Project 2

System Synthesis and Modeling

Zack Fravel

010646947

zpfravel@uark.edu

Objective

The parent objective of Project 2 was to design a 7:1 multiplexer, that is seven inputs to one output, as well as get more familiar with the ModelSim tools and the design flow. After we design the mux we're also tasked to create a suitable testbench and analyze the simulation to verify our mux design.

Design and Simulation

The idea of the multiplexer is fairly straight forward, we'll have multiple inputs and one output with additional selector inputs that route the desired input to the output. Below is the circuit diagram showing the idea.

Mux Diagram

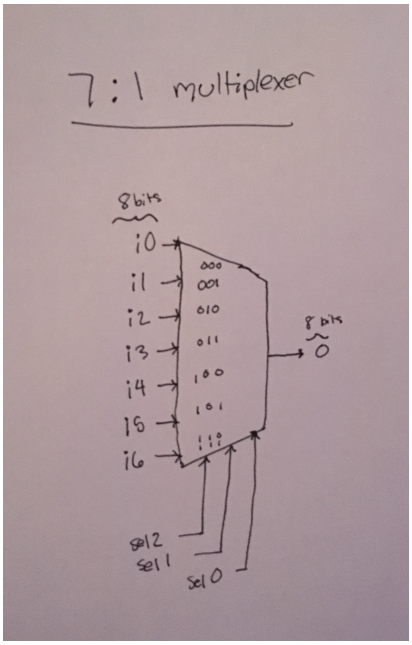

Figure 1

Basically you'll have seven inputs, and one output. In order to distinguish between all seven inputs we need three selector bits, which technically could account for up to eight inputs.

However, that just means we have a don't-care case and we'll just output the last input for both "110" and "111."

I have also attached the source verilog files to the end of this report showing how it is implemented in the ModelSim tools. Basically, all that is needed is to declare the necessary ports, and create an always@ block that is sensitive to all the inputs to the mux, especially the selectors. In this always@ block you have a case statement for sel2, sel1, sel0 in that order. Then below that you're able to set the declarations for each output (using the format 3'b000 to represent sel2=0, sel1=0, sel0=0). Once you've accounted for all the cases and assigned the respective inputs to the output at each case (using non-blocking assignment).

Finally, we need a way to test our design. I went with writing a testbench as my method of simulation. In my testbench I declare all the inputs and outputs as registers and instantiate the module I designed using the format [Project2 test (in0, in1, … sel2);]. After that, I created an initial block and after the begin statement I assign each of the seven outputs to a different 8 bit number (seven powers of 2). At the top of my testbench I use the compiler directive `timescale 1ns/1ns so that I can use #20 to spread my test cases 20 ns apart. Then below my initial input assignments I test each selector case 20 ns at at time.

Results, Analysis, and Conclusion

Once I got both the testbench and module files compiling I was able to start simulating. In the simulation I set the radix of the output to decimal so it's more obvious to see on the screenshot when the output changes what it's value is changing to. The testbench initializes all the inputs, waits 40 ns and then starts going through each selector case 20 ns at at time. The resulting output can be observed on the bottom line.

Simulation Waveform

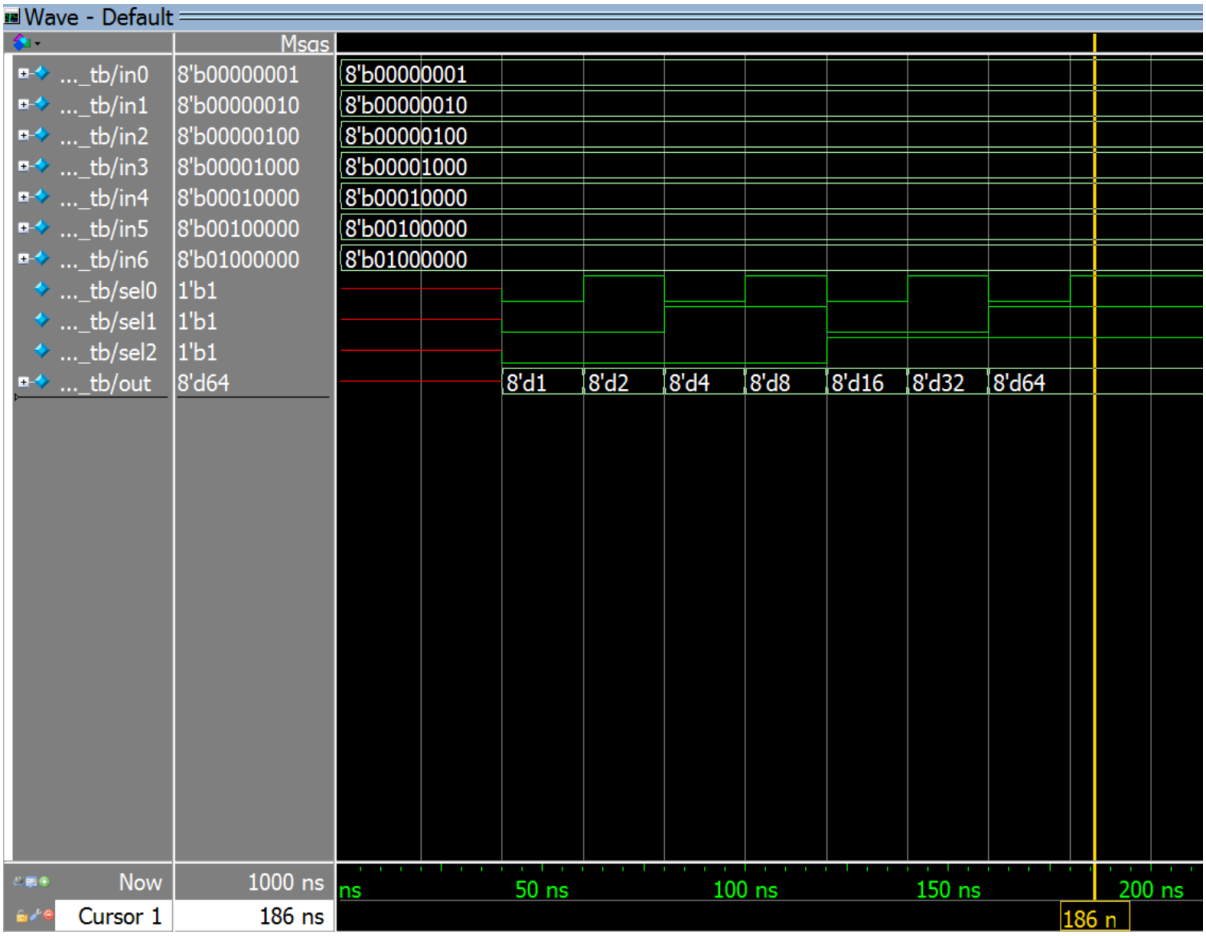

Figure 2

It can be seen above that the testbench goes through all eight combinations of selector bits within the 200 ns that is shown in the screenshot. Each time the selector changes the output also changes to it's respective input. Again, as stated earlier, the seventh input gets asserted to the output for both the 110 and 111 cases, since we only have seven inputs for eight cases.

With all that laid out, we should have all we need taken care of for Project 2. To reiterate, we successfully designed a 7:1 multiplexer with an associated testbench that tests all possible cases and showed its functionality with the waveform simuation. All source files mentioned are included.

Project2.v

```
// Zack Fravel
// System Synthesis and Modeling
// Project2.v

`timescale 1ns / 1ns
module Project2 (
// Port Declarations
                input[7:0] i0, i1, i2, i3, i4, i5, i6,
                output reg[7:0] o,
                input sel0, sel1, sel2
        );

// 7:1 Mux
always @(i0 or i1 or i2 or i3 or i4 or i5 or i6 or
         sel0 or sel1 or sel2)
begin
        case ({sel2, sel1, sel0})
                3'b000: o <= i0;
                3'b001: o <= i1;
                3'b010: o <= i2;
                3'b011: o <= i3;
                3'b100: o <= i4;
                3'b101: o <= i5;
                3'b110: o <= i6;
                3'b111: o <= i6;
        endcase
end

endmodule
```

Project2_tb.v

```
// Zack Fravel
// System Synthesis and Modeling
// Project2_tb.v

`timescale 1ns / 1ns
module Project2_tb;
// Declarations

   // Inputs
   reg[7:0] in0, in1, in2, in3, in4, in5, in6;

   // Select input bits
   reg sel0, sel1, sel2;

   // Output Wire
   wire[7:0] out;

   // Instantiate 7:1 mux
   Project2 test(in0, in1, in2, in3, in4, in5, in6, out, sel0, sel1, sel2);

// Testbench

// Set Inputs

initial // Test stimulus
  begin
        in0 = 8'b00000001;
        in1 = 8'b00000010;
        in2 = 8'b00000100;
        in3 = 8'b00001000;
        in4 = 8'b00010000;
        in5 = 8'b00100000;
        in6 = 8'b01000000;

    #20 // wait before testing

    #20 sel2 = 0; sel1 = 0; sel0 = 0;  // 000
    #20 sel2 = 0; sel1 = 0; sel0 = 1;  // 001
    #20 sel2 = 0; sel1 = 1; sel0 = 0;  // 010
    #20 sel2 = 0; sel1 = 1; sel0 = 1;  // 011
    #20 sel2 = 1; sel1 = 0; sel0 = 0;  // 100
    #20 sel2 = 1; sel1 = 0; sel0 = 1;  // 101
    #20 sel2 = 1; sel1 = 1; sel0 = 0;  // 110
    #20 sel2 = 1; sel1 = 1; sel0 = 1;  // 111

  end

endmodule
```