**HW1 Modeling Concepts**

Verilog HDL modeling language supports three kinds of modeling styles: gate-level, dataflow, and behavioral. The gate-level and dataflow modeling are used to model combinatorial circuits whereas the behavioral modeling is used for both combinatorial and sequential circuits. This lab illustrates the use of all three types of modeling by creating simple combinatorial circuits using ISE or Vivado software tool.

1. Gate-level Modeling

The gate-level modeling is useful when a circuit is a simple combinational, as an example a multiplexer. Multiplexer is a simple circuit which connects one of many inputs to an output. In this part, you will create a simple 2-to-1 multiplexer and extend the design to multiple bits.

a) Create a 2-to-1 multiplexer using gate-level modeling. Specifically, create and add the Verilog module with three inputs (x, y, s) and one output (m) using gate-level modeling.

![Multiplexer schematic](image)

b) Create a two-bit wide 2-to-1 multiplexer using gate-level modeling.

2. Dataflow Modeling

Dataflow modeling style is mainly used to describe combinational circuits. The basic mechanism used is the continuous assignment. In a continuous assignment, a value is assigned to a data type called net. The syntax of a continuous assignment is

```
assign [delay] LHS_net = RHS_expression;
```

Where LHS_net is a destination net of one or more bit, and RHS_expression is an expression consisting of various operators. The statement is evaluated at any time any of the source operand value changes and the result is assigned to the destination net after the delay unit. The gate level modeling examples listed in Part 1 can be described in dataflow modeling using the continuous assignment. For example,

```
assign out1 = in1 & in2; // perform and function on in1 and in2 and assign the result to out1
assign out2 = not in1;
assign #2 z[0] = ~(ABAR & BBAR & EN); // perform the desired function and assign the result after 2 units
```
The target in the continuous assignment expression can be one of the following:

1. A scalar net (e.g. 1st and 2nd examples above)
2. Vector net
3. Constant bit-select of a vector (e.g. 3rd example above)
4. Constant part-select of a vector
5. Concatenation of any of the above

Let us take another set of examples in which a scalar and vector nets are declared and used

```verilog
count, cin; // scalar net declaration
wire [3:0] sum, a, b; // vector nets declaration
assign {cout, sum} = a + b + cin; // A and B vectors are added with CIN and the result is assigned to a concatenated vector of a scalar and vector nets
```

Note that multiple continuous assignment statements are not allowed on the same destination net.

a) Model a two-bit wide 2-to-1 multiplexer using dataflow modeling with net delays of 3 ns. Specifically, create and add the Verilog module with two 2-bit inputs (x[1:0], y[1:0]), a one bit select input (s), and two-bit output (m[1:0]) using dataflow modeling. Each assignment statement should have 3 units delay. As an example, a one-bit 2-to-1 multiplexer can be described as follows:

```verilog
assign #3 m = (~s & x) | (s & y); // 3 units delay
```

b) Create a testbench and Simulate the design for 100 ns and show the output.

3. Behavioral Modeling

Behavioral modeling is used to describe complex circuits. It is primarily used to model sequential circuits, but can also be used to model pure combinatorial circuits. The mechanisms (statements) for modeling the behavior of a design are:

initial Statements

always Statements

A module may contain an arbitrary number of initial or always statements and may contain one or more procedural statements within them. They are executed concurrently (i.e. to model parallelism such that the order in which statements appear in the model does not matter) with respect to each other whereas the procedural statements are executed sequentially (i.e. the order in which they appear does matter). Both initial and always statements are executed at time=0 and then only always statements are executed during the rest of the time. The syntax is as follows:

```verilog
initial [timing_control] procedural_statements;
always [timing_control] procedural_statements;
```
where a procedural_statement is one of:

procedural assignment
conditional_statement
case_statement
loop_statement
wait_statement

The initial statement is non-synthesizable and is normally used in testbenches. The always statement is synthesizable, and the resulting circuit can be a combinatorial or sequential circuit. In order for the model to generate a combinatorial circuit, the always block (i) should not be edge sensitive, (ii) every branch of the conditional statement should define all output, and (iii) every case of case statement should define all output and must have a default case. More detailed coverage of this topic is covered in Lab 7. The destination (LHS) should be of reg type; either scalar or vector. For example,

```
reg m; // scalar reg type
reg [7:0] switches; // vector reg type
```

a) Create a 2-to-1 multiplexer using behavioral modeling. Specifically, Create and add the Verilog module with three inputs (x, y, s) and one output (m) using behavioral modeling. Use the example code given below:

```
reg m;
always @ (x or y or s)
begin
  if(s==0)
    m=y;
  else
    m=x;
end
```

b) Create a two-bit wide 2-to-1 multiplexer using behavioral modeling. Specifically, create and add the Verilog module with two-bit input (x[1:0], y[1:0]), a one bit select input (s), and two-bit output (m[1:0]) using behavioral modeling.

4. Mixed-design Style Modeling

Complex systems can be described in Verilog HDL using mixed-design style modeling. This modeling style supports hierarchical description.

As an example of a mixed-style modeling, following diagram shows how one can build a 3-to1 multiplexer using multiple instances of 2-to-1 multiplexer. It also shows the symbol and the truth table.
In the above diagram, \( u, v, w \) are data inputs whereas \( S0, S1 \) are select signals, and the output is \( m \). It uses two instances of 2-to-1 multiplexer.

a) Model a 3-to-1 multiplexer using 2-to-1 multiplexers. Specifically, create a top-level Verilog module with three data inputs \( (u, y, w) \), two select inputs \( (s0, s1) \), and one bit output \( (m) \) using the previously defined 2-to-1 multiplexer. You can use any style designed 2-to-1 multiplexer (1, 2, or 3). Wire them up as shown in the above diagram.

5. Model a BCD to 7-Segment Decoder.

A 7-segment display consists of seven segments, numbered \( a \) to \( g \) which can be used to display a character. Depending on the input type, a type conversion may be needed. If want to display a binary coded decimal (BCD) using 4-bit input, a BCD to 7-segment decoder is required. The table below shows the bit pattern you need to put to display a digit (note that to turn ON a segment you need to put logic 0 on the segment and the anode of the display needs to be driven logic 0 on this board).

<table>
<thead>
<tr>
<th>Input</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010 to 1111</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Where \( x \) is don't care.
a) Create a top-level Verilog module, named bcdto7segment_dataflow with 4-bit data input (x[3:0]), anode enable output signals (an[3:0]), and 7-bit output (seg[6:0]) using dataflow modeling (Hint: You will have to derive seven expressions for the 7 segments on paper). Assign appropriate logic to an[3:0] in the model so you can display only on the right most display.