CS 194-6
Digital Systems Project Laboratory

Lecture 4: Testing

2008-10-6

John Lazzaro
(www.cs.berkeley.edu/~lazzaro)

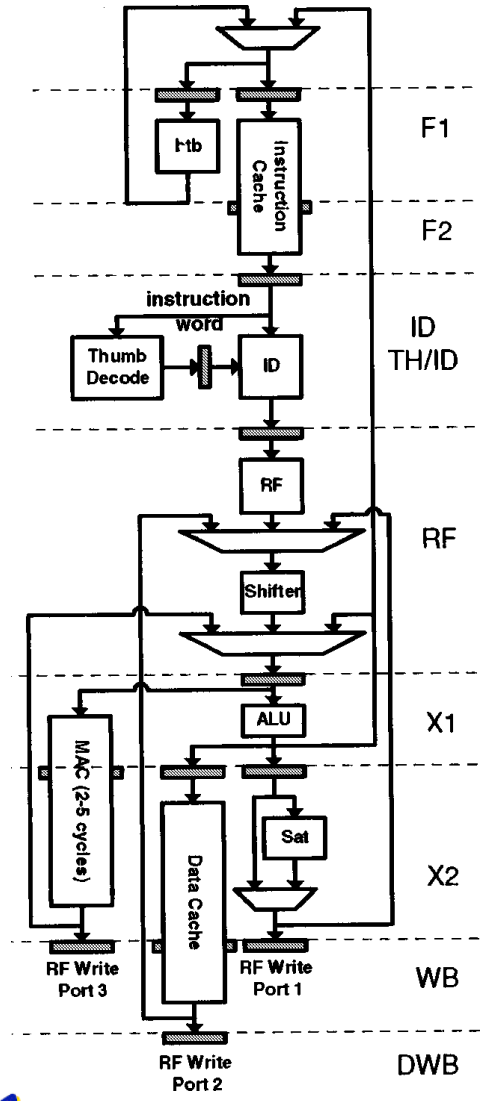
TA: Greg Gibeling

www-inst.eecs.berkeley.edu/~cs194-6/

Today: Testing Processors

- Making a processor test plan
- Unit testing techniques
- State machine testing

Lecture Focus: Functional Design Test



testing goal

Not manufacturing tests ...

The processor design correctly executes programs written in the supported subset of the ISA

Clock speed? CPI? Upcoming lectures ...

Intel XScale ARM Pipeline, IEEE Journal of Solid State Circuits, 36:11, November 2001

Four Types of Testing

Big Bang: Complete Processor Testing

Top-down testing

- complete processor testing

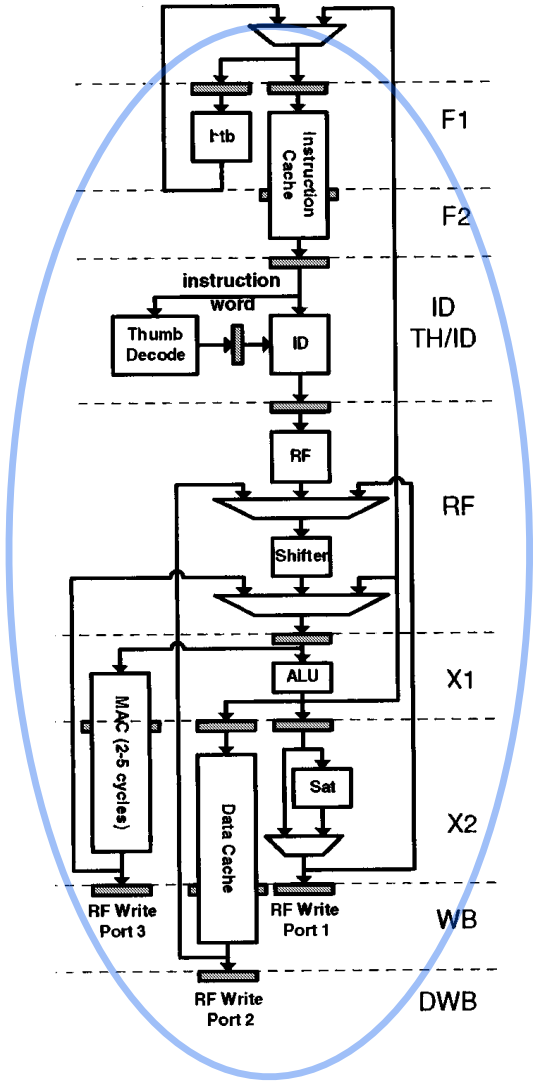
Bottom-up testing

how it works

Assemble the complete processor.

Execute test program suite on the processor.

Check results.



What makes it tricky ...

Observation: On a prototype processor, the correctness of every executed instruction is suspect.

Consequence: One can never be totally sure of the correctness of instructions that surround an "instruction under test".

Methodical Approach: Unit Testing

Top-down testing

complete processor testing

- unit testing

Bottom-up testing

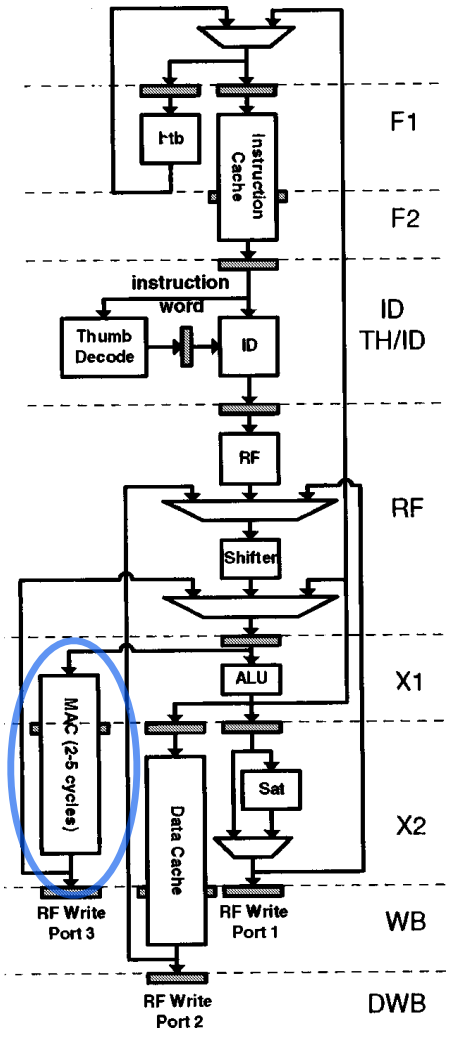
how it works

Remove a block from the design.

Test it in isolation against specification.

What if the specification has a bug?

What if team members do not use the exact same specification?



Climbing the Hierarchy: Multi-unit Testing

Top-down testing

complete processor testing

- multi-unit testing

unit testing

Bottom-up testing

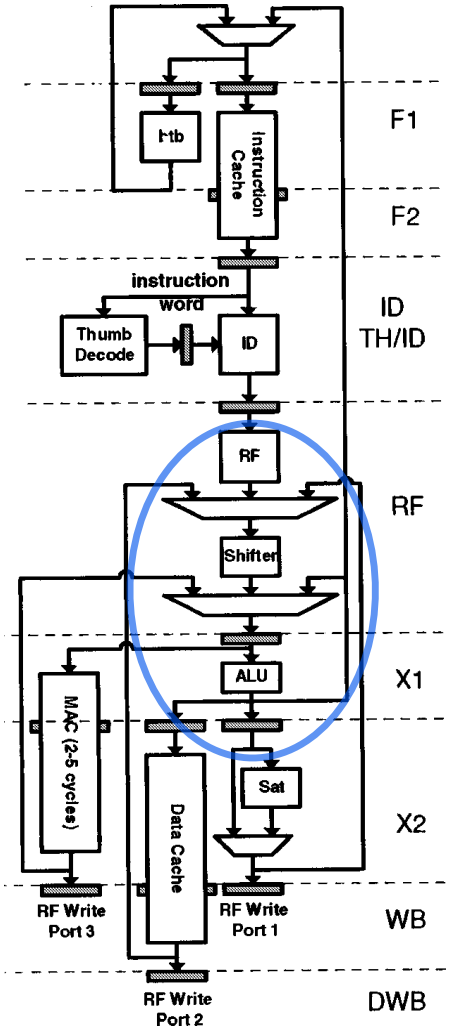
how it works

Remove connected blocks from design.

Test in isolation against specification.

How to choose partition?

How to create specification?



Processor Testing with Self-Checking Units

Top-down testing

complete processor testing

- processor testing with self-checks

multi-unit testing

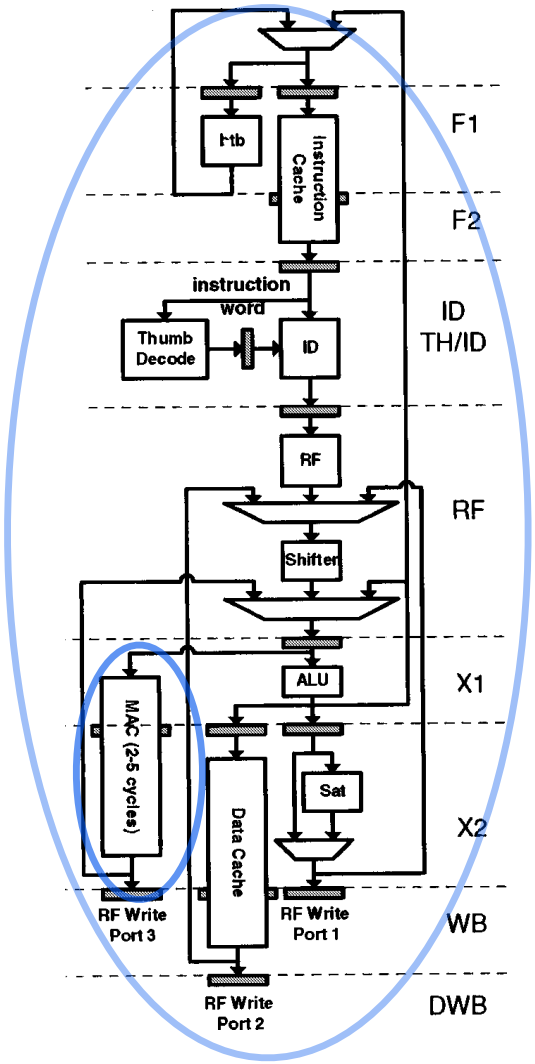
unit testing

Bottom-up testing

how it works

Add self-checking to units

Perform complete processor testing



Good for Xilinx? ModelSim?

Why not use self-checks for all tests?

Testing: Verification vs. Diagnostics

Top-down testing

- complete processor testing
- processor testing with self-checks
- multi-unit testing
- unit testing

Bottom-up testing

- **Verification:**

  A yes/no answer to the question "Does the processor have one more bug?"

- **Diagnostics:**

  Clues to help find and fix the bug.

Which testing types are good for verification? For diagnostics?

Xilinx: Observability and Controllability

Top-down testing

complete processor testing

processor testing with self-checks

multi-unit testing

unit testing

Bottom-up testing

- **Observability:**

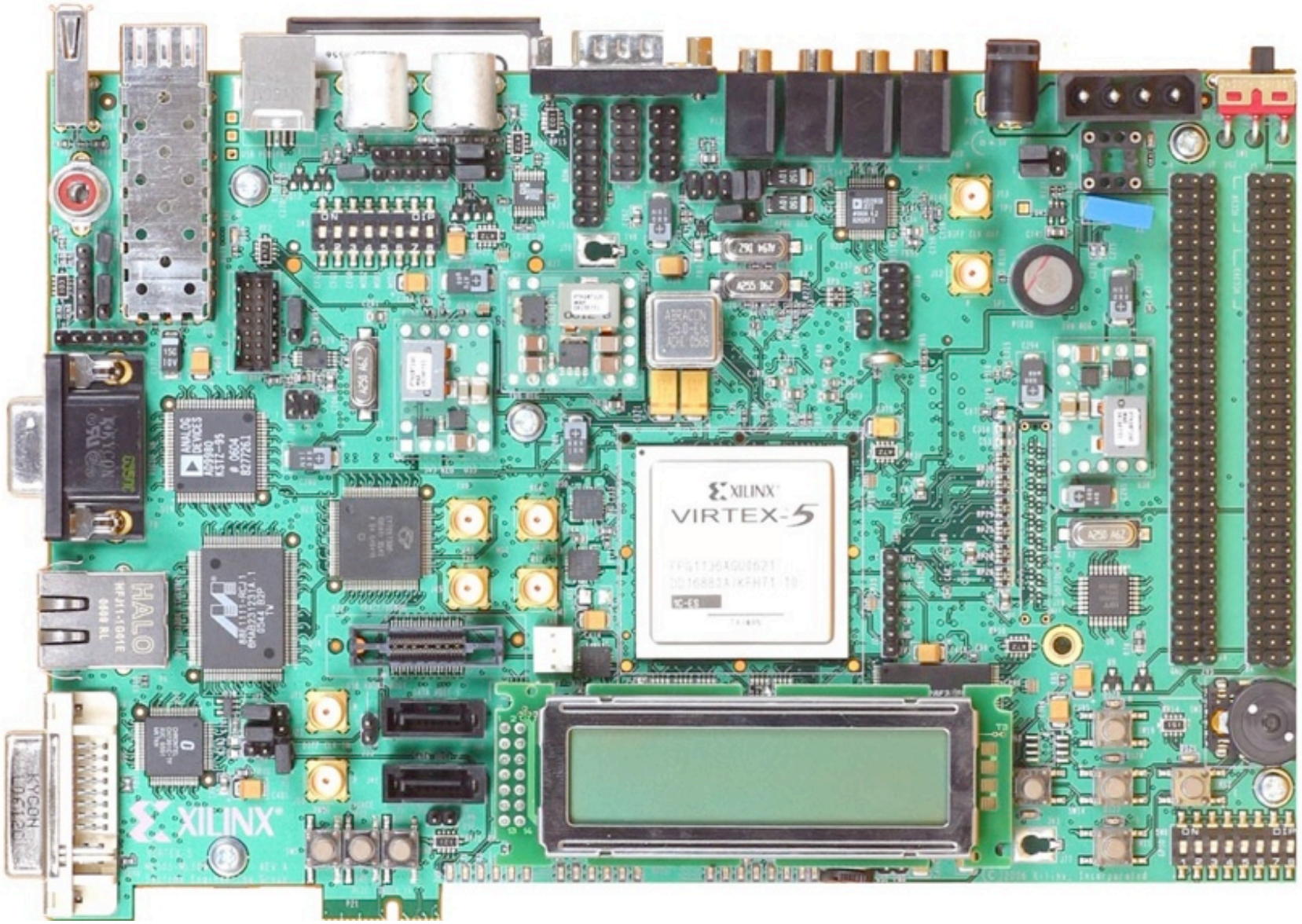
  Can I sense the state I need to diagnose a bug on the board?

- **Controllability:**

  Can I force a flip-flop into known state to diagnose bugs on the board?

Use ChipScope for observability.

Also, switches and LEDs on the board.



Writing a Test Plan

The testing timeline ...

Top-down testing

complete processor testing

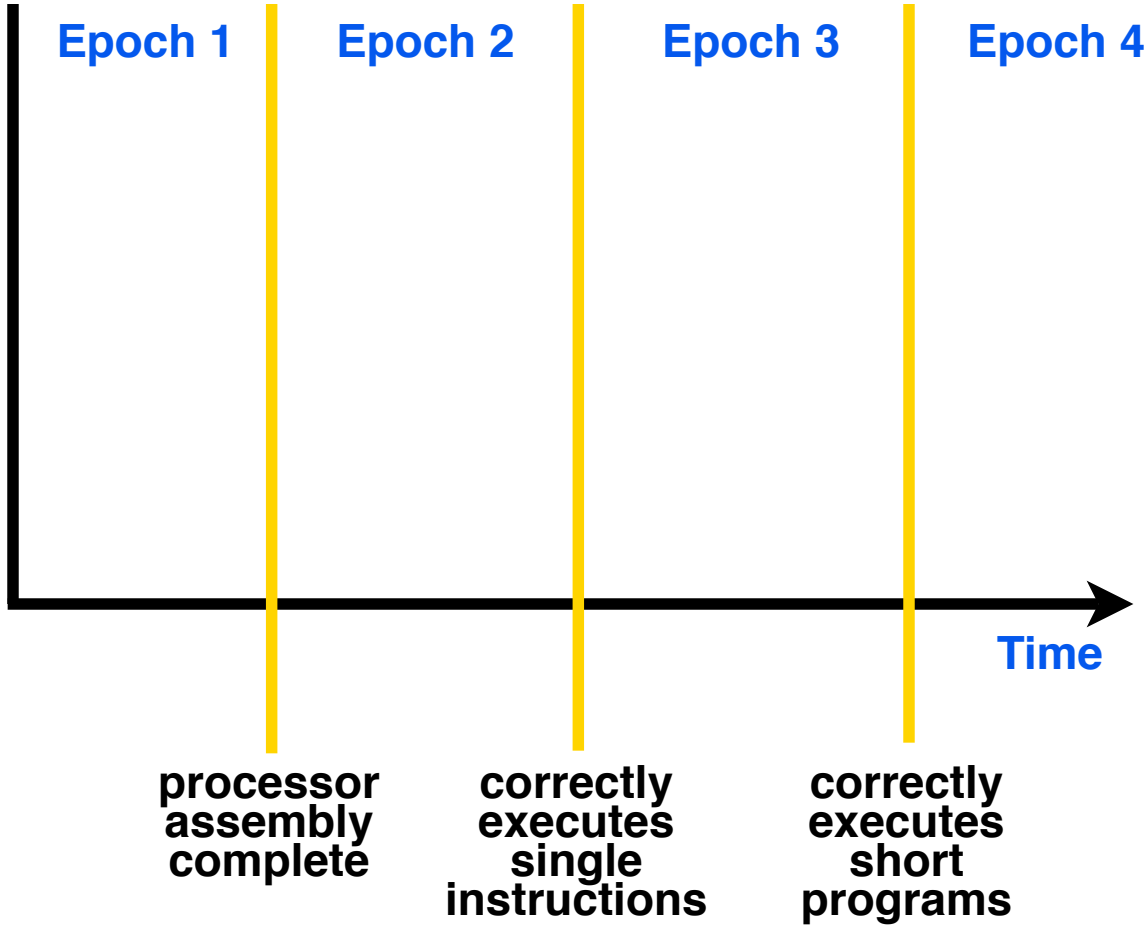
processor testing with self-checks

multi-unit testing

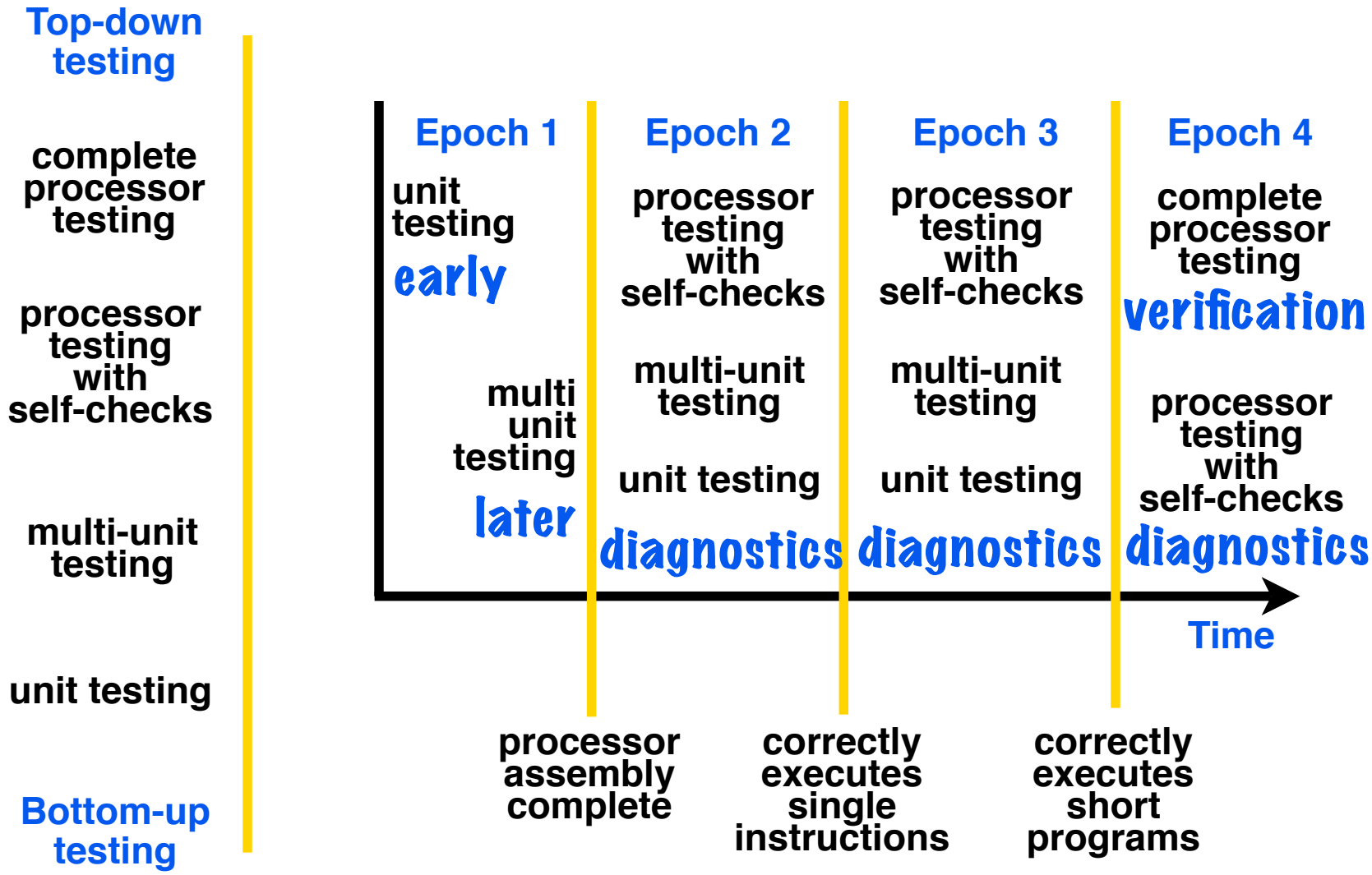
unit testing

Bottom-up testing

Plan in advance what tests to do when ...



An example test plan ...



Spr 05: "Works in Modelsim, not on board"

In the end, Team Ergo failed because they didn't figure out how to handle some write buffer conditions. They passed most tests but not that one.

As far as checkoffs go, Ergo passed the following in simulation: basic, corner, hammer, 3/8 tests for base, extra. Nothing worked on board.

Ted Hong, TA Spring 05.

Solving "Works in ModelSim, not on board"

Top-down testing

complete processor testing

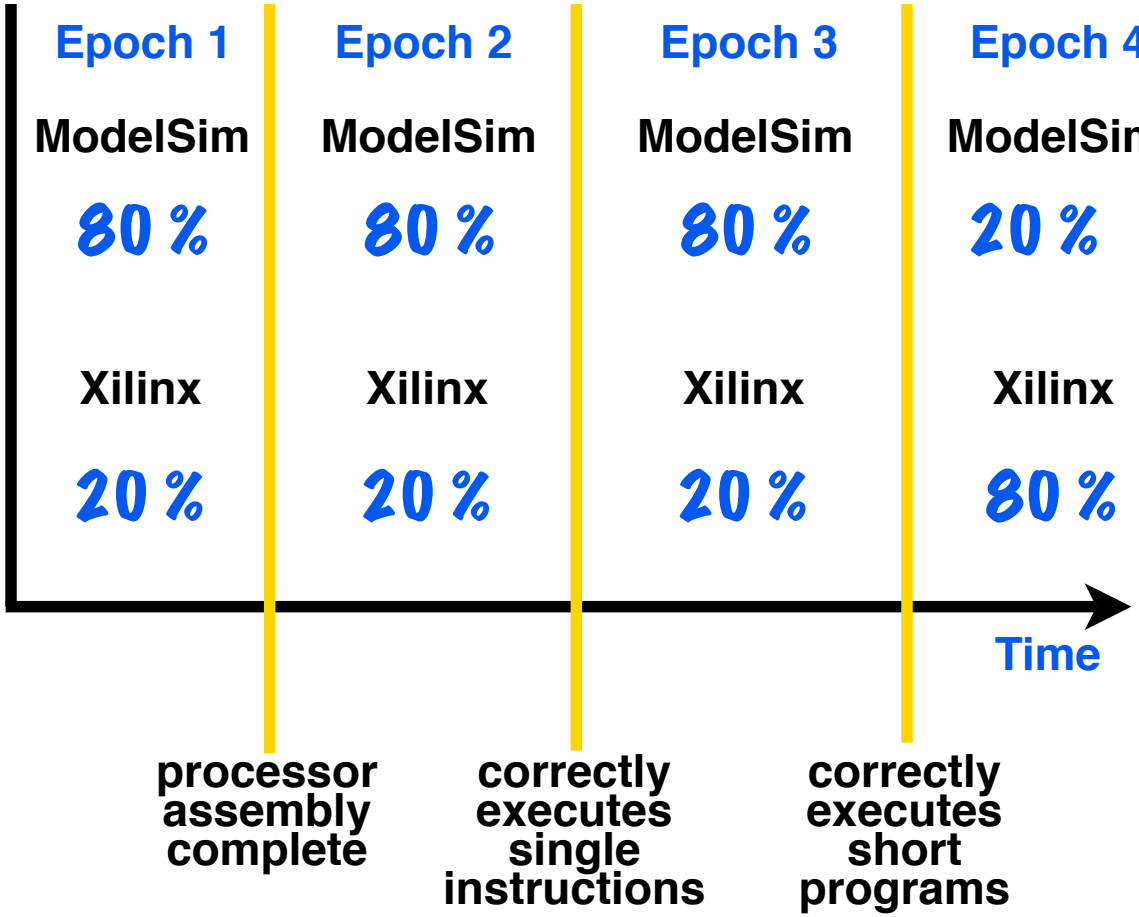
processor testing with self-checks

multi-unit testing

unit testing

Bottom-up testing

Solution: get confidence in "going to board" earlier ...

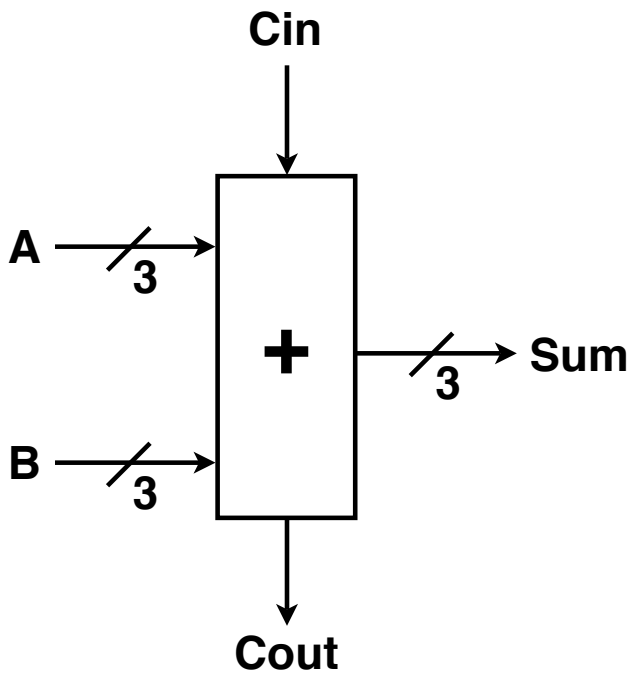


Catch "warnings and errors", signal name misspellings.

Errors: "latch generated", "combinational loop detected", etc

Unit Testing

Combinational Unit Testing: 3-bit Adder



Number of input bits ? 7

Total number of possible input values?

$2^7 = 128$

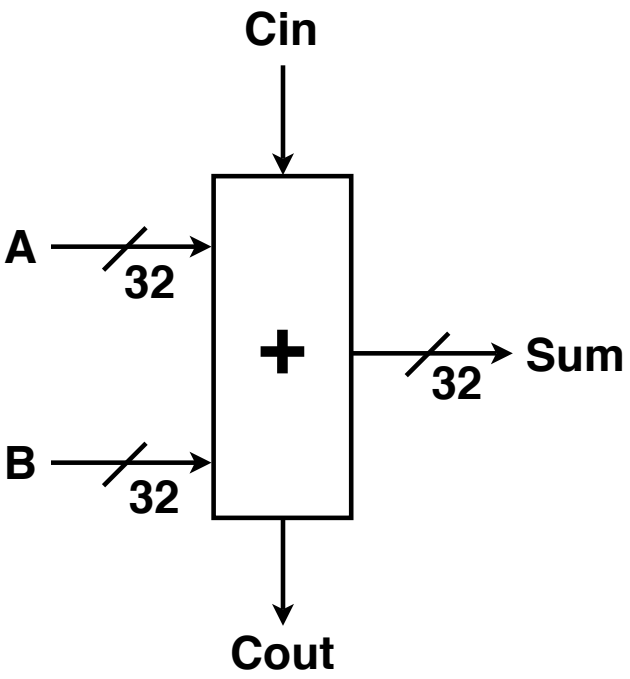
Just test them all ...

Apply "test vectors" 0,1,2 ... 127 to inputs.

100% input space "coverage"

"Exhaustive testing"

Combinational Unit Testing: 32-bit Adder



Number of input bits ? 65

Total number of possible input values?

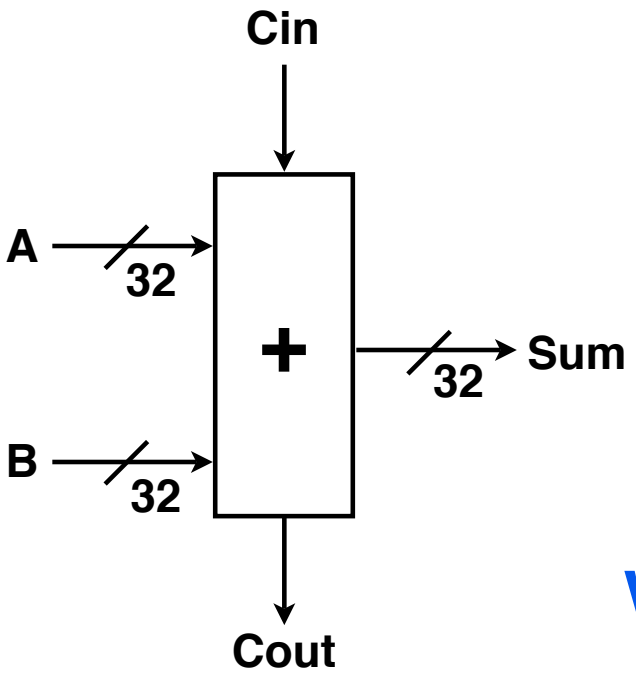
$2^{65} = 3.689e+19$

Just test them all?

Exhaustive testing does not "scale".

"Combinatorial explosion!"

Test Approach 1: Random Vectors

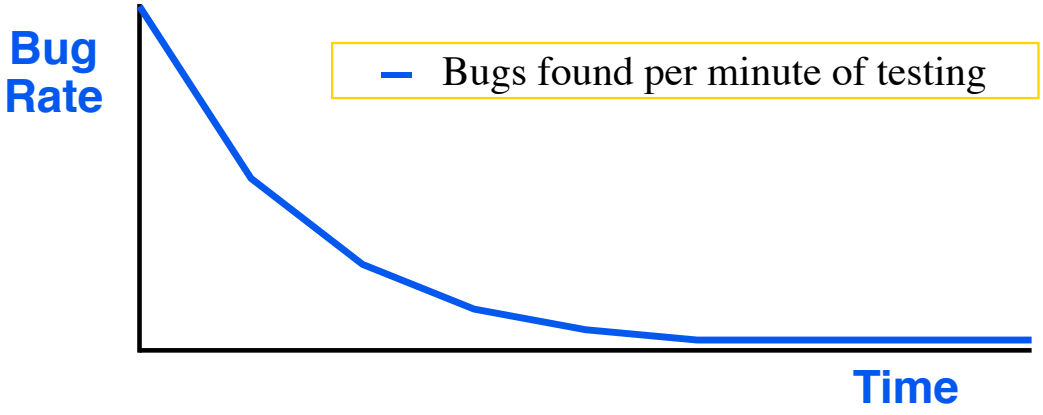


how it works

Apply random A, B, Cin to adder.

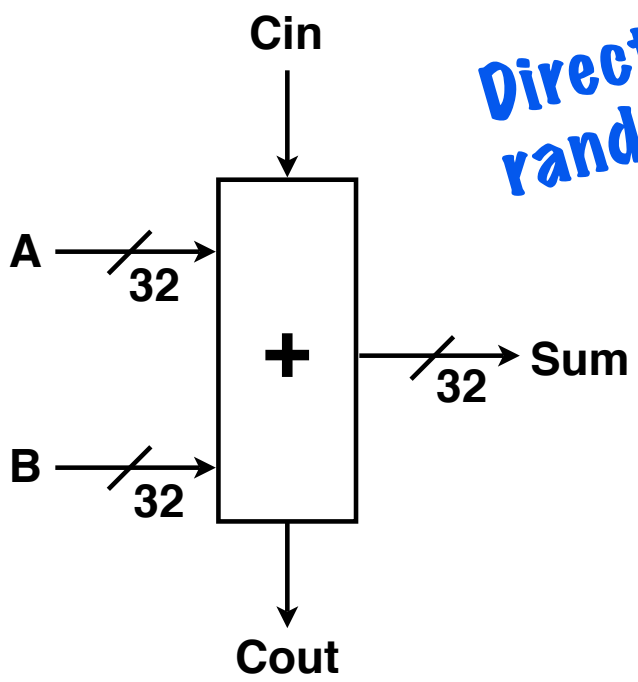
Check Sum, Cout.

When to stop testing? Bug curve.



How? Use $random to set inputs to the testbench.

Test Approach 2: Directed Vectors



Directed random?

how it works

Hand-craft test vectors to cover "corner cases"

A == B == Cin == 0

"Black-box": Corner cases based on functional properties.

Examples ?

"Clear-box": Corner cases based on unit internal structure.

Examples ?

State Machine Testing

CPU project examples:

DRAM controller state machines

Cache control state machines

Branch prediction state machines

Spring 05: Final project FSM woes ...

Neither groups passed the checkoff today. They both had it working in simulation, but could not push to board.

It seems that the problem was not in their cache design, but in their ability to perform testing using finite state machines on the board. Both groups underestimated the amount of time it would take to make a working fsm, and both ran into errors.

Dave Marquardt, TA Spring 05.

Specification: Traffic Light Controller

Inputs

CLK Change Rst

Outputs

R (red) →

Y (yellow) →

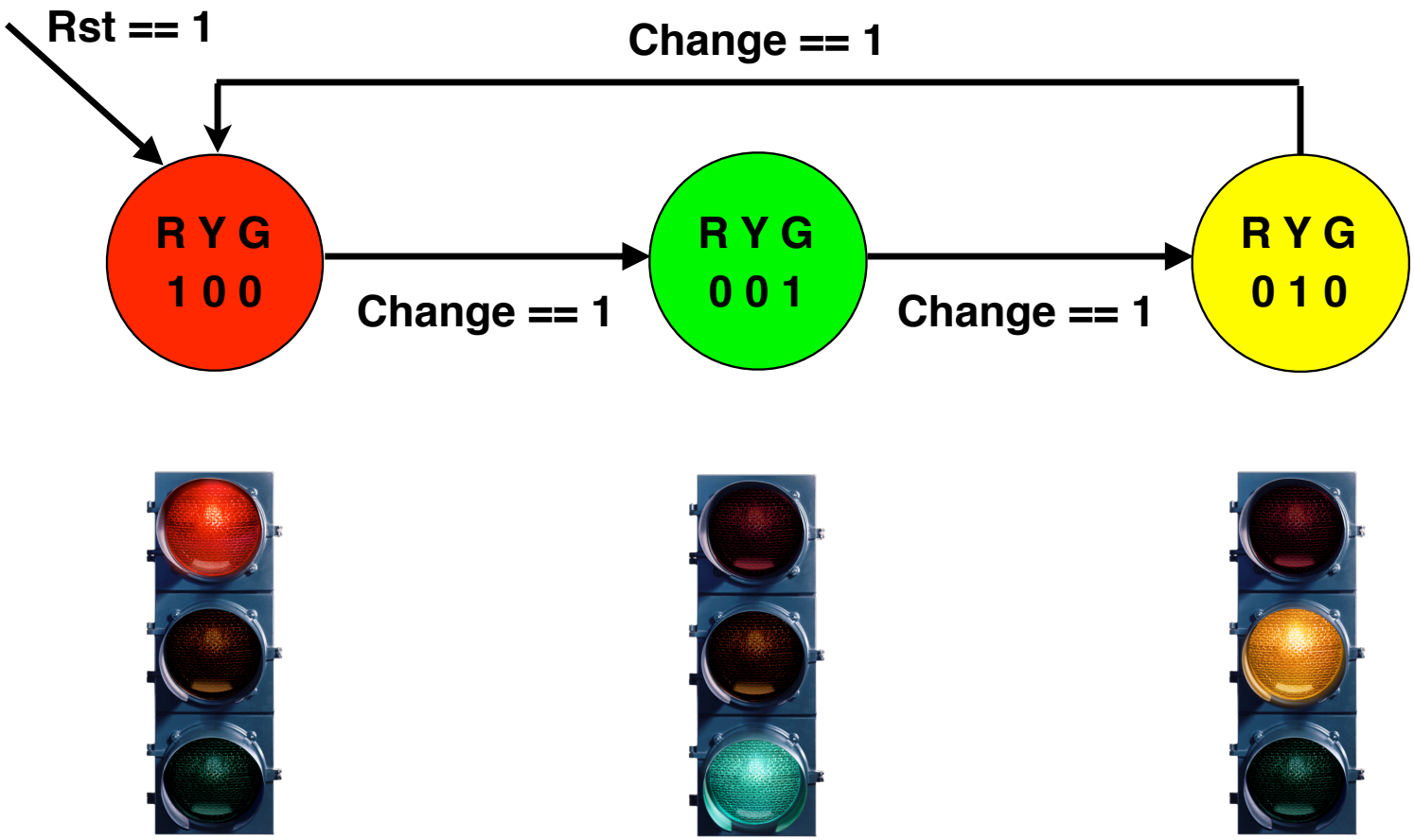
G (green) →

If Change == 1 on positive CLK edge traffic light changes

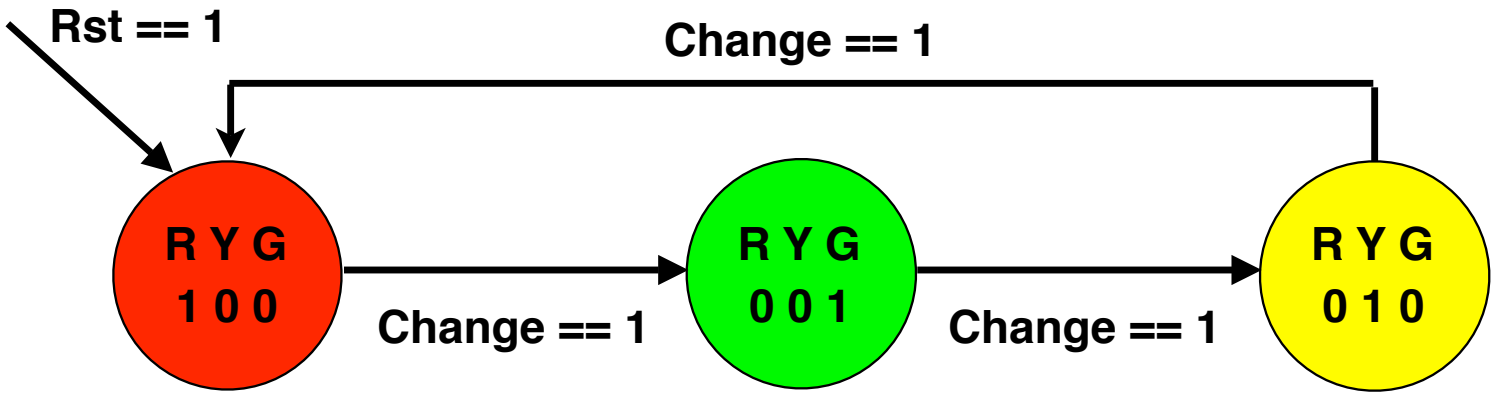
If Rst == 1 on positive CLK edge R Y G = 1 0 0

R	Y	G
1	0	0

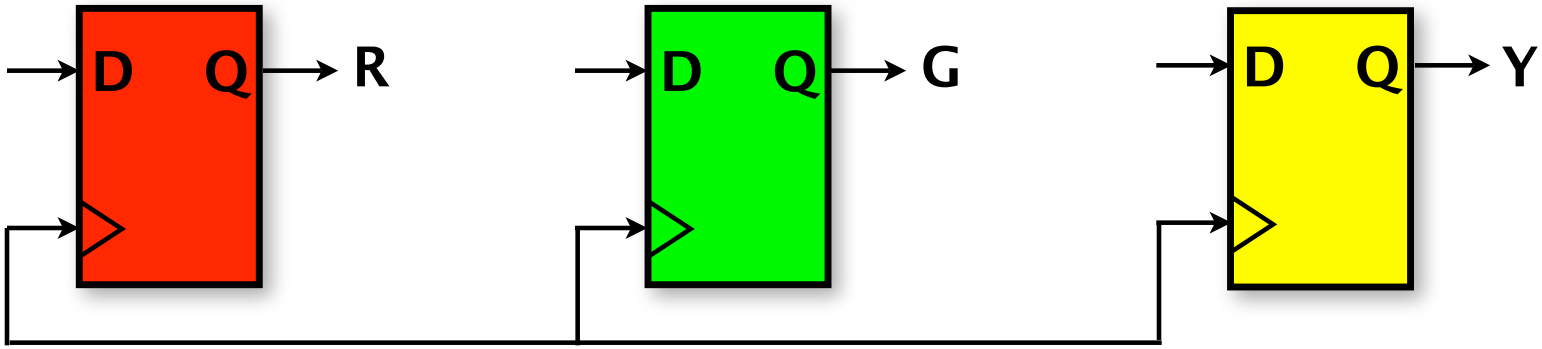
State Machine: Traffic Light Controller



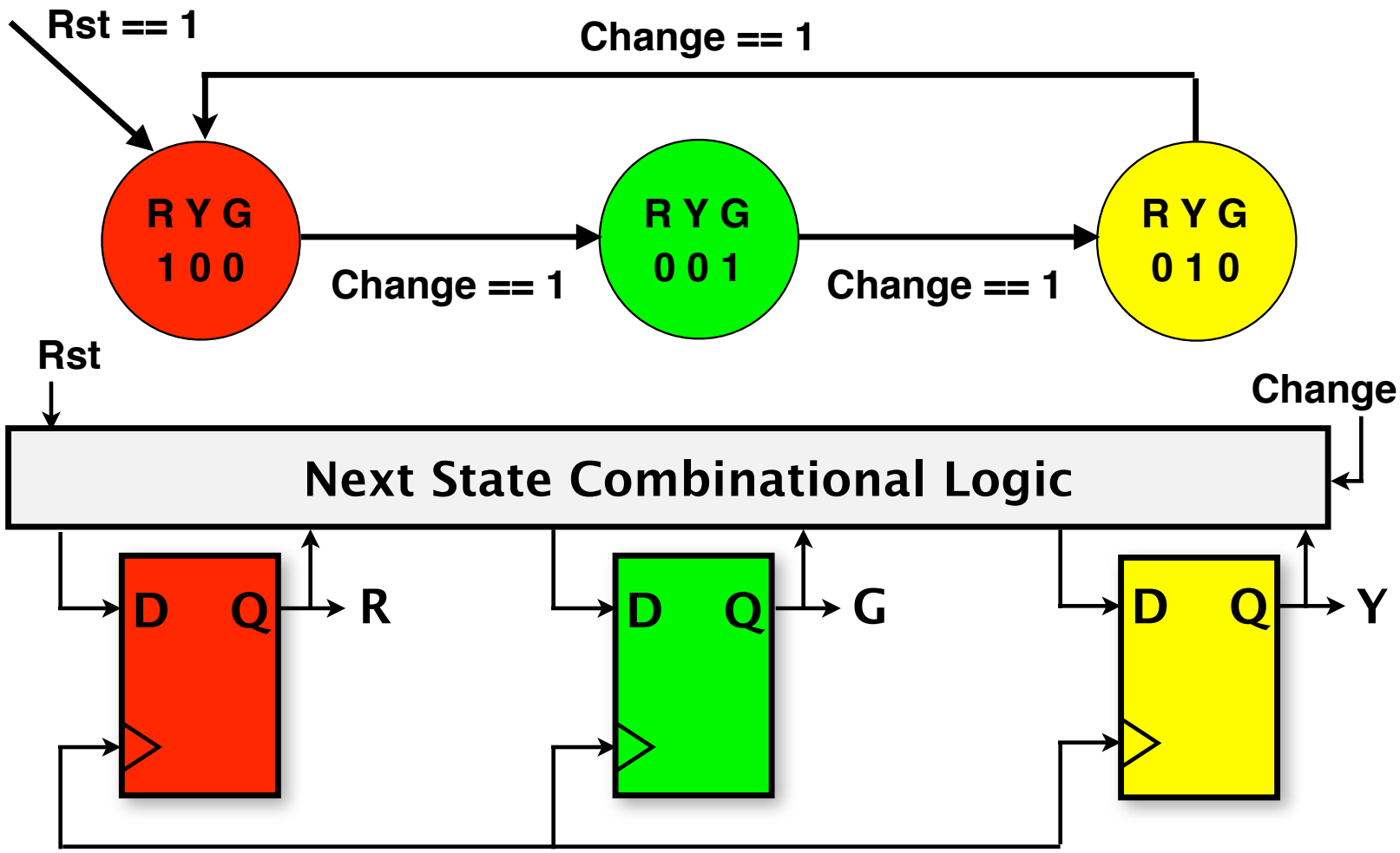
State Assignment: Traffic Light Controller



"One-Hot Encoding"

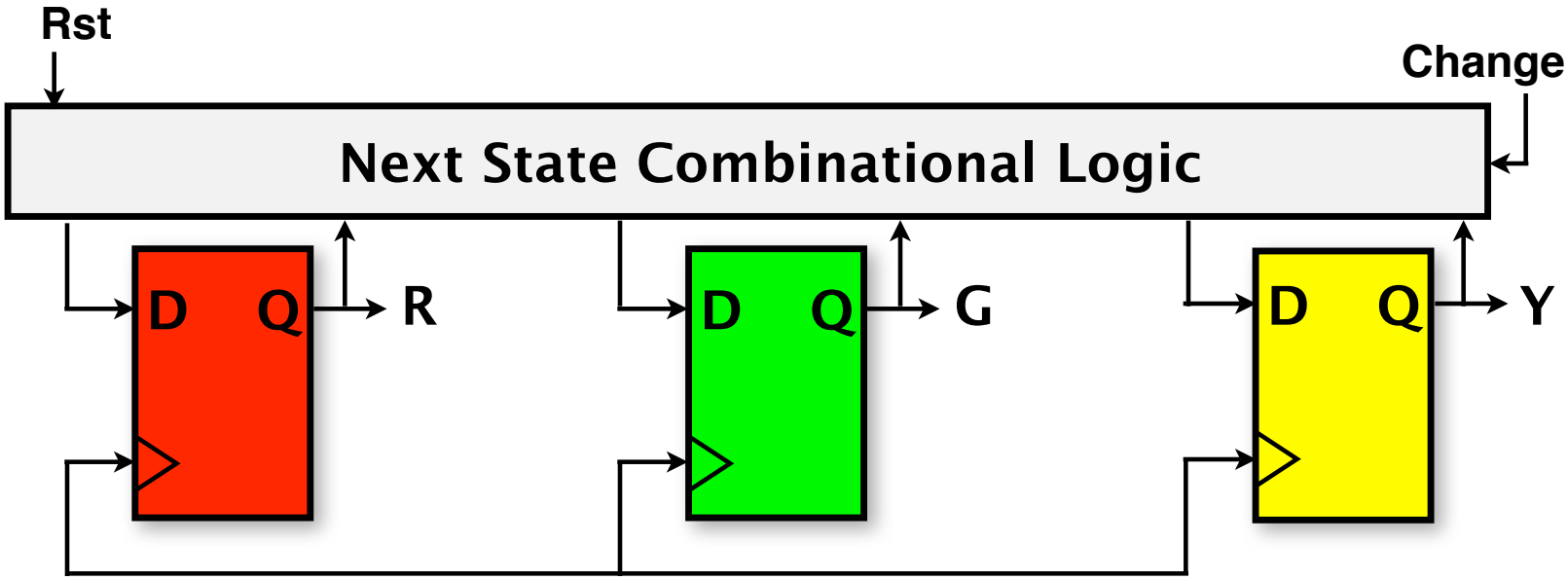


Next State Logic: Traffic Light Controller



State Machine Testing

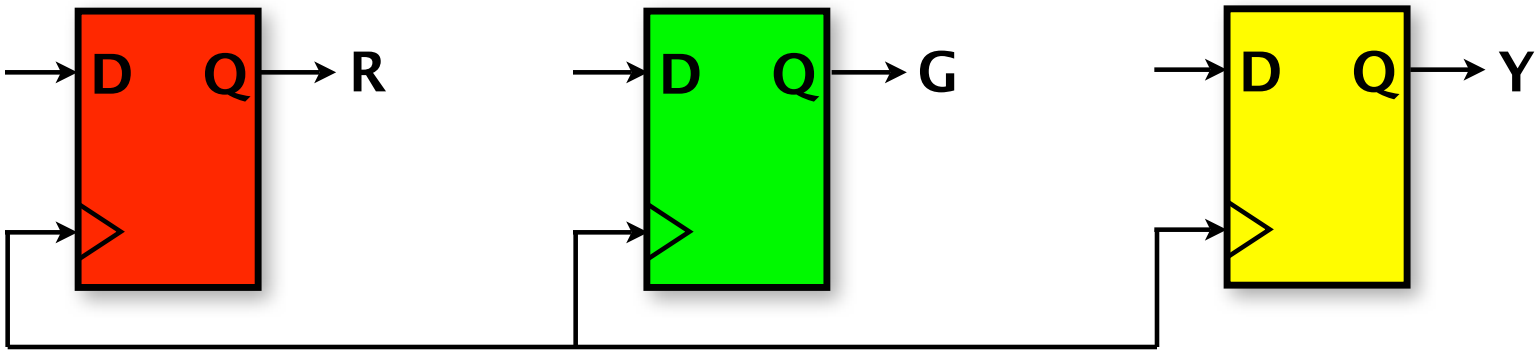
Testing State Machines: Break Feedback



Isolate "Next State" logic.
Test as a combinational unit.

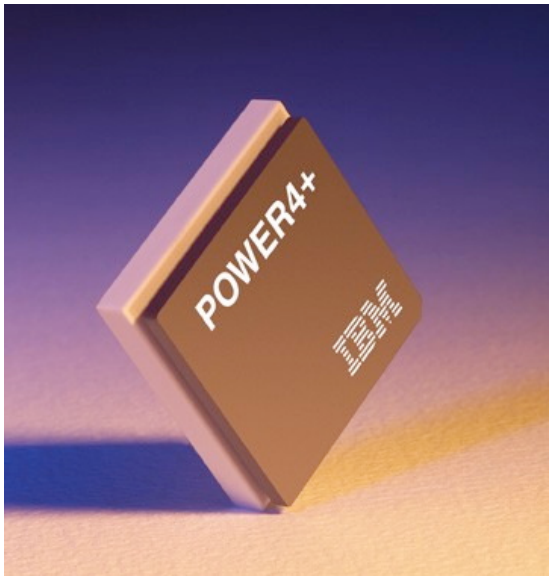
Easier with certain Verilog coding styles?

State Verilog: Traffic Light Controller

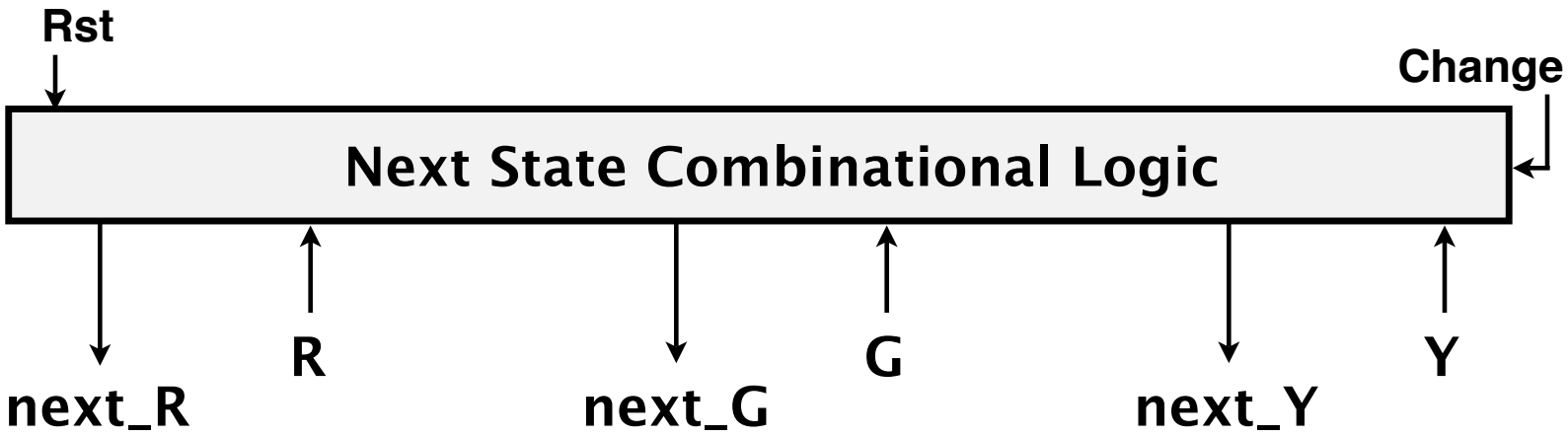



```
wire    next_R, next_Y, next_G;
output  R, Y, G;

ff ff_R(R, next_R, CLK);
ff ff_Y(Y, next_Y, CLK);
ff ff_G(G, next_G, CLK);
```



Next State Verilog: Traffic Light Controller




```
wire    next_R, next_Y, next_G;

assign next_R = rst ? 1'b1 : (change ? Y : R);
assign next_Y = rst ? 1'b0 : (change ? G : Y);
assign next_G = rst ? 1'b0 : (change ? R : G);
```

Verilog: Complete Traffic Light Controller

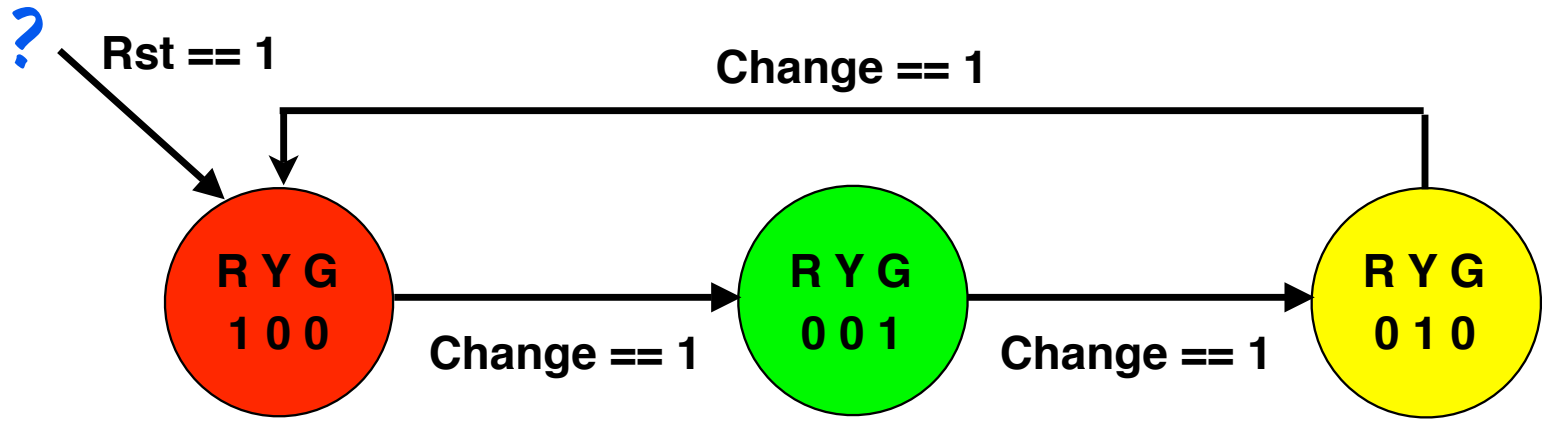
```
wire    next_R, next_Y, next_G;
output  R, Y, G;

assign next_R = rst ? 1'b1 : (change ? Y : R);
assign next_Y = rst ? 1'b0 : (change ? G : Y);
assign next_G = rst ? 1'b0 : (change ? R : G);

ff ff_R(R, next_R, CLK);
ff ff_Y(Y, next_Y, CLK);
ff ff_G(G, next_G, CLK);
```

State Machine Testing II

Testing State Machines: Arc Coverage



Force machine into each state.
Test behavior of each arc.

Is this technique always practical to use?

Conclusion -- Testing Processors

- **Bottom-up test for diagnosis, top-down test for verification.**
- **Make your testing plan early!**
- **Unit testing: avoiding combinatorial explosions.**

Next Monday:

M 10/13	**L5:** Pipelined CPU Design

This Friday:
Project check-ups, 125 Cory, 10 AM