Lab Lecture 5
Debugging
2/20/2004

Gabriel Eirea
Greg Gibeling
Today

- Debugging Techniques
- Simulation vs Hardware Debugging
- The Debugging Lab
- ChipScope Overview
Techniques

- Part I: Bottom-up Testing
- Part II: Designing Test Hardware
- Part III: Testing of FSMs
- Part IV: Probing Internal Signals
Simulation vs. Hardware Debug

- Debugging in Simulation:
  - Slow run time
  - Fast debug time (waveform, text)
  - Very high coverage (90-100%)
  - Easy to fix!

- Debugging in Hardware:
  - Fast run time
  - Slow debug time
  - No or little internal visibility
  - It might be too late to fix (ASICs)
Test Case Generation

- Complete Coverage: Exhaustive Testing
- FSM: Test all Transitions and States
- In practice: Goal is to get 95%+ Coverage
- Correct by Design: Only the Coverage Specified by Design
Part I: Bottom Level Testing (1)

```plaintext
one_bit_comp
```

```
g_i → g0 (a > b)
e_i → e0 (a = b)
a_j →
b_j →
```

What does it mean if \( g0 == 0 \) and \( e0 == 0 \)?
Part I: Bottom Level Testing (2)

- Ideal Testing: Exhaustive Testing
  - Every possible input combination has been tested. Can we do it here?
- Determine the truth table first
- Write a testbench that test every combination of the circuit
- This circuit is 100% tested!
Part I: Intermediate Testing (1)
Part I: Intermediate Testing (2)

- Can we do Exhaustive Testing Here?
- Two sets of four bit inputs:
  - 16 * 16 = 256 possible inputs/outputs
- Use a for or a while loop to go through all 256 combinations
  - Test benches are the ONLY PLACE YOU MAY USE FOR LOOPS
reg[3:0] ma[1:16]; // Create a memory array

initial begin

    // read the file specified and put the values in 'ma'
    $readmemh("data_vector", ma);
    for(i=1; i<=16; i= i + 1) begin

        // Remember to advance the time forward
        #(`half_cycle * 4)
        in = ma[i];
        $display("in = %d, peak = %d", in, peak);

    end

end
Part I: System Level Testing (2)

- Read a test vector from a file
- Make sure you get a good coverage with your test vectors
  - Designing test vectors isn’t easy
  - It’s a bit of a black art
- Read up on the $ statements in the Verilog reference (online)
Part II: ChipScope (1)

- Software based logic analyzer
  - Get results on the computer
- Put a logic analyzer right into the FPGA
  - ICON – Connects FPGA to software
  - ILA – Does the actual analysis
- More flexible than the bench analyzers
- But not as timing accurate
Part II: ChipScope (2)

- Steps to use ChipScope
  - Generate an ICON
  - Generate an ILA
  - Connect the ILA to the ICON in your design
  - Synthesize, and implement your design
  - Program the CaLinx board
  - Run the ChipScope Pro Analyzer
PartII: ChipScope (3)

- Similarities/Differences
  - Triggering is just like before
  - Data/Trigger can be MUCH bigger
  - Data is captured synchronously
  - Much easier to view waveforms
PartII: ChipScope (4)

- You will NEED ChipScope
- Plus maybe some additional hardware
  - You’re comparing a Counter and a BrokenCounter
  - ChipScope can’t do it all for you
  - You’ll need some extra hardware
- Detailed ChipScope Tutorial
  - [http://inst.eecs.berkeley.edu/~cs150/Documents.htm](http://inst.eecs.berkeley.edu/~cs150/Documents.htm)
PartIII: Test FSM (1)

- You must exercise every transition
- Very every state and output
- You do not need to correct this FSM
  - You just need to draw a correct bubble and arc diagram for your TA
- This is boring, but necessary
  - You will have bugs in your own FSMs
PartIII: Test FSM (2)
Part IV: Probe Internal Signals

- Simulate the circuit to make sure it works properly
- Set probe points on internal signals
- Download to the board
  - Use the bench analyzer
  - Or ChipScope
Part IV: Design Your Solution

- First, find the problem signal
- Second, think the problem out
- Third, design a solution
  - Perhaps consult your TA
  - You will need to really think this out
  - If you try and just build it, it probably won’t work quite right
- Fourth, implement your design