Name: ______________________________

Student ID: ___________________

MIDTERM

*Closed book except for one 8½ x 11 sheet. No calculators or electronics devices. All work is to be your own - **show your work** for maximum partial credit.*

1) (10pts) Circle True or False for the following:

a. **True/False** On average one will spend 3X more time validating their DUT code than they did creating it.

b. **True/False** Any statemachine coded as a Moore machine can be coded as a Mealy machine in the same or fewer states.

c. **True/False** Verilog can't be used for designs that target FPGA's.

d. **True/False** In addition to structural, dataflow, & behavioral, Verilog also supports switch level modeling.

e. **True/False** Within a given module, one cannot mix styles of verilog used. For instance behavioral and dataflow used in the same module.

f. **True/False** A design utilizing asynchronous reset should have the reset deassert on the same clock edge that the flops trigger on.

g. **True/False** Synthesis uses delay #'s specified in verilog during the mapping process (mapping of logic equations to standard cells in library) to try to achieve the specified delay.

h. **True/False** It is a good idea to use ***`define*** for state encodings to make the code more readable.

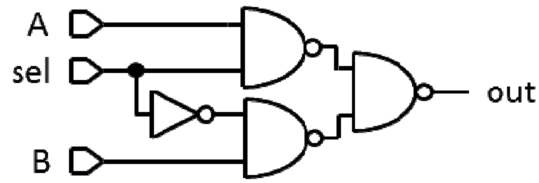
i. **True/False** If a clock signal transitioned: **0 → x** the simulator would consider it a positive edge?

j. **True/False** Verilog has limited file manipulation functions, which is why most people find creative ways to adapt the **$readmemh/$readmemb** functions to their file reading needs.

2) (4pts) 2. Complete the sensitivity list for the always block below. Do not use *

```
always@(                                  ) begin
   if (a) out = in1;
   else out = in2 + 1;
end
```

3) (8pts) Code the circuit shown in all three verilog styles we learned (structural, dataflow, and behavioral)



4) (4pts) In the following ***casez*** block of code, what is displayed when ***sel*** is 0, 1, x and z?

```
always@(sel)
    casez(sel)
      1'b0 : $display("Output 0");
      1'b1 : $display("Output 1");
      1'bx : $display("Output x");
      1'bz : $display("Output z");
endcase
```

5) (5pts) In the following code snippets, will setting ***rst_n*** to 0 in the initial block trigger the always block? **Why or why not**?

```
initial begin
   rst_n = 0;  <---- Is the always block triggered by this line?
   clk = 0; etc..
end

always@(posedge clk, negedge rst_n)
   if(!rst_n)
      state = 1'b0;
   else
      state = 1'b1;
```

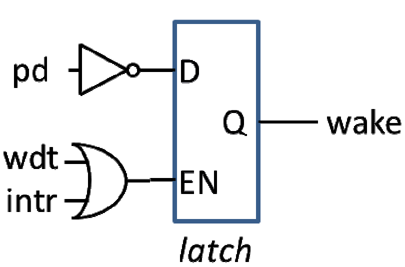
6) (8pts) Complete the behavioral Verilog for a module that infers a 16-bit wide register (bank of flops). When ***load*** is asserted the register should load a sign extended version of the 8-bit input ***dst***. When ***div*** is asserted the register take its current value divided by 2. If neither ***load*** nor ***div*** is asserted the register should maintain. The register should be asynchronously reset (active low) by ***rst_n***.

```
module div_reg16(clk, rst_n, load, div, dst, divReg);

input clk,rst_n,load,div;
input [7:0] dst;
output [15:0] divReg;

endmodule
```

7) (7pts) Complete the verilog code to implement the functionality shown in the schematic.



Note: This does not demonstrate good design practice! It's just a test question.

```
module wake(input pd,wdt,intr, output       wake);

  always @(                              )

endmodule
```

8) (8pts) Gives values for a, b, c and d as they change in each time interval. Please fill in the table provided:

Note: Number of lines provided in table not necessarily related to correct answer.

```
module timing();

  integer a, b, c, d;

  initial begin
      a = 4;
      b = 3;
      c = 2;
      d = 1;
   #1 c = a + 1;
      b = #1 c + 4;
  end

  always@(c) begin
    #2 b <= c + 4;
     d <= #1 b + 2;
    #2 a <= d - 3
  end

endmodule;
```

time:	a	b	c	d

9) (3pts) When implementing a counter that needs to count to 3000. I could define my counter signal as type integer. This would compile and simulate fine in both ModelSim and Synopsys. Why don't I want to do this?

10) (5pts) In the code shown below what type of flop timing is being modeled? (circle one)

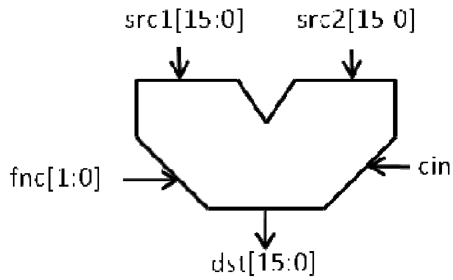
(setup time) (clock to Q delay) (hold time)

```
always @(posedge clk)
   if (rst_n) q <= #(1.5) 1'b0;
   else q <= #(2.1,1.5) d;
```

Why are two timing sets specified for the non-reset case? What is going on there?

11) **(10pts)** Complete the code for the continuous assign statement to implement an ALU with the functions outlined in the table below. **Strive for easy to read code!**

fnc[1:0]	Description
2'b00	**PrtLow:** dst[15:9] = zeros dst[8] = even parity of low byte of src2 dst[7:0]=src2[7:0]
2'b01	**ByteSwap:** dst = src1 with high and low bytes swapped
2'b10	**2sComp:** dst = 2's compliment of src1
2'b11	**Add:** dst = add with carry

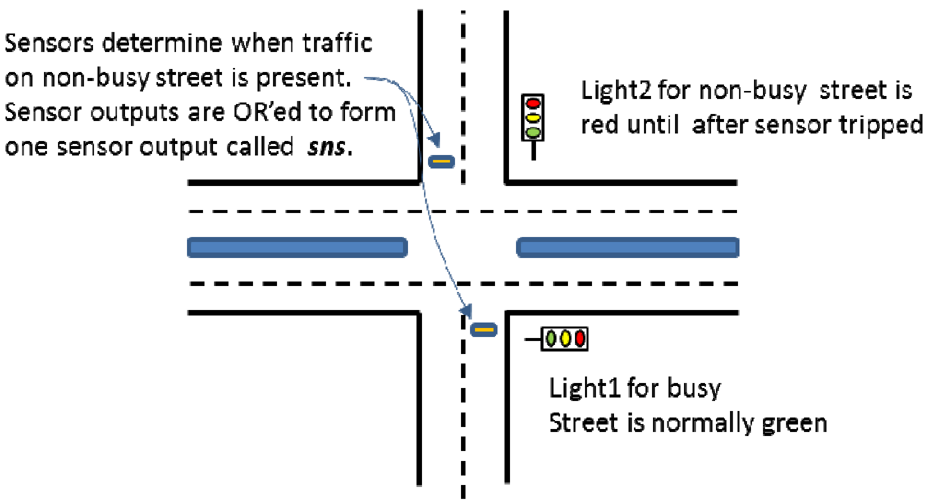


(No Always blocks!)

```
assign dst =
```

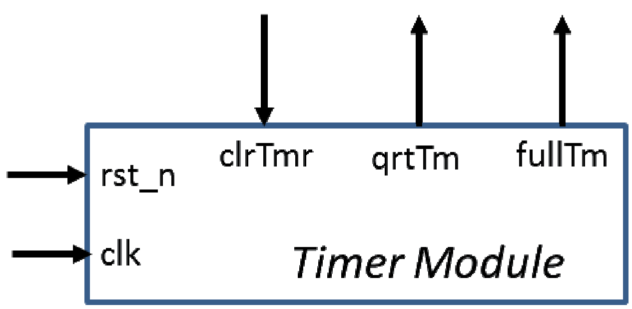
12) **(28pts)** Design Problem (stay calm…its really not that hard) (read entire problem statement before doing anything).

You are to design a state machine to control traffic lights. We have an intersection with a busy street controlled by Light1, and a not so busy street controlled by Light2. Typically, Light1 is green, Light2 is red. When a car drives up to the intersection in the non-busy street, they trip a sensor (indicated to the SM by ***sns*** signal). At that point the SM will reset a timer (assert ***clrTmr***), and **wait for a full timer period** (indicated by ***fullTm***) prior to changing the busy street (light1) yellow. Light1 should stay yellow for a quarter of a timer period (indicated by ***qrtTm*** if the timer was propery cleared). Then the non-busy street (controlled by light2) should get a green light, and the busy street should transition to red. The non-busy street should get a green light for a full timer period. Again on the transition back to the busy street being green, there needs to be a quarter timer period where Light2 was yellow, and Light1 maintains red.



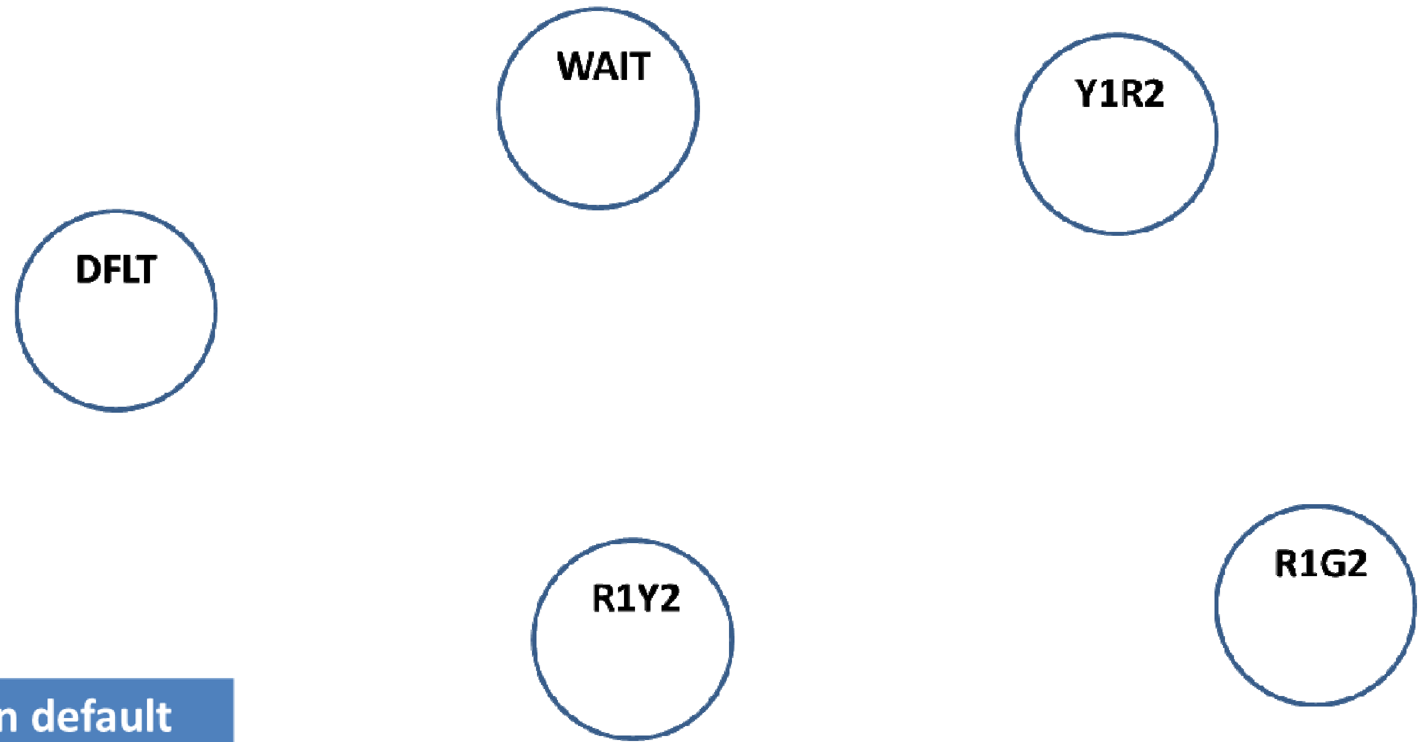
State Machine Outputs:	
G1	Turn on green for light1
Y1	Turn yellow for light1
R1	Turn red for light1
G2	Turn on green for light2
R2	Turn yellow for light2
Y2	Turn red for light2
clrTmr	Clears timer

State Machine Inputs:	
clk	Clock
rst_n	Active low reset
qrtTm	Input indicating timer is at 1023 clocks
fullTm	Input indicating timer is at 2047 clocks
sns	Turn yellow for light2



A timer module exists to assist the SM in timing light durations

10. (continued)



Complete the bubble diagram, indicating transition conditions And non-default output values on each arc. Fill in the default output value table.

Fill in default outputs:	
G1	
Y1	
R1	
G2	
R2	
Y2	
clrTmr	

10. (Continued) Complete the Verilog for the SM. Pretend **Cummings** was grading.

```
module traffic_sm(clk, rst_n, sns, qrtTm, fullTm, R1, Y1, G1, R2, Y2,
                  G2, clrTmr);

input clk, rst_n, sns, qrtTm, fullTm;
output R1, Y1, G1, R2, Y2, G2, clrTmr;

endmodule
```

10. (continued) Complete the code for the timer module, assuming ***fullTmr*** is asserted at a full count of a timer that is WIDTH bits wide. Where WIDTH is a parameter.

```
module timerModule(clk, rst_n, clrTmr, qrtTm, fullTm);

parameter WIDTH = 16; // default is 16, minimum value of WIDTH is 3

input clk, rst_n;        // reset is asynch active low
output qrtTm; fullTm;    // fullTm is asserted at full 16-bit cnt











endmodule
```