Logic Analyzers

EECS150 Spring 2006 – Lab Lecture #5

David Lin
Greg Gibeling

2/17/2006 EECS150 Lab Lecture #5 1

Today
- Lab #3 Solution
- Synplify Warnings
- Debugging Hardware
- Administrative Info
- Logic Analyzer
- ChipScope
- ChipScope Demo – Not on webcast!

Lab #3 Solution (1)
- Simple Solution
  - Use the standard 2 (or 3) block FSM format
  1. Always @ (posedge Clock) block that instantiates the register that contains state.
  2. Combinational logic block that responds to inputs and state changes by updating nextState wire and outputs.
  3. Optionally, block that updates outputs.

Lab #3 Solution (2)
- Cleaning Up Your Verilog FSM Code
  always @ (ps) begin
    case (ps)
      STATE_Init: begin
        Open = 1'b0;
        Prog1 = 1'b0;
        Prog2 = 1'b0;
        Error = 1'b0;
        if (Decode1 & Enter)
          ns = STATE_Ok1;
        else if (~Decode1 & Enter)
          ns = STATE_Bad1;
        end
      ...
      STATE_Ok2: begin
        Open = 1'b1;
        Prog1 = 1'b0;
        Prog2 = 1'b0;
        Error = 1'b0;
      ...

Lab #3 Solution (3)

```verilog
always @ (ps) begin
    Open = 1'b0;
    Prog1 = 1'b0;
    Prog2 = 1'b0;
    Error = 1'b0;
    case (ps)
        STATE_Init: begin
            if (Decode1 & Enter)
                ns = STATE_Ok1;
            else if (~Decode1 & Enter)
                ns = STATE_Bad1;
        end
        ...
    endcase
    ...
end
```

How about using assign statements for outputs?

Synplify Warnings (1)

- Why Bother?
  - “@W” in the Synthesis Report (Errors are “@E”)
  - Part of your project grade
    - Major warnings will cost points
  - Knowing these will make your life easier
    - Saves debugging
    - Always run synthesis before simulating in ModelSim!

Incomplete Sensitivity List

- ModelSim will use the sensitivity list
- Synplify pretty ignores it

Synplify Warnings (2)

```verilog
input [15:0] A, B;
output[31:0] Sum;
output COut;
// Adder
always @ (A or B)
    {COut, Sum} = A + B;

input Clock;
reg [31:0] Count;
// Counter
always @ (posedge Clock)
    Count <= Count + 1;
```

OK!

Synplify Warnings (3)

```verilog
input [1:0] select;
input A, B, C;
output Out;
reg Out;
// Mux
always @ (select or A or B or C) begin
    case (select)
        2'b00: Out = A;
        2'b01: Out = B;
        2'b10: Out = C;
        default: Out = 1'bx;
    endcase
end
```

Latch Generated

```verilog
input [1:0] select;
input A, B, C;
output Out;
reg Out;
// Mux
always @ (select or A or B or C) begin
    case (select)
        2'b00: Out = A;
        2'b01: Out = B;
        2'b10: Out = C;
        default: Out = 1'bx;
    endcase
end
```

Dk incomplete Sensitivity
Synplify Warnings (4)

- Combinational Loop
  - Must remove the loop or add a register
  - Multiple assignments to wire/\text{reg}
    - Nothing should be assigned to in more than one place!

Synplify Warnings (5)

- FPGA\_TOP2 always has warnings
  - Un-driven Input
  - Unconnected Output
  - These are truly unneeded pins
    - Things like the audio chips...
  - In your modules these are a problem
    - Synplify will optimize your design
    - Unconnected modules removed

Synplify Warnings (6)

- Why bother?
  - Getting rid of warnings saves debugging
  - Synplify warnings will result in lost points
- Warnings are the only syntax check
  - Verilog is a forgiving language
    - Undeclared variables default to 1 bit wires
  - This isn’t a good thing

Debugging Hardware (1)

- Debugging Algorithm
  - Hypothesis: What’s broken?
  - Control: Give it controlled test inputs
  - Expected Output: What SHOULD it do?
  - Observe: Did it work right?
  - If it broke: THAT’S GREAT!
    - If we can’t break anything like this then the project must be working...
Debugging Hardware (2)

- Using the logic analyzer / ChipScope
  - The most reliable tool you have
    - When used properly
  - Use the triggers effectively
    - Trigger on recurring sequences
    - Trigger on errors
    - An unstable display is useless
  - Compare synthesis to simulation
  - ChipScope is almost as good as simulation

Debugging Hardware (3)

- Before you change anything
  - Understand exactly what the problem is
  - Find an efficient solution
  - Evaluate alternative solutions
- After the change
  - Fixes may make things worse sometimes
    - May uncover a second bug
    - May be an incorrect fix
  - Repeat the debugging process

Administrative Info (1)

- Lab/Project Partners
  - If you don’t have a partner, stay after lab lecture and we’ll help you get partnered up.
- Remote access to Xilinx tools
  - Use Remote Desktop Connection to access kramnik.eecs.berkeley.edu.
  - A link to the kramnik set-up guide is on the documents page.
  - Also useful for transferring files to and from your U:\ drive.

Administrative Info (2)

- The grade book is online. Use your web account to access it.
- We will update the grade book roughly once a week.
- Hours spent on labs/checkpoints will not affect your grade.
Exhaustive FSM Testing
- Very similar to Part3 of Lab #4
- You’ll be mapping the whole FSM
  - No bubble-and-arc to start from
- No single step
  - Takes an input every cycle at 27MHz
  - Much too fast to see on the LEDs
  - Logic Analyzer!

Logic Analyzer
- HP54645D Mixed Signal Oscilloscope
  - Analog Oscilloscope
  - Digital Logic Analyzer
- Graphs Signals vs. Time
  - Like a timing diagram
- Invaluable for Debugging
  - This is your only tool for examining your clocks
  - Easy to see trends in signals...

Procedure
- Set up the Logic Analyzer
- Synthesize the design
- Write a test pattern and set SW10
- Press Single on the logic analyzer
- Press Reset to start the test
- Examine the waveforms
  - Build a bubble-and-arc diagram
The Logic Analyzer (1)

- Graphs Voltage vs Time
  - Takes real signals from a CUT
  - Can show both analog and digital signals
  - Great for signal quality, delay, timing

The Logic Analyzer (2)

- 16 Digital Inputs
  - Excellent Debugging Tool
  - Not very many input signals

The Logic Analyzer (3)

- Read the Lab.
- This is just a quick 10min demo.

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
  - Can create busses
  - Advanced triggering support
**ChipScope (2)**

- Steps to use ChipScope
  - Generate an ICON
  - Generate an ILA
  - Connect the ILA to the ICON
  - Synthesize, and implement your design
    - With the ILA and ICON
  - Program the CaLinx board
  - Run the ChipScope Pro Analyzer
    - Runs over the JTAG, not Slave Serial connection!

**ChipScope (3)**

- Logic Analyzer Similarities/Differences
  - Triggering is similar
    - Can be set to show waves before trigger
    - Can trigger on repeated or combined events
  - Data/Trigger can be MUCH bigger
    - Up to 256bits wide
    - As many samples as Block RAM on the FPGA
  - Data is captured synchronously
    - Can’t look at clocks
  - Much easier to view waveforms

**ChipScope (4)**

- ChipScope is useful to verify
  - In this lab we’re using it just to make absolutely sure
  - You will NEED ChipScope
    - You cannot debug a large design (i.e. your project) without it
    - Bench analyzers won’t show enough signals
  - It helps to master use of HP Logic Analyzer and ChipScope early on.
    - You want to have ready knowledge of both tools for when you’re working on the project.

**ChipScope (4)**

- Detailed ChipScope Tutorial
  - [http://inst.eecs.berkeley.edu/~cs150/sp06/Documents.php#Tutorials](http://inst.eecs.berkeley.edu/~cs150/sp06/Documents.php#Tutorials)
  - Get used to reading technical documents and tutorials.
    - It’s a useful and necessary job skill for engineers.
ChipScope Demo

- Aren't you glad you decided not to webcast today?