CS 152
Computer Architecture and Engineering

Lecture 4 – Testing and Teamwork

Congrats on Lab 1 !

2005-9-8

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

TAs: David Marquardt and Udam Saini

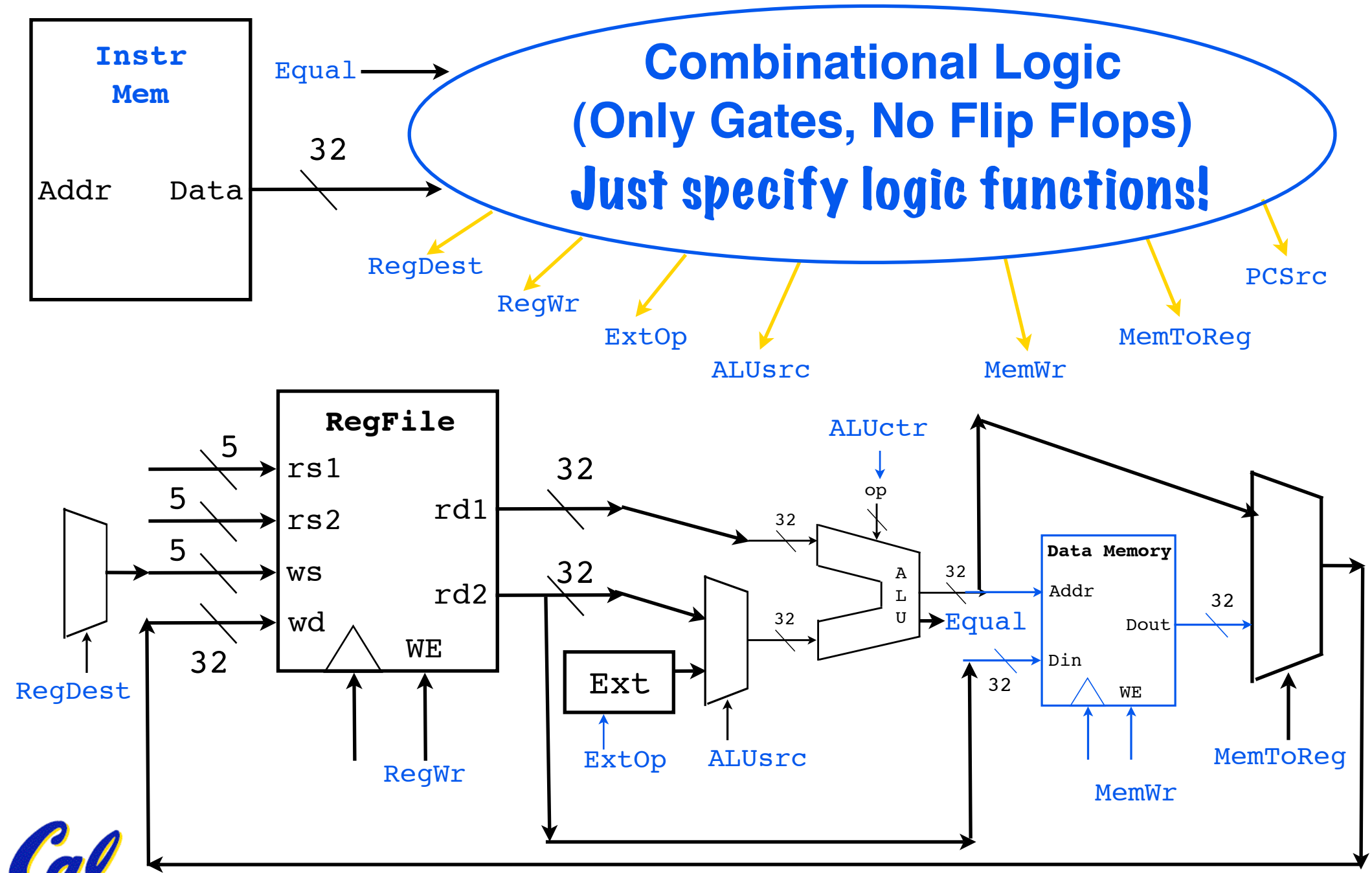
www-inst.eecs.berkeley.edu/~cs152/

Thoughts on Lab 1, Fall 05 ...

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".

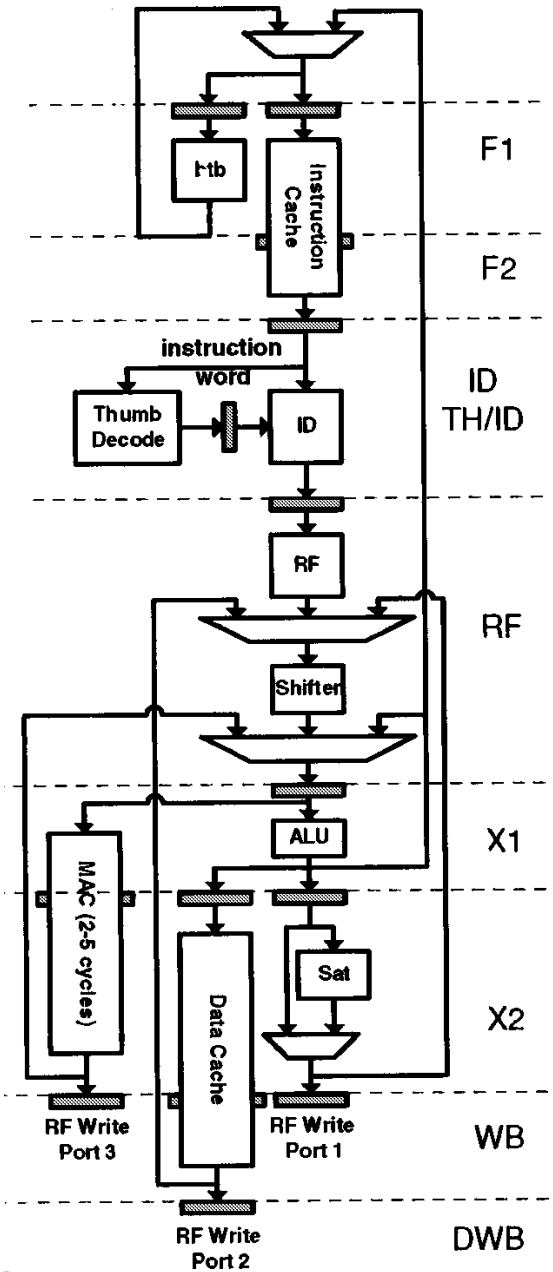
Last Time: Single-Cycle Processors



Today: Testing Processors + Teamwork

- Making a processor test plan
- Unit testing techniques
- State machine testing
- Teamwork: Lessons learned from previous CS 152 classes.

Lecture Focus: Functional Design Test



testing goal

Not manufacturing tests ...

The processor design correctly executes programs written in the supported subset of the MIPS 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 (Lab 1)

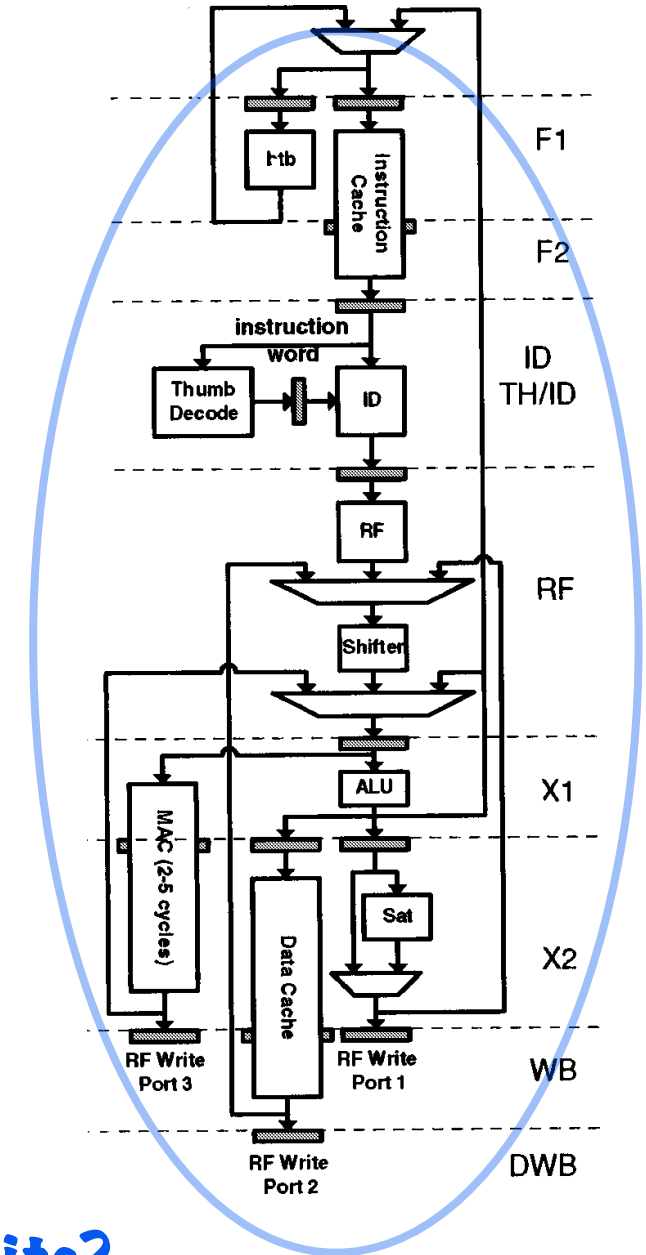
Bottom-up testing

how it works

Assemble the complete processor.

Execute test program suite on the processor.

Check results.



What makes a good test program suite?

Methodical Approach: Unit Testing

Top-down testing

complete processor testing

- unit testing

Bottom-up testing

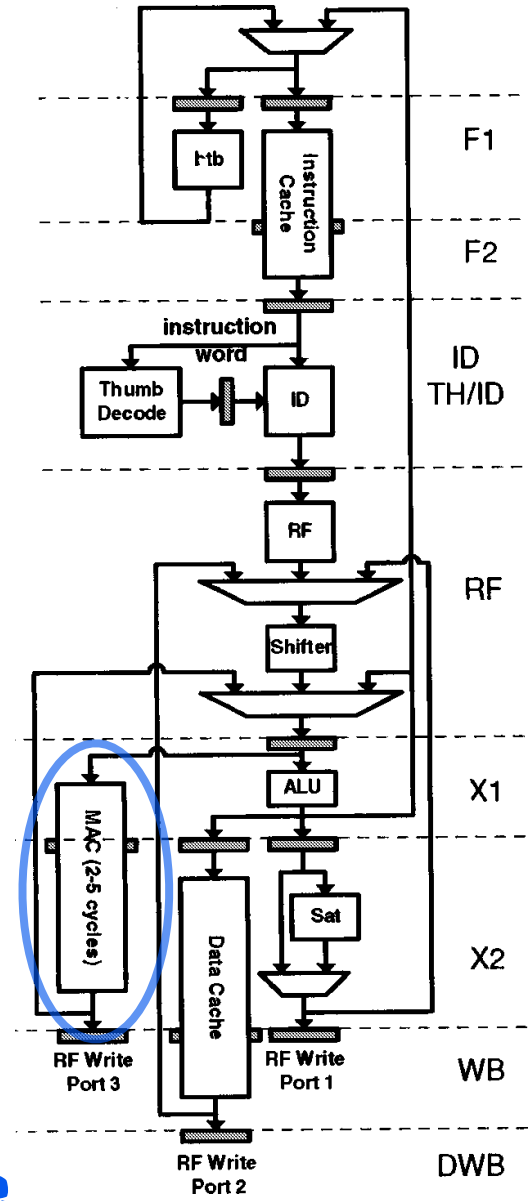
how it works

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

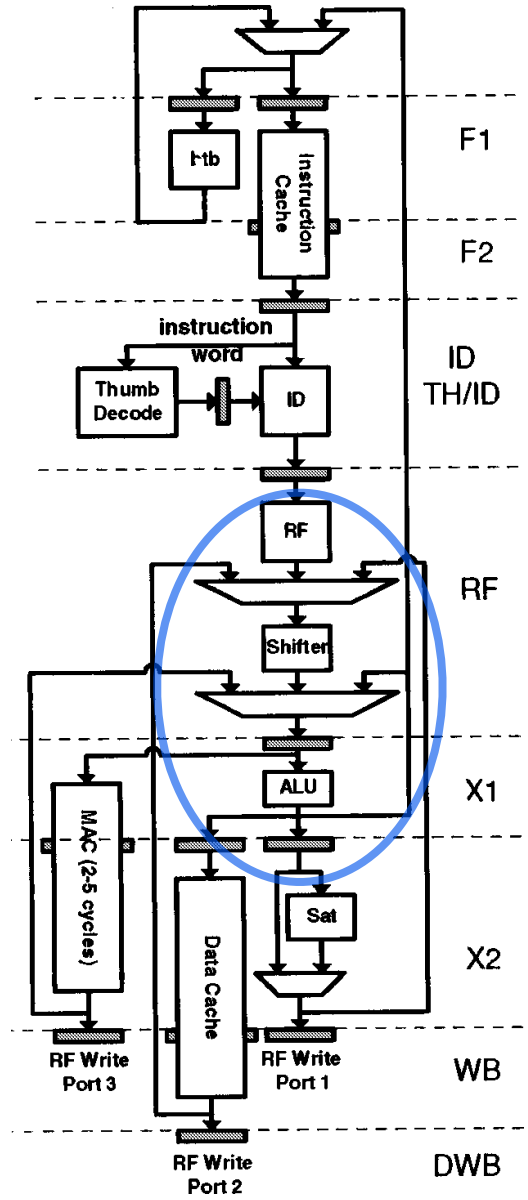
how it works

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

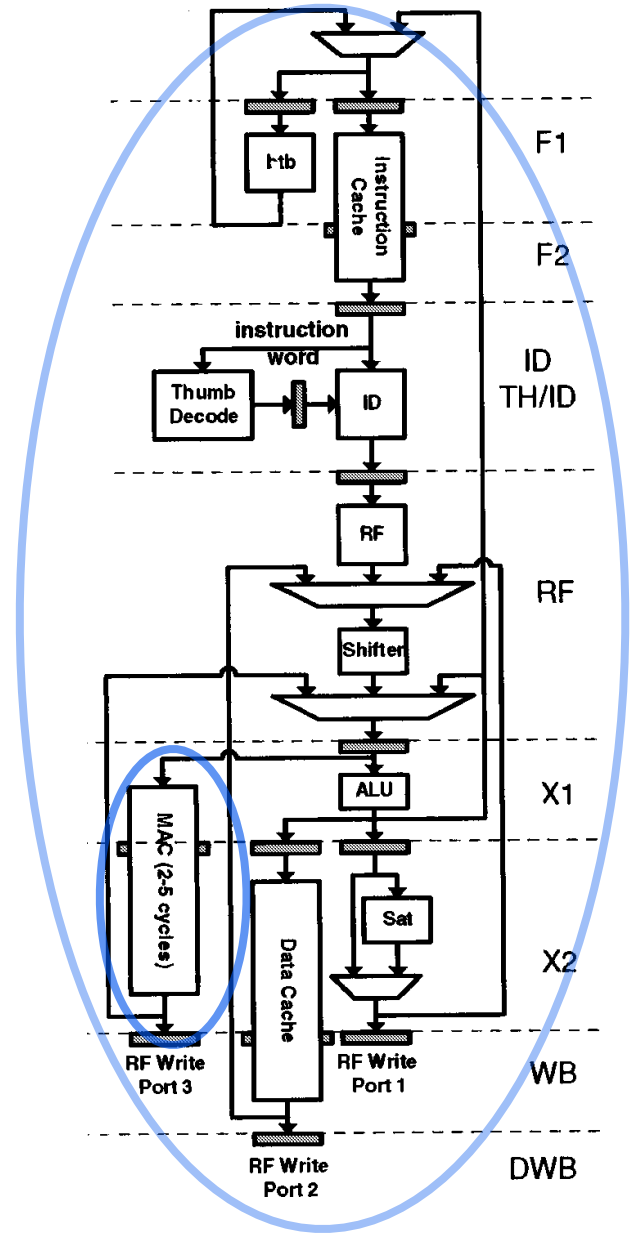
unit testing

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?

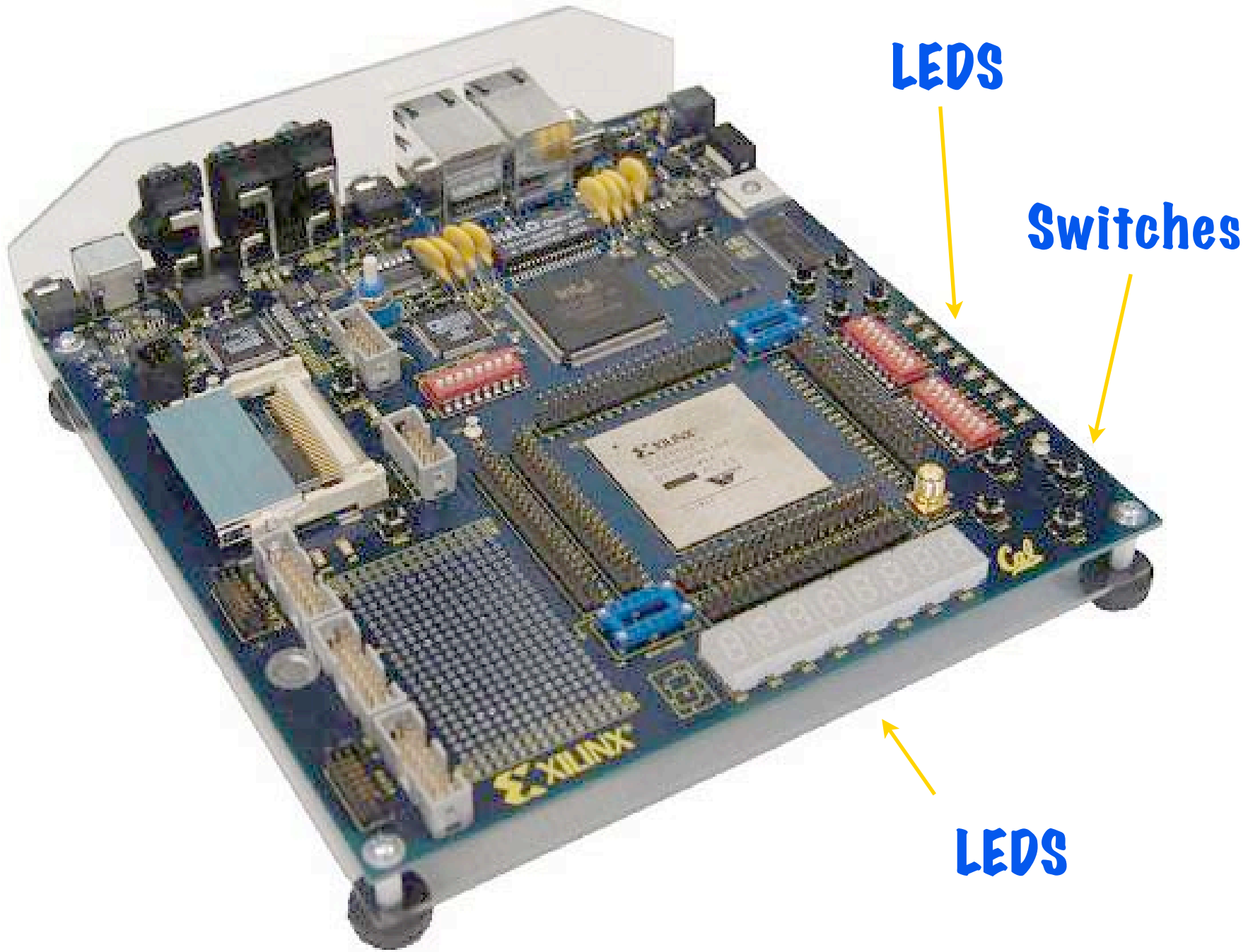
- **Controllability:**

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

For early labs, use ChipScope for observability.

For later labs ...

Use switches and LEDs on the board ...



Writing a Test Plan

The testing timeline ...

Top-down testing

complete processor testing

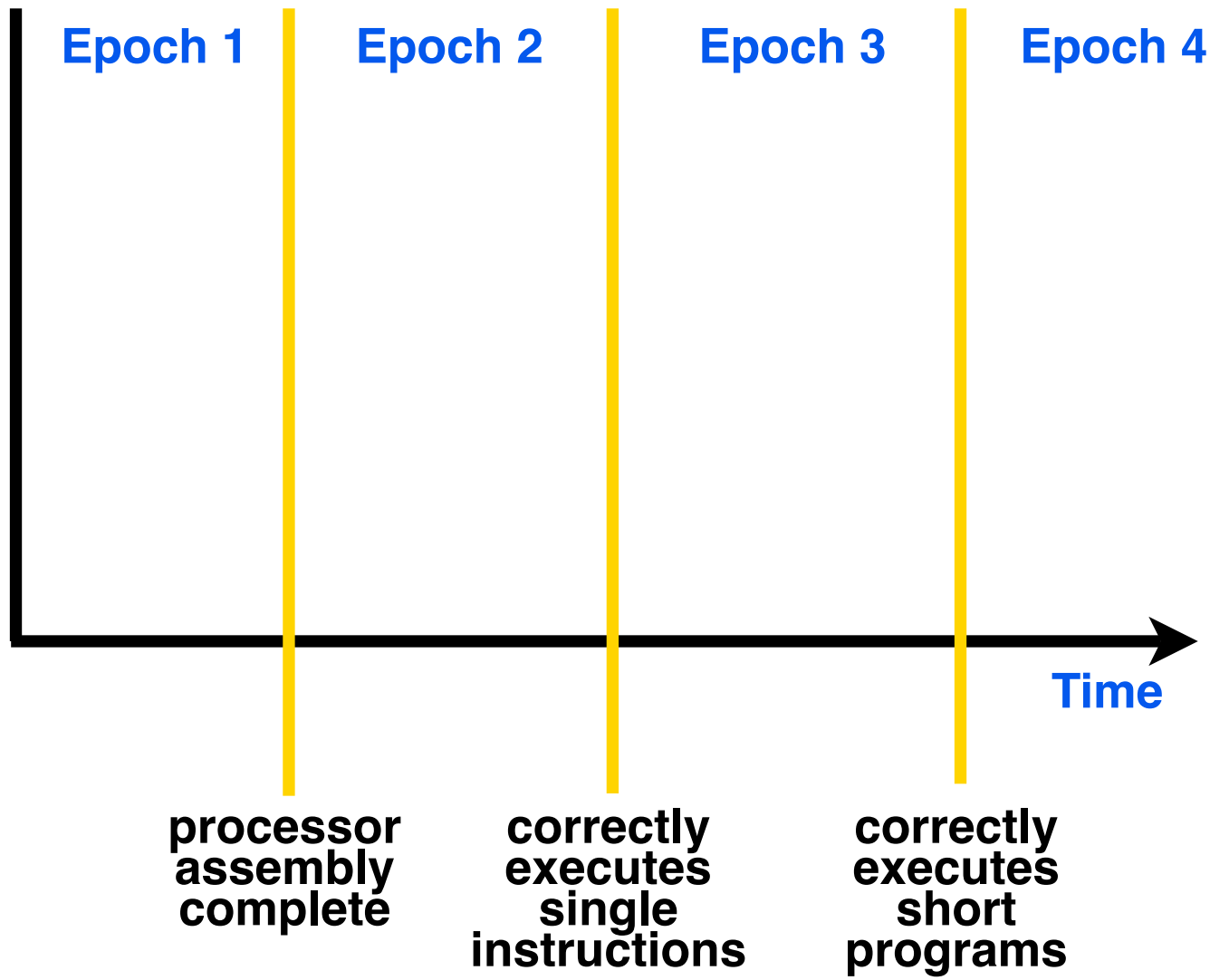
processor testing with self-checks

multi-unit testing

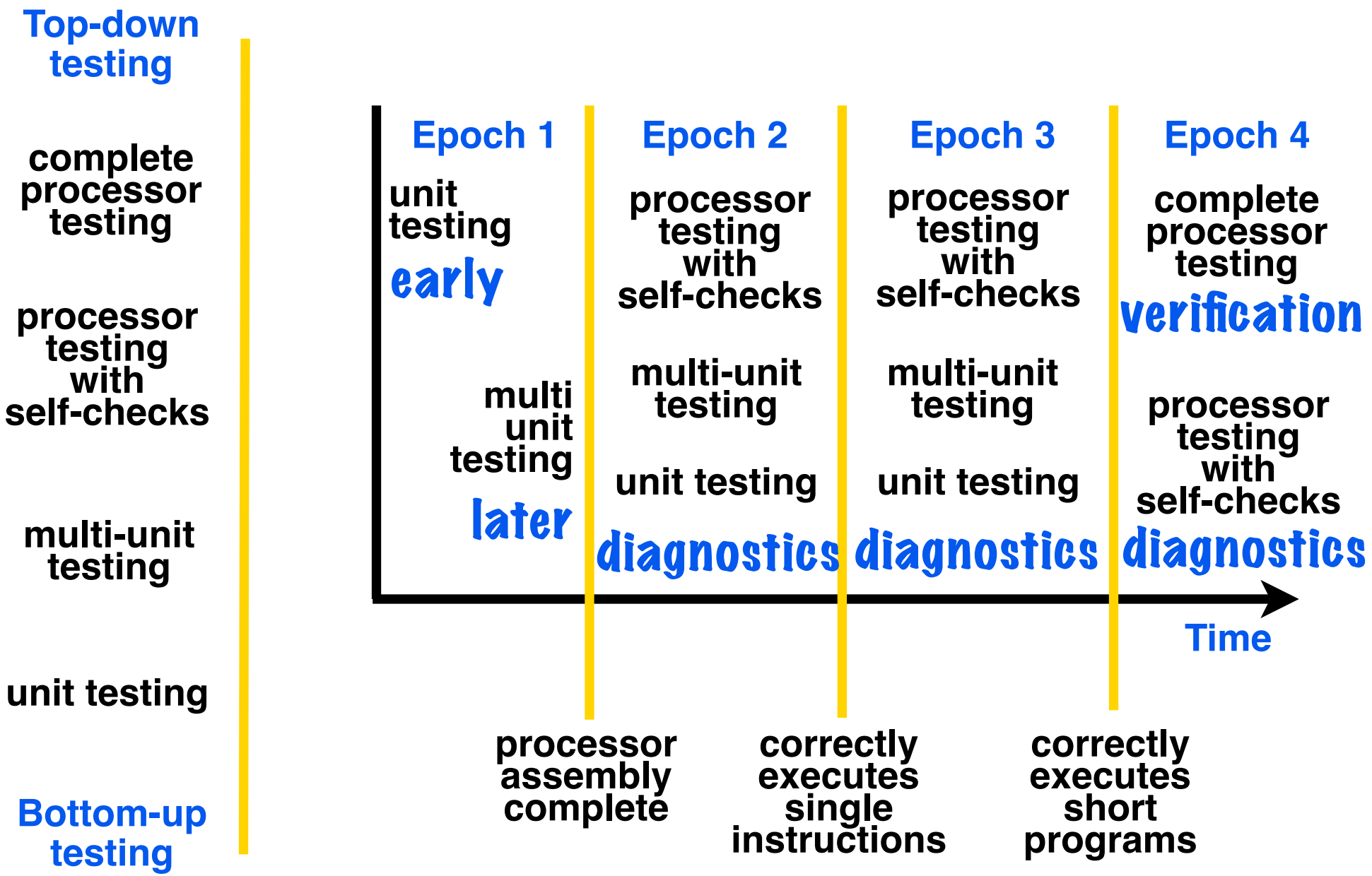
unit testing

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

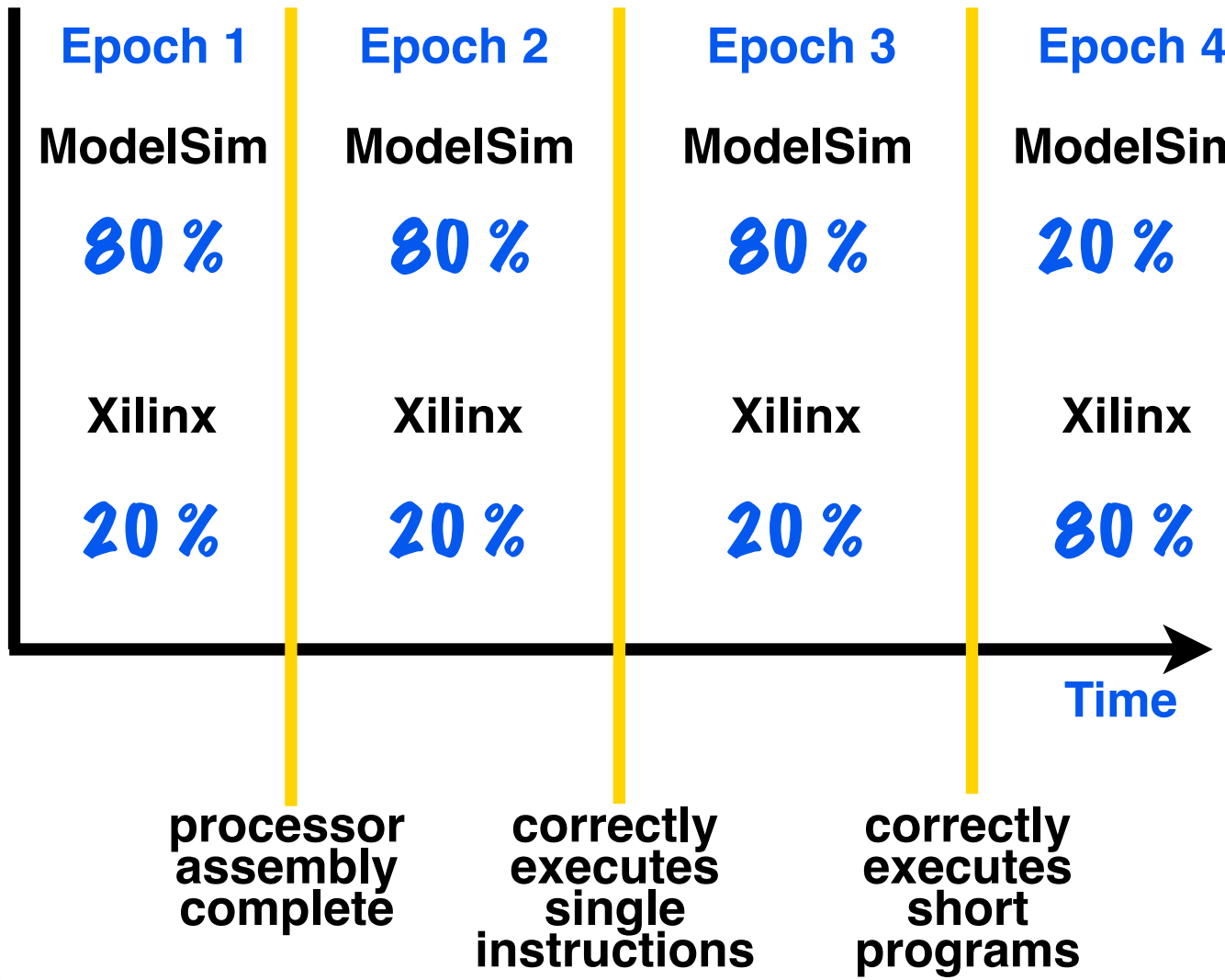
processor testing with self-checks

multi-unit testing

unit testing

Bottom-up testing

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

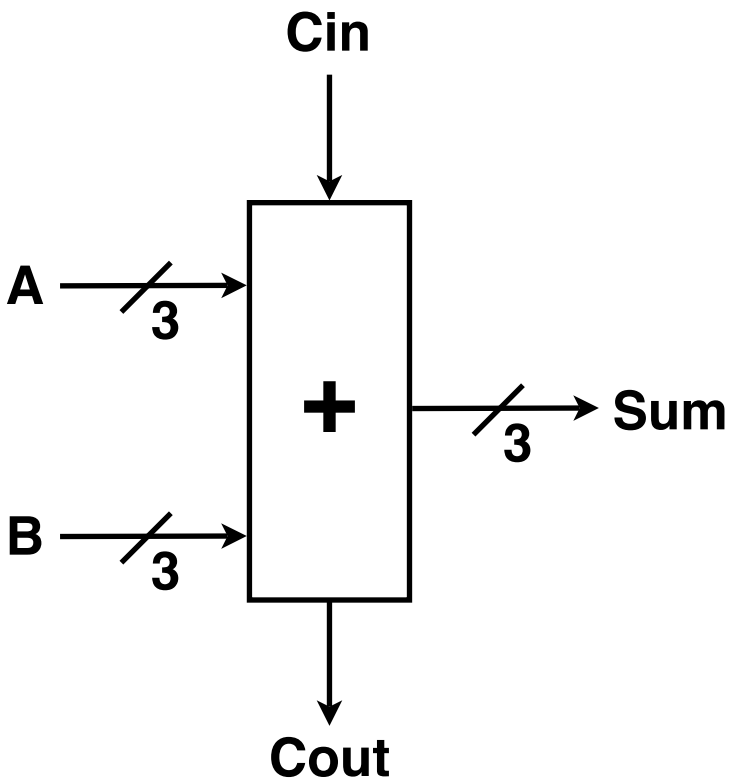


Catch "warnings and errors", signal name misspellings.

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?

$2^7 = 128$

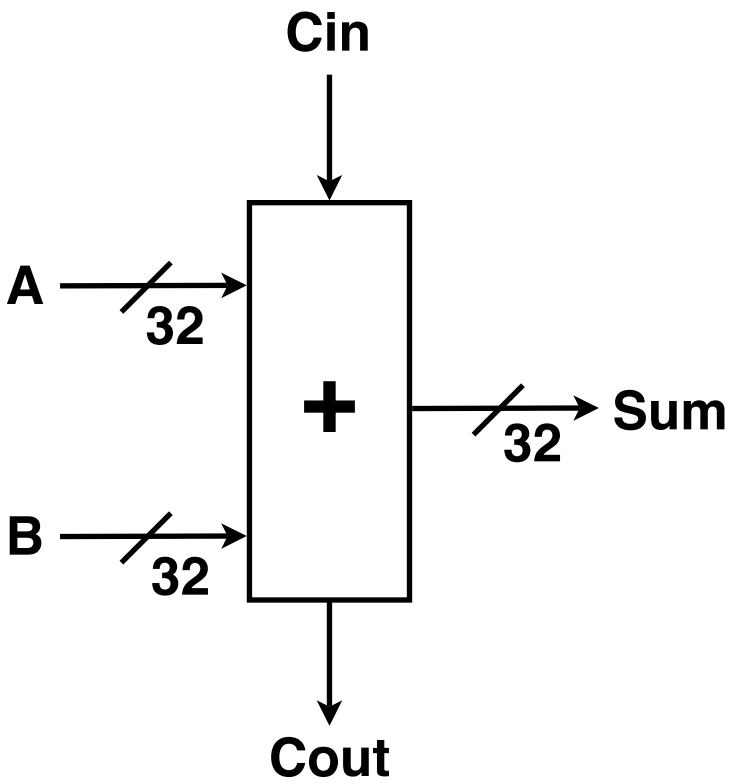
Just test them all ...

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?

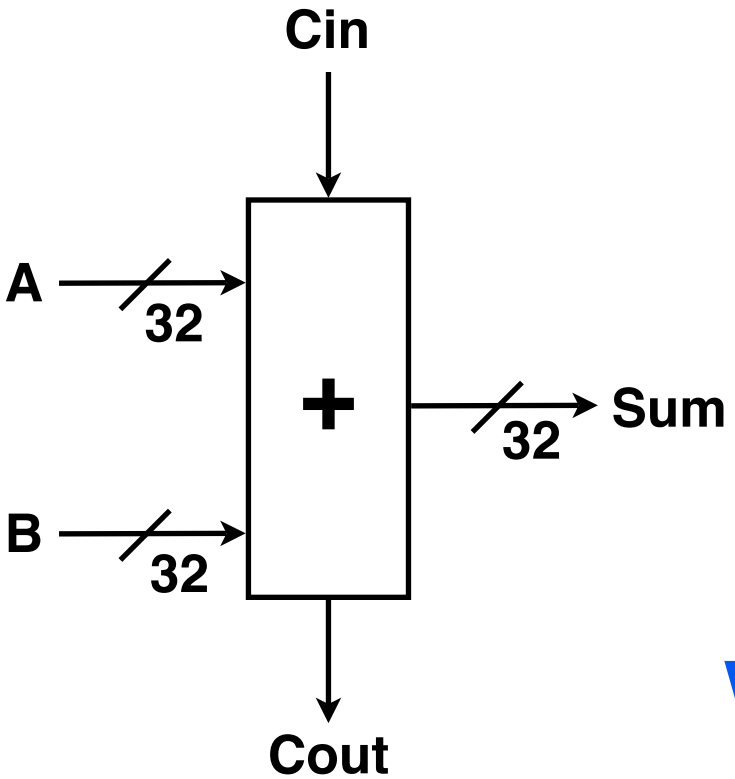
$2^{65} = 3.689e+19$

Just test them all?

Exhaustive testing does not "scale".

"Combinatorial explosion!"

Test Approach 1: Random Vectors

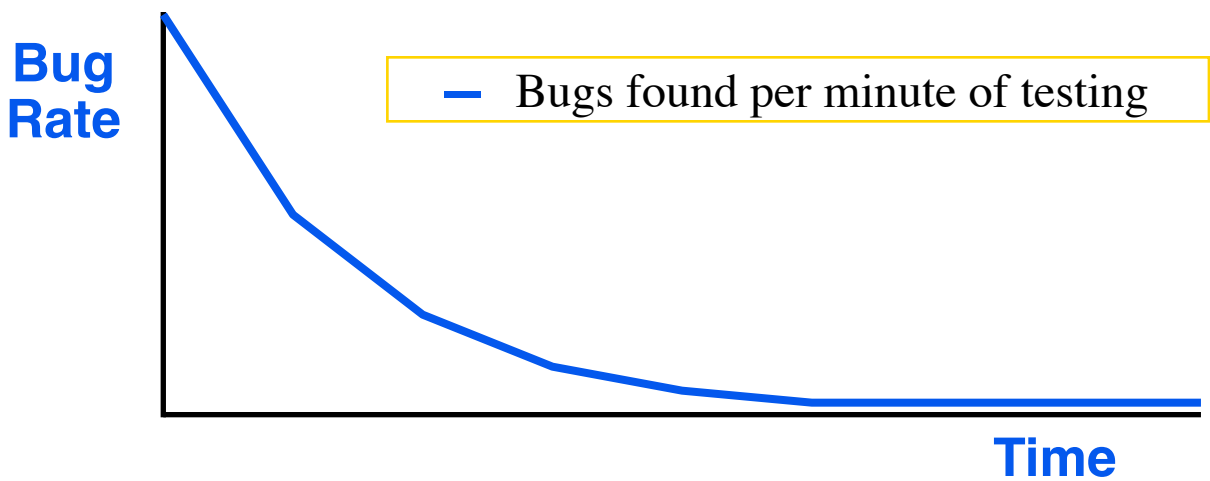


how it works

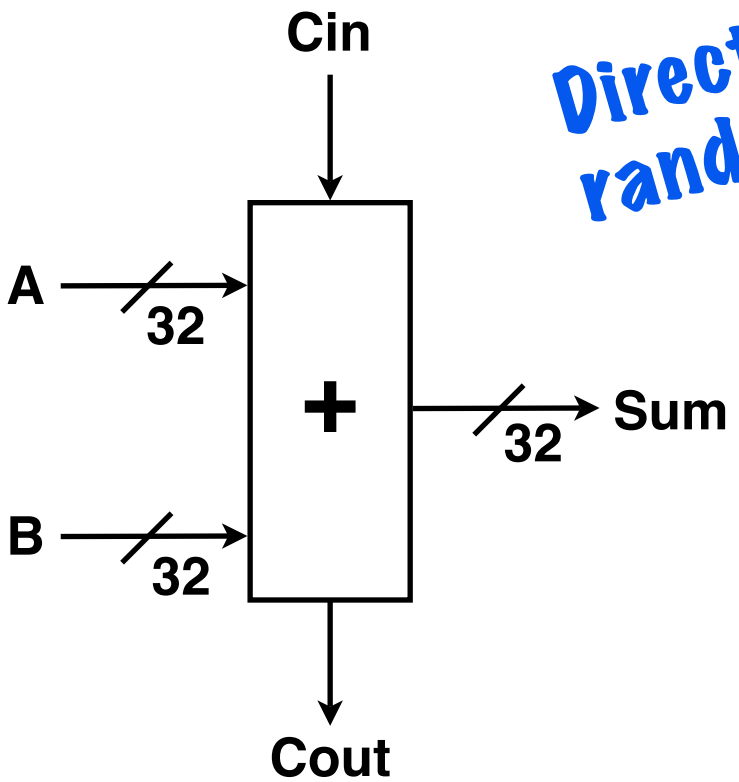
Apply random A, B, Cin to adder.

Check Sum, Cout.

When to stop testing? Bug curve.



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

152 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

CLK Change Rst

R (red) →

Y (yellow) →

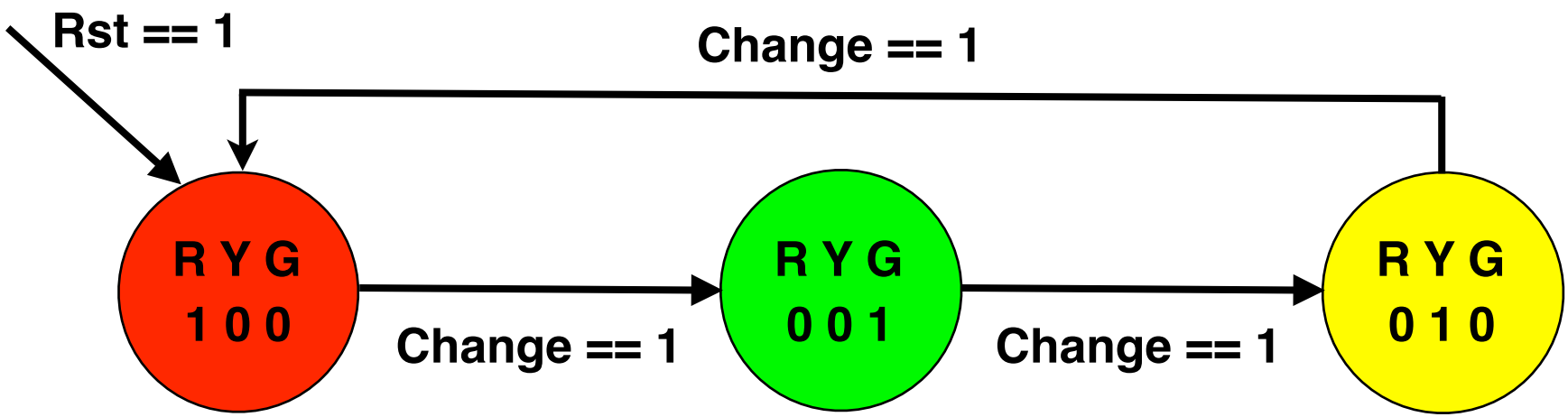
G (green) →

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

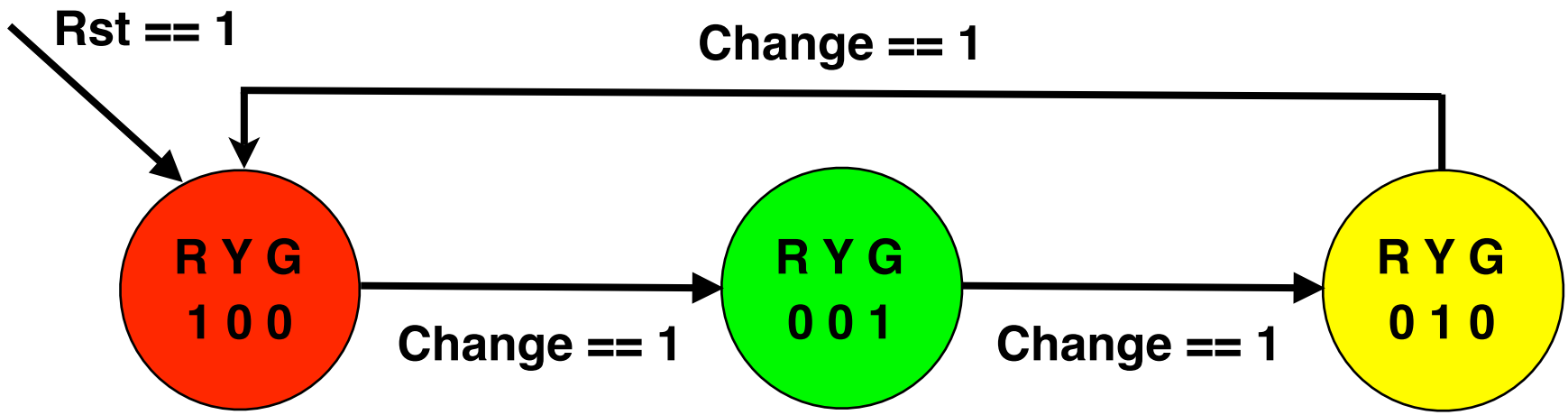
If Rst == 1 on positive CLK edge R Y G = 1 0 0

R	Y	G
1	0	0

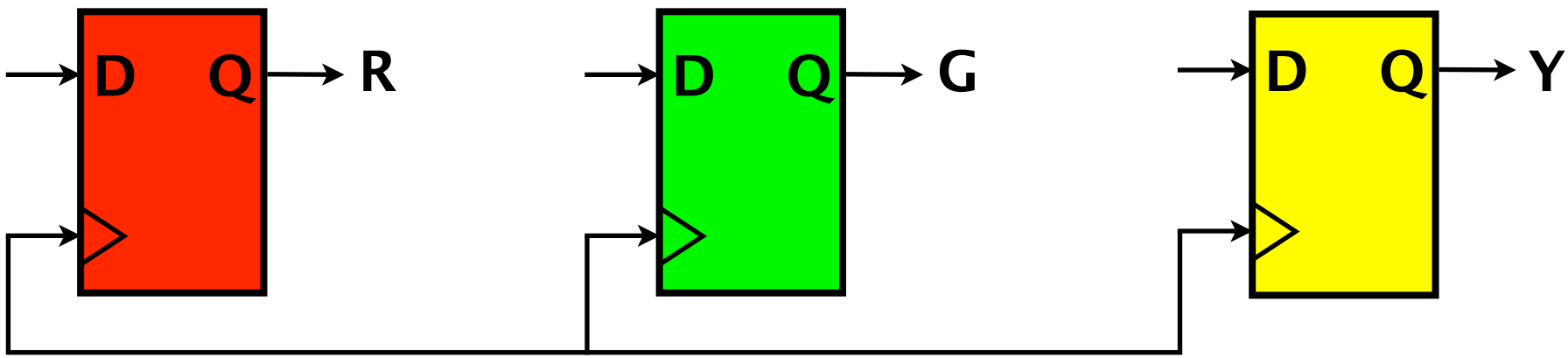
State Machine: Traffic Light Controller



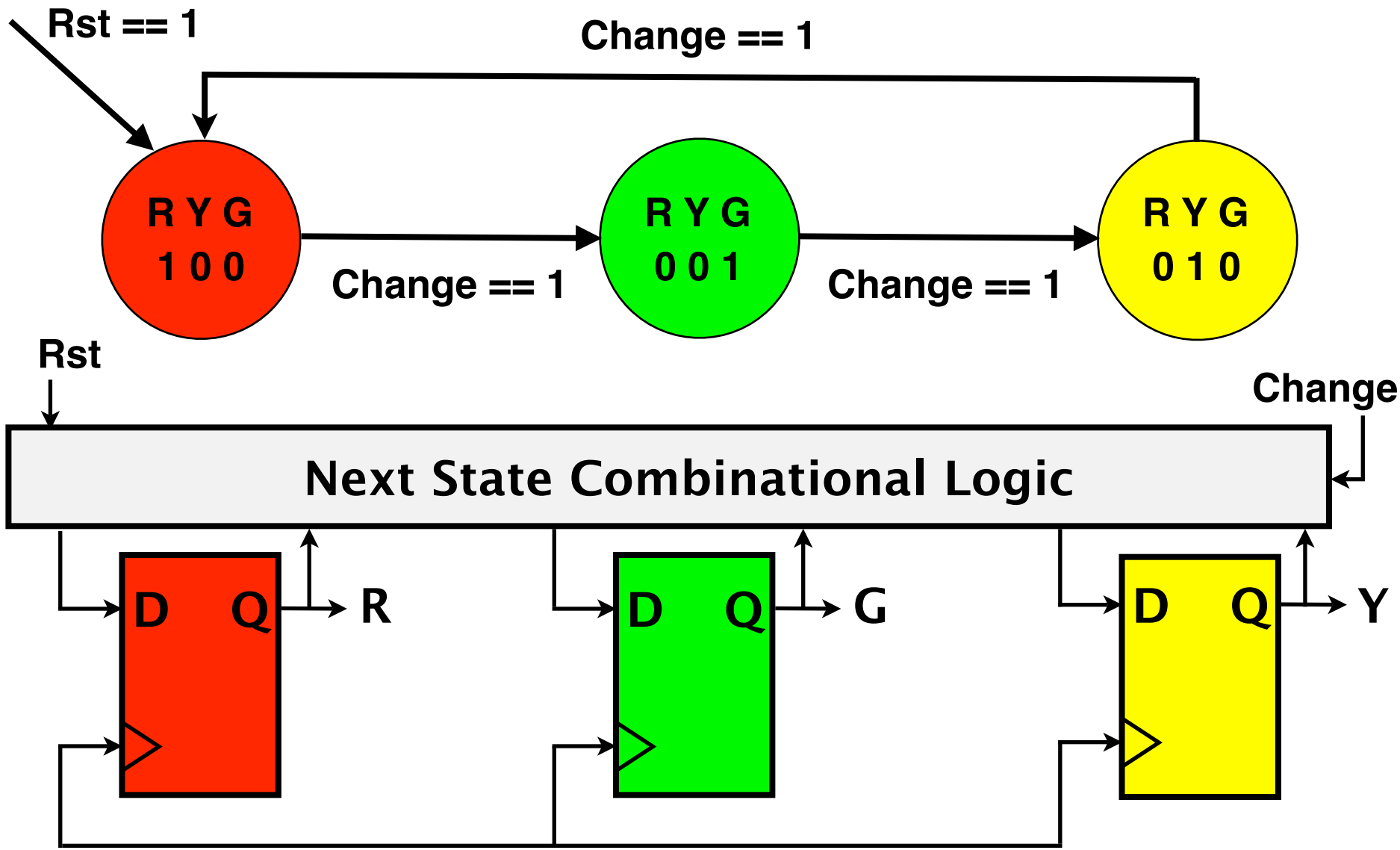
State Assignment: Traffic Light Controller



"One-Hot Encoding"

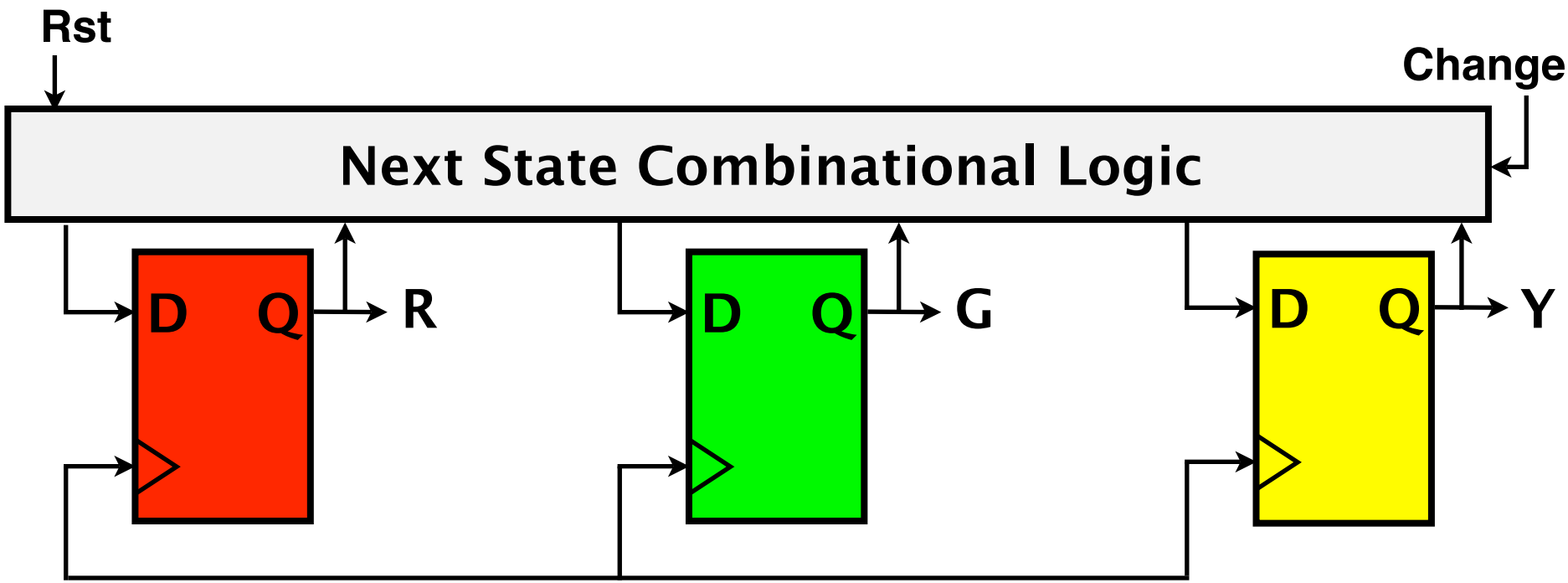


Next State Logic: Traffic Light Controller



State Machine Testing

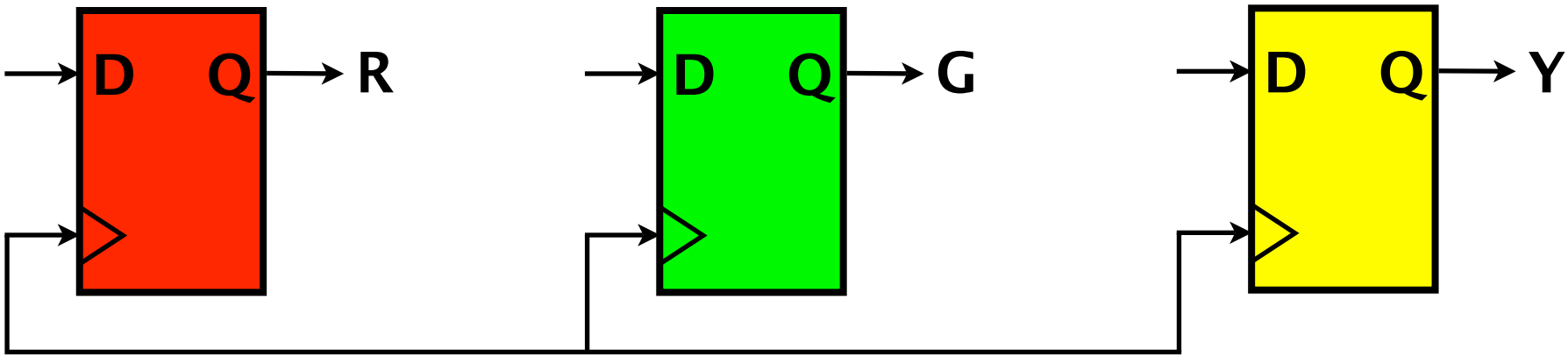
Testing State Machines: Break Feedback



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

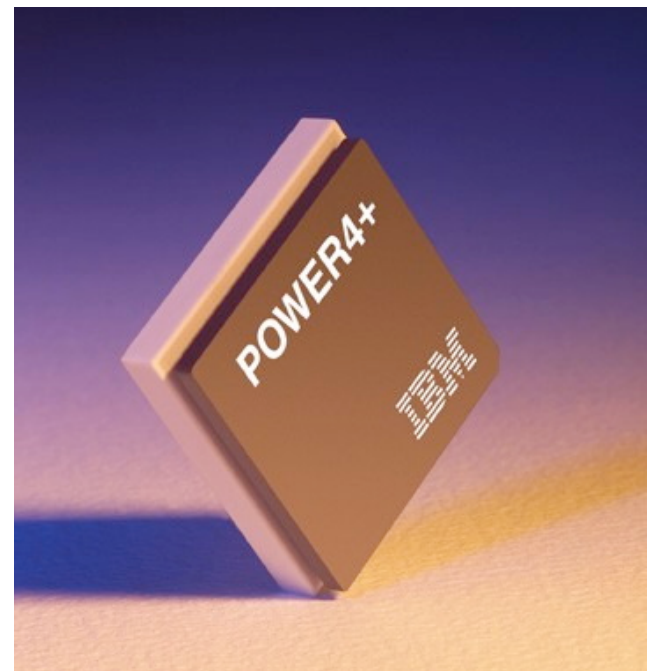
Easier with certain Verilog coding styles?

State Verilog: Traffic Light Controller

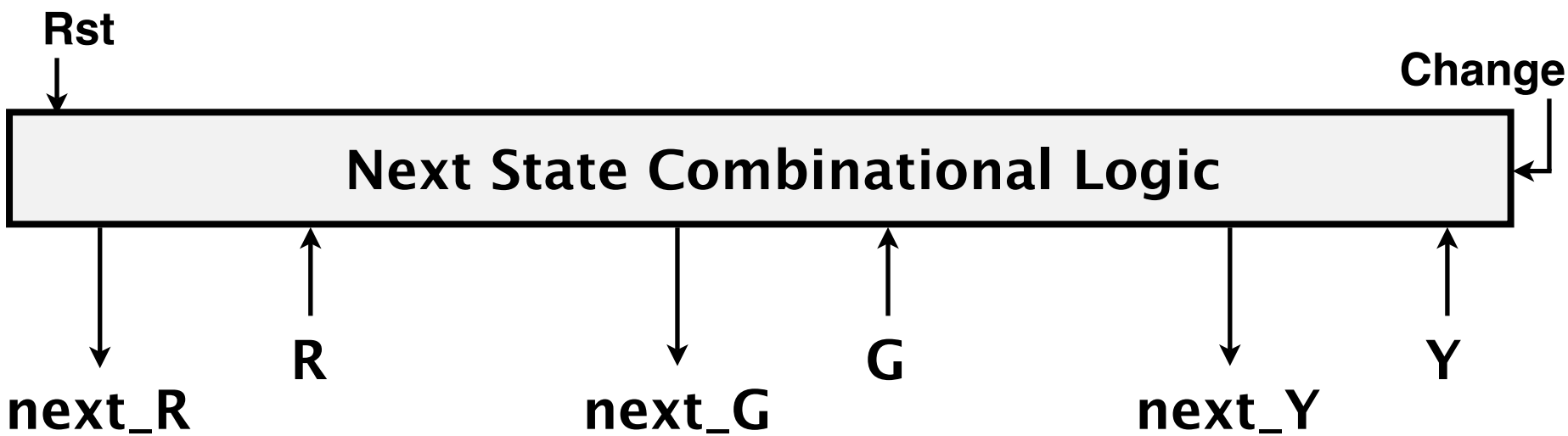



```
wire    next_R, next_Y, next_G;
output  R, Y, G;

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

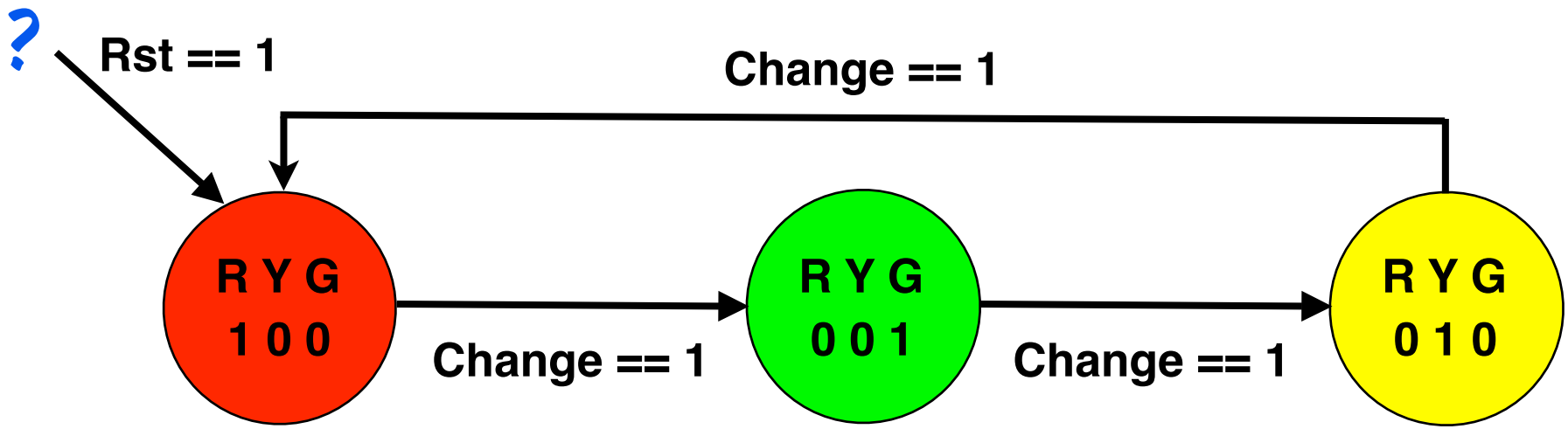
```
wire    next_R, next_Y, next_G;
output  R, Y, G;
```

```
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?

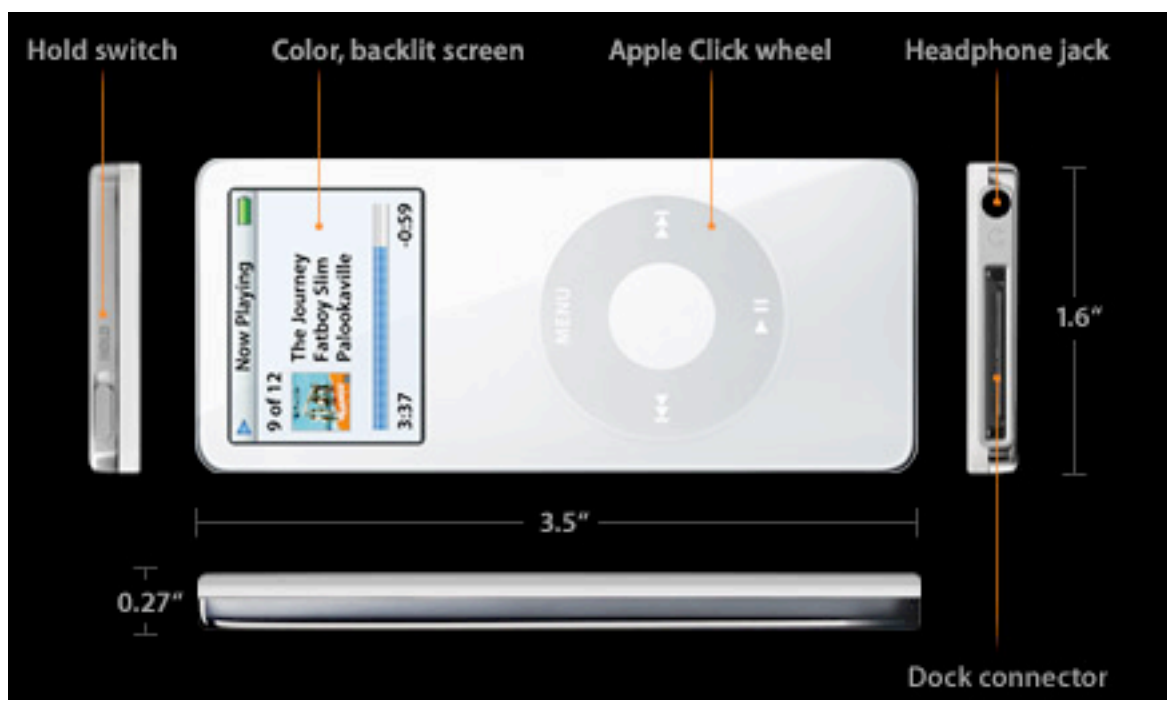
Conclusion -- Testing Processors

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

Administrivia: Upcoming deadlines ...

- **Thursday:** Lab 2 preliminary design document due to TAs via email, 11:59 PM.
- **Friday:** "Design Document Review", in session 125 Cory. For non-150s, 150 Lab Lecture 2, 2-3 PM, 125 Cory.
- **Monday:** Lab 2 final design document due to TAs via email, 11:59 PM.

In the news: iPod nano ...



Teamwork

CS 152
Computer Architecture and Engineering

What Went Right, What Went Wrong

2004-12-13

Dave Patterson, John Lazzaro
Doug Densmore, Ted Hong, Brandon Ooi

End-of-term presentation to CS hardware faculty ...

www-inst.eecs.berkeley.edu/~cs152/

152 F04: Executive Summary

- **Successful Start:** Lab 2 (Single Cycle Processor) and Lab 3 (Pipelines) went well. Most groups finished on time.

- **Stressful End:** Lab 4 (Caches): 1 group on time, 3 (?) were late, 1 never worked. Lab 5 (Final Project): 1 perfect project, 1 near miss, 2 worked in simulation.

What did we do after Lab 4?
We held a "town meeting" in class ...

Lab 4 “Town Meeting”

Held during one of the last Fall 04 classes …

Lab 4: Reflections from the TAs

- Everyone worked hard. Only in retrospect did most students realize they also had to work smart.
- Example: Only one group member knows how to download to board. Once this member falls asleep, the group can't go on working ...
- Solution: Actually use the Lab Notebook to document processes. An example of working smart.

Lab 4: Reflections from the TAs

- **Example**: Group has a long design meeting at start of project. Little is documented about signal names, state machine semantics. Members design incompatible modules, **suffer**.

- **A Better Way:** Carry notebooks (silicon or paper) to meetings, and force documentation of the decisions on details.

Lab 4: Reflections from the TAs

- **Example**: Comprehensive test rigs seen as a "checkoff item" for Lab report, done last. Actual debugging proceeds in haphazard, painful way.

- **A Better Way:** One group spent 10 hours up front writing a cache test module. Brandon "The best cache testing I've ever seen". They finished on time. An example of working smart.

Other Teamwork Topics

CAD: Never blindly trust a CAD tool

- **Example**: We recommended using the CoreGen multiplier generators to Fall 04. The tool was **buggy** (my bad). The most successful groups realized this **early** and switched methods.

- **Lesson:** Most CAD has bugs, and we can't know them all. Be paranoid -- never blindly trust any CAD tool !

Always check CAD tool "warnings and errors"! Synplicity examples:

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

CAD: Technical issues ...

- **Verilog: Carefully written Verilog will yield identical semantics in ModelSim and Synplicity. If you write your code in this way, many "works in Modelsim but not on Xilinx" issues disappear.**

Always check log files, and inspect output tools produce!

- **Backups: Use CVS, but also make safety copies off-site regularly (gmail). New CVS users often lose work as they are learning how to use CVS. Beware of CVS NT permissions issues.**

CAD and Testing: Asset Management

- Agree on where Verilog files will reside in the file directory structure.
- Agree on placement of test bench Verilog and hardware Verilog files.
- Agree on standard way to name files, and standard way to name Verilog modules, variables, parameters, ...
- Don't copy files -- include them. Each file should exist once in file tree.

The Hardest Part: Finding Things

Every semester a lot of work and precious time is lost by not being able to find the most current files or deleting good work on accident.

I think we should also warn them to only save Verilog, Chipscope, and bit files to their drives as groups invariably run out of space on their drives saving a bunch of Xilinx projects. It takes 10 seconds to make a new Xilinx project once you know how.

Dave Marquardt, TA Spring 05.

Remember: CPUs must meet ISA spec

A lot of points were needlessly lost last semester on working processors that didn't follow spec.

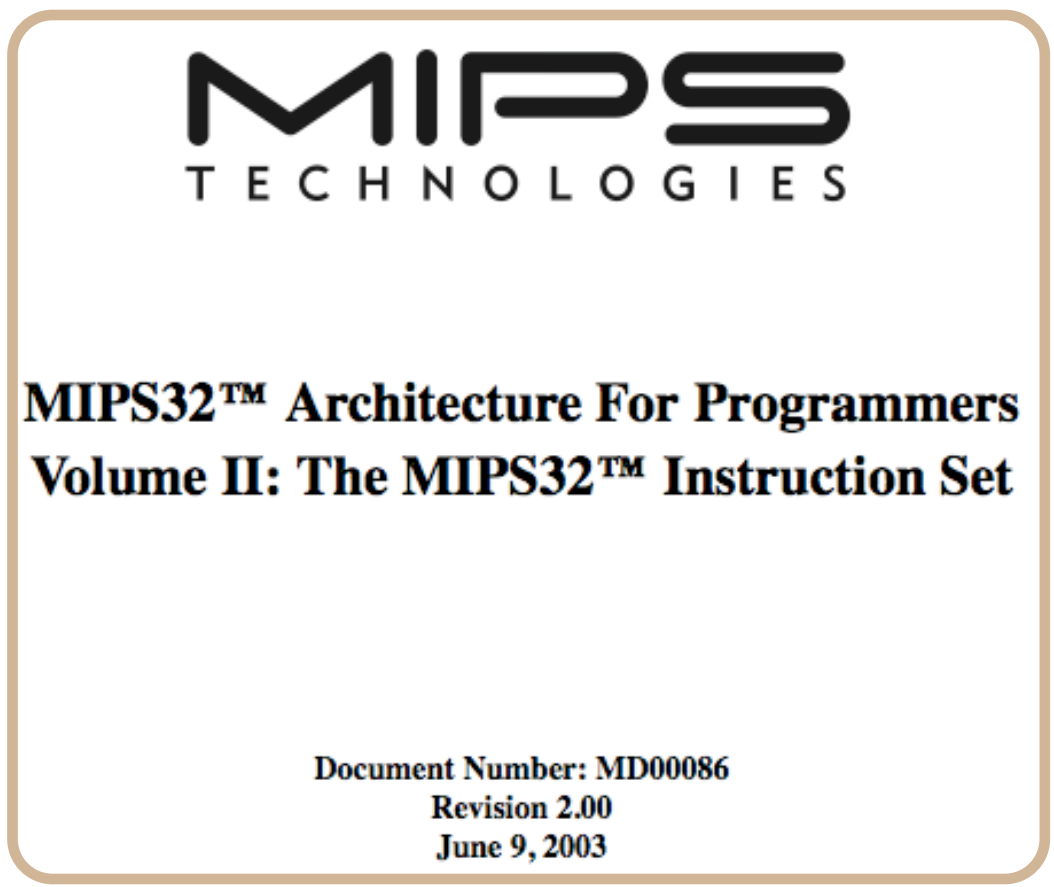
I think this means students should be reading over the lab spec more carefully (perhaps twice before starting and referring back to it often during development).

Dave Marquardt, TA Spring 05.

MIPS Instruction Set

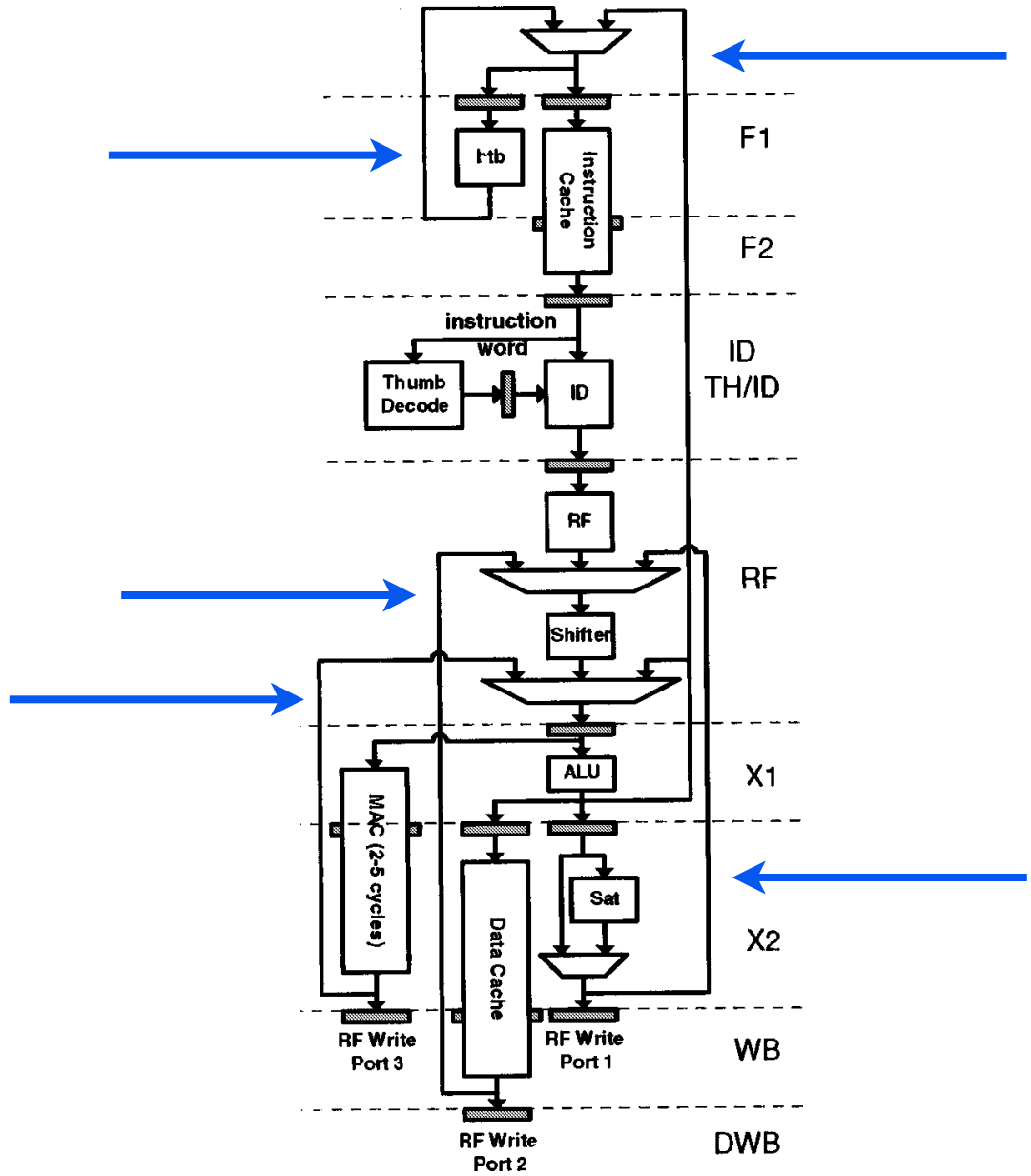
Use the MIPS ISA document as the final word on the ISA (+ labs).

Not P&H!



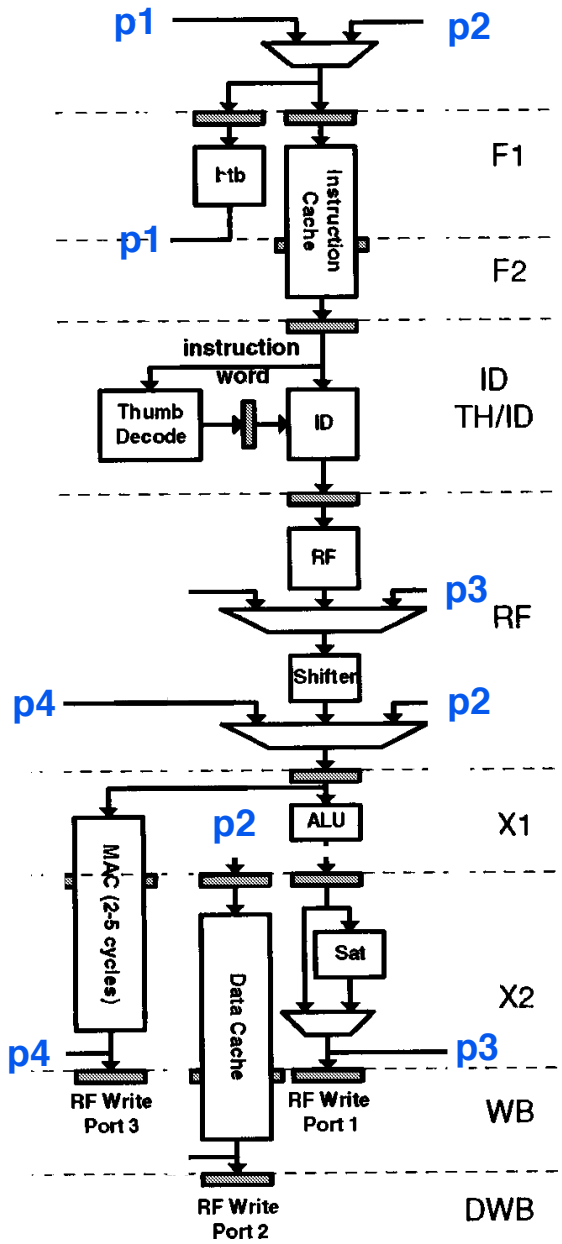
MIPS ISA document available on Resourses page on class website.

Schematics: This schematic uses wires ...



This schematic uses labels ...

Which is easier to understand?



Group Dynamics: How to Disagree

- **Example**: 3 members want to do the design one way; member number 4 does not agree.
- **Solution #1:** Voting. "Fair". But, what if the "loser" was technically correct?
- **Solution #2:** Consensus. Keeping in mind the goal (correctly working CPU on the board on schedule), what option brings the group closer to the goal?

Never lose sight of the goal!

Team Final Presentations

Ergo 152
aka RAID 2

Group Philosophy

-We tried to work as a team for most of the project.

-Golden Rule: If it isn't broken, don't fix it.

Why?:

- We could all understand what was going on
- Team unity
- We tried to keep things simple

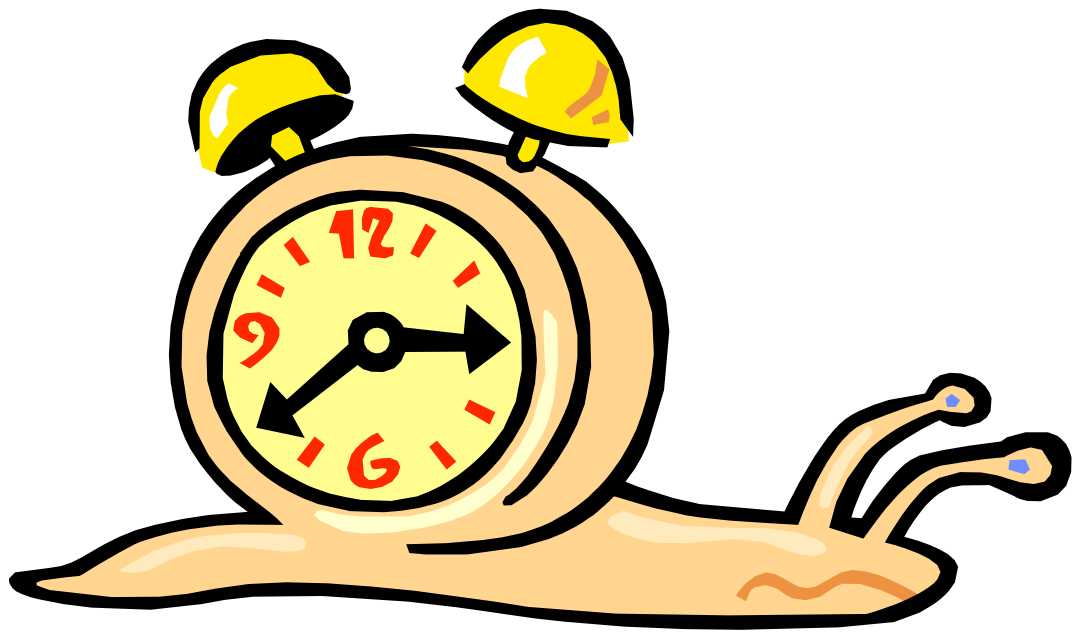
Analysis of philosophy

Pros:

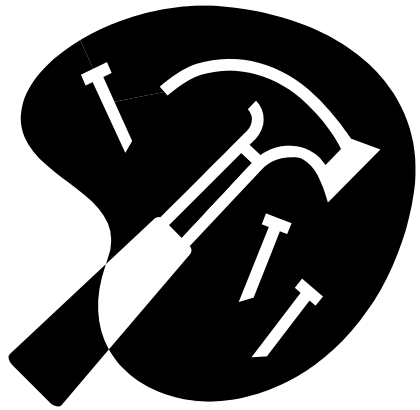
+ Knew what was going on
+ One set of code (Coherency)
+ Four heads are better than one
+ FUN!

Cons:

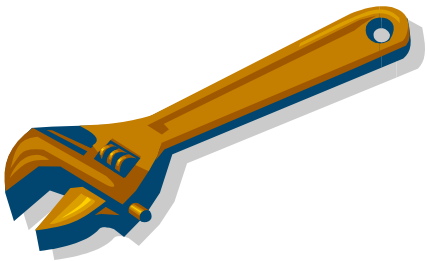
- Slow



How we used the tools



- Design in simulation first
- Look at waveforms
- Useful output to Sim terminal
- Push to board
- Use LEDs efficiently
- Use Synplify Pro to find combinational loops

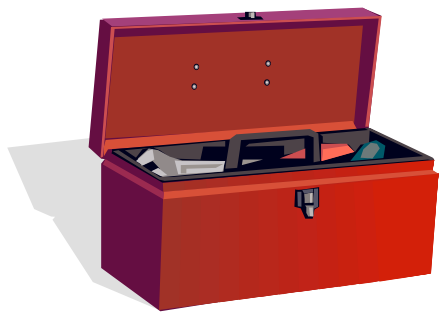


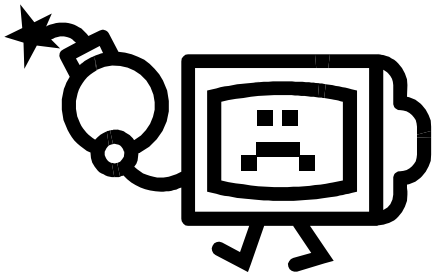
Advice for tools

- Get good at looking at waveforms
- Understand the waveforms
- Know how to locate bugs and trace backwards to find it.
- Look at signals and see what those signals depend upon
- Learn how to use DIP switches and LEDs to your advantage

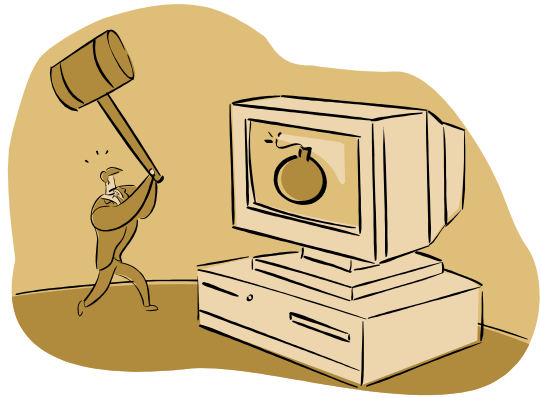
Advice for tools cont'd

- Attend CS150 and do its labs
- Learn how to do things such as critical path in CS150
- Go through CS150 manuals to learn to use Chipscope, TFTP, etc.
- Use Kramnik remote desktop to work from home

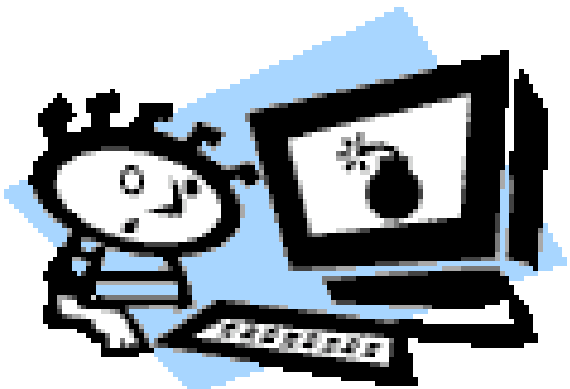




What went wrong



- Ran out of time
- Didn’t think of all the bugs in preliminary design
- Weren’t prepared for checkoffs
- Didn’t comment well enough
- Didn’t carefully consider timing issues



What went right

- Successfully finished most of the labs
- Kept board files and simulation files in separate folders
- No inner group conflict because we discussed and agreed upon solutions to most issues together

Advice for next semester

- Spend a lot of time working on design document
- Make sure all corner cases are well documented and tested
- Understand the provided modules (Simu and look at the waveform to understand)
- Work together as a team

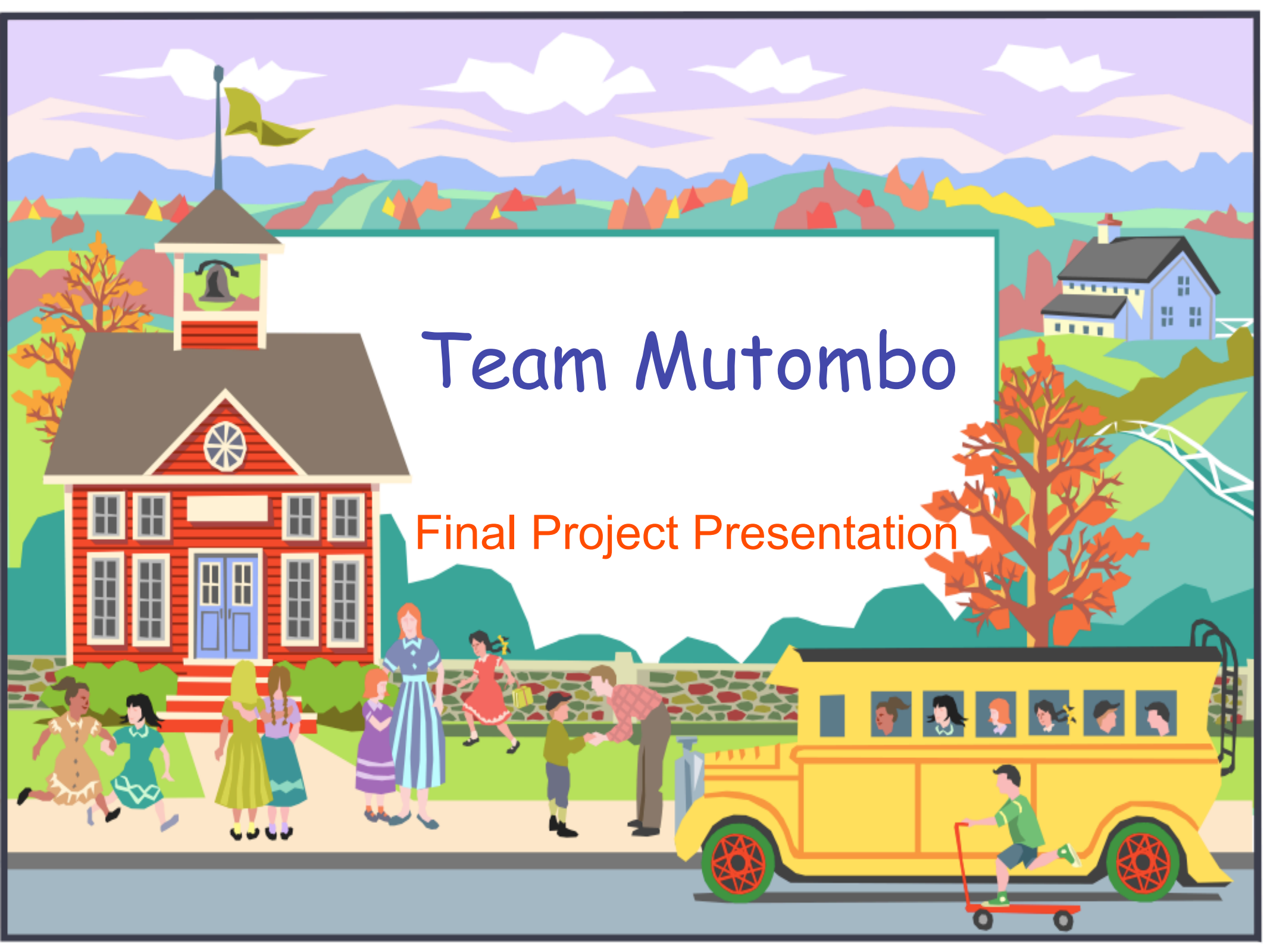
Advice for next semester cont'd

- Don't be afraid to ask for help
- Keep up with work in other classes
- Be prepared to spend at *least* 40hrs/week in the lab
- Don't take other project classes concurrently
- Backup the right copies of files!
- Fix all of your warnings in Synplify
- Hide your food

Suggestions for the course

- Testbench checkoff before writing code is difficult
- Make sure you are very clear with what is wanted in each checkoff (Confirm with TA if necessary)
- Provide better documentation for given modules

Team Final Presentations


Team Mutombo
Final Project Presentation

Testing Techniques

- Finish programming early so there's plenty of time for debugging
- Rerun all tests when any changes are made
- Don't modify tests so the CPU passes
- Don't write tests the night before the test vectors are due.

CAD

- Save wave forms between sessions
- Make a default project navigator .npl
- Save transcripts after running simulation tests
- Make a custom modelsim.ini file

Maintaining Good Group Dynamics

- Keep the atmosphere light
- When stuck, play games
- Play games during synthesizing
- Play games when not in lab
- Play games when members not present
- Play games

Design Methodology

- Break all modules into manageable pieces. (Example: *MegaModules)
- Agree on a naming scheme & stick to it
- Write names of all wires on block diagram
- Keep block diagram neat

*MegaModules is a trademark of TM

Problems

1) SDRAM Clock
2) Clock Skew (Buttons and CPU)
3) State Machine Synthesis Issues
4) Write Buffer Cases
5) Stalling Logic Bugs

Problem 1

- The SDRAM Clock on the FPGA Top was the divided clock.
- Solution: Don’t name the divided clock, memory clock. However, it was overlooked on our part.

Problem 2

- Pressing Reset and Step buttons caused odd errors on the board.
- Solution: Put Reset and Release on the processor clock. When in stepping mode, make Reset and Release the raw signal.

Problem 3

- Cache controller state machines cause many synthesis errors including: Combinational Loop and Latch Errors.
- Solution: Don’t listen to Daniel ?????
- Use Moore machines to prevent loops and make sure each wire is in every case to prevent latches.

Problem 4

- Write buffer contents updated while being sent to DRAM
- Solution: Don’t update the write buffer slot going to DRAM. Make sure you deal with 2 copies of what’s in write buffer correctly.

Problem 5

- Stalling Logic was incorrect
- Solution: Break up signal into sub-wires, to reduce confusion and facilitate debugging.

Suggestions

- Use cvs & descriptive comments when you checkin
- Use multiple computers to maximize efficiency.
- Use 125 instead of 119. (plasma screens for gaming)
- Take 150 first! (for state machines)

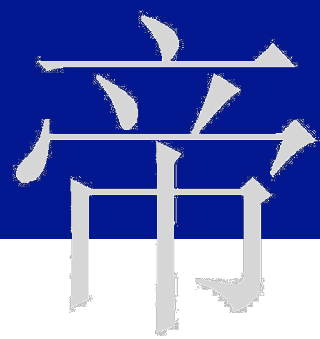
Suggestions Cont..

- Create shared email folder/acct for quick access to info
- Sleep at home, not in lab

Team Final Presentations

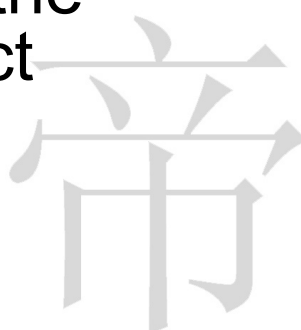
Rapid Multiply Deep-Pipelined Processor

ETERNAL EMPIRES



In Conclusion… (1/2)

- Design document stage is VERY important
  - Perhaps give more time – 2 days isn't enough
  - Less hacking, more planning – Wires all over the place, files all over the place
- Naming conventions are VERY important
  - XLXN_653 takes forever to decode…
  - Modelsim does not complain about missing wire declarations or upper/lower case discrepancies
  - Number starts as 0 or 1, i.e. mux inputs
- Need a better directory structure
  - _utb.v files shouldn't be in the same directory as the modules they're testing – takes forever to unselect them for synthesis.



In Conclusion… (2/2)

- Datapath drawings are VERY important
  - The schematic editor is not enough
- Xilinx Schematic Editor is the worst product ever shipped
- Simplify the design
  - Do as much of the logic and design outside of schematics
  - Similar functionality can be merged into one module

Coming up next week ...

T 9/13	Timing
W 9/14	
Th 9/15	Performance
F 9/16	
Sa 9/17	
Su 9/18	
M 9/19	
T 9/20	Pipelining I
W 9/21	
Th 9/22	Pipelining II

Top-down view of how signals move through your processor in time.

Why we pipeline ...

How to pipeline ...