THE UNIVERSITY OF ALABAMA IN HUNTSVILLE
Department of Electrical and Computer Engineering

CPE 322 Digital Hardware Design Fundamentals

Simulation Assignment

(12% of Final Grade)

Purpose

The purpose of this simulation assignment is to augment the student's understanding of Verilog, in designing, debugging, and synthesizing a circuit. Simulation often provides the quickest way to take a Verilog design concept from initial concept to execution. In this simulation, select problems from the CPE 322 Exam I makeup work will be assigned and explored.
Note that several of the code snippets are pasted in as images instead as raw text – this was intentional to force the student to understand what the code was trying to accomplish (mostly testbench code is provided).

Problem 1: Ring Oscillator

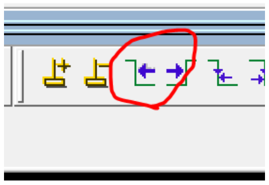
From problem 3.5, as discussed in the lecture, a ring oscillator is a useful circuit for characterizing the speed achieved on a particular die, which depends on the process characteristics, voltage, and temperature of operation.

In this case, a testbench isn't necessary. The start of the following module is provided as:

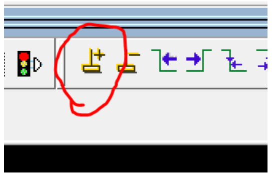
```
`timescale 1ps/1ps
module ring_osc (output z);
  reg [8:0] inv;
  integer i;

  initial begin
    inv = 0;
  end

  assign z = inv[0];
  always @(*) begin
    #200 inv[0] <= ~inv[8];
    ... your code here...
  end
endmodule
```

To get the above code to a complete state, the student should add a "for loop", incrementing on the integer i (already declared) to set values of int[1] through int[8], including an inertial delay. Different behavior will occur depending on whether int[1] through int[8] are assigned with blocking versus non-blocking operators.
QUESTION: HOW DOES THE CIRCUIT BEHAVE WITH BLOCKING? HOW DOES IT BEHAVE FOR NON-BLOCKING?

Using the Wave window to view the signals, place a cursor on the z output rising edge (by clicking the z output, then hitting either button for finding the previous/next transition:



Then click the add-cursor button:



Use the buttons to find the next rising edge on the output z, and noticing that the cursor display shows the relative difference between the two.

QUESTION: WHAT IS THE REPORTED DIFFERENCE BETWEEN CURSORS ON TWO CONSECUTIVE RISING EDGES? SHOW A SCREENSHOT OF YOUR SIMULATION WAVEFORM AS PART OF THE SUBMISSION.

Now finally, update the code to make the ring oscillator 10 inverters deep – this requires a modification to the inv declaration and a modification to the for loop iteration bounds.

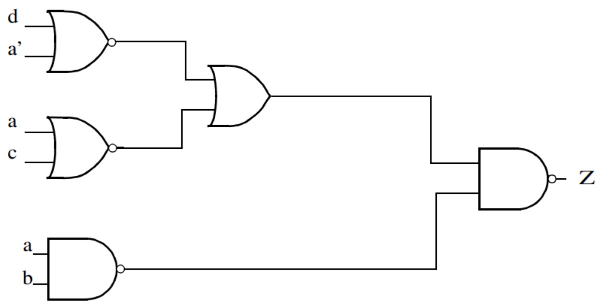
QUESTION: WHAT HAPPENS TO THE z OUTPUT WITH 10 INVERTERS?

Problem 2: Hazard Detection

The logic network from problem 1.5 has one 1-hazard, and it can be found through working the problem. It can also be found by simulation via brute force – by going through the following sequence in a Verilog testbench:

a changes while b, c, and d are set to 8 different values
b changes while a, c, and d are set to 8 different values
c changes while a, b, and d are set to 8 different values
d changes while a, b, and c are set to 8 different values

```
For the network shown below, find any/all static 1-hazards.
For any 1-hazard found specify the conditions which will cause
the hazard to appear at the output (i.e. specify the logic
values of the variables which are constant and the variable
which is changing).
```



Set up a testbench with a single always procedural block, as follows:

```
always begin
  a = 0; b = 0; c = 0; d = 0;
  #10; // wait for 1 nsec (or psec, depending on timescale)
  a = 1; b = 0; c = 0; d = 0; // an example transition on a while b, c, and d remained at 000
  // ADD MORE CODE HERE
end
```

Then set up the above logic network *with 1 nsec inertial delays for each gate*. Then show the single 1-hazard, either through analytically solving the problem and testing the network for the 1-hazard found, or by brute-force by the testbench.

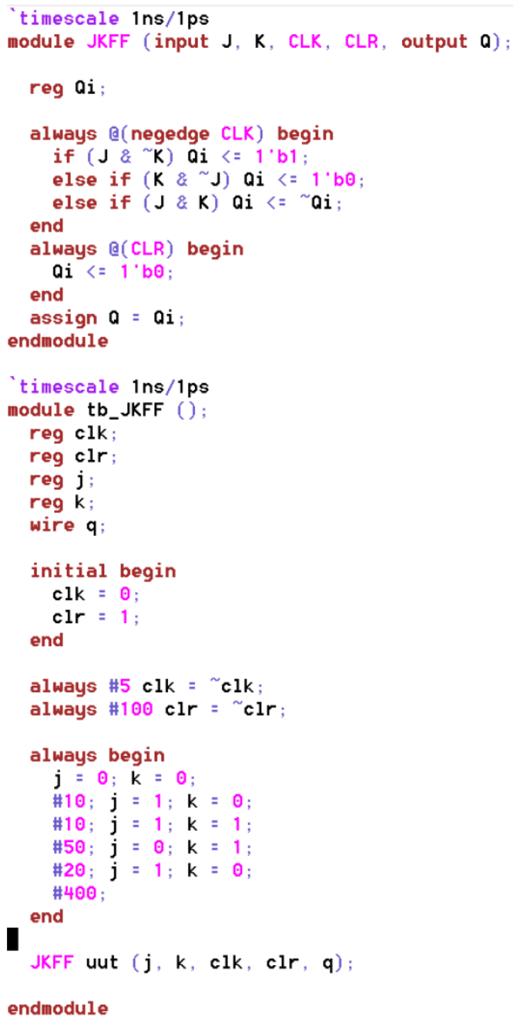
CAPTURE A SCREENSHOT OF THE 1-HAZARD FOR SUBMISSION

Problem 3: Rule for Verilog/VHDL – single procedural block for reg assignment

Many students missed full credit on problem 4.3 because they split the active-high asynchronous clear into a different always @(CLR) block, instead of properly combining it with the always @(negedge clk) block. Here is the problem statement:

Write a short Verilog description of a falling-edge triggered JK Flip-Flop, with J and K inputs, and an active-high asynchronous clear input CLR that sets the output Q immediately to 0.
`module JKFF (input J, K, CLK, CLR, output Q)`

Simulate the following code, running it for 1 us



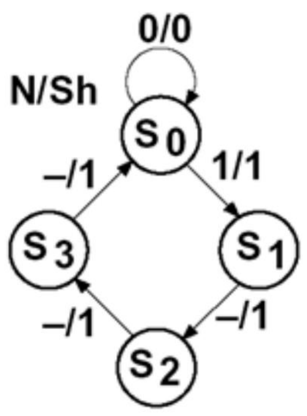
```
`timescale 1ns/1ps
module JKFF (input J, K, CLK, CLR, output Q);

  reg Qi;

  always @(negedge CLK) begin
    if (J & ~K) Qi <= 1'b1;
    else if (K & ~J) Qi <= 1'b0;
    else if (J & K) Qi <= ~Qi;
  end
  always @(CLR) begin
    Qi <= 1'b0;
  end
  assign Q = Qi;
endmodule

`timescale 1ns/1ps
module tb_JKFF ();
  reg clk;
  reg clr;
  reg j;
  reg k;
  wire q;

  initial begin
    clk = 0;
    clr = 1;
  end

  always #5 clk = ~clk;
  always #100 clr = ~clr;

  always begin
    j = 0; k = 0;
    #10; j = 1; k = 0;
    #10; j = 1; k = 1;
    #50; j = 0; k = 1;
    #20; j = 1; k = 0;
    #400;
  end

  JKFF uut (j, k, clk, clr, q);

endmodule
```

QUESTION: FOR THE FIRST 100 nsec, THE CLR INPUT IS HELD HIGH. DOES IT HOLD THE Q OUTPUT LOW, AS REQUIRED? SHOW A SCREENSHOT OF YOUR SIMULATION WAVEFORM AS PART OF THE SUBMISSION.

Now, combine the two always @(…) procedural blocks into a single one, debugging it until the syntax is right and the Q output stays low no matter what the J and K inputs are while CLR is held high.

Problem 4: FSM Coding

In problem 7.B, the following FSM was drawn:



The above is a Mealy state machine, so the expectation is that when the FSM is at state S0 and the N input is driven high, immediately the Sh output will be driven high (without waiting for the following clock edge).
FIRST, GENERATE THE VERILOG CODE FOR THE ABOVE.

SECOND, SIMULATE IT WITH THE FOLLOWING TESTBENCH (note that the name of the module for the above is called FSM (input N, clk, rst, output Sh))

```
`timescale 1ns/1ps
module tb_FSM ();
  reg clk;
  reg rst;
  reg N;
  wire Sh;

  initial begin
    clk = 0;
    rst = 1;
  end

  always #5 clk = ~clk;
  always #20 rst = 0;

  always begin
    N = 0;
    #40; N = 1;
    #20; N = 0;
    #80; N = 1;
    #20; N = 0;
  end

  FSM uut (N, rst, clk, Sh);

endmodule
```

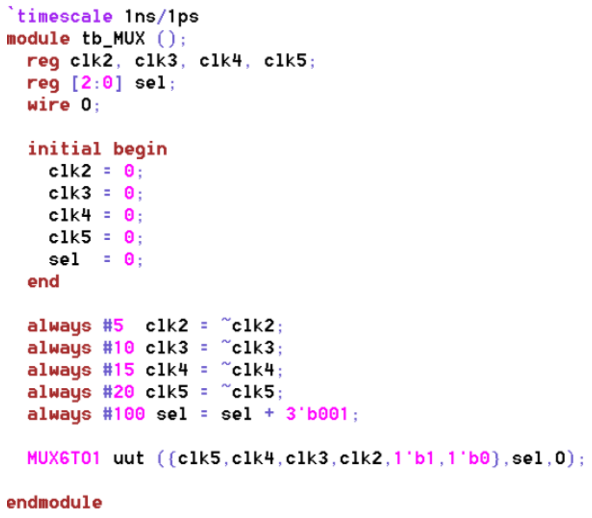
QUESTION: AT TIME t=40 nsec, THE N INPUT GOES HIGH. WHAT IS THE VALUE ON THE Sh OUTPUT AT THAT TIME? SHOW A SCREENSHOT OF YOUR SIMULATION WAVEFORM AS PART OF THE SUBMISSION.

Problem 5: Verilog 6:1 MUX

In problem 10.2, the student was asked to create a 6:1 mux. The S input of a 6:1 mux has 3 bits, which causes it to naturally encode 8 values (3’b000 – 3’b111). The problem statement mentioned specific requirements for the cases when S was out of bounds (6 or 7), with respect to the range of inputs I[5:0]. Nevertheless, several students neglected to code for cases Sel=0 to Sel=5, and only coded 6 and 7:

Develop a Verilog model for a 6-to-1 multiplexer where the general inputs represent a 6-bit bus and the output is a single signal. When S[2:0] is 6, choose I[2] as the output, and when S[2:0] is 7, choose I[3] as the output. If using a case statement, please use a default clause; if using an if/else statement, assign O in all clauses including a final "else" clause to avoid creating a latch. Label the inputs I[5:0], S[2:0], and the output O.
module MUX6TO1 (input [5:0] I, [2:0] S, output O)

Here, the student must code the full 6:1 multiplexer, meaning that all 6 bits of the I[5:0] input vector can be selected to send to the O output. Use the following testbench to show that all 6 inputs are properly forwarded to the O output, including the cases where S is out-of-bounds. Please use the stated requirement to include a “default” clause or final “else” in the code.



```
`timescale 1ns/1ps
module tb_MUX ();
  reg clk2, clk3, clk4, clk5;
  reg [2:0] sel;
  wire O;

  initial begin
    clk2 = 0;
    clk3 = 0;
    clk4 = 0;
    clk5 = 0;
    sel  = 0;
  end

  always #5  clk2 = ~clk2;
  always #10 clk3 = ~clk3;
  always #15 clk4 = ~clk4;
  always #20 clk5 = ~clk5;
  always #100 sel = sel + 3'b001;

  MUX6TO1 uut ({clk5,clk4,clk3,clk2,1'b1,1'b0},sel,O);

endmodule
```

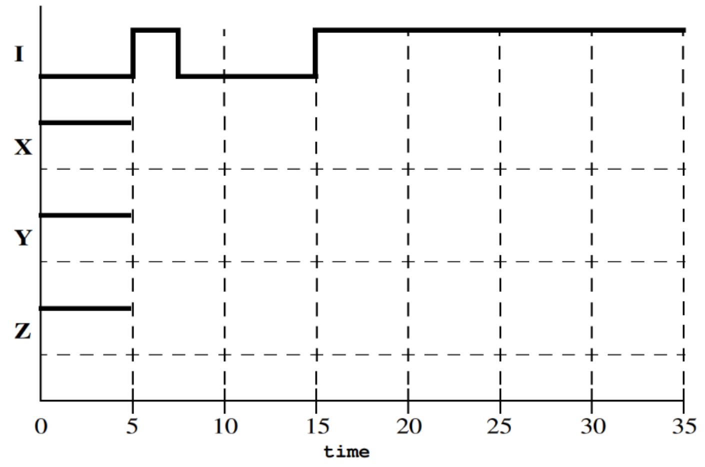
AFTER CODING THIS, SHOW A SCREENSHOT OF YOUR SIMULATION RUNNING FOR 1 usec, WHICH CAPTURES ALL 8 STATES OF THE S INPUT, AND SHOULD SHOW 6 DISTINCT PATTERNS.

Problem 6: Delays

In problem 11.3, the following delay question was posed. Using Modelsim (or other such simulator) is a quick way to slam-dunk this problem and get the correct answer. In fact, the use of Modelsim in solving this problem in homework is advised.

Complete the Timing Diagram for the simulation of the following Verilog Model where input I is driven in the manner that is shown on the diagram.

```
module v_model(input I, output reg X, output Y,Z);
   always @(I) X <= #5 ~I;
   not #(5) G1 (Y,X);
   assign #5 Z = ~Y;
endmodule;
```



Use the following code as a testbench. Note at t=0 the X, Y, and Z signals are all X'es (unknown), but at time t=100 nsec, they are all at 1 like the above diagram shows.

```
`timescale 1ns/1ps
module tb_v_module ();
  reg I;
  wire X, Y, Z;
  initial begin
    I = 0;
  end

  always begin
    I = 0;
    #5; I = 1;
    #2.5; I = 0;
    #7.5; I = 1;
    #50; I = 0;
    #25; I = 1;
    #5; I = 0;
    #5; I = 1;
  end
  v_module uut (I,X,Y,Z);

endmodule
```

SUBMIT A SCREENSHOT OF THE WAVEFORM TO SHOW t=0 AND t=100 nsec (TWO SCREENSHOTS TO SHOW BOTH TIMES SEPARATELY IS ALSO ACCEPTABLE).