CME341 Assignment 3

This assignment consists of 9 questions, of which the first 7 MAY be considered for marking purposes. Questions 8 and 9 are optional, but recommended.

1. This purpose of this question is to demonstrate some of the capabilities of the simulator. Make a new project called "simulator_exercise" for the following verilog prototype.

```
module simulator_exercise (
    input [7:0] x,
    output reg [7:0] y );

always @ *
    y=x;
endmodule
```

   (a) Compile the project.

   (b) Write a testbench. Generate a test signal for `x` that counts in binary from 0 to 255. It is to increment every 100 ns. This can be done as shown below:

```
‘timescale 1 ns/ 1 ps
module simulator_exercise_testbench ();
reg [7:0] x;
wire [7:0] y;
                initial #100000 $stop;
initial
   x = 8’h00;
always
   #100 x = x + 8’h01;
simulator_exercise cct_1(.x(x), .y(y));
endmodule
```

(c) Load the simulation and add `x` and `y` to the wave window. **In the wave window**, right click on the blue diamond that marks the signal `x`. It will be called `/simulator_exercise_testbench/x`. This will pop up a menu. Select **Radix** $\longrightarrow$ **Hexadecimal**.

(d) Run the simulation and observe the screen. The signal `x` should count up in hexadecimal. Change the radix to binary. The display should instantly change to display `x in binary`.

(e) Change the format from "literal" to "analog" by right clicking on the signal `/simulator_exercise_testbench/x` and selecting `Format` $\longrightarrow$ `Analog(automatic)`.

(f) Change the units on the time axis in the wave window by right clicking just below the axis and selecting **Grid and timeline properties ....** Then change the units in the time units box to `ns`.

(g) Change the format back to "literal" and the radix back to "Hexadecimal" and then make a wave.do file (see chapter 1 notes for a refresher).

(h) Reload the simulation i.e. select **Simulate** $\longrightarrow$ **Start Simulation**. Run "do wave.do" in the transcript window. Verify all the settings, including the radix setting, has been restored to the wave window.

2. Generate a schematic diagram for each of the circuits described by the Verilog HDL modules below. As a reminder, the sensitivity list in an "always" statement need not be specified explicitly. A wild card, which is a star, i.e. "*", can be used. Thus the always statement "always @ (a or b) ", which appears in module circuit_1 below, could be equivalently replaced with "always @ *".

(a)
```
module circuit_1 ( a, b, c);
input [7:0] a,b;
output [5:0] c;

reg [5:0] c;

always @ (a or b)
begin
```

```
if (a[7:5] == 3'b111)
c = b[5:0];
else if (a[7:6] == 2'b11)
c = b[6:1];
else  c = b[7:2];
end
endmodule
```

(b)
```
module circuit_2 ( a, b, c);
input [1:0] a,b;
output  c;

reg c;

always @ (a or b)
begin
if ( b == (2'b10 | 2'b01)  )
c = &(b ^ a);
else
begin c = c ^ ~&a; end
end
endmodule
```

(c)
```
module circuit_3 ( gate,d,q);
input gate,d;
output  q;

reg q;

always @ (gate or d)
begin
if ( gate == 1'b1 )
q = d;
else  q = q;

end
endmodule
```

3. Suppose circuit_1 was compiled by Quartus. There would be unused input pins in the circuit produced by the compiler. Which input pins

are unused? Remember `a` is an 8-pin input.

4. The Verilog HDL in question 2c above (`module circuit_3`) produces a circuit that, depending on the input, can have positive feedback. For what values of the inputs does the circuit have positive feedback?

   NOTE: Digital circuits with positive feedback are marginally stable. They are stable if the output is a constant 1'b1 or constant 1'b0, but it is possible for a transient (narrow pulse) to make them oscillate.

5. Write a behavioral Verilog HDL description of a circuit that detects whether or not an 8 bit vector has value 8'd211. Compile your Verilog HDL description and test the circuit generated by the Quartus compiler using the Modelsim-Altera simulator.

6. Sometimes typographical errors in Verilog programs can cause bugs in the circuit. In large programs the presence of these bugs can go undetected for a long time and then when they are discovered they are hard to find. This question shows the cause and effect of a few typographical errors that have plagued some CME341 students of previous years.

   Some typographical errors cause the compiler to produce an error, some cause the compiler to produce a warning and some slip under the compiler's radar. The Verilog HDL descriptions below have caused problems in the past. Enter the descriptions, compile them and simulate them to observe the effect. Describe the cause and effect i.e. if there is a warning message, what is it, what is the problem with the circuit, etc.

   (a) Incorrectly specifying a number, i.e. writing 3'b1100 instead of 4'b1100, can cause problems. Explain why the output the way it is and what the compiler warning message is saying. Your explanation may be (if this is what is really happening): " The typo has x being compared to a vector that has been declared to be 3 bits. The compiler believes the declaration to be true, ignores the most significant bit specified and uses the 3 least significant bits as the 3 bit number. Since x is a 4 bit vector, the compiler extends the 3 bit vector to which x is being compared, to a 4 bit vector by placing a 0 in the most significant bit. "

```
module number_typo (x , y);
  input [3:0] x;
  output y;
  reg y;

  always @ x
  if (x == 3'b1100) // should be 4'b1100
    y = 1;
  else
    y = 0;
endmodule
```

(b) Either not declaring a wire that should be declared a wire vector or declaring the wrong number of wires in the vector.

i. The Verilog description below has no errors. It is used as a benchmark. Compile the description. Arrange the simulator as follows: NOTE: The circuit requires two modules: the top module and the prototype for the adder. You can type in both modules in the same file. If you use separate files, add the file with the adder module to the project.

```
module bus_width (x , y);
  input [3:0] x;
  output [3:0] y;

  wire [3:0] W1;   // this is correct but will be
                   // changed in a subsequent question
  adder adder_1(.in_1(x),
                .in_2(4'b0001),
                .out_1(W1)
                );
  adder adder_2(.in_1(W1),
                .in_2(4'b0100),
                .out_1(y)
                );

  endmodule
```

```
module adder(in_1, in_2, out_1);
  input [3:0] in_1, in_2;
  output [3:0] out_1;
  reg [3:0] out_1;

  always @ (in_1 or in_2)
    out_1 = in_1 + in_2;
endmodule
```

ii. Change the wire declaration in the Verilog HDL description above to "wire [1:0] W1;". Compile and test the resulting circuit. Explain exactly what has happened to make the output the way it is and what the compiler warnings are saying.

iii. Remove the wire declaration in the Verilog HDL description i.e. delete the line "wire [3:0] W1;". Compile and test the resulting circuit. Explain exactly what has happened to make the output the way it is and what the compiler warnings are saying. Hint: The compiler assumes an undeclared wire is a single wire and not a vector wire.

7. Write a Verilog description of any combination logic circuit you want, as long the description contains a case statement. Do not use "don't cares" in the case items (also sometimes called case alternatives). Compile your Verilog description and test the resulting circuit using the simulator.

8. Consider the Verilog description:

```
module and_or_invert (in_0, in_1, select, out);
input in_0, in_1, select;
output out;
reg out;
always @ (in_0 or in_1 or select)
if (select == 1'b0) out = ~in_0;
else out = ~in_1;
endmodule
```

(a) Draw a schematic diagram of the Verilog description.

(b) Check to see if your schematic is correct by doing the following:

   i. Translate your schematic back into a Verilog HDL description, but make it as a structural description (the original description is a behavioral description).

   ii. Compile the structural description and test the resulting circuit with the Modelsim-Altera simulator.

   iii. Enter the behavioral description that is given above and compile it. Then test the resulting circuit using the Modelsim-Altera simulator.

   iv. If the structural and behavioral descriptions perform exactly the same function, then your hardware schematic must be correct. Check the test results to make sure the Verilog descriptions produce circuits that perform the same function.

9. Compile and simulate each of the three circuits to verify your answers to questions 3 and 4. Be sure to set up a test sequence in the testbench that has all combinations of the inputs.