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

Top-down testing
- Assemble the complete processor.
- Execute test program suite on the processor.
- Check results.

Why is this method appealing?

Bottom-up testing
- What makes a good test program suite?
Methodical Approach: Unit Testing

Top-down testing
- how it works
  - Remove a block from the design.
  - Test it in isolation against specification.
  - What if the specification has a bug?

Bottom-up testing

Four Types of Testing (continued)

Climbing the Hierarchy: Multi-unit Testing

Top-down testing
- how it works
  - Remove connected blocks from design.
  - Test in isolation against specification.
  - How to choose partition? How to create specification?

Administrator - 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.

Processor Testing with Self-Checking Units

Top-down testing
- how it works
  - Add self-checking to units
  - Perform complete processor testing

Why not use self-checks for all tests?

Testing: Verification vs. Diagnostics

- 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)

Fill in the testing timeline: Answer:

epoch 1
processor complete
unit testing

ePOCH 2
multi-unit testing

ePOCH 3
self-checks
unit testing

ePOCH 4
top-down testing

Which testing types are good for each epoch?

- Epoch 1: processor complete unit testing
- Epoch 2: multi-unit testing
- Epoch 3: self-checks unit testing
- Epoch 4: top-down testing

Testing Mechanics: Asset Management

- adder.v: One directory holds all datapath units.
- 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?

Follow a file naming convention! Why?

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

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?
\[ 2^{65} = 3.689 \times 10^{19} \]
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.

Bug curve.
Bugs found per minute of testing

Time

Test Approach 2: Directed Vectors

how it works
Hand-craft test vectors to cover “corner cases”
A == B == Cin == 0

“Black-box”: Corner cases based on functional properties.
“Clear-box”: Corner cases based on unit internal structure.

Examples?

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.