ECE 3544: Digital Design 1
Homework Assignment 3 (80 points)

Name: **Jonathan Lemarroy**

For each design that you represent in a Verilog module, copy your source code into the document. For each design that you simulate in ModelSim, include waveforms displaying the correct operation of each module

In addition, submit the .v file containing each design and each test bench that you write. Your files should contain header information as described in Project 1 and be neatly formatted and commented. Submit your files separately on Canvas. Do not put them into an archive. *Make sure that you upload all of your files.*

Problem 1 (10 points)

A **two-out-of-five code** is a five-bit binary code having exactly two ones. Since there are ten two-out-of-five codes, the encoding scheme can be used as a means of representing decimal digits.

Write a Verilog **structural** model for a circuit that determines whether or not a five-bit input `code` is a valid two-out-of-five code. The output `valid` should equal 1 if `code` is valid. It should equal 0 otherwise. Use the module declaration below as a starting point for your model. You may add wires as needed.

```
// Filename:    problem1.v
// Author:      Jonathan Lemarroy
// Date:        06 October 2020
// Version:     1
// Description: I was not able to implement this using only 10 gates. This was written
//              just so the other problems will still compile.

module problem1_jdl25175(code, valid);
    input  [4:0] code;
    output       valid;
    wire   [2:0] count;

    assign count = code[4] + code[3] + code[2] + code[1] + code[0];

    assign valid = (count == 3'b010) ? 1'b1 : 1'b0;

endmodule
```

This is important!

Use **only Verilog's built-in gate primitives** to represent the operators that make up your logic circuit. Imagine that gates come at a premium in this problem; your model should use no more than a total of ten logic gates. You may use logic gates of any type, and you may use logic gates having any number of inputs. Perform a multi-level transformation to reduce the gate count of the circuit. Comment on how your implementation compares to a two-level circuit that implements the same function.

Helpful Hints for Writing Test Benches as Procedures

The Verilog code shown below represents one way to write a **test bench**. *Here are some very important teaching points:*

- The circuit represented by `problem1` is the one that we want to test.
- If a variable is the target of a procedure – that is, if the procedure changes the value of the variable – *then the variable must be typed as a `reg`.* Therefore, the input ports of the design under test generally have `reg` type in the test bench, since the procedure targets a value to generate the input cases.
- In a test bench, the output ports of the design under test generally have `wire` type, since the output value is driven by the circuit as controlled by the inputs.
- The FOR-loop is contained in a procedural block –an `initial` block in this particular example. Structures such as IF-statements, CASE statements, FOR-loops and WHILE-loops *must be contained within procedural blocks* such as `initial` blocks and `always` blocks. *These structures cannot appear outside of procedural blocks!*
- Procedures represented by `initial` blocks are *not generally synthesizable*, but since a test bench is not meant to be synthesizable, the `initial` block provides us with the means to sequence an input test set starting at simulation time t = 0.
- `#20` represents a time delay of 20 time steps (usually nanoseconds). The presence of delays prevents the procedure from executing in zero-time. Delays are not synthesizable, but are appropriate for use in test benches for simulation purposes.

```
// Time Unit = 1 ns (#1 means 1 ns)
// Simulation Precision = 1 ns
`timescale 1ns/1ns

// Filename:    tb_problem1_for.v
// Author:      Jason Thweatt
// Date:        04 June 2017
// Version:     1
// Description: This file contains a test bench for a 5 input circuit.
//              It uses a FOR-loop in an initial block to apply all
//              combinations of the inputs for a period of 20 ns each.

module tb_problem1_for();
   reg  [4:0] ins;
   wire       out;
   reg  [4:0] count;

   problem1 dut(ins, out);    // Instantiate the design being tested.

   initial begin
      for(count = 0; count < 32; count = count + 1) begin
         ins = count;
         #20;
      end
   end

endmodule
```

Problem 2 (5 points)
Explain why the test bench described above will execute forever. Write a working alternative that uses a FOR-loop.

(If you aren't wondering why `count` was declared separately from `ins` and then assigned to `ins` in the procedure, then maybe you should be. The solution is not as simple as that, but it is a factor that should enter into your thinking.)

Answer: Because the count reg only consists of 5 bits, so it can only count to 31 before the carry bit is lost and its value would hold 00000 again making it start over and never reaching a condition that would be less than 32.

```
// Time Unit = 1 ns (#1 means 1 ns)
// Simulation Precision = 1 ns
`timescale 1ns/1ns

// Filename:    problem2.v
// Author:      Jonathan Lemarroy
// Date:        06 October 2020
// Version:     1
// Description: This file contains a test bench for a 5 input circuit.
//              It uses a FOR-loop in an initial block to apply all
//              combinations of the inputs for a period of 20 ns each.

module tb_problem1_for();
    reg  [4:0] ins;
    wire       out;
    reg  [5:0] count;

    problem1_jdl25175 dut(ins, out);    // Instantiate the design being tested.

    initial begin
        for(count = 6'd0; count < 6'd32; count = count + 6'd1) begin
            ins = count;
            #20;
        end
    end

endmodule
```

Problem 3 (5 points)
Bearing in mind the deficiency contained in the test bench as it was given to you in Problem 2, rewrite the test bench of Problem 2 as a working WHILE-loop.

Answer:

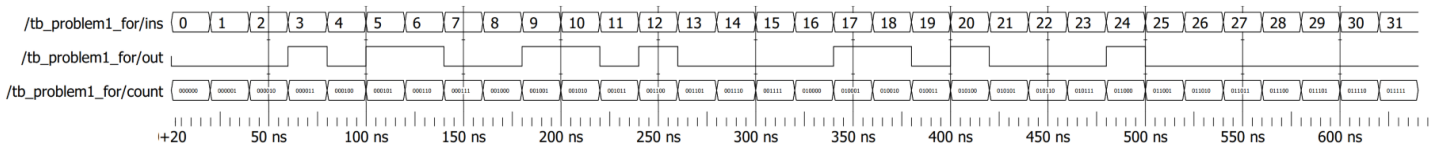
```
// Time Unit = 1 ns (#1 means 1 ns)
// Simulation Precision = 1 ns
`timescale 1ns/1ns

// Filename:    problem3.v
// Author:      Jonathan Lemarroy
// Date:        06 October 2020
// Version:     1
// Description: This file contains a test bench for a 5 input circuit.
//              It uses a FOR-loop in an initial block to apply all
//              combinations of the inputs for a period of 20 ns each.

module tb_problem1_while();
    reg  [4:0] ins;
    wire       out;
    reg  [5:0] count;

    problem1_jdl25175 dut(ins, out);    // Instantiate the design being tested.

    initial begin
        count = 6'd0
        while(count < 6'd32) begin
            ins = count;
            #20;
            count = count + 6'd1;
        end
   end

endmodule
```

Problem 4 (10 points)

Use ModelSim to simulate the operation of the circuit that you modeled in Problem 1. You may use your corrected test bench from Problem 2, or the test-bench you wrote in Problem 3.

Follow the instructions provided for creating a project in ModelSim and performing the simulation.

Waveform:



Problem 5 (10 points)

Write a Verilog **behavioral** model for the system of Problem 1. Your model may use continuous assignment and dataflow operators, or a procedural model using an `always` block. However, you should make your model as simple as possible. *You need not follow the gate restriction of Problem 1, but you should also take advantage of the functionality of dataflow operators to implement your model differently.* Write your model for efficiency and synthesizability.

Use the module declaration below as a starting point for your model. You may add wires as needed. You may assign the `reg` type to an entity to make it into a variable separately from declaring the entity as an `output`. Remember that any value targeted by a procedure must be typed as a `reg`.

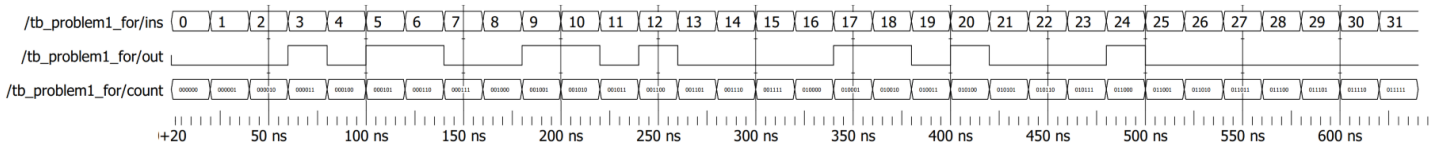
Answer:

```
// Filename:    problem5.v
// Author:      Jonathan Lemarroy
// Date:        06 October 2020
// Version:     1
// Description: This circuit returns valid if exactly 2 bits out of the 5 bit code are 1s.
//              This is the continuous assignment implementation

module problem5_jdl25175(code, valid);
    input  [4:0] code;
    output       valid;
    wire   [2:0] count;

    assign count = code[4] + code[3] + code[2] + code[1] + code[0];

    assign valid = (count == 3'b010) ? 1'b1 : 1'b0;

endmodule
```

Use ModelSim to simulate the operation of the circuit that you modeled in this problem. You should be able to use a test bench similar in structure to the one that you used to simulate your design in Problem 1. *Of course, you will have to change the name of the design under test to test your design in this problem.*

Follow the instructions provided for creating a project in ModelSim and performing the simulation.

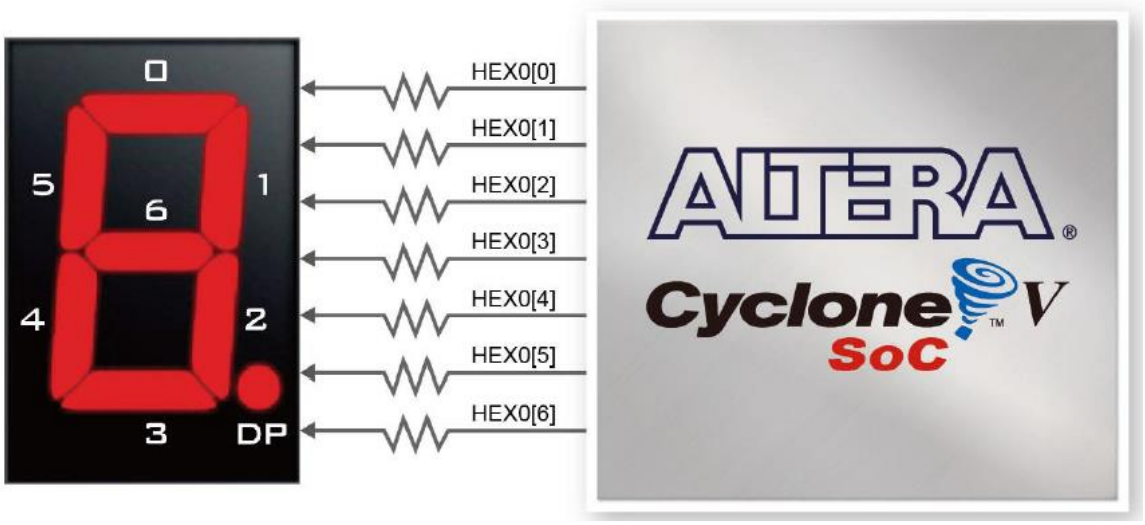
Waveform:



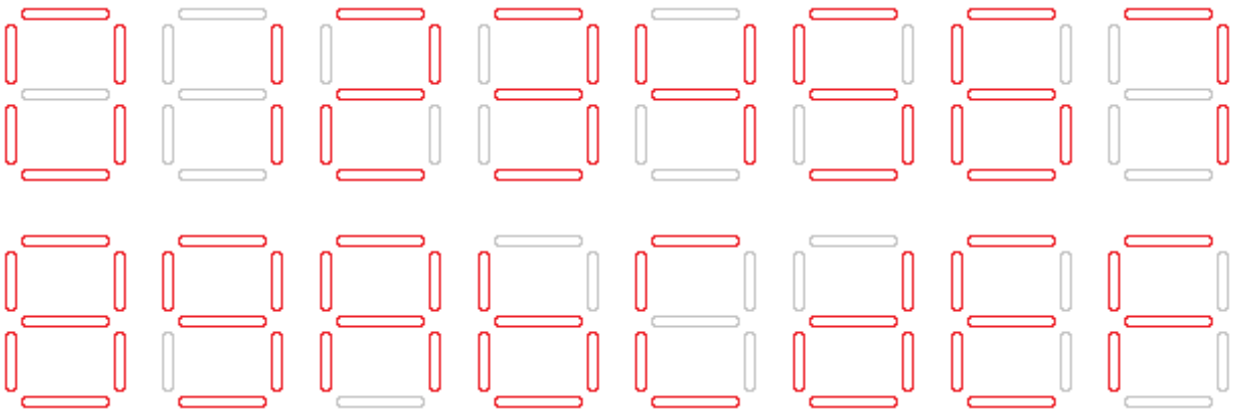
Helpful Hints for Writing Seven-Segment Display Drivers

A seven-segment display is often used to display characters. For the seven-segment displays on your DE1-SoC board, *each of the segment outputs is ON when logic-0 is applied to it and is OFF when logic-1 is applied to it.*

The diagram below shows the interface between our FPGA and one of the seven segment displays that the board is capable of driving. The segments are labeled 0 to 6. (Ignore the decimal point DP.) The labels indicate the position that each driver value has in a seven-bit vector.



If we wanted the display to show characters that correspond to the hexadecimal digits, they might appear as follows:



Problem 6 (20 points)

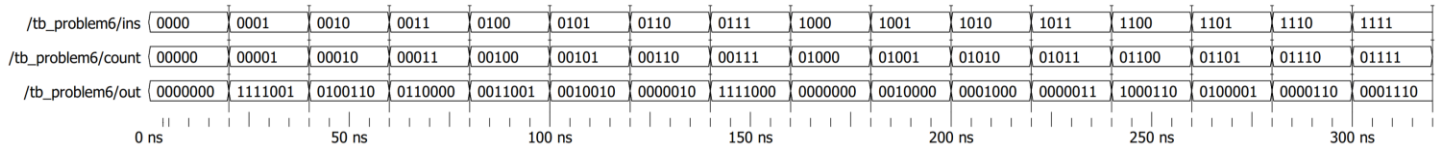
Write a Verilog model for a circuit that accepts a four-bit input representing an unsigned binary value and produces an output that drives the seven-segment display to show the corresponding hexadecimal digit. In this problem, use **continuous assignments with dataflow operators**. Make sure that the order of the signals in the vector `hex_display` corresponds to the order of the driver values shown in the diagram above.

Answer:

```
// Filename:    problem6.v
// Author:      Jonathan Lemarroy
// Date:        06 October 2020
// Version:     1
// Description: This circuit maps a hexidecimal input to a seven-segment display.
//              This is the continuous assignment implementation.

module sevensegdecoder_cont_jdl25175(hex_digit, hex_display);
    input  [3:0] hex_digit;
    output [6:0] hex_display;

    assign hex_display = (hex_digit == 4'h0) ? 7'b0000000 :
                         (hex_digit == 4'h1) ? 7'b1111001 :
                         (hex_digit == 4'h2) ? 7'b0100110 :
                         (hex_digit == 4'h3) ? 7'b0110000 :
                         (hex_digit == 4'h4) ? 7'b0011001 :
                         (hex_digit == 4'h5) ? 7'b0010010 :
                         (hex_digit == 4'h6) ? 7'b0000010 :
                         (hex_digit == 4'h7) ? 7'b1111000 :
                         (hex_digit == 4'h8) ? 7'b0000000 :
                         (hex_digit == 4'h9) ? 7'b0010000 :
                         (hex_digit == 4'hA) ? 7'b0001000 :
                         (hex_digit == 4'hB) ? 7'b0000011 :
                         (hex_digit == 4'hC) ? 7'b1000110 :
                         (hex_digit == 4'hD) ? 7'b0100001 :
                         (hex_digit == 4'hE) ? 7'b0000110 :
                         (hex_digit == 4'hF) ? 7'b0001110 : 7'b1111111;

endmodule
```

Use ModelSim to simulate the operation of the circuit that you modeled in this problem. You should be able to use the same test bench for this circuit and the one that you implement in Problem 7 – in fact, you should be able to test both designs in the same test bench at the same time, if you are careful with the naming conventions that you use for the ports of both circuits.

Waveform:



Problem 7 (20 points)

Repeat Problem 6 using a **procedural model**. Employ an **`always`** block that contains **`IF`**-statements or a **`case`** statement. Make sure that the order of the signals in the vector `hex_display` corresponds to the order of the driver values shown in the diagram. Make your model as simple as possible. Write your model for efficiency and synthesizability.

Use the following module declaration. You may add wires as needed. You may assign the `reg` type to an entity to make it into a variable separately from declaring the entity as an `output`. Remember that any value targeted by a procedure must be typed as a `reg`.

Answer:

```
// Filename:    problem7.v
// Author:      Jonathan Lemarroy
// Date:        06 October 2020
// Version:     1
// Description: This circuit maps a hexidecimal input to a seven-segment display.
//              This is the procedural implementation.

module sevensegdecoder_proc_jdl25175(hex_digit, hex_display);
    input  [3:0] hex_digit;
    output [6:0] hex_display;
    reg    [6:0] temp;

    always @(hex_digit) begin
        case(hex_digit)
            4'h0    : temp <= 7'b0000000;
            4'h1    : temp <= 7'b1111001;
            4'h2    : temp <= 7'b0100110;
            4'h3    : temp <= 7'b0110000;
            4'h4    : temp <= 7'b0011001;
            4'h5    : temp <= 7'b0010010;
            4'h6    : temp <= 7'b0000010;
            4'h7    : temp <= 7'b1111000;
            4'h8    : temp <= 7'b0000000;
            4'h9    : temp <= 7'b0010000;
            4'hA    : temp <= 7'b0001000;
            4'hB    : temp <= 7'b0000011;
            4'hC    : temp <= 7'b1000110;
            4'hD    : temp <= 7'b0100001;
            4'hE    : temp <= 7'b0000110;
            4'hF    : temp <= 7'b0001110;
            default : temp <= 7'b1111111;
        endcase
    end

    assign hex_display = temp;

endmodule
```

Use ModelSim to simulate the operation of the circuit that you modeled in this problem. You should be able to use the same test bench for this circuit and the one that you implemented in Problem 6 – in fact, you should be able to test both designs in the same test bench at the same time, if you are careful with the naming conventions that you use for the ports of both circuits.

Waveform: