CS152 – Computer Architecture and Engineering

Lecture 3 – Testing Processors

2004-09-07

Dave Patterson

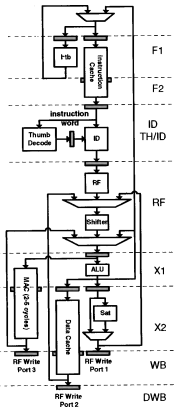
(www.cs.berkeley.edu/~patterson)

John Lazzaro

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

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

Last Lecture: Clocked Logic Review



design description

Schematic Diagrams

Timing Diagrams

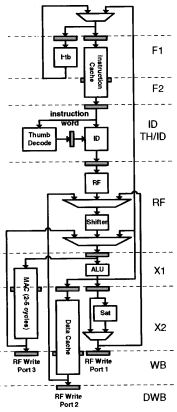
Verilog

Intel XScale ARM Pipeline, JSSC 36:11, November 2001

Outline - Testing Processors

- **The four types of testing.**
- **Making a test plan.**
- **Unit testing techniques.**

Lecture Focus: Testing 152 Projects



testing goal

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

Clock speed? CPI? Upcoming lectures ...

Four Types of Testing

Big Bang: Complete Processor Testing

Top-down testing

- **complete processor testing (Lab 1)**

Bottom-up testing

how it works

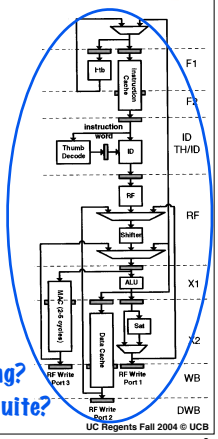
Assemble the complete processor.

Execute test program suite on the processor.

Check results.

Why is this method appealing?

What makes a good test program suite?



Methodical Approach: Unit Testing

Top-down testing

complete processor testing

- unit testing

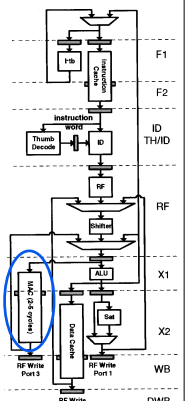
Bottom-up testing

how it works

Remove a block from the design.

Test it in isolation against specification.

What if the specification has a bug?



Administrivia - Mini-Lab 1 a Success!

- Survey due today!
- Mini-Lab 2 this Friday (9/10). Remember to do the pre-lab!
- Lab 1 due Monday 9/13. Don't wait to get started!
- First homework out soon -- due 9/15.

Four Types of Testing

(continued)

Climbing the Hierarchy: Multi-unit Testing

Top-down testing

complete processor testing

- multi-unit testing

unit testing

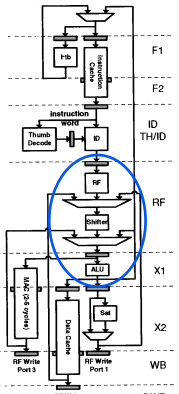
Bottom-up testing

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

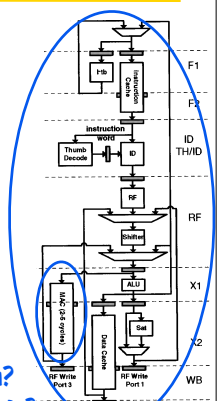
how it works

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?

Writing a Test Plan

(peer instruction)

Test Plan: Fill in the testing timeline

Top-down testing

complete processor testing

processor testing with self-checks

multi-unit testing

unit testing

Bottom-up testing

Which testing types are good for each epoch?

Epoch 1 | Epoch 2 | Epoch 3 | Epoch 4

Time

processor assembly complete

correctly executes single instructions

correctly executes short programs

Fill in the testing timeline: Answer:

Top-down testing

complete processor testing

processor testing with self-checks

multi-unit testing

unit testing

Bottom-up testing

Which testing types are good for each epoch?

Epoch 1	Epoch 2	Epoch 3	Epoch 4
unit testing early	processor testing with self-checks	processor testing with self-checks	complete processor testing verification
multi unit testing later	multi-unit testing	multi-unit testing	processor testing with self-checks diagnostics
	unit testing diagnostics	unit testing diagnostics	

Time

processor assembly complete

correctly executes single instructions

correctly executes short programs

Testing Mechanics: Asset Management

adder.v

Adder unit.

adder_utb.v

Unit testing test bench for adder.

adder_sc.v

Self-checking wrapper for adder unit.

Instantiate into _utb and _sc, not copy-paste! Why?

```
% ls
dpath/
%
% ls dpath/adder dpath/regfile dpath/shifter
dpath/adder:
adder.v         adder_sc.v      adder_utb.v

dpath/regfile:
regfile.v       regfile_sc.v    regfile_utb.v

dpath/shifter:
shifter.v       shifter_sc.v    shifter_utb.v
%
% ls dpath
adder/                  mid_dpath.v             regfile/
mid_dpath.mutb.v        mid_dpath_sc.v          shifter/
```

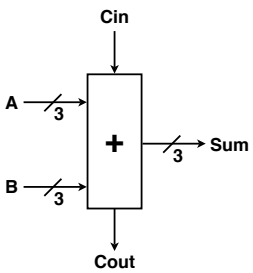
One directory holds all datapath units.

Follow a file naming convention! Why?

mid_dpath is a multi-unit: instantiates adder, regfile, shifter.

Unit Testing

Combinational Unit Testing: 3-bit adder



Number of input bits ? 7

Total number of possible input values?

$2^7 = 128$

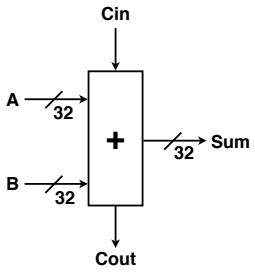
Just test them all ...

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?

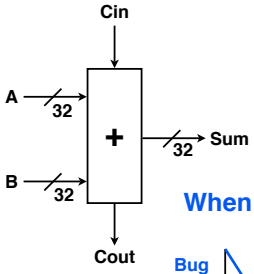
$2^{65} = 3.689e+19$

Just test them all?

Exhaustive testing does not "scale".

"Combinatorial explosion!"

Test Approach 1: Random Vectors

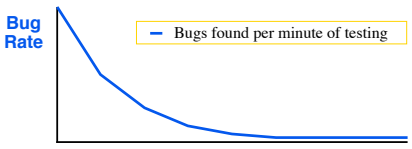


how it works

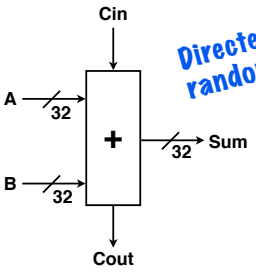
Apply random A, B, Cin to adder.

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

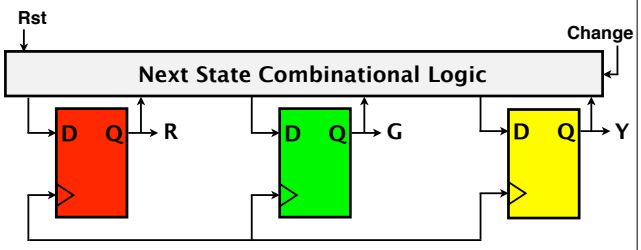
"Black-box": Corner cases based on functional properties.

Examples ?

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

Examples ?

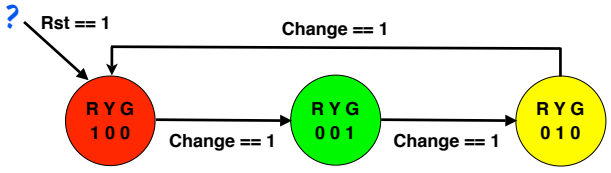
Testing State Machines: Break Feedback



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

Easier with certain Verilog coding styles?

Testing State Machines: Arc Coverage



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

Is this technique always practical to use?

Final Thought: When bugs "escape" ...

(Testing our financial trading system), we found a case where our software would get a bad calculation. Once a week or so.

Eventually, the problem turned out to be a failure in a CPU cache line refresh. This was a hardware design fault in the PC.

The test suite included running for two weeks at maximum update rate without error, so this bug was found.

Eric Ulevik

Conclusion -- Testing Processors

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