EECS 270 Winter 2025
Discussion #5: ModelSim Tutorial II and Bug Fixing Practice
By: Mick Gordinier

ModelSim Tutorials

1. **Displaying Letters on ModelSim (Changing / Creating new radices)**
    a. Have student create the project and add the DisplayLetters project file
    b. Update the text font of the DisplayLetters file on ModelSim
        i. Tools → Edit Preferences… → Fonts → textFont (to ~16)
    c. Have students walk through how to set up the test bench
        i. Reg, wire, instantiation, initial begin, all test cases
    d. Let students run on their own local ModelSim waveform
    e. Update the wave font of the waveform
        i. Tools → Edit Preferences… → Fonts → waveFont (to ~16)
    f. Show how you can change to common radices (decimal, binary, hex)
    g. Save the project macro as 'DisplayLettersWave.do' file
    h. Close the simulation and open the .mpf and load the .do file
    i. In 'DisplayLettersWave.do', give code to copy for letter display
        i. radix define DisplayStates {
            7'b0001000 "DISP_A",
            7'b0000011 "DISP_B",
            7'b1000110 "DISP_C",
            7'b0100001 "DISP_D",
            -default binary
           }
    j. Let students change to new radix display
    k. Lets students create a new radix SelStates
        i. Should create 4 states, one for each SEL opcode option
        ii. SEL_A, SEL_B, SEL_C, SEL_D

```
`timescale 1ns/1ns

module DisplayLetters(
  input [1:0] sel,
  output [6:0] letterDisp
);

  localparam A = 7'b0001000;
  localparam B = 7'b0000011;
  localparam C = 7'b1000110;
  localparam D = 7'b0100001;

  assign letterDisp =
          (sel == 2'd0) ? A :
          (sel == 2'd1) ? B :
          (sel == 2'd2) ? C : D;
```

```
endmodule

module LetterDisplayTestbench;
  // Make the testbench
endmodule
```

2. **Buggy Code #1: Magnitude Comparator**
    - Code provided contains a few different types of bugs that students need to find
    - Bugs in code
        - If students try to compile on ModelSim, there will be a compiler error
            - **In MagnitudeComp declaration of ports, there is an extra comma after the output A_eq_B**
        - If student try to compile again after changing, there will be 2 additional compiler errors alerting that in MagnitudeTestbench
            - **There is a comma after the 'wire' keyword**
            - **There is a semicolon missing after the re-assigning of 'A' in Test #3**
        - Once fixed, there will be no more compiler errors. BUT, once the students attempt to simulate from the ModelSim library, there will be an error load design
            - **There needs to be an instantiation name for MagnitudeComp, like MagnitudeComp mc(A, B, A_gt_B, A_lt_B, A_eq_B);**
        - NOW, there are no more compiler or loading errors, but there is now an incorrect implementation.
            - **In MagnitudeTestbench, missing A_eq_B as output parameter. This won't result in a compiler error. You don't need to specify all the outputs.**
            - **A_gt_B and A_lt_B are swapped in MagnitudeComp. When doing testing student should see this when looking at the waveform**
            - **BONUS: Test #5 is set up incorrectly. A and B are 3-bits. The range of unsigned values are between 0-7. We set A to 8 = 4'b1000. Thus, ModelSim will ignore the most significant bit and make A = 0 → Test will result in A == B instead of A > B**

```
`timescale 1ns/1ns

// Comapring between 2 unsigned binary numbers
// Outputs boolean A > B, A < B, A == B
module MagnitudeComp (
  input [2:0] A, B,
  output A_gt_B, A_lt_B, A_eq_B,
);

  assign A_gt_B = (A < B);
  assign A_lt_B = (A > B);
  assign A_eq_B = (A == B);

endmodule


module MagnitudeTestbench;
  reg [2:0] A, B;
  wire, A_gt_B, A_lt_B, A_eq_B;
```

```
  MagnitudeComp(A, B, A_gt_B, A_lt_B);

  // Cannot test every single case
  // Need to test specific cases
  initial begin

    // Test #1: A == B and both are zero
    A = 3'd0;
    B = 3'd0;
    #5;

    // Test #2: A == B and both are non-zero
    A = 3'd4;
    B = 3'd4;
    #5;

    // Test #3: A > B
    A = 3'd6
    B = 3'd2;
    #5;

    // Test #4: A < B
    A = 3'd1;
    B = 3'd7;
    #5;

    // Test #5: A > B
    A = 3'd8;
    B = 3'd0;

  end
endmodule
```

3. **Buggy Code #2: Selector**
    - Code provided contains a few different types of bugs that students need to find
    - Bugs in code
    - Compiler errors
        - **Commas missing in port list in Project1**
        - **Semicolons used instead of commas in Selector port list**
    - Once compiler errors taken care of
        - **In SelectorTestbench, we are not properly passing all of the parameters. KEY is not being passed as an input. It will cause the ModelSim wave to look very strange to due a ton of port mismatching**
        - **In Project1 Set_A, Set_B, Out_F are declared as 1-bit instead of multi-bit**
        - **In Selector, it is driving to wire 'out_F' instead of 'Out_F'. Compilers do not give errors if we drive undeclared wires. Doing assigns will implicitly declare the wire**
        - **In Selector, we are doing a LOGICAL OR instead of a BITWISE OR, causing the output to be only one-bit instead of 7-bits**
        - **In Project1, we are doing a LOGICAL NOT instead of a BITWISE NOT, cause the output to be only one-bit instead of 7-bit**

```
`timescale 1ns/1ns

// File Name: Project1.v
module Project1(
        input KEY  // KEY = 0 --> SW[6:0], KEY = 1 --> SW[16:10]
        input [17:0] SW
        output [6:0] LEDR
        output [6:0] HEX0
);

        wire Set_A, Set_B, Out_F;

  assign Set_A = SW[16:10];
  assign Set_B = SW[6:0];

        Selector s(KEY, Set_A, Set_B, Out_F);

  assign LEDR = Out_F;
  assign HEX0 = !Out_F;

endmodule // Project1


// If sel == 0 --> Out_F = Set_B
// If sel == 1 --> Out_F = Set_A
module Selector(
  input sel;
  input [6:0] Set_A;
```

```
  input [6:0] Set_B;
  output [6:0] Out_F;
);

  /* DON'T CHANGE BEGIN */

  wire [6:0] sel_and_A, n_sel_and_B;

  // Extending sel to be 7-bit with the same values
  // Performing Bitwise AND operations across all bits
  assign sel_and_A = {7{sel}} & Set_A;
  assign n_sel_and_B = {7{~sel}} & Set_B;

  /* DON'T CHANGE END */

  // Performing the OR of the selector
  assign out_F = sel_and_A || n_sel_and_B;

endmodule


module SelectorTestbench;
  reg KEY;
  reg [17:0] SW;
  wire [6:0] LEDR, HEX0;

  Project1 p1(SW, LEDR, HEX0);

  /* DON'T CHANGE BEGIN */

  initial begin

    KEY = 1'b0;
    SW = 18'd0;
    #5;

    KEY = 1'b0;
    SW[6:0] = 7'd35;
    SW[16:10] = 7'd5;
    #5;

    KEY = 1'b0;
    SW[6:0] = 7'd21;
    SW[16:10] = 7'd121;
    #5;

    KEY = 1'b1;
    #5;

    SW[6:0] = 7'd37;
    SW[16:10] = 7'd5;

  end
```

```
  /* DON'T CHANGE END */

endmodule
```