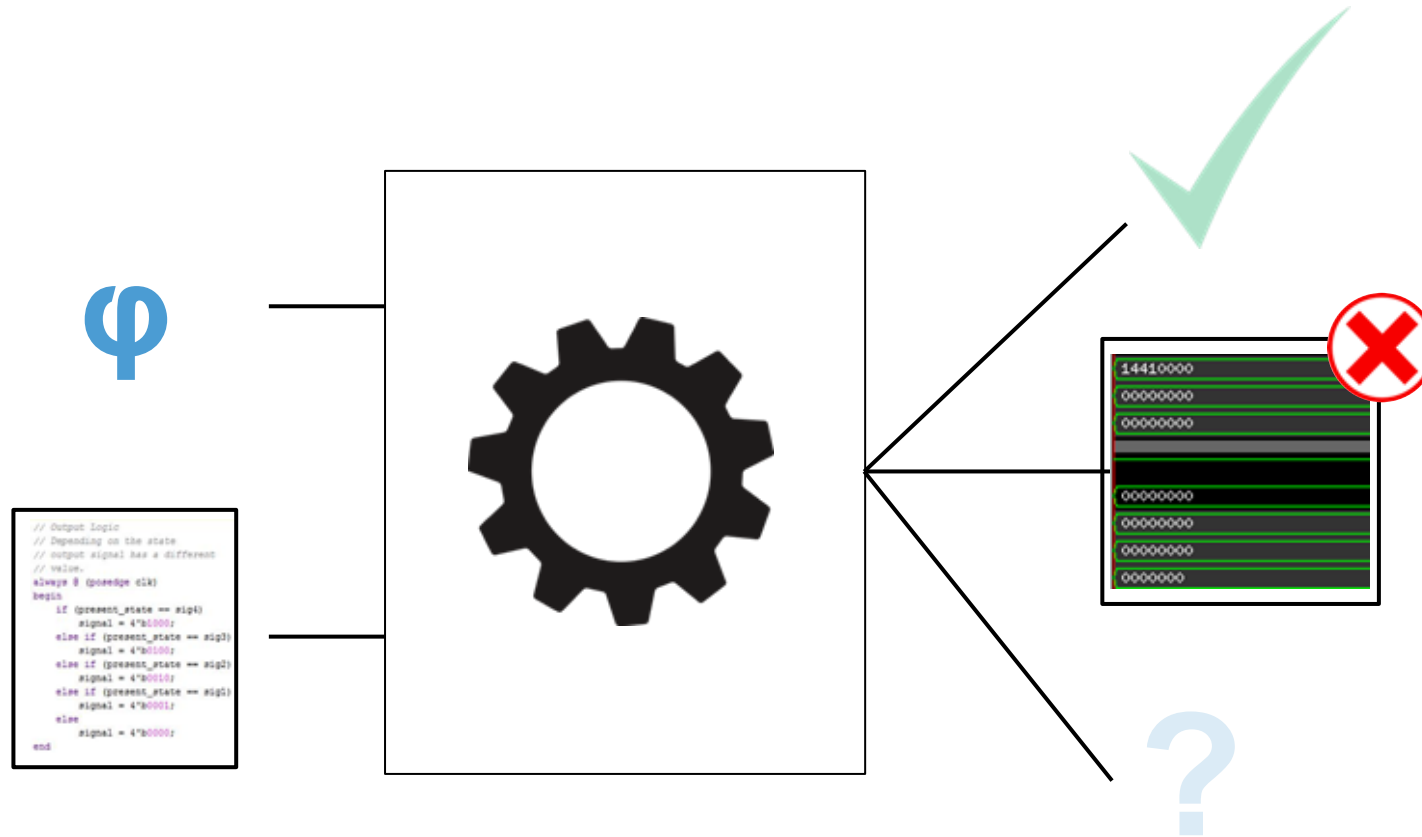
Simulation-Based Testing: Limitations



```
if (x == 13):  
    ERROR  
if (x > 0):  
    count := count + 1  
else:  
    count := count - 1
```

 Coverage

Model Checking

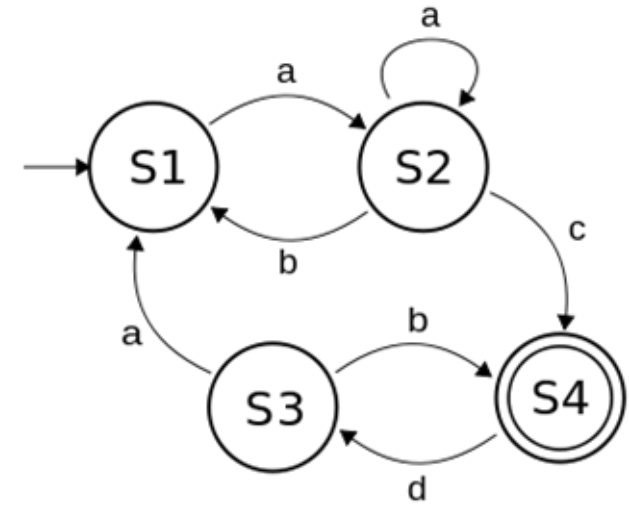


Model-Checking: Limitations



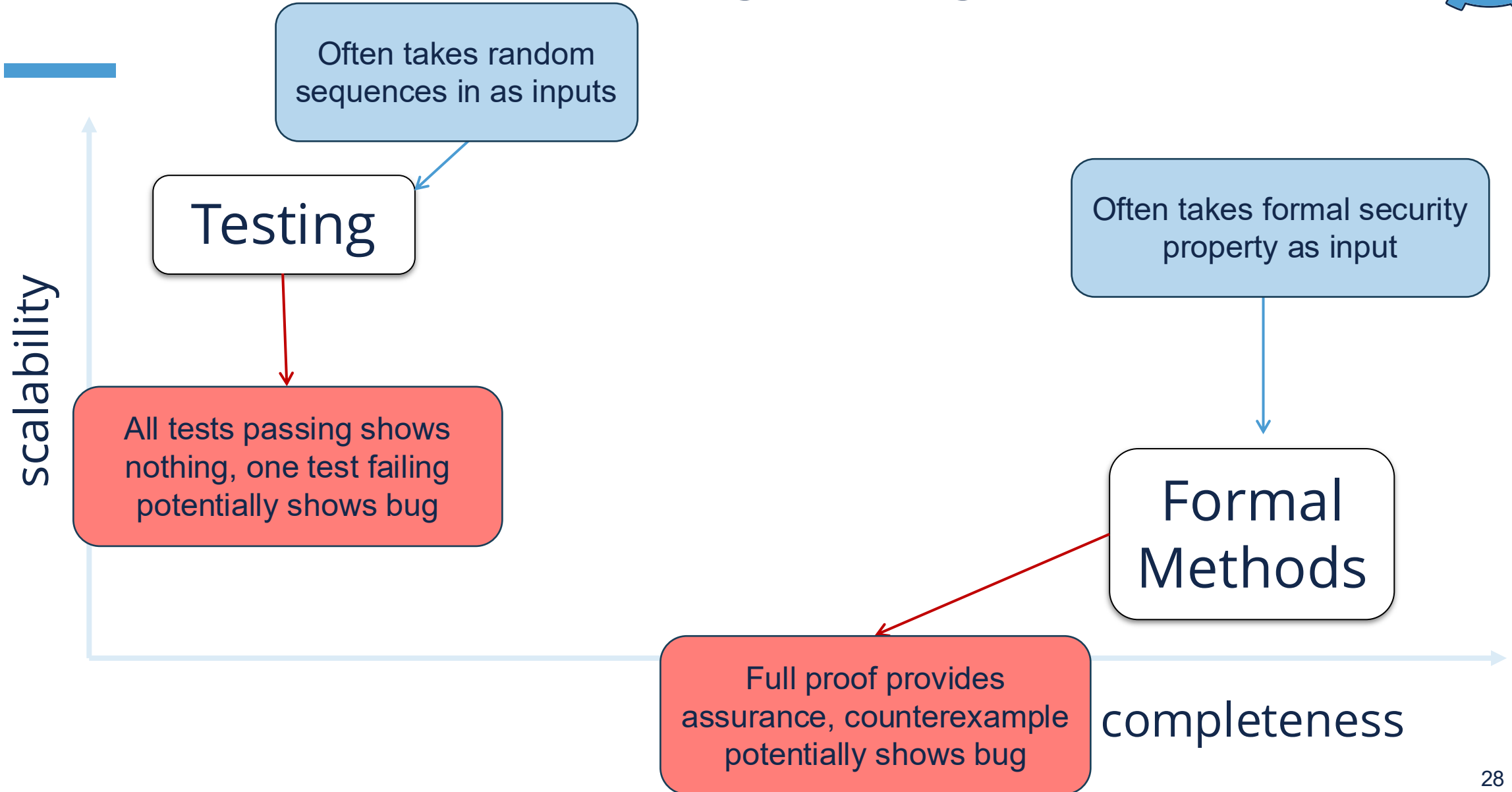
```
// Output Logic
// Depending on the state
// output signal has a different
// value.
always @ (posedge clk)
begin
    if (present_state == sig4)
        signal = 4'b1000;
    else if (present_state == sig3)
        signal = 4'b0100;
    else if (present_state == sig2)
        signal = 4'b0010;
    else if (present_state == sig1)
        signal = 4'b0001;
    else
        signal = 4'b0000;
end
```

formal model



Scalability

Different Approaches to Bug Finding



Another Key Component in Bug Finding...



- **Security experts** identifying **common weaknesses** and **documenting vulnerabilities**.
- The writing of **good security properties** and **specifications**.
- This then allows the community as a whole to vet and check their HW design for the same bugs and flaws.

UNIT 1: TRUSTING THE MACHINE

Hardware CWEs



Why CWEs?

- They help security professionals compare bugs and properties across designs instead of treating every flaw as totally unique.
- Different from CVE! How?



Common Weakness Enumeration

A community-developed list of SW & HW weaknesses that can become vulnerabilities



Examples of Hardware CWEs

- Improper access control for protected resources
- Improper isolation of shared resources
- Incorrect privilege management
- Debug/test interface exposed or insufficiently protected
- Side-channel exposure through timing, power, or shared state

Some Hardware Background: Debug Interface

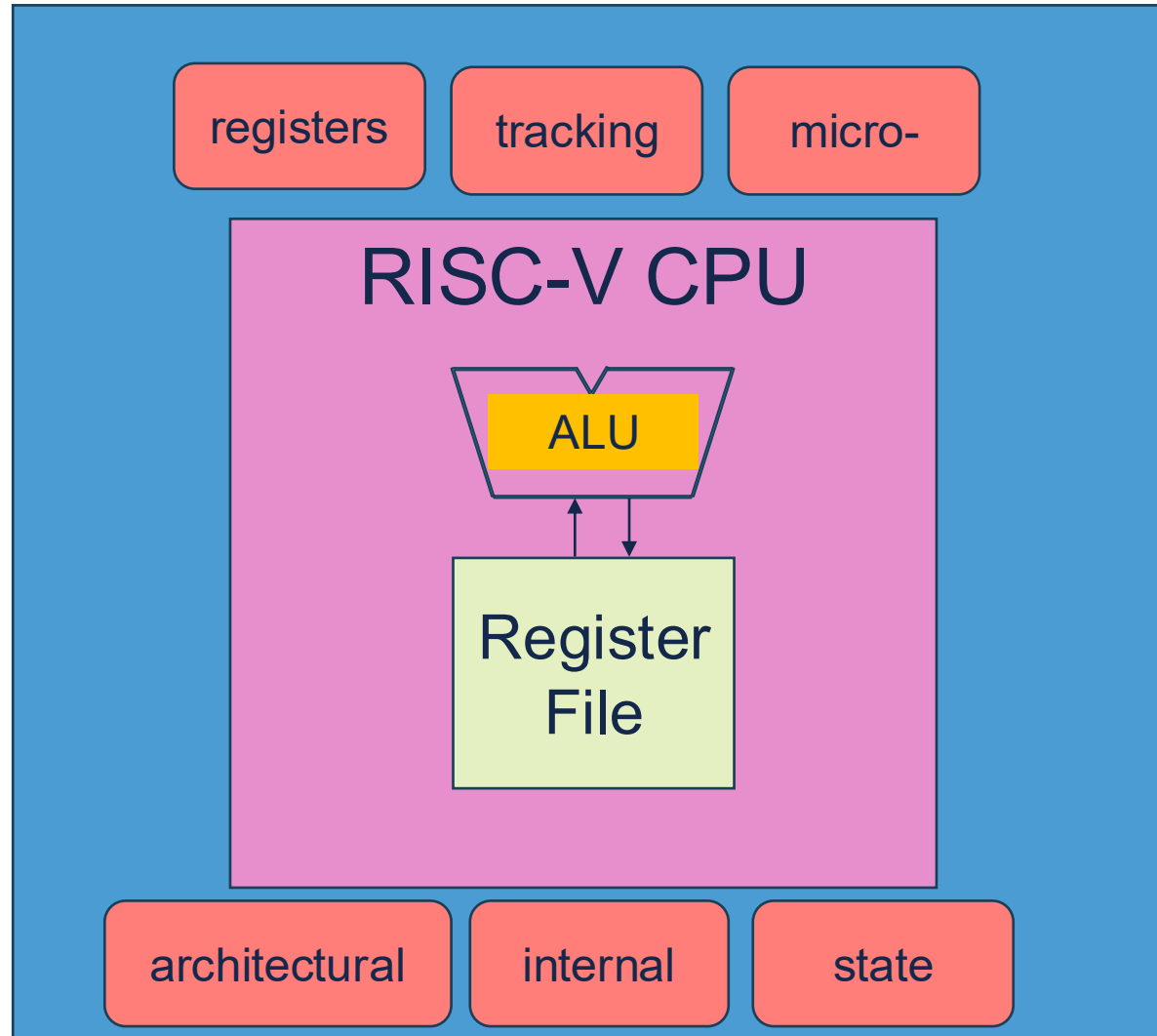


Desired behavior:

- Imagine some boolean variable ***debug_mode***
- Only when this is **TRUE**, should the processor be able to inspect the contents of the red registers.

Q: what type of users should have access this to state?

Most ISAs only expose behavior/state at clock-cycle boundaries or after the end of an instruction, for example





Examples of Hardware CWEs

- Improper access control for protected resources
- Improper isolation of shared resources
- Incorrect privilege management
- Debug/test interface exposed or insufficiently protected
- Side-channel exposure through timing, power, or shared state

Activity: CWEs

1. Go to the MITRE Website and find the top 10 Hardware CWEs.
2. Spend a few minutes reading the docs + researching what folks have to say about your CWE. Be able to explain in plain English:
 - What goes wrong?
 - What hardware state/resource is involved?
 - What could an attacker gain?
3. If you were to try to write down the formal spec or property. Use simple if–then pseudocode:

IF _____ happens, **THEN** _____ must / must not happen?
Would it help you if you could add timing/ordering into your property?

4. Share with your groupmates.

Be ready to report: CWE number + one-sentence explanation + one property you would want to verify. Then discuss:

- What do your CWEs have in common?
- Which seems easiest to detect during hardware design?
- Which seems hardest to test for?
- Any terminology or behavior that surprised you?