CS 152
Computer Architecture and Engineering

Lecture 4 – Testing and Teamwork

Congrats on Lab 1 !

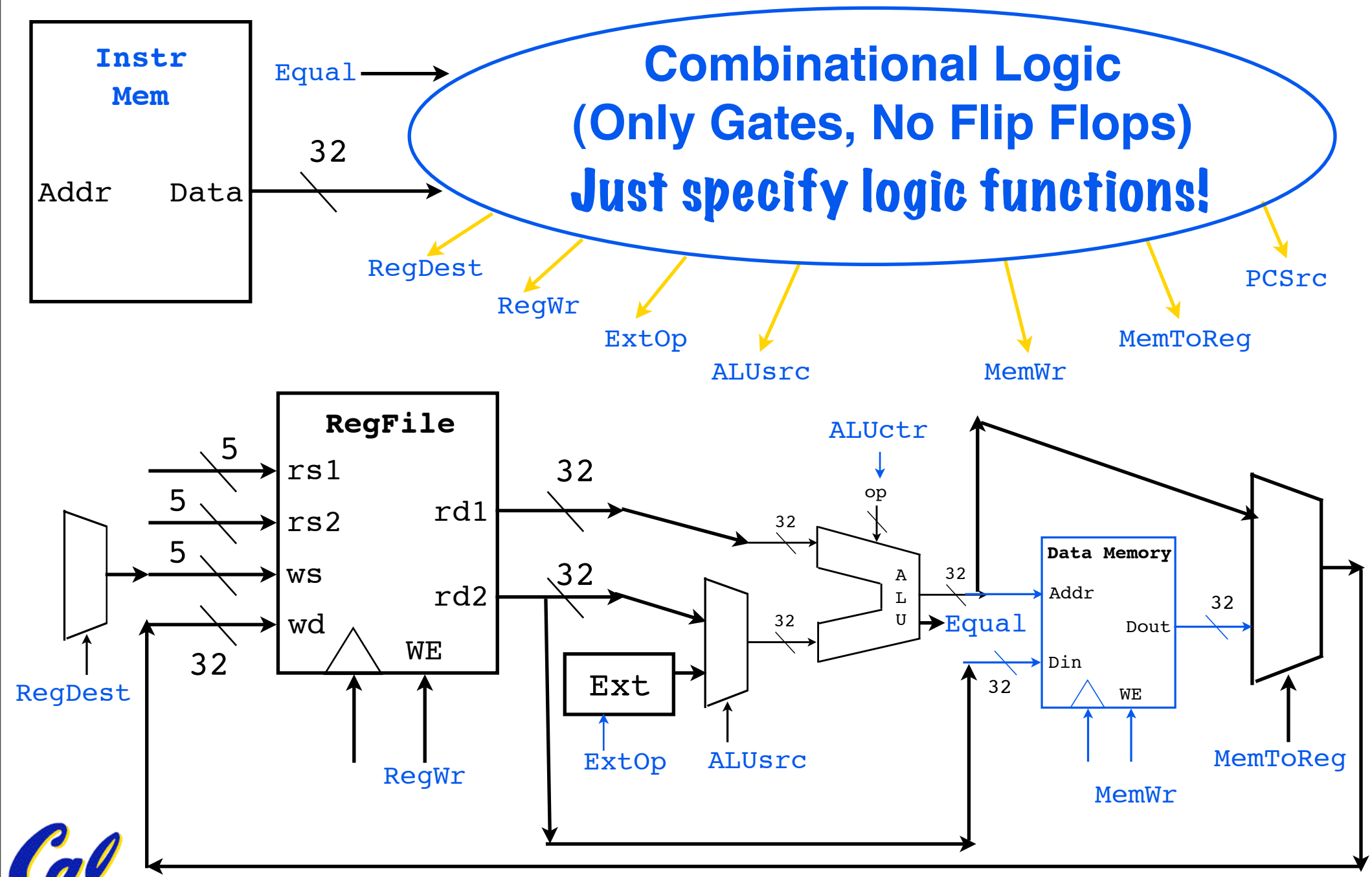
2006-9-7

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

TAs: Udam Saini and Jue Sun

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

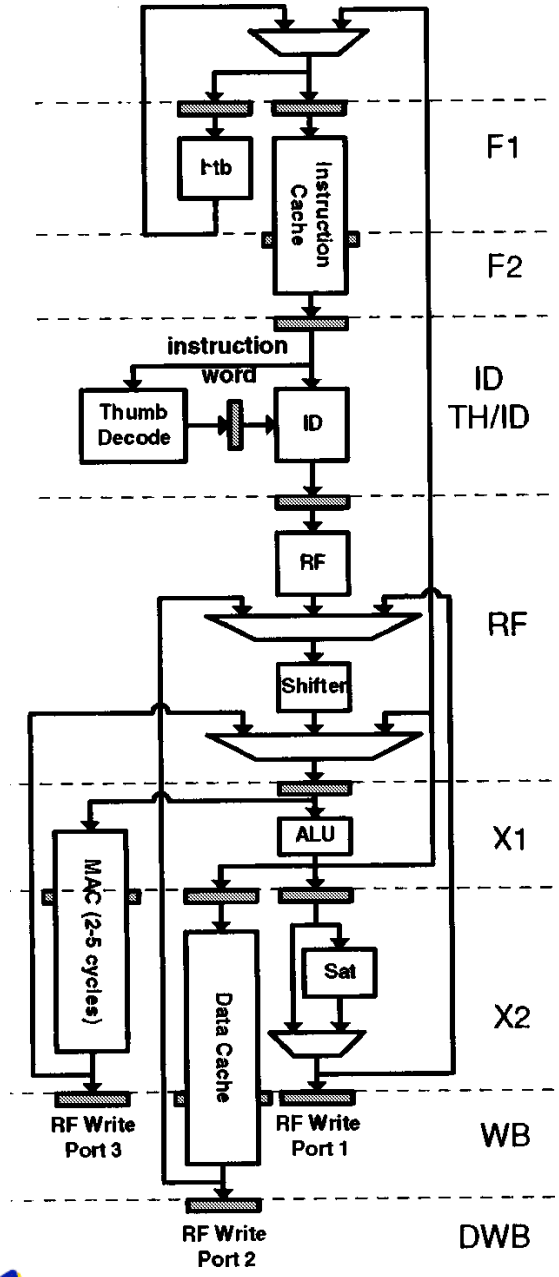
Last Time: Single-Cycle Processors



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)

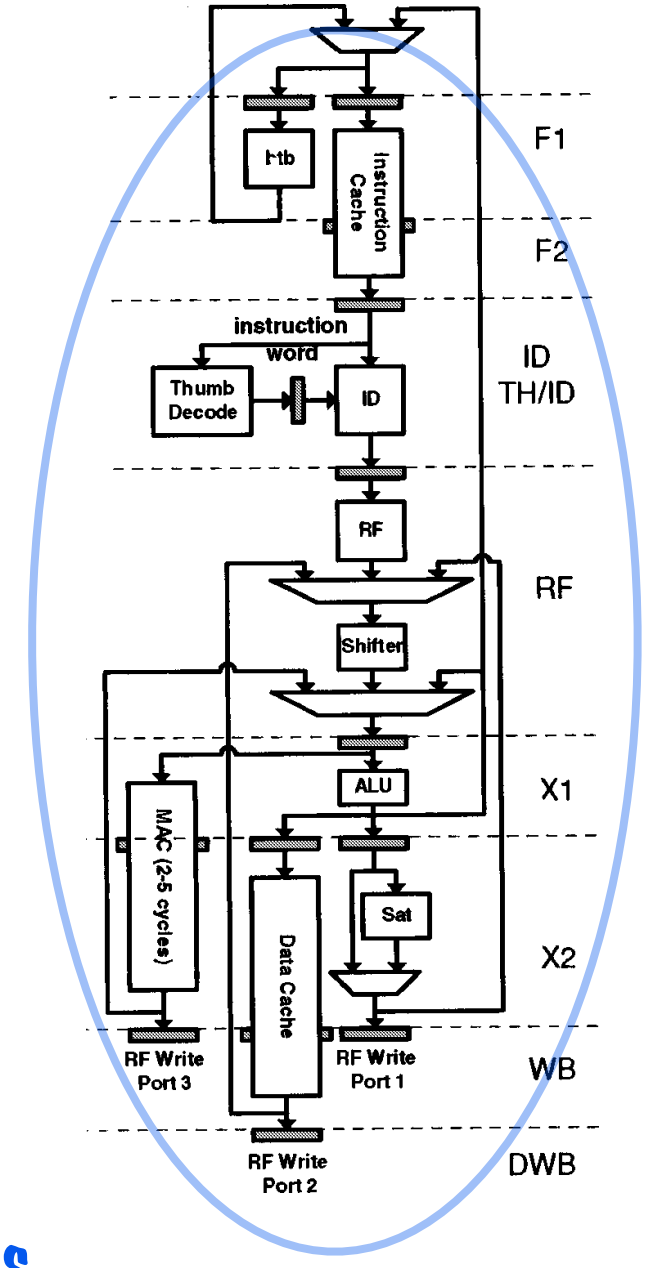
Bottom-up testing

how it works

Assemble the complete processor.

Execute test program suite on the processor.

Check results.



This is how TAs test on checkoff days ...

Methodical Approach: Unit Testing

Top-down testing

complete processor testing

- unit testing

Bottom-up testing

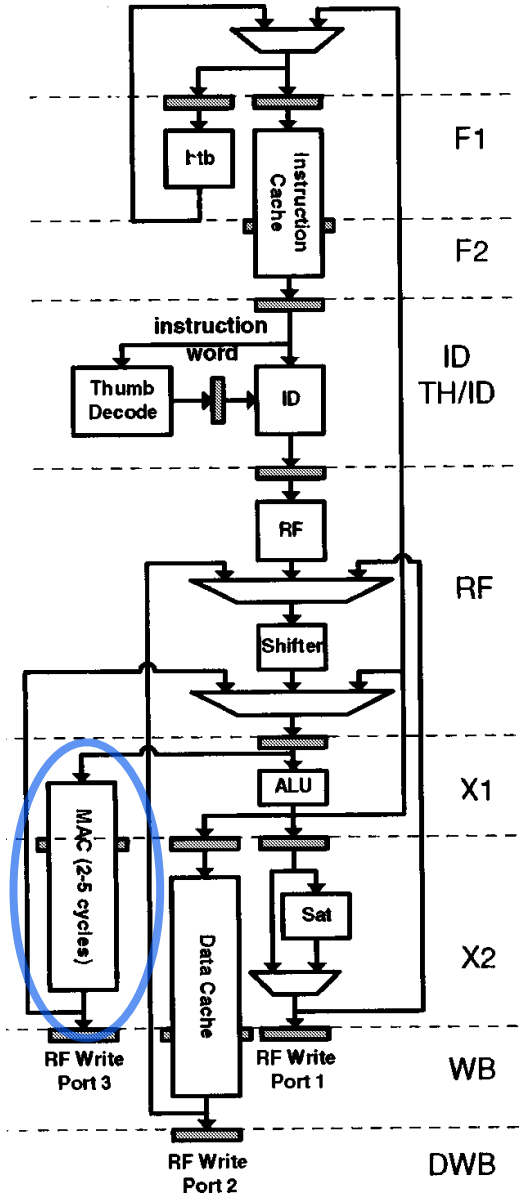
how it works

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

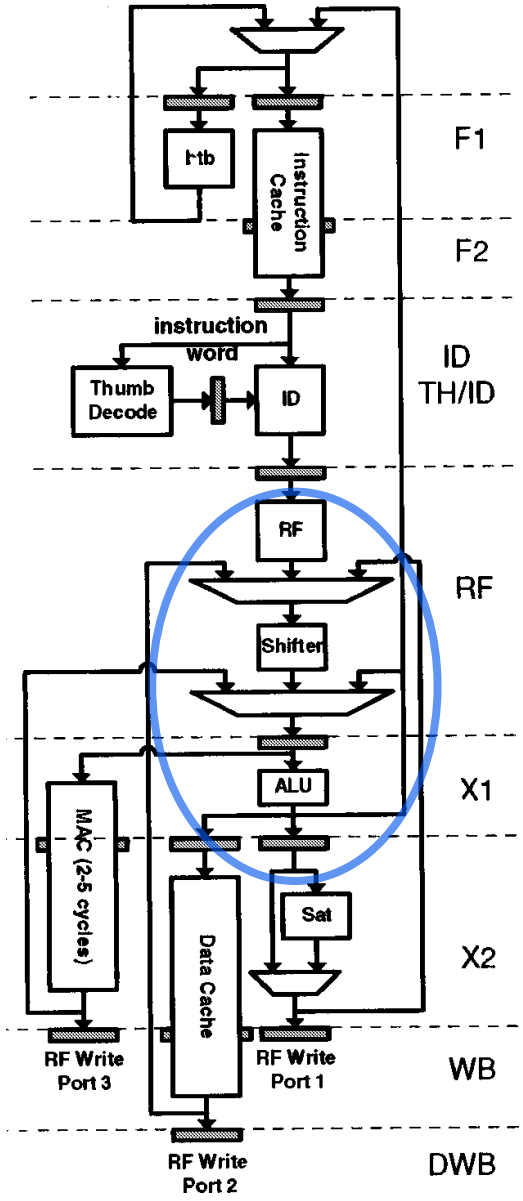
how it works

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

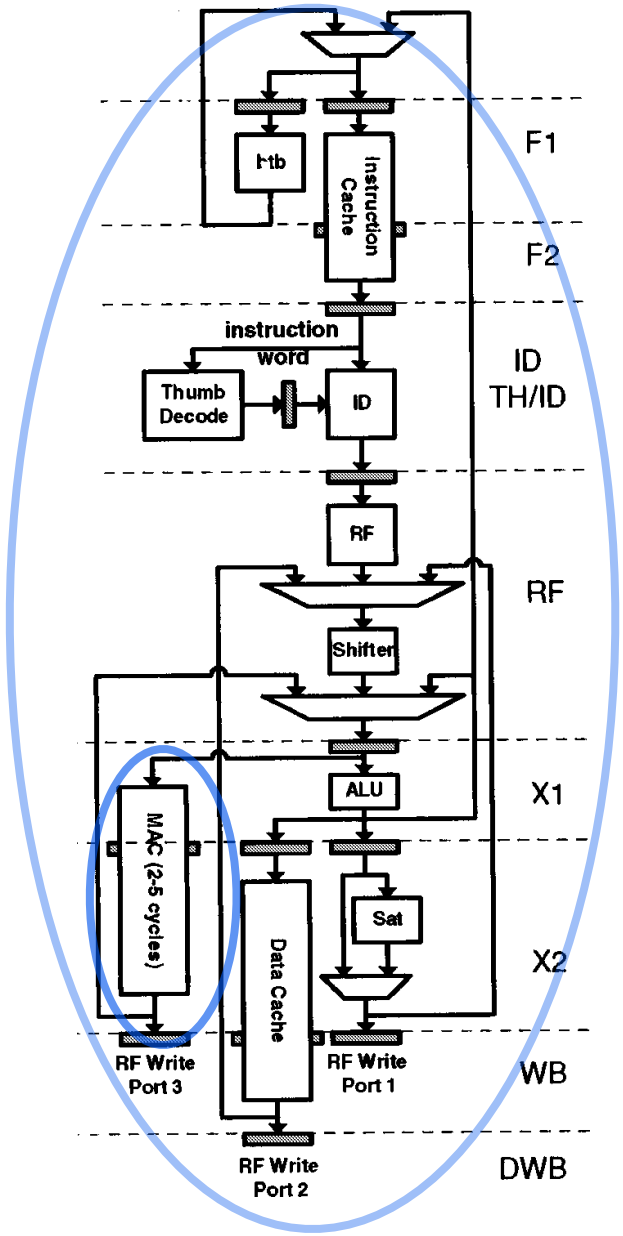
unit testing

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?

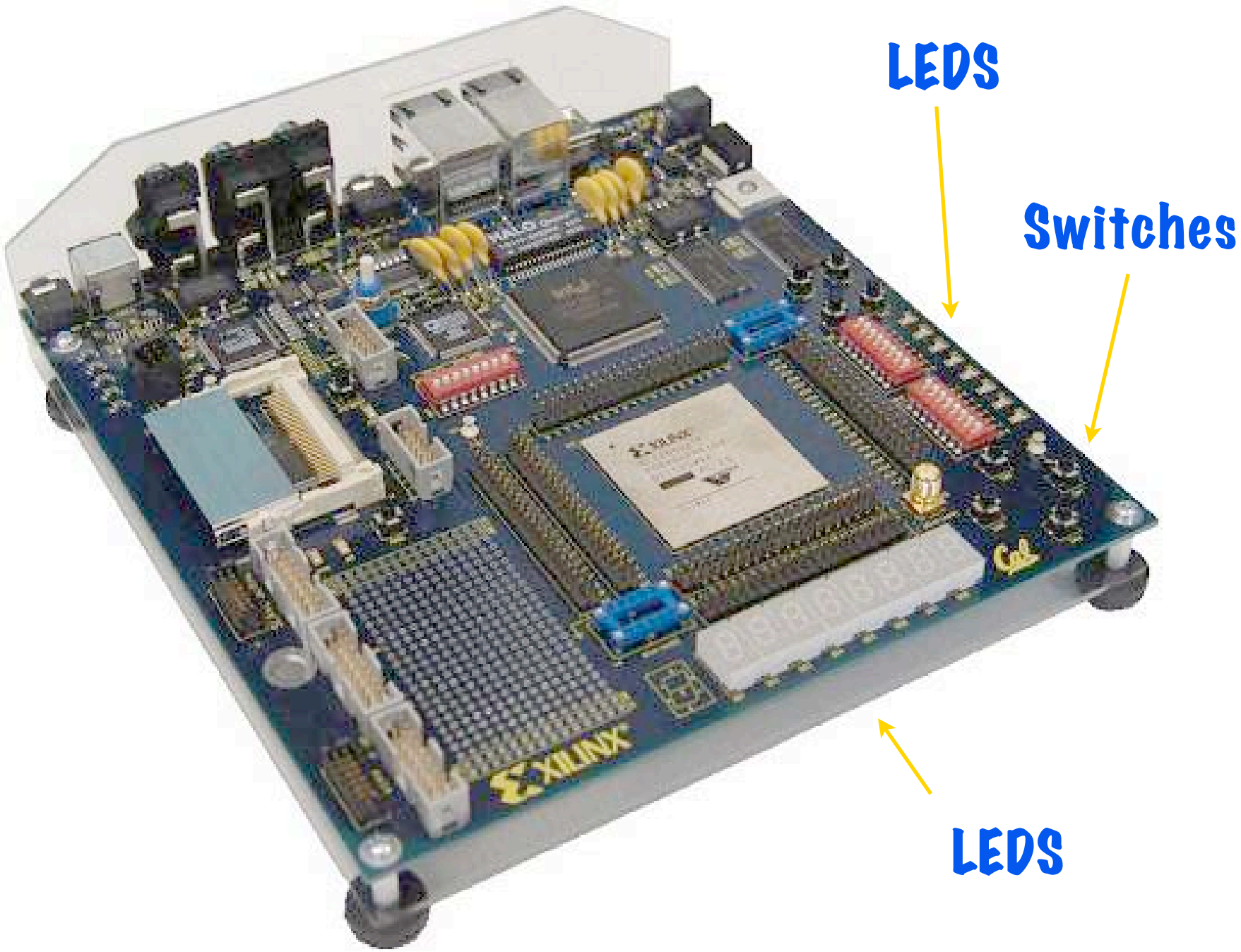
- **Controllability:**

  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

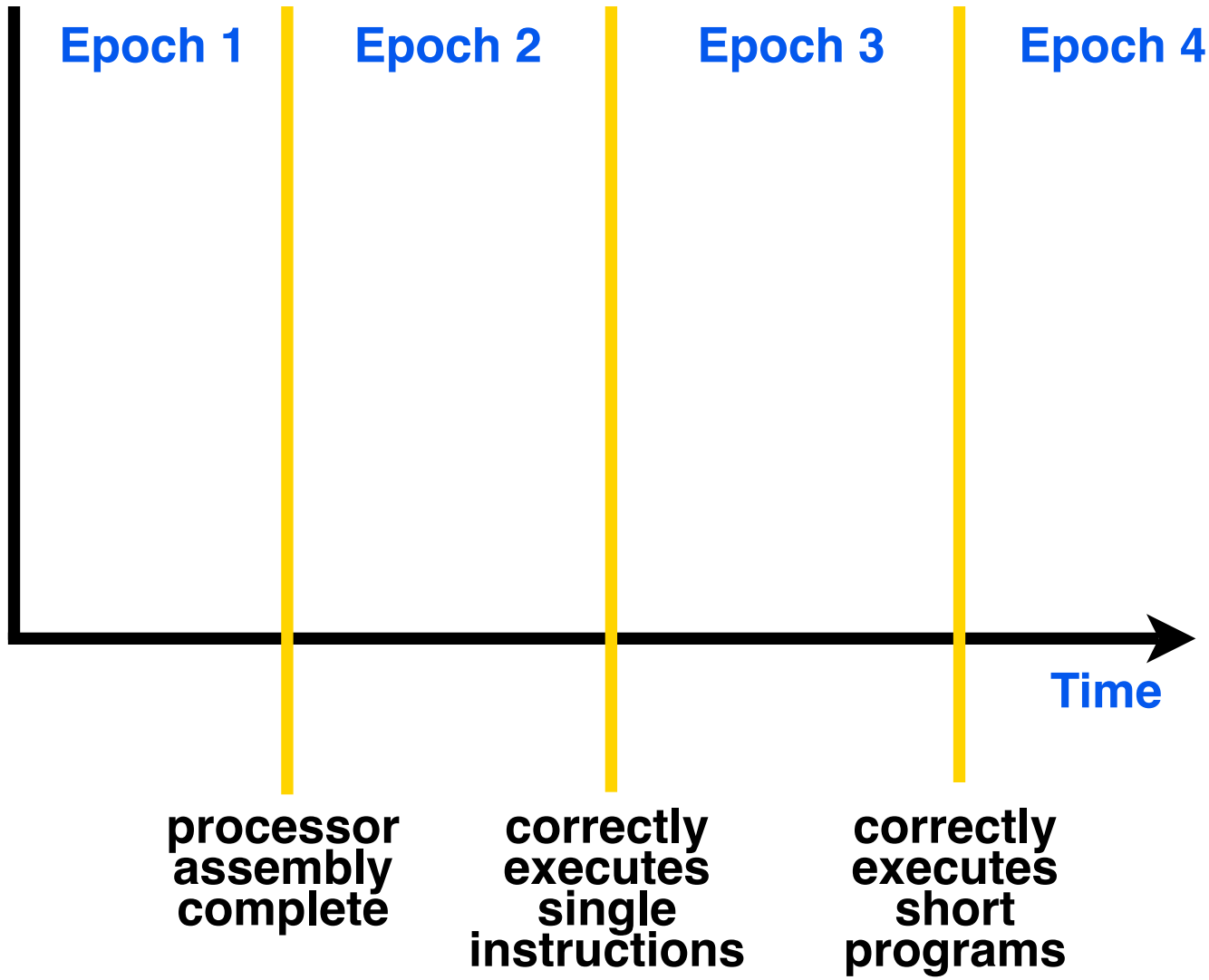
processor testing with self-checks

multi-unit testing

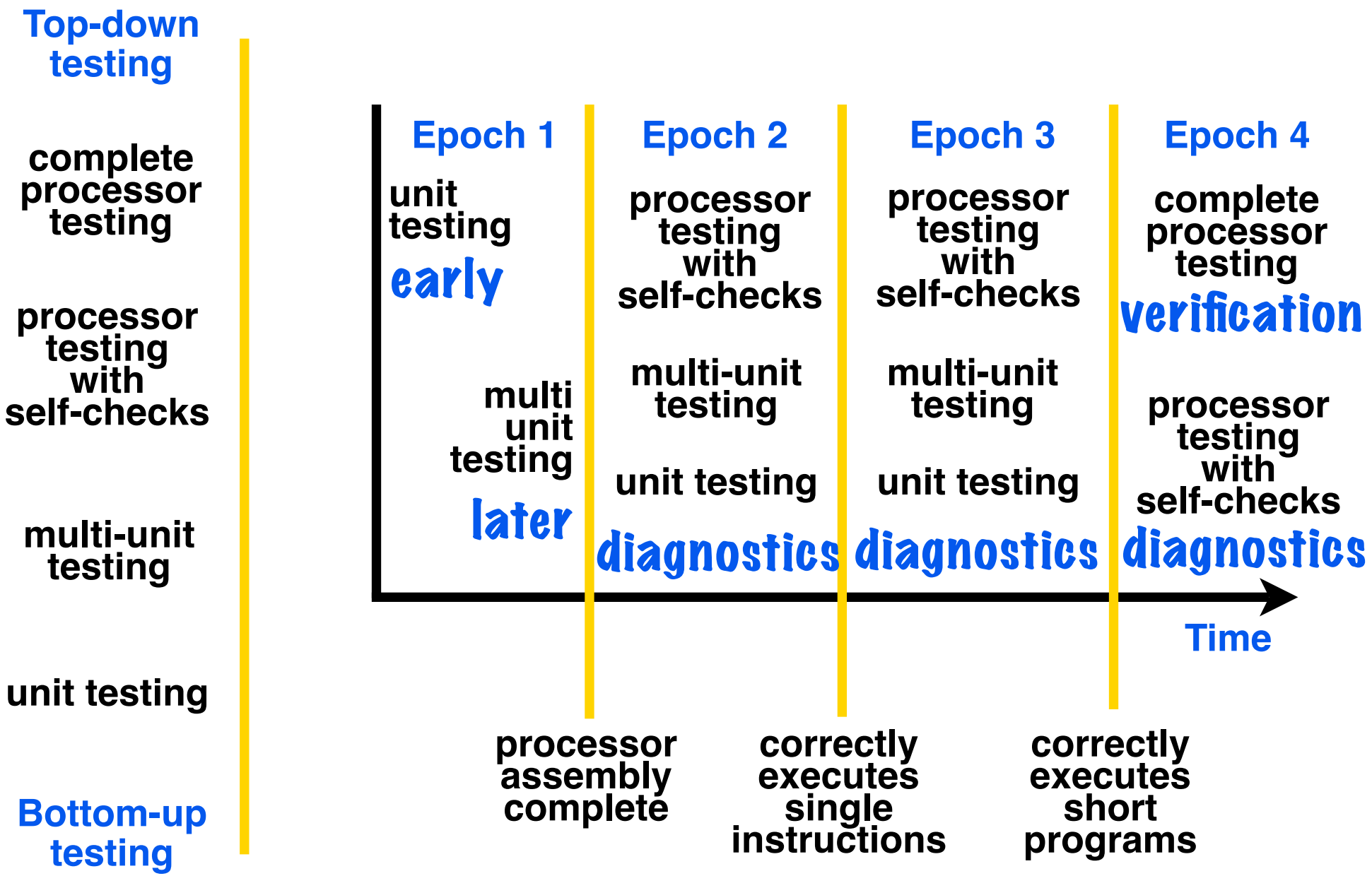
unit testing

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

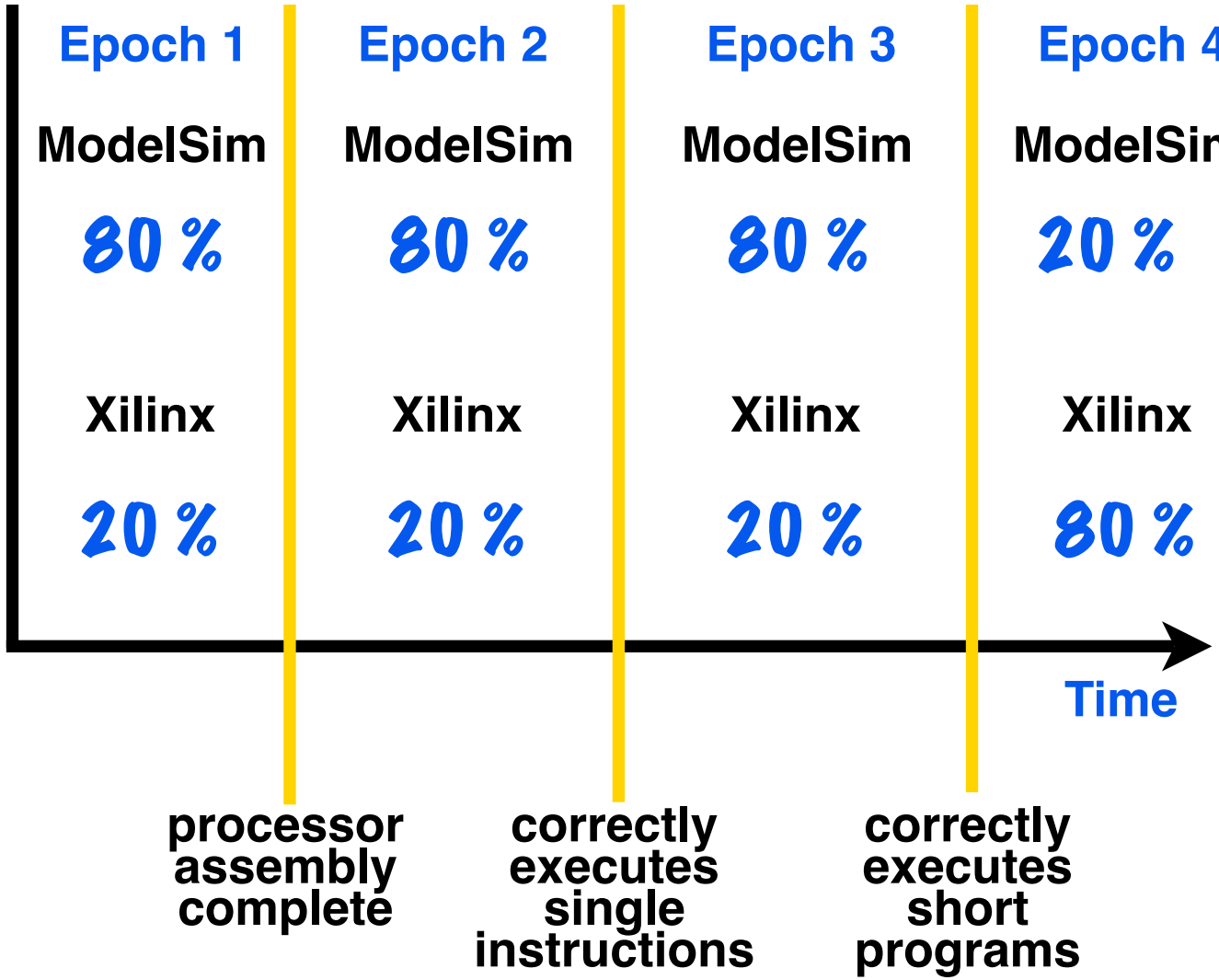
processor testing with self-checks

multi-unit testing

unit testing

Bottom-up testing

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

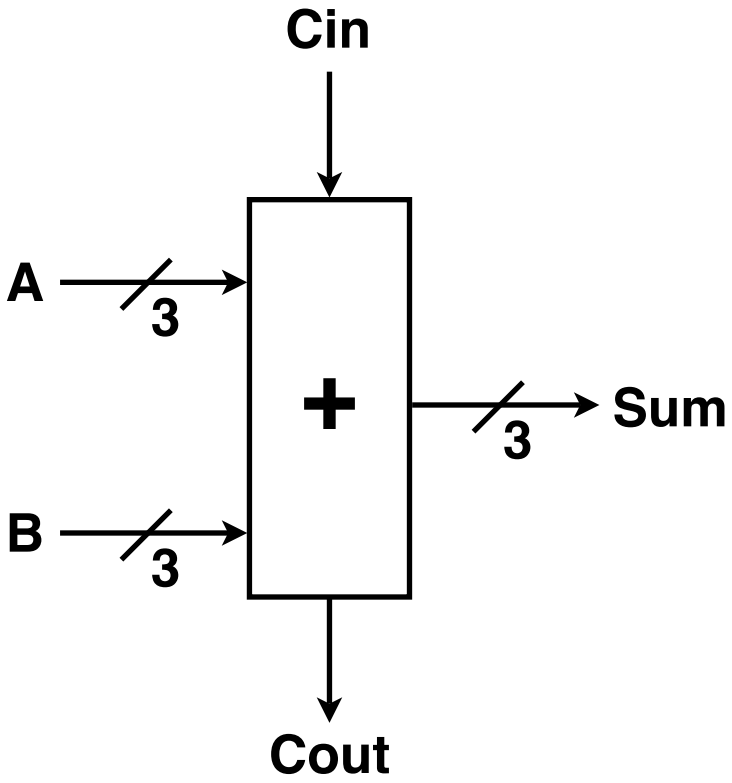


Catch "warnings and errors", signal name misspellings.

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?

$2^7 = 128$

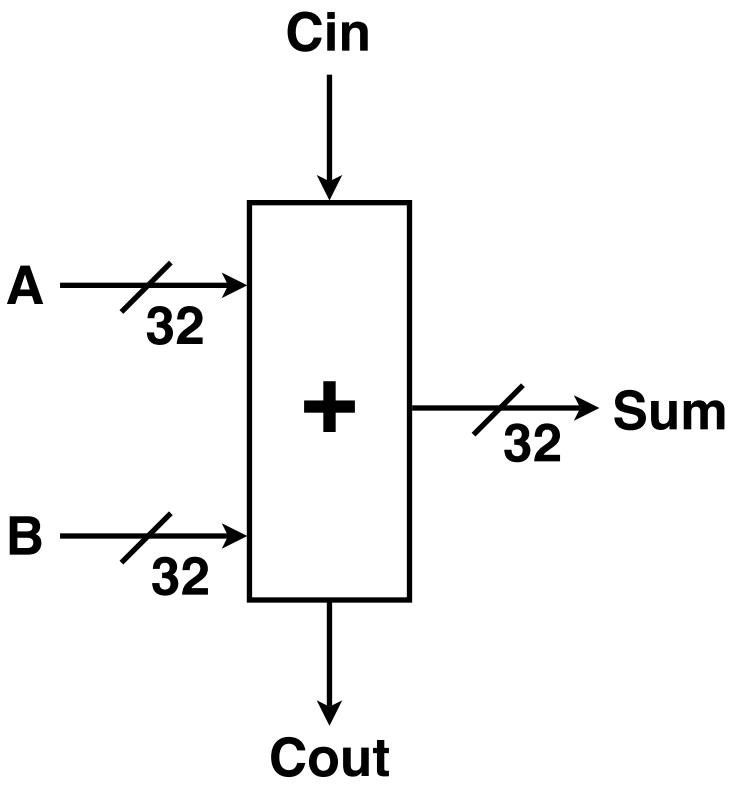
Just test them all ...

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?

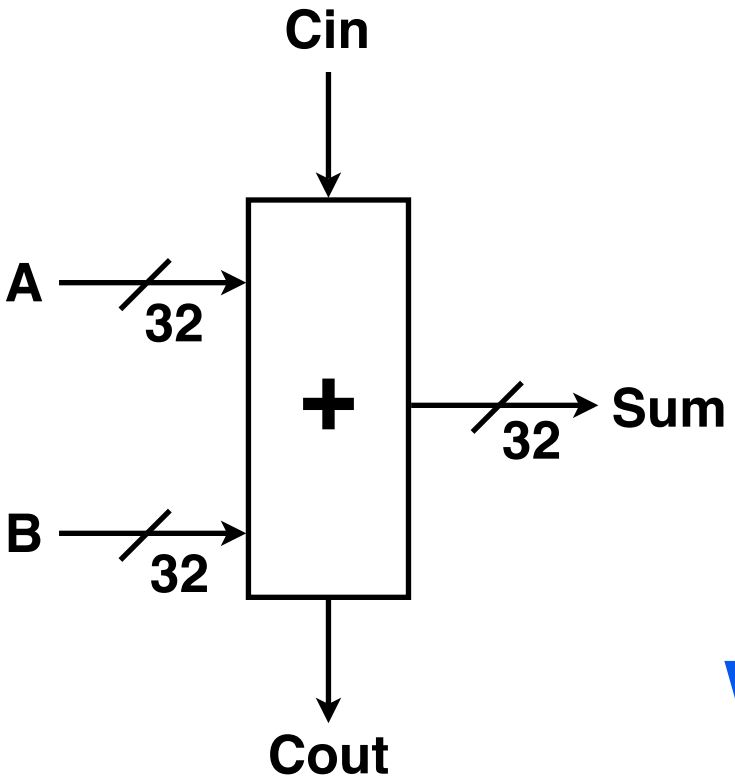
$2^{65} = 3.689e+19$

Just test them all?

Exhaustive testing does not "scale".

"Combinatorial explosion!"

Test Approach 1: Random Vectors

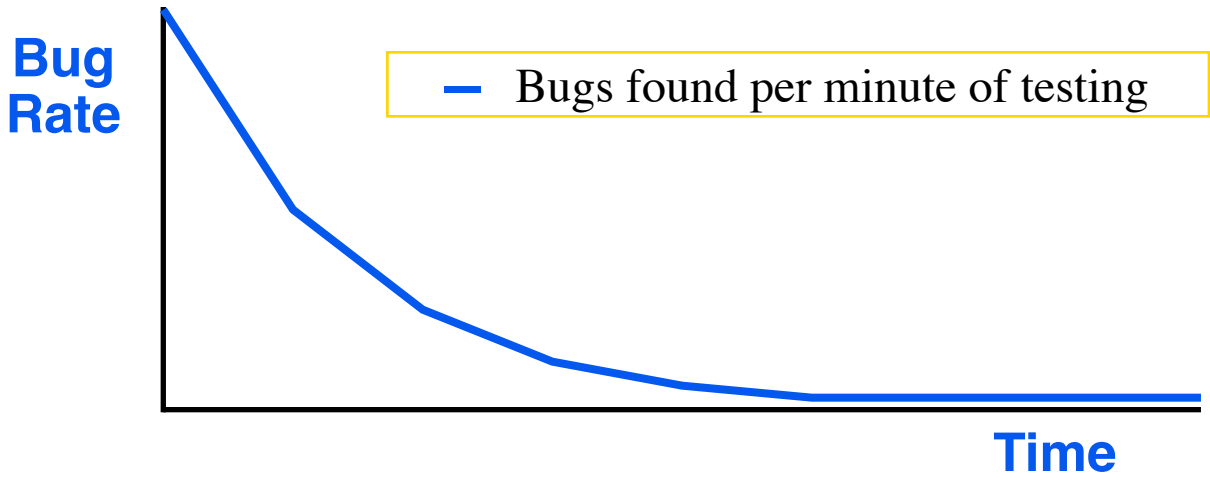


how it works

Apply random A, B, Cin to adder.

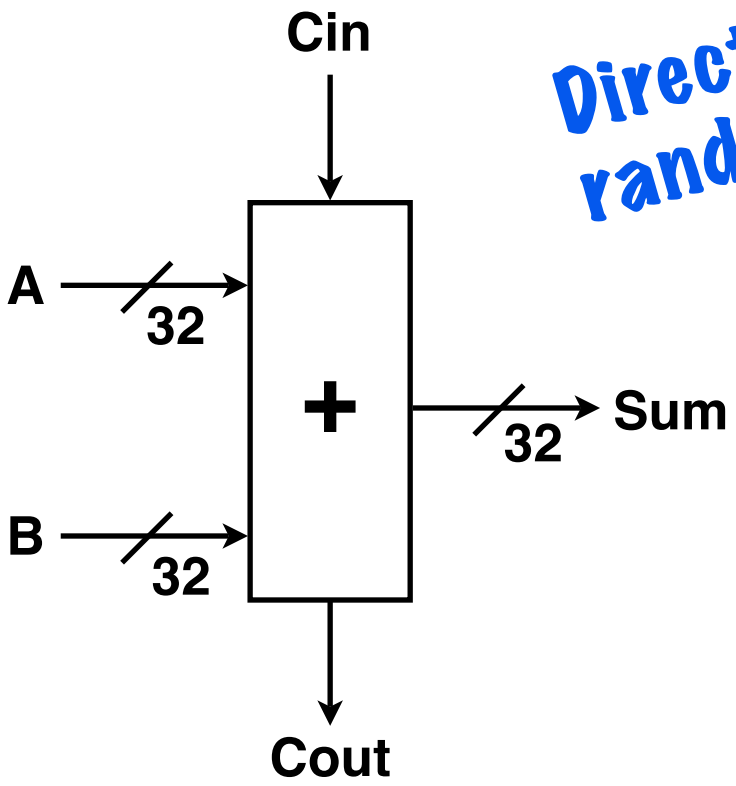
Check Sum, Cout.

When to stop testing? Bug curve.



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

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

Inputs

CLK Change Rst

Outputs

R (red) →

Y (yellow) →

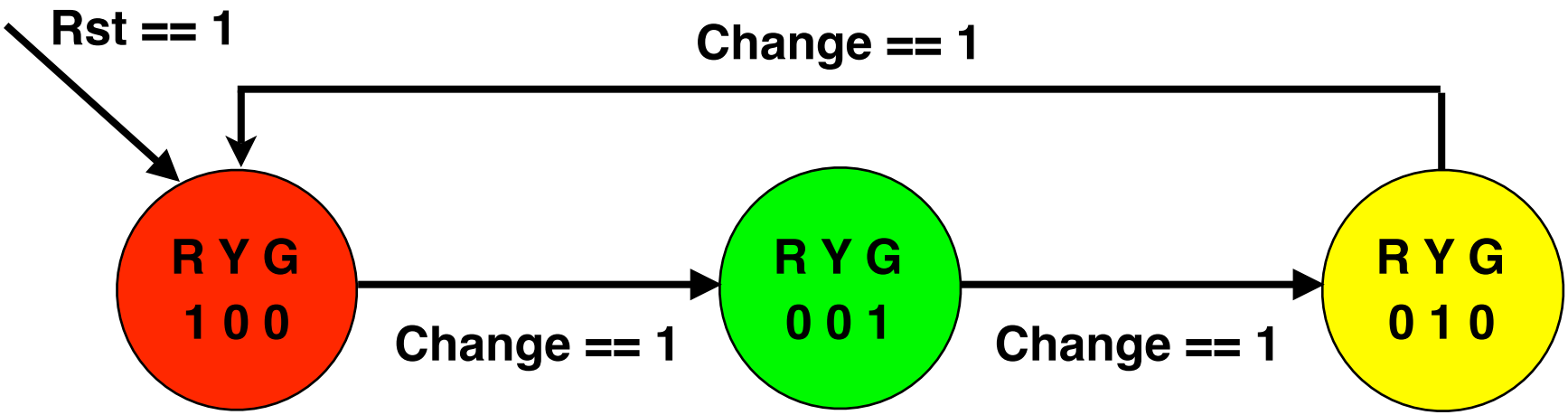
G (green) →

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

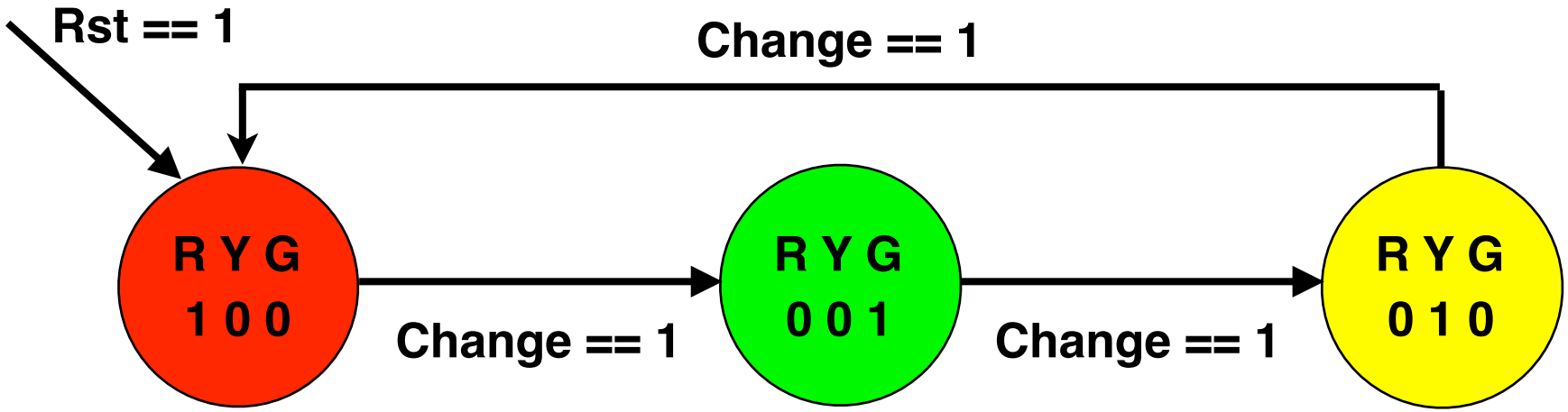
If Rst == 1 on positive CLK edge R Y G = 1 0 0

R	Y	G
1	0	0

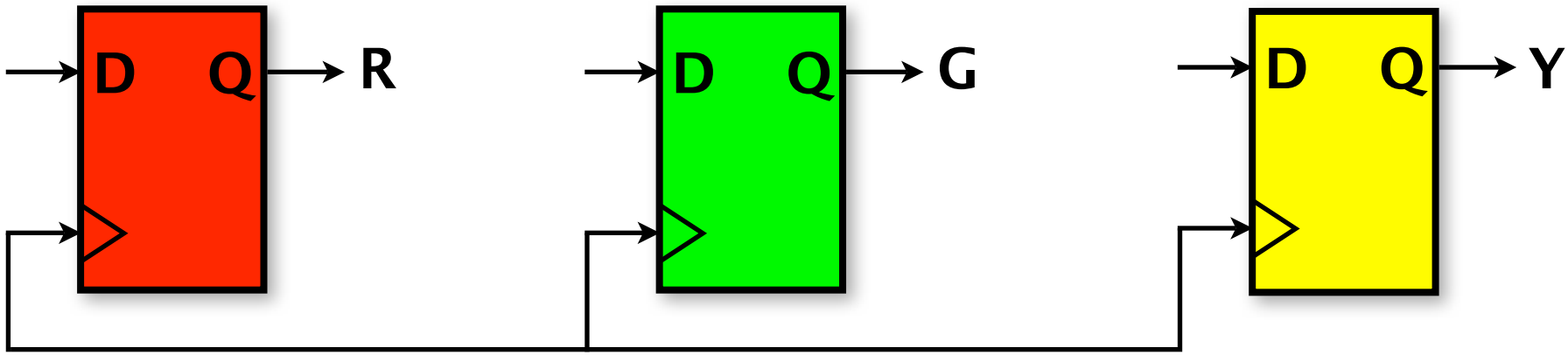
State Machine: Traffic Light Controller



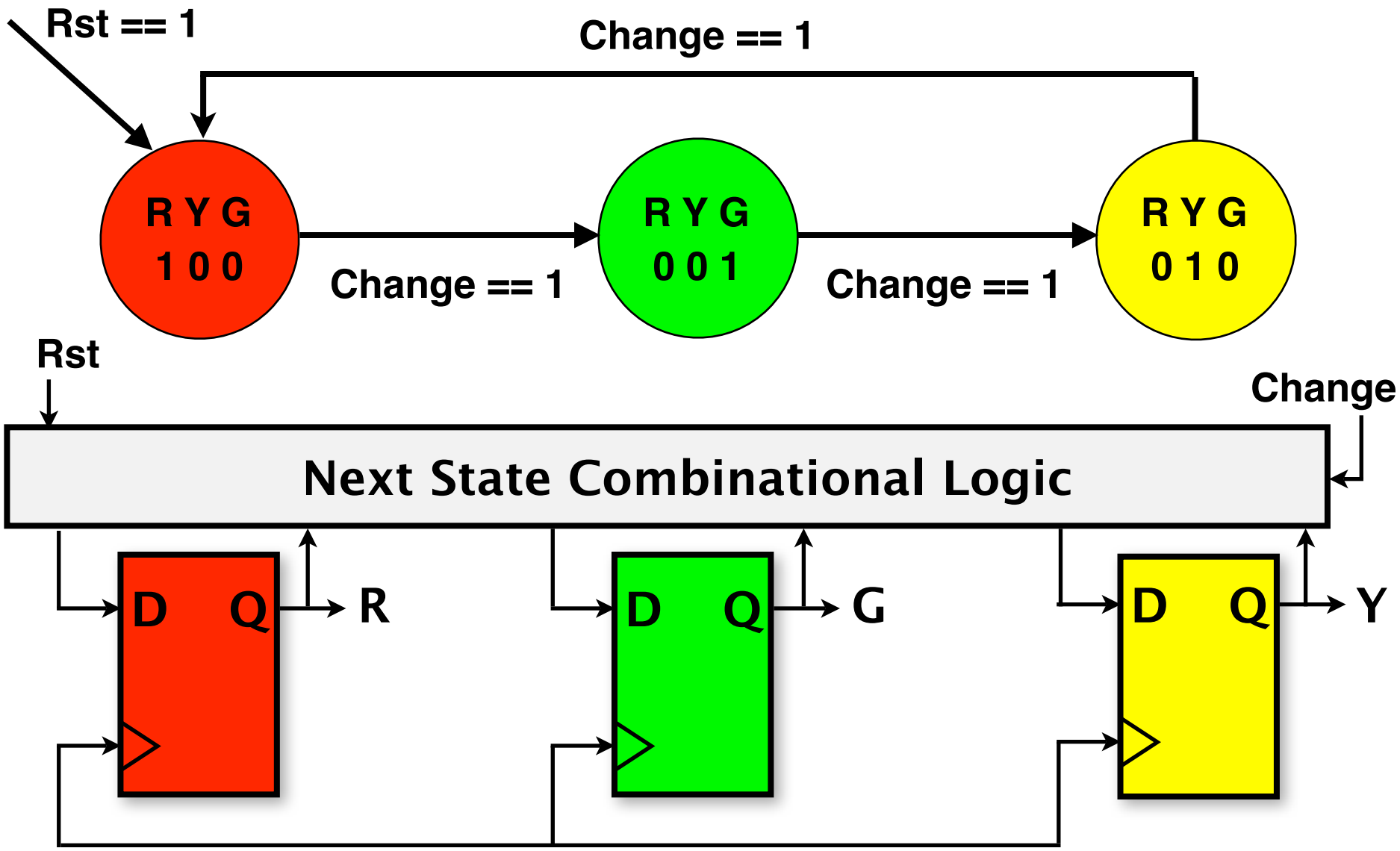
State Assignment: Traffic Light Controller



"One-Hot Encoding"

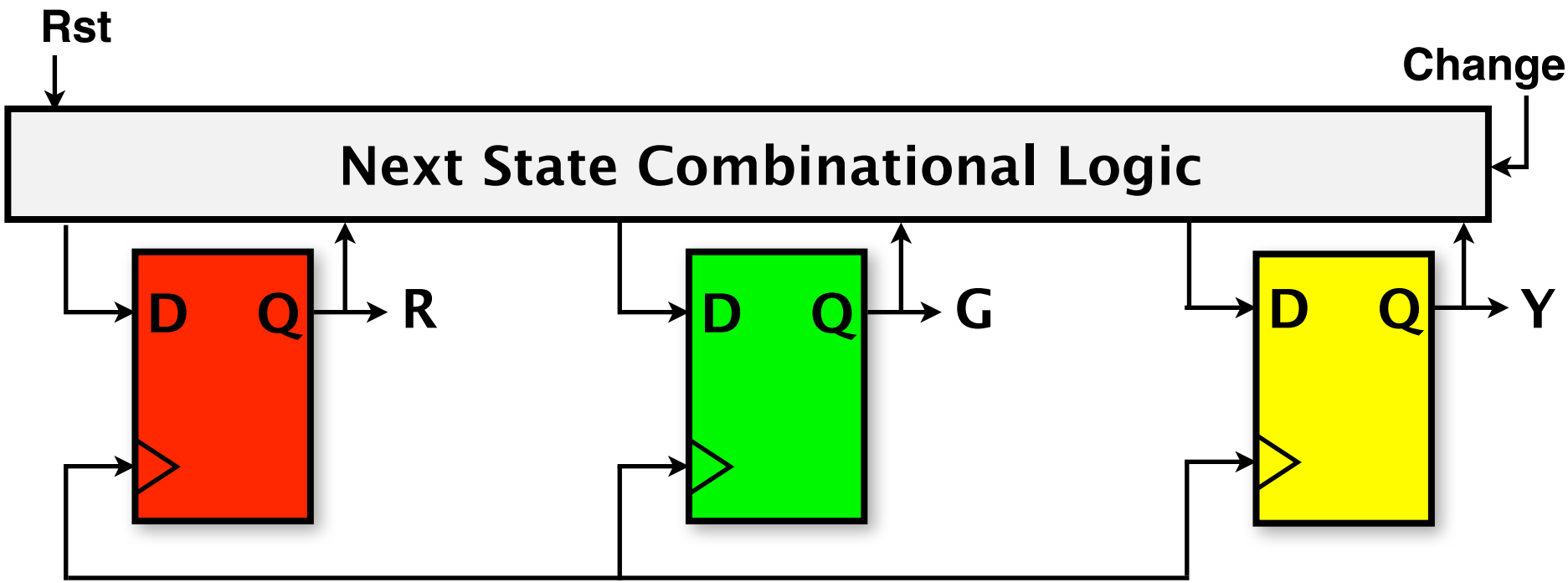


Next State Logic: Traffic Light Controller



State Machine Testing

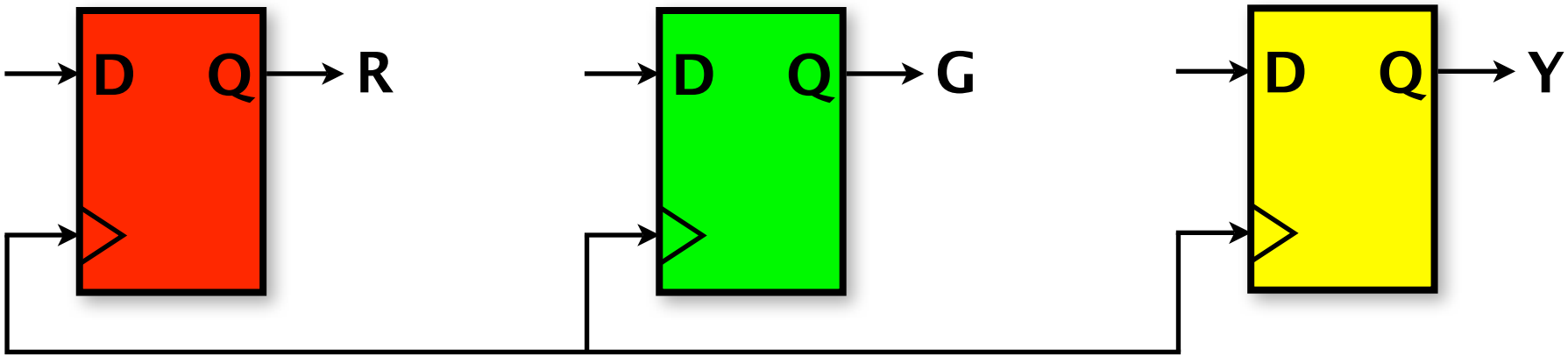
Testing State Machines: Break Feedback



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

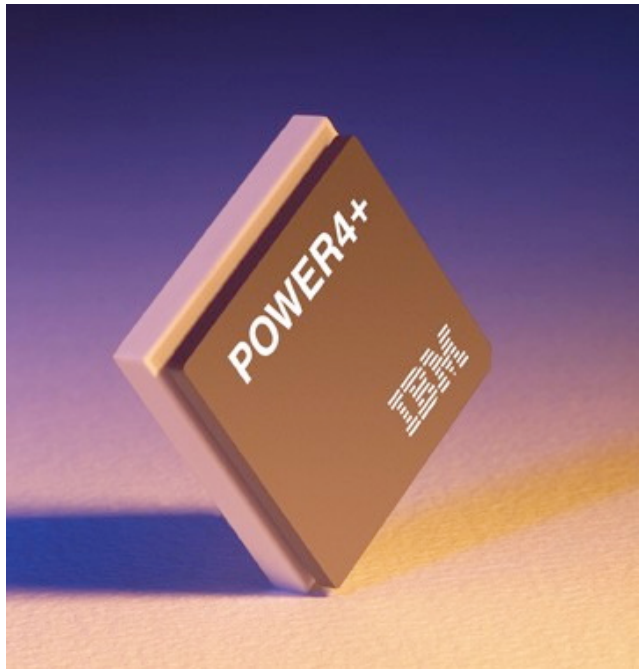
Easier with certain Verilog coding styles?

State Verilog: Traffic Light Controller

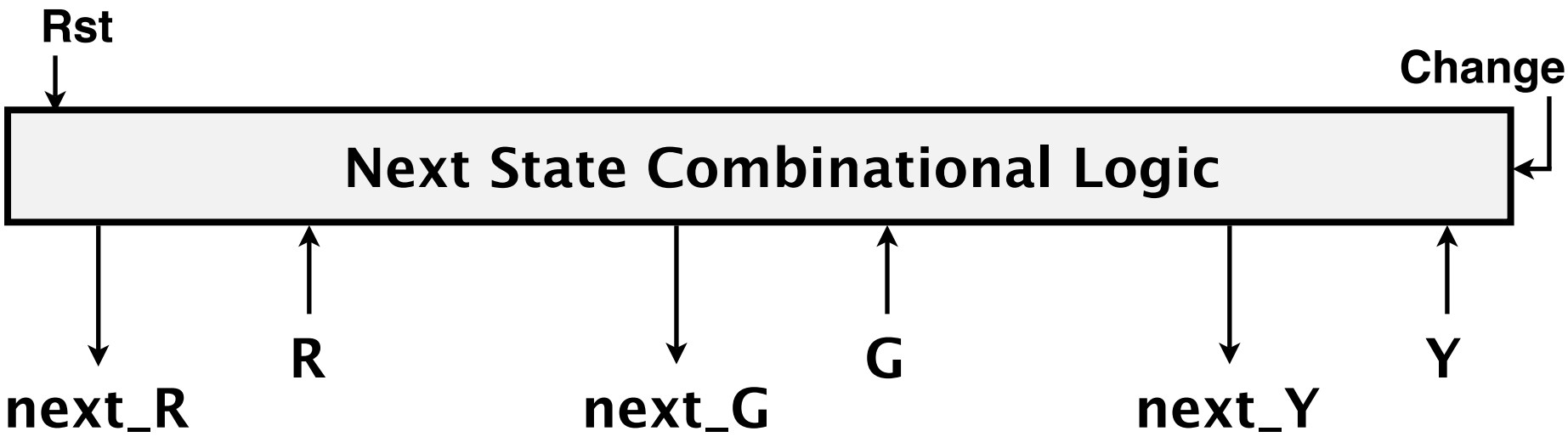



```
wire    next_R, next_Y, next_G;
output  R, Y, G;

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

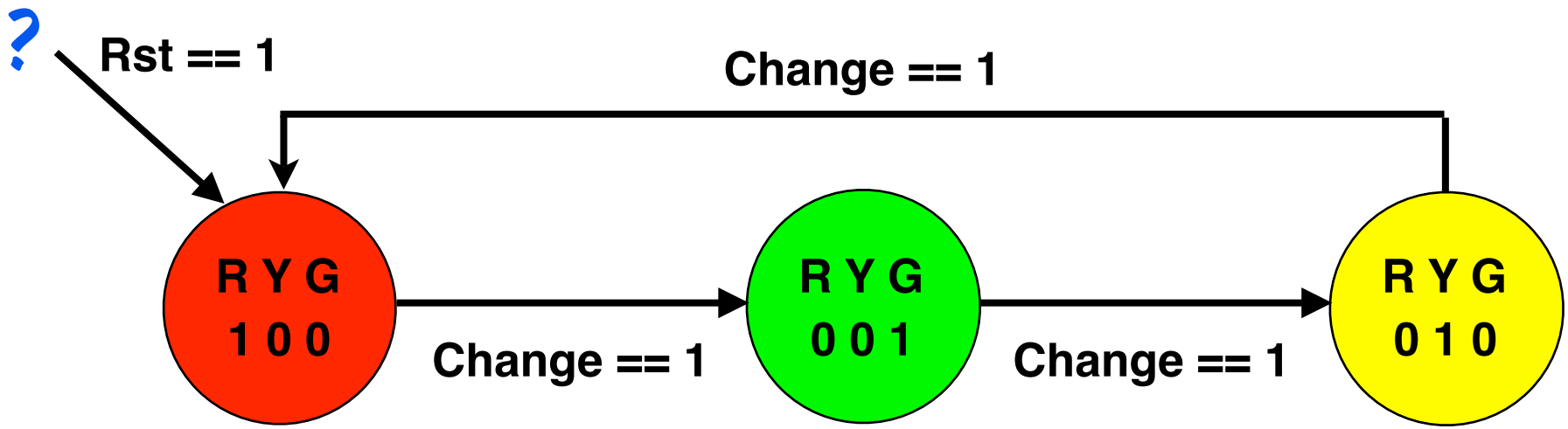
```
wire    next_R, next_Y, next_G;
output  R, Y, G;
```

```
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.

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.
- **Monday:** Lab 2 final design document due to TAs via email, 11:59 PM.

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.

In their
own
words ...

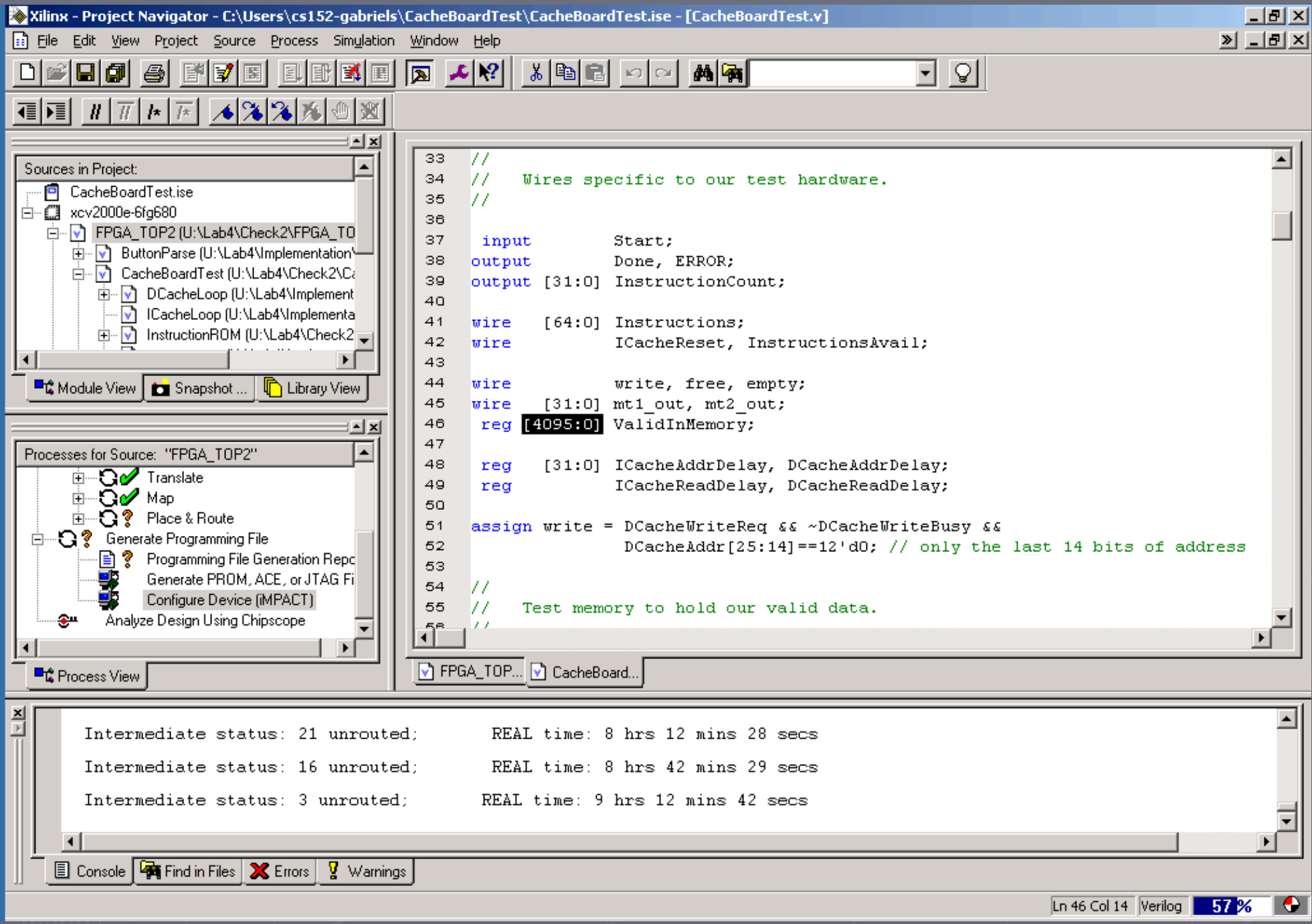
Slides from Fall 04,
Spring 05, Fall 05
CS152 Final Project
Presentations.

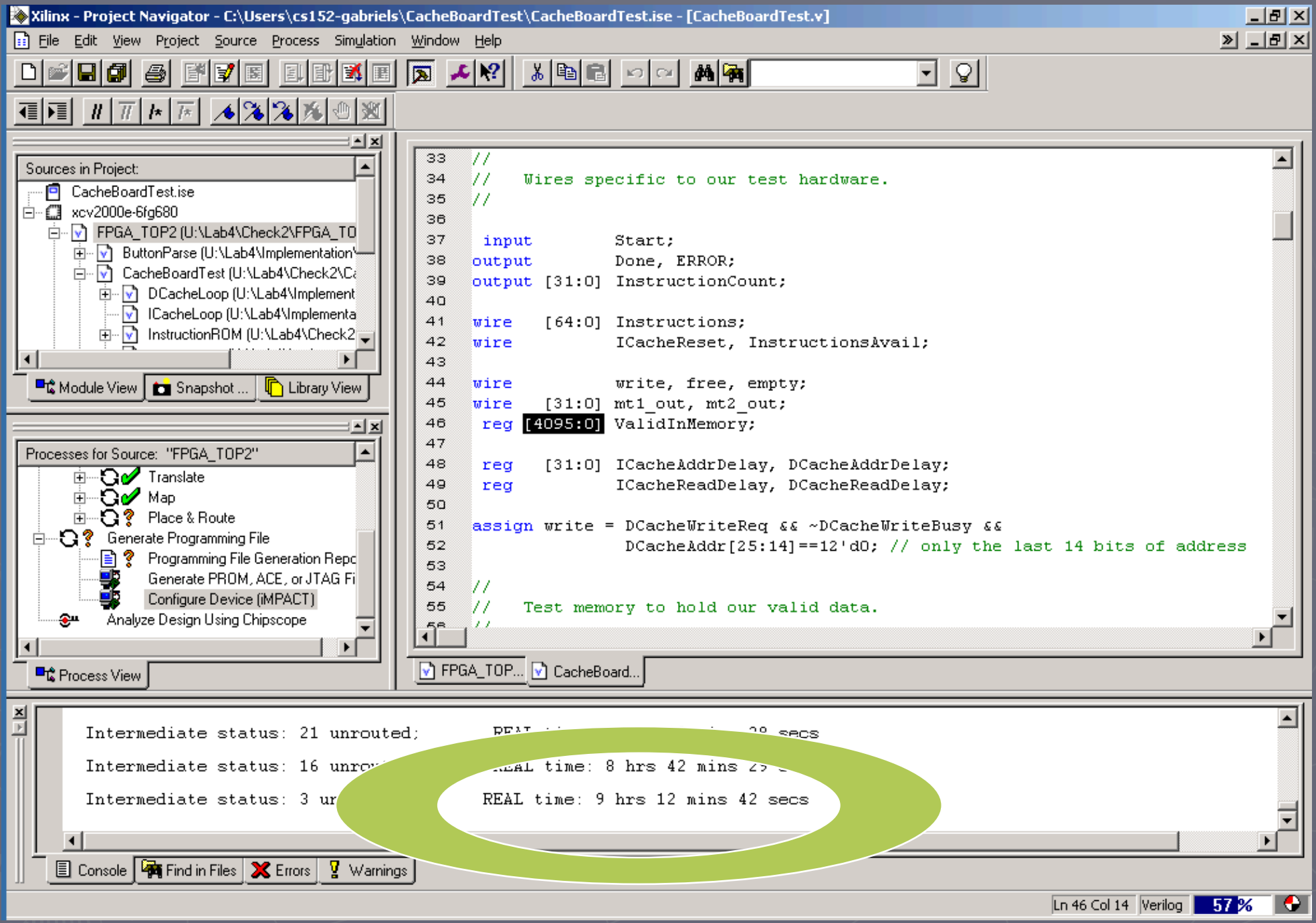
+ a few staff comments.

POSEDGE

Why Posedge?

- Because negedge logic is bad
- Actually, it's because keeping a positive attitude helps you get through the tough times when:
  - Your project won't compile
  - You've had no sleep
  - You realize that the reason it's not compiling is dumb, and you could've fixed it sooner


Xilinx - Project Navigator - C:\Users\cs152-gabriels\CacheBoardTest\CacheBoardTest.ise - [CacheBoardTest.v]
File Edit View Project Source Process Simulation Window Help
Sources in Project:
CacheBoardTest.ise
xcv2000e-6fg680
FPGA_TOP2 (U:\Lab4\Check2\FPGA_TO
ButtonParse (U:\Lab4\Implementation\
CacheBoardTest (U:\Lab4\Check2\C
DCacheLoop (U:\Lab4\Implement
ICacheLoop (U:\Lab4\Implementa
InstructionROM (U:\Lab4\Check2
Module View
Snapshot ...
Library View
Processes for Source: "FPGA_TOP2"
Translate
Map
Place & Route
Generate Programming File
Programming File Generation Repo
Generate PROM, ACE, or JTAG Fi
Configure Device (iMPACT)
Analyze Design Using Chipscope
Process View
33 //
34 // Wires specific to our test hardware.
35 //
36
37 input Start;
38 output Done, ERROR;
39 output [31:0] InstructionCount;
40
41 wire [64:0] Instructions;
42 wire ICacheReset, InstructionsAvail;
43
44 wire write, free, empty;
45 wire [31:0] mt1_out, mt2_out;
46 reg [4095:0] ValidInMemory;
47
48 reg [31:0] ICacheAddrDelay, DCacheAddrDelay;
49 reg ICacheReadDelay, DCacheReadDelay;
50
51 assign write = DCacheWriteReq && ~DCacheWriteBusy &&
52 DCacheAddr[25:14]==12'd0; // only the last 14 bits of address
53
54 //
55 // Test memory to hold our valid data.
FPGA_TOP...
CacheBoard...
Intermediate status: 21 unrouted; REAL time: 8 hrs 12 mins 28 secs
Intermediate status: 16 unrouted; REAL time: 8 hrs 42 mins 29 secs
Intermediate status: 3 unrouted; REAL time: 9 hrs 12 mins 42 secs
Console
Find in Files
Errors
Warnings
Ln 46 Col 14
Verilog
57 %



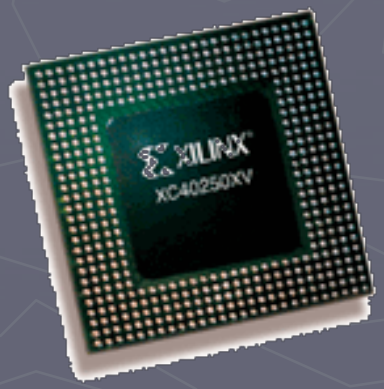
- 9 hours, 12 minutes, 42 seconds… and not yet completed…

Why Posedge?

- That last error was because we thought it would be a good idea to declare a 4096 bit register.
- It wasn't…
- This is why it helps to get sleep!

Posedge Philosophies

- Think top-down about your modules
  - What is the bigger picture for any module that you're creating?
- Think ahead for where it fits in
  - How is it going to interface with the system?



Posedge Philosophies

- Favor correctness over performance
  - In general, you'll get more credit for something that works slowly than something that doesn't work at all!
- Clear interfaces across the entire design
  - This includes making sure that you divide your work properly across all the team members

Bugs We Squished

- Wire misspelling! Took **9 hours** to find
- Cache corner cases in the write buffer
  - Make sure you read the lecture slides thoroughly, there were some good hints there
- Human-induced bugs
  - There was a misinterpretation between how we implemented a board-level Reset and how the TAs wanted it done…

Bugs We Squished

- But overall, we didn't have many difficult bugs in the project
- Why? We took our time with our designs and followed the principles on the next slide…

Development Strategies

- Synthesize your code even before you use ModelSim to simulate it
  - The Synplify compiler will catch many errors for you that the ModelSim one will not
- Search the synthesis log for…
  - Combinational loop detected
  - Latch generated

Combo Loops!

Testing Strategies

- Conceptual testing
  - Rather than exhaust every single possible input and output, understand **why** something works the way that it does
- Randomized testing
  - Used it for pretty much every major module
  - Saves you some effort in the long run

Human Strategies

- Group design effort
  - Everyone is clear on the specs
- Modular work effort
  - Divide the work between all your teammates
  - Avoid having 4 people working on 1 screen
- But help each other test
  - Fresh eyes catch different bugs

Advice for Newbies

- Don’t randomly poke at your code when it doesn’t work, that usually doesn’t help
- Don’t randomly poke other people when their work doesn’t work, be friendly!

Ergo 152

How we felt:

Always chasing the goal…Most of the time we reached it

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?:

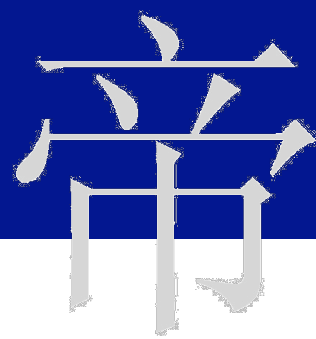
- We could all understand what was going on
- Team unity
- We tried to keep things simple

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?

Rapid Multiply Deep-Pipelined Processor

ETERNAL EMPIRES



On the design itself ...

- Design document stage is VERY important
  - 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
- Devise a good directory structure
- Datapath drawings are VERY important
- Simplify the design
  - Do as much of the logic and design outside of schematics
  - Similar functionality can be merged into one module

CAD and Testing: Asset Management

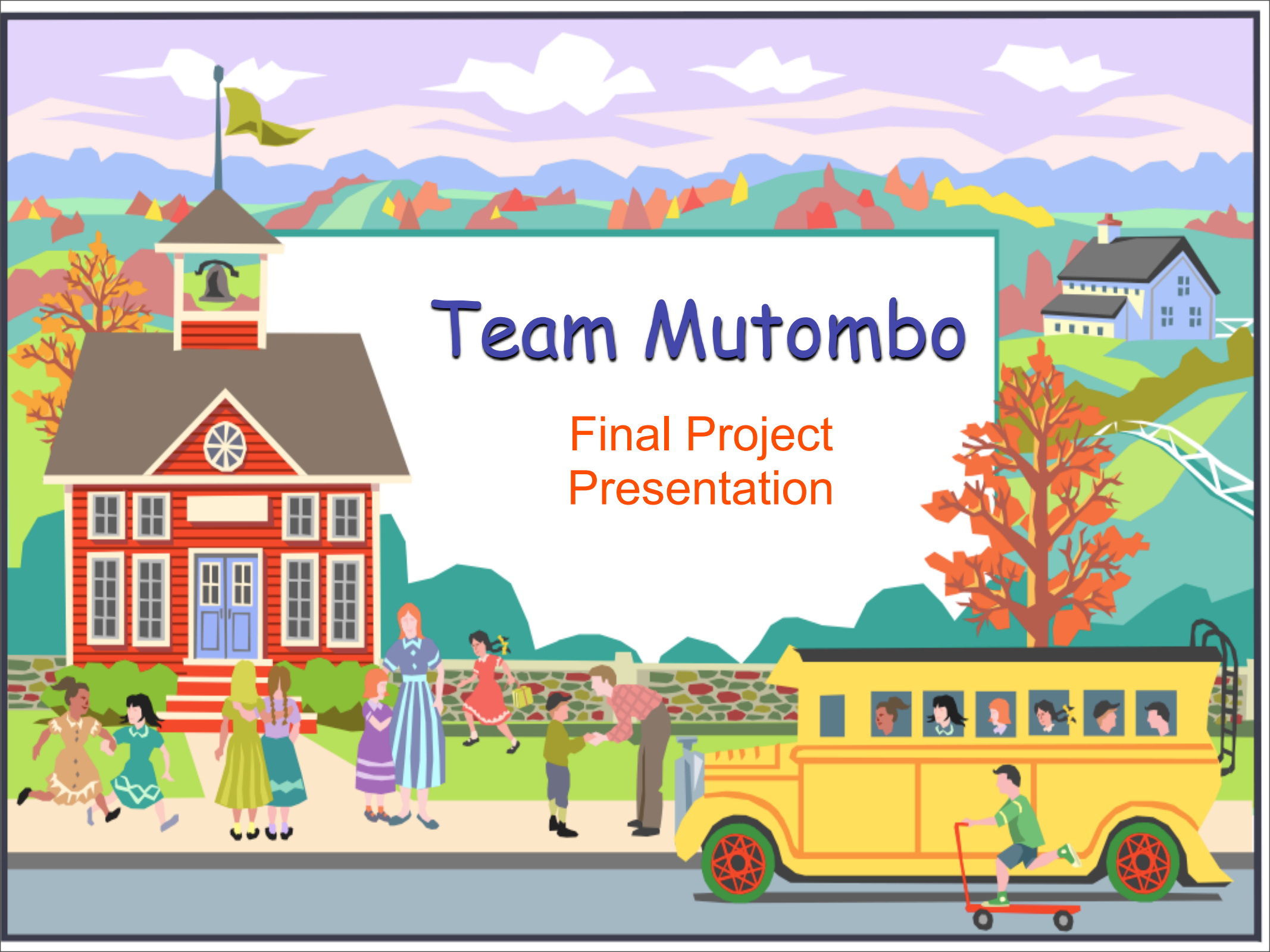
- Agree on where Verilog files will reside in the file directory structure.
- Agree on placement of test bench Verilog and hardware Verilog files.
- Agree on standard way to name files, and standard way to name Verilog modules, variables, parameters, ...
- Don't copy files -- include them. Each file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


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

CAD

- Save wave forms between sessions
- Make a default project navigator .npl
- Save transcripts after running simulation tests
- Make a custom modelsim.ini file
- Use cvs & descriptive comments when you checkin
- Use multiple computers to maximize efficiency.

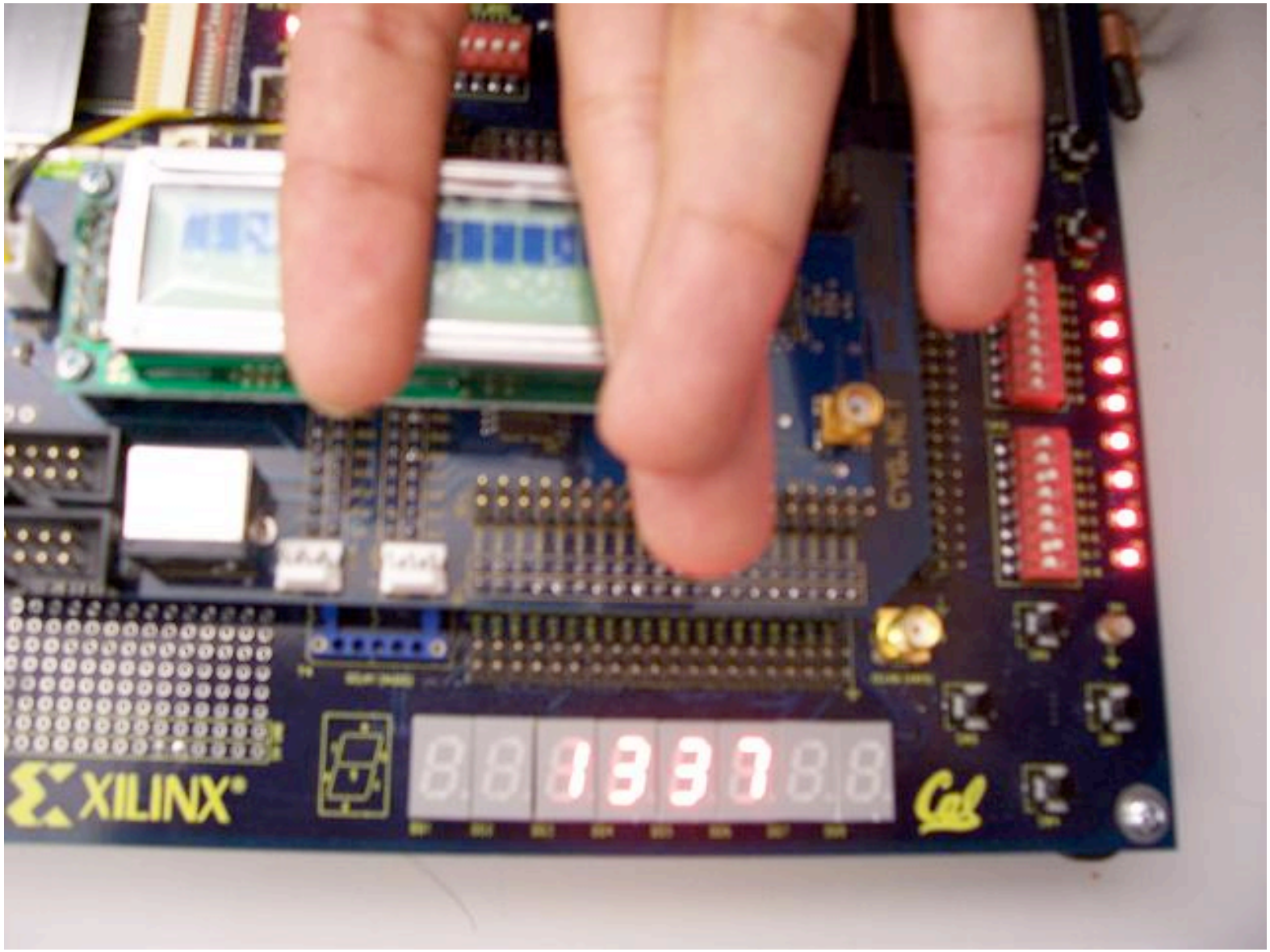
Design Methodology

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

Always have a fallback ...

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.


XILINX
Cal

CS152 Final Presentation

The Four Bytes

Fall 2005

Maintaining Group Dynamics

- Try to collaborate as much as possible and as early as possible
- Organize who is responsible for what early on
- Make sure to communicate to each other what you worked on/accomplished when not working in a group.
- Use a versioning system so that you don’t overwrite each other’s updates

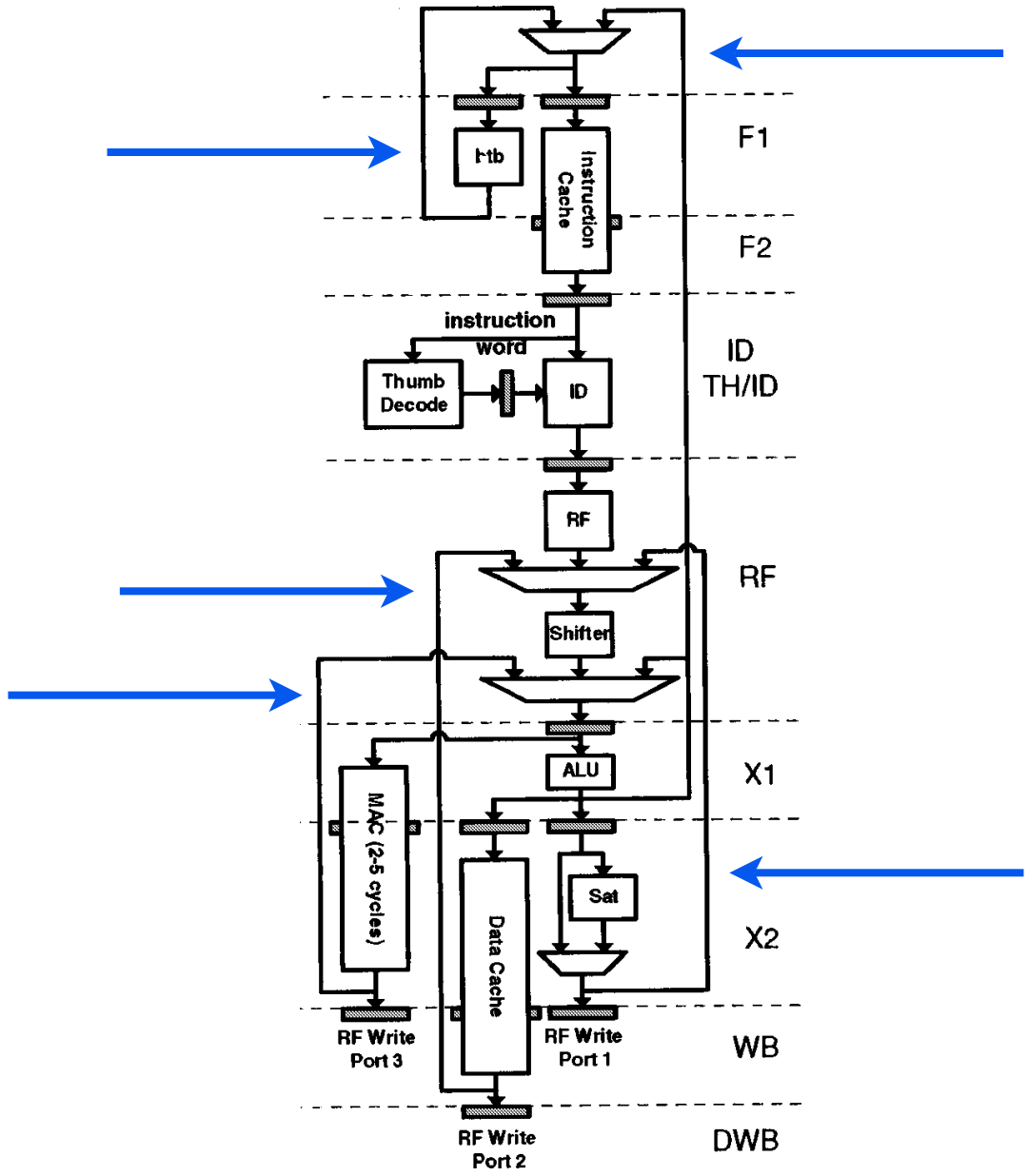
Design Methodology

- Design early on as much as possbile
- Make sure all flip flops have their reset ports connected to a reset signal (can get weird things happening on the board if you don’t).
- Break modules into subsections whenever possible to maintain simplicity and abstraction
- Understand pre-written code given to you by the TA’s. (TFTP was the hardest part of Lab 3)

Design Methodology

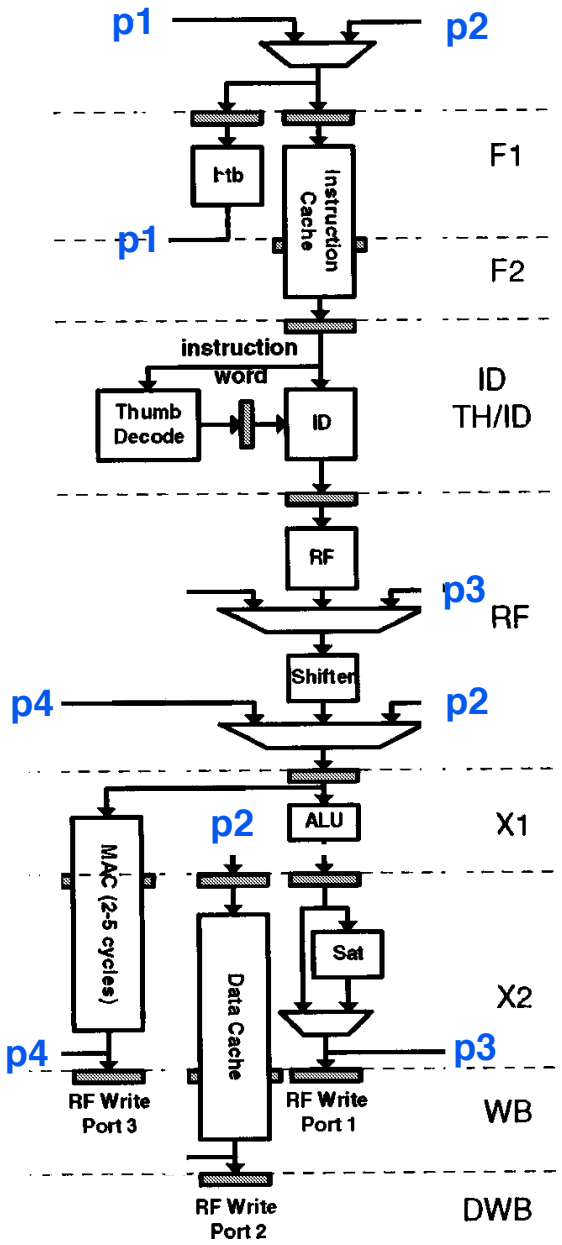
- Come up with what you name your input and output ports early, so that you won't have conflicts later
- Make good schematics. Doing this gives a better understanding of the higher level design.
- Create modules that are as independent as possible from other modules

Schematics: This schematic uses wires ...



This schematic uses labels ...

Which is easier to understand?



Problems

- Memory Mapped I/O
  - Difficult to get time correctly
  - Pay attention to which signal are synchronous and which are asynchronous
  - Understand how this module interacts with other modules in the processor

Problems

- Handling Clock Boundaries
  - Make sure to look at how positive edges of different clocks can interact (ie ButtonParser, SDRAM arbiter, etc.)
  - Make sure to use different clocks when doing simulation to try to root out these type of bugs.

Problems

- Don’t fall into trap of using one giant module
  - Makes it really hard to find problems
  - Too many things are happening at the same time
  - To solve this, break things apart into sub-modules and use layers of abstraction.

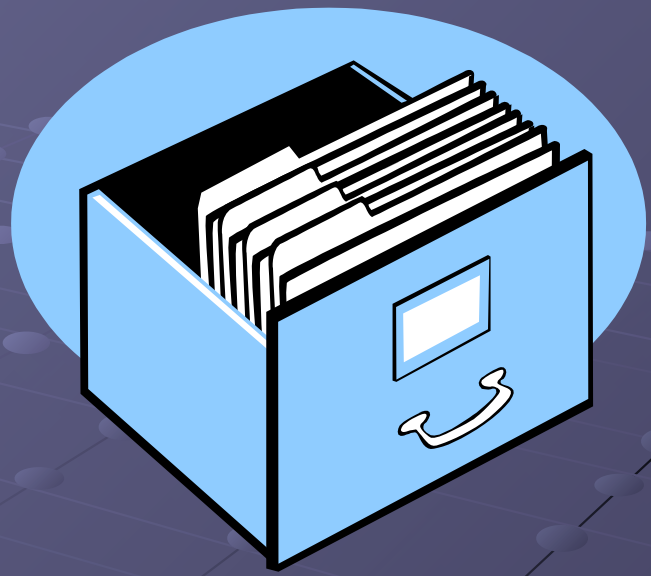
Final Project Presentation

Team Opeerrand

CS 152 Fall 2005

- Know your add/drop/ grade-option deadline!

Project Organization



- CVS is your friend…
  - … if you actually use it
  - and remember to update
- Read the Design Doc
  - … several times
- Working schematics before coding

Design

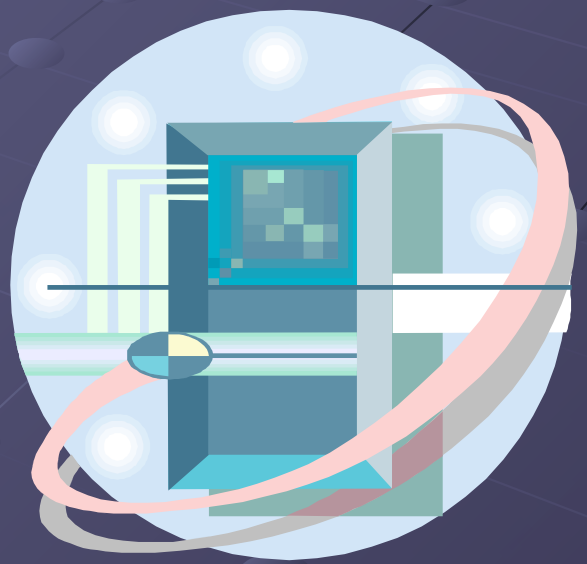
- Spend more time on Design Doc
- Reiterate: consistent signal naming
- Pen & paper
  - not too low-tech
- Split up the work

Simulation

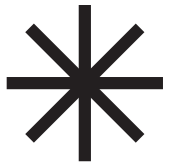
- Save all waveforms
- Transcript display, not just waves
- Error signal if possible
- Use random number generators

Synthesis

- Read the warnings
- Read the warnings (not a typo)
- Start working on FPGA_TOP early
- Not all boards work equally
- ChipScope?



CAD and Verilog ...



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 "warnings and errors", and inspect output tools produce!

Synplicity examples: latch generated", "combinational loop detected", etc

Also: If your CPU is not wired to a Calinx LED, Synplicity may optimize away your CPU !!

More Testing

- Start writing test code early
  - Don’t just rely on check-off tests
  - Reuse old project test code
- Don’t skip unit testing
- Everyone should help test (4>1)
- Use all board LEDs/switches

Reset & PC

- PC counter at reset
  - Especially on the board
- Watch the first instruction
  - Don’t lose or repeat
- Remember to reset cache tags
- Behavior under stalling
  - Different stalls may affect PC differently

Remember: CPUs must meet ISA spec

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

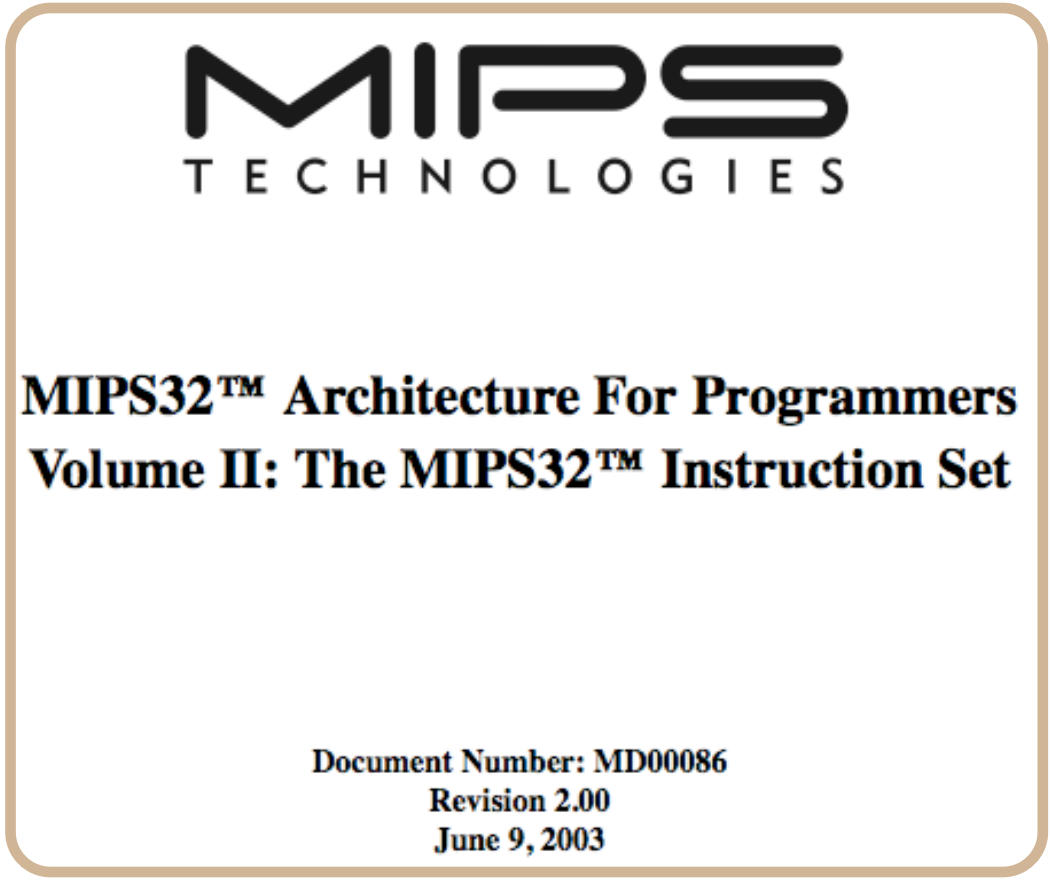
I think this means students should be reading over the lab spec more carefully (perhaps twice before starting and referring back to it often during development).

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.

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 ...