Last (family) name: _____

First (given) name: _____

Student I.D. #: _____

Department of Electrical and Computer Engineering
University of Wisconsin - Madison

ECE 551 Digital System Design and Synthesis

Instructor: Kewal K. Saluja

Midterm Exam

Wednesday, March 13, 2002
113 Psychlogy
***7:15--8:30PM* (75 minuts)**

Instructions:

1. Closed book examination. Only one cheat sheet allowed.
2. Five points penalty if fail to enter name or ID number.
3. No one shall leave room during last 5 minutes of the examination.
4. Upon announcement of the end of the exam, stop writing on the exam paper immediately. Pass the exam to isles to be picked up by the instructor or a TA. The instructor will announce when to leave the room.
5. Failure to follow instructions may result in forfeiture of your exam and will be handled according to UWS 14 Academic misconduct procedures.

Problem	*Points*	*Score*
1	20	
2	10	
3	10	
4	15	
5	15	
6	10	
7	10	
8	10	
Total	100	

1. (20 points) *General Questions*
   Answer the following briefly and to the point.

(a) (2 points) Write the binary equivalent of the following and indicate those which are invalid.

4'b10xz = _____
.175 = _____
8'o3_6 = _____
-8'd3 = _____

(b) (2 points) Determine the values of signals in each of the following based on verilog standard:

For a two input **nor** gate if the inputs are x and 1, what will be the output

For a two input **and** gate if the inputs are 1 and z, what will be the output

For a two input **nand** gate if the output is 1, what can be the possible input values

(c) (2 points) What is the difference between the following two declarations

```
reg [1:8] reg_a
reg reg_a [1:8]
```

(d) (2 points) Reduction AND and bitwise AND, both these are written as "&", how do they differ and how are these recognized.

(e) (2 points) Is the following construct correct? If not what is the error?

```
wire y;
nand (y, a, b);
nor (y, c, d);
buf (out, y);
```

(f) (2 points) What is the difference between the following two

```
wire scalared [31:0] bus_A
wire vectored [31:0] bus_A
```

(g) (2 points) True or False:

A verilog net variable can be assigned value by a continuous assignment

All module ports must be scalars

(h) (2 points) what is the difference between transport and inertial delay?

(i) (4 points) Consider the following two descriptions of a 2-to-4 mux. Which of these two descriptions is preferred and why?

```
module mux_1(out, in_1, in_2, in_3, in_4, sel);
input   in_1, in_2, in_3, in_4;
input [1:0] sel;
output out;
 reg out;
 always @(sel)
      case (sel)
          2'b00 : out = in_1;
          2'b01 : out = in_2;
          2'b10 : out = in_3;
          2'b11 : out = in_4;
       endcase
  endmodule


module mux_2(out, in_1, in_2, in_3, in_4, sel);
input   in_1, in_2, in_3, in_4;
input [1:0] sel;
output out;
 reg out;
 always @(sel)
      case (sel)
          2'b00 : assign out = in_1;
          2'b01 : assign out = in_2;
          2'b10 : assign out = in_3;
          2'b11 : assign out = in_4;
       endcase
  endmodule
```

2. (10 points) Structural, RTL, and behavior descriptions

Structural description of a circuit in verlog module form is given below:

```
module structure(y, a, b, c, d);
input  a, b, c, d;
output y;
 wire n;
   nand g1 (n, a, b, c);
   nand g2 (y, n, d);
endmodule
```

a) (2 points) Draw the structure of the circuit

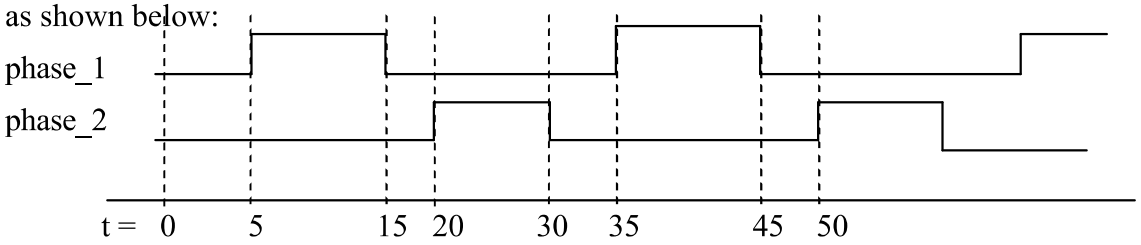
b) (4 points) Write the above module in RTL/Dataflow style using continuous assignment

```
module structure_RTL(y, a, b, c, d);

endmodule
```

c) (4 points) Write the above module using procedural/behavioral style modeling

```
module structure_Proc(y, a, b, c, d);

endmodule
```

3. (10 points) *Test bench - Clock generation*
A circuit uses two phase cloking and the two phases of the clock, phase_1 and phase_2, are as shown below:



a) (2 points) What is the clock period for each phase? All times are in nanoseconds.

b) (8 points) Complete the coding. You are required to use both **always** blocks.

```
module two_phase_clock;
    reg phase_1, phase_2;
initial
    begin
       phase_1 = 1'b0;
       phase_2 = 1'b0;
       #5 phase_1 = 1'b1

    end

// used the following to generate phase_2
always @(posedge phase_1)
    begin




     end
// used the following to generate phase_1
always @(posedge phase_2)
    begin




     end
endmodule
```

4. (15 points) *Blocking/nonblocking assignment; inter/intra statement delays*

In each of the following parts sufficient time steps are provided, use only as many as you need.

a) (5 points) If the following module is executed what will be the values of the variables at each time instant. Assume initially all values are x.

```
module two_begins;
    initial begin
        #1 a = 0;
           b =  1;
        #2 c = 2;
        end
    initial begin
           a = 5;
        #2 c = 0;
        #2 b = 4;  a = c;
        end
endmodule
```

time	a	b	c
0			
1			
2			
3			
4			
5			

b) (5 points) Determine the values of variable at each time instant when the following initial block is executed:

```
initial  begin
          a = 0; b = 1; c =2;
       #1  a = 3;
       #3  b = c;
           c <=  # 2 a;
        end
```

time	a	b	c
0			
1			
2			
3			
4			
5			
6			
7			

c) (5 points) Determine the values of variable at each time instant when the following initial block is executed

```
initial begin
        a = 0; b = 1; c =2;
    fork
     #1  a = 3;
     #3  b = c;
         c <=  # 2 a;
    join
      end
```

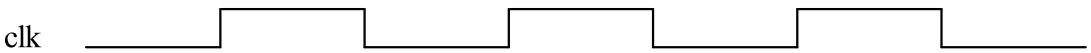
time	a	b	c
0			
1			
2			
3			
4			
5			
6			
7			

5. (15 Points) *Flip-flop modeling*
Consider the following verilog model of a storage element with an active low clr_n signal.

```
module poor_flip_flop(q, d, clr_n, clk);
    input d, clr_n, clk;
    output q;
    reg q;

    initial q = 0;

    always
        begin
            if (clr_n != 0) @(posedge clk) #1 q = d;
            else #1 q = 0;
        end
endmodule
```

(a) (3 points) Does this flip-flop have a synchronous or asynchronous clear?

(b) (12 points) An expert analyzer of verilog claims that "this flip-flop model will create a 1-time unit glitch on the output in some cases when clear is asserted low". Can you identify such a condition. Do so by identifying the input values (d, clr_n), and the state (q) of the flip-flop relative to the clock (shown below). Draw the corresponding waveforms of all signals for at least two clock cycles.



clr_n

d

q

6. (10 points) *Debug*
Correct all syntax errors in the verilog description of a parity checker given below. Please also pay attention to the functionality of the design.

```
module buggy_parity_Checker (rxData, error);
    parameter parity = 0;              // parity =  0 means rxData should even parity
    input scalared [7:0] rxData;
    reg scalared [6:0] temp;
    wire newParity;
     output error;
     reg error;

        assign error = rxDara[8] ^ newParity

     always @
         if (parity = = 1)
               not [6:0] (temp, rxData)
                newParity = ^ temp;
          else
                newParity = ^ rxData[6:0];

 endmodule ;
```

7. (10 Points) *UDP*

:.

(a) (2 points) Answer the following brief questions

How many inputs can a UDP have?

How many outputs can a UDP have?

How many transitions can be specified in a row while describing a sequential UDP?

Is **inout** a legal variable for use in a UDP?

(b) (8 points) Write a UDP that implements the following combinational function:

F = (A ^ B) | C

In the table use as few entries as possible using appropriate symbols. Recall that the output x is a default output for input conditions not explicitly listed in the table.

8. (10 Points) *Finite State Machine*

a) (5 points) Five different styles were discussed in class to implement an FSM. Enumerate these styles and indicate the FSM models (Mealy and/or Moore) that can be implemented by each o the five styles. Give a brief reason as to why the style that combines all three always blocks into one block can not be used to specify a Mealy model.

b) (5 points) For the FSM described below, draw its state diagram and indicate if it is a Mealy or a Moore Machine.

```
module FSM(clk, rst, in, out);
    input   clk, rst, in;
    output out;
    reg [1:0] state, nextstate;
    reg out;
    parameter  s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
     always @(posedge clk or posedge rst)
          if (rst = = 1) state <= s0; else state <= nextstate;

     always @(in or state)
          case (state)
            s0 :  if (in = = 1) nextstate <= s1; else nextstate <= s0;
            s1 :  if (in = = 1) nextstate <= s2; else nextstate <= s0;
            s2 :  if (in = = 1) nextstate <= s2; else nextstate <= s3;
            s3 :  if (in = = 1) nextstate <= s1; else nextstate <= s0;
          endcase

      always @(in or state)
          case (state)
              s0 :  out <= 0;
              s1 :  out <= 0;
              s2 :  out <= 0;
              s3 :  out <= 1;
          endcase
    endmodule
```