Name: ____Eric Hoffman_____

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) (4pts) Circle True or False for the following:

a. **True/False** An advantage of a Moore machine is the output(s) can't glitch.

b. **True/False** An unconnected input signal to a module will be assigned a "z" (high impedance) value by the simulator.

c. **True/False** It is possible to infer latch behavior with a continuous assign statement (dataflow verilog).

d. **True/False** It is good practice to have the master reset signal de-assert on the opposite clock edge than the style of flops you use.

2) (6pts) Multiple Choice

a. System verilog offers an ***always_ff*** block. This differs from standard ***always*** block in the following way:
   i. It ensures the signals assigned within it will synthesize to flops.
   ii. It does not differ that much, only ensures synthesis will throw a warning if the assignments within it do not form a flop.
   iii. It does not need a sensitivity list because it knows flops are being inferred.

b. System verilog offers an ***always_comb*** block. This differs from standard ***always*** block in the following way:
   i. You are allowed to use * in the sensitivity list as opposed to individually specifying each signal.
   ii. It does not need a sensitivity list because it knows combinational logic is being inferred.
   iii. Synthesis will ensure the logic inferred by the block has no sequential elements (latches or flops).

c. We favor HDL's over schematic capture because:
   i. Data path widths (8,16,32,…) can be abstracted
   ii. Synthesis does the bulk of the tedious work
   iii. Design files (ASCII text) are very portable between different CAD vendors
   iv. All of the above
   v. i & ii.

d. Meta-stability:
   i. Occurs when alpha particles impart charge on a node of a latch or flop
   ii. Much like the Loch ness monster, is an elaborate hoax
   iii. Can occur when the input of a flop changes coincident with the clock
   iv. Can be solved by using a high gain inverter.
   v.

3) (5pts) Which of the following clock generators will work best, why?

```
initial
  clk = 1'b0;
```

```
always @(clk)
   clk = #5 ~clk;
```

```
always @(clk)
  clk <= #5 ~clk;
```

```
initial
  forever clk = ~clk;
```

Which of the [illegible] best and why?

The first one can't self trigger due to use of blocking, so clock will stop after first edge. The third one does not have a delay to the clock would be infinitely fast.

4) (5pts) In the table below fill in the result of the expressions (include bit width of the result). **A = 6'h01**, and **B = 6'h2A**

Expression:	Result (include width i.e. 4'bxxxx)
{A,B[4:0]}	**11'b00000101010**
^B	**1'b1**
~A	**6'b111110**
{{2{B[5]}},B}	**8'b11101010**

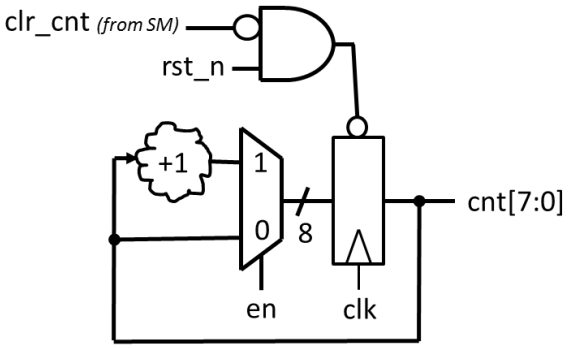
5) (4pts) A lazy digital designer tries to infer an **active low asynchronous reset D-flop** in the following manner. Does this work? Explain **why** or **why not**.

```
module D_Flop(input clk, rst_n, d; output reg q);
always@(posedge clk, rst_n)
 if(!rst_n)
   q<= 0;
 else
   q<=d;
endmodule
```

Lazy Digital Designer Thinks: "Why write negedge when specifying sensitivity list, simply rst_n should also work

The block will also trigger on the rise of rst_n. This will then cause the block to evaluate, and q<=d which is not proper operation.

6) (5pts) Complete the verilog code to implement the counter shown.

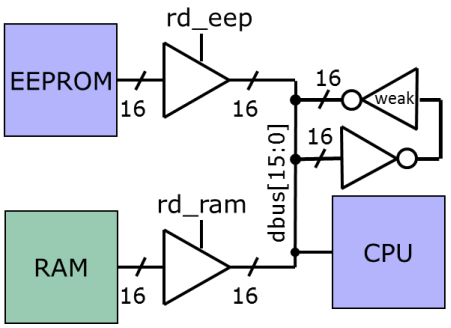



```
module cntr(clk,rst_n,clr_cnt,en,cnt);

input clk,rst_n,clr_cnt,en;
output reg [7:0] cnt;

assign rstn = ~clr_cnt & rst_n;

always @(posedge clk, negedge rstn)
  if (!rstn)
    cnt <= 8'h00;
  else if (en)
     cnt <= cnt + 1;

endmodule
```

7) (4pts) In the counter shown above. Given ***clr_cnt*** is coming from a state machine, describe in detail what might go wrong with this implementation.

Any signal coming from a SM can glitch. If clr_cnt glitches low the counter will be reset at that time since it is coming in the asynch clear path. This would be bad. If clr_cnt comes in through the "front door" as it should then it can only have its affect at the posedge of clock, so as long as the logic is settled by then we are good.

8) (4pts) Realize the tri-state bus indicated in the diagram using **dataflow** (not structural) Verilog in space provided.



(2pts) What would happen if both ***rd_eep*** and ***rd_ram*** were asserted at the same time?
Contention…which results in 16'hxxxx being on dbus.

(3pts) What is the purpose of the feed forward and feedback inverter structure?. To ensure the bus is not high impedance. This may not be important for functionality since the CPU is not reading dbus, but it is important for power consumption since floating inputs to gates make them consume current. I graded very lenient on this…the one person who answered perfect got Extra.

```
module TRIbus(eeprom, ram, rd_eep, rd_ram, dbus);
input [15:0] eeprom, ram;
input rd_ram, rd_eep;
output [15:0] dbus;
// Write your code here in the space provided
assign dbus = (rd_eep) ? eeprom : 16'hzzzz;
assign dbus = (rd_ram) ? ram : 16'hzzzz;


endmodule
```

nan)

9) (6pts) A designer wished to make a counter that would count up when enabled. The code they came up with is as follows:

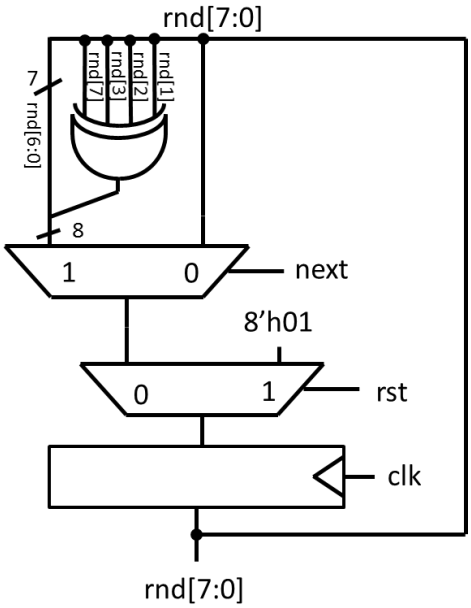
```
assign count = cnt_en ? count + 1 : count;
```

Draw how this would synthesize. Describe how you think it would behave in silicon.

Most everyone got the drawing right. Just a 2:1 mux with incrementor in one path, and count feeding back to itself.

For how it would behave I was looking for the fact that it would either increment very fast because there was no clock, or behave poorly/unpredictably due to the combinational feedback loop and lack of flop.

10) (6pts) A LFSR is a digital circuit that can be used to obtain a pseudo-random number sequence. Below is a diagram of an LFSR. Complete the Verilog to implement it.




```
module LFSR (clk,rst,next);
input clk,rst;
input next;
output reg [7:0] rand;


always @(posedge clk)
  if (rst)
    rnd <= 8'h01;
  else if (next)
    rnd <= {rnd[6:0],rnd[7]^ ^rnd[3:1]);



endmodule
```

(3pts) Simulate in this circuit (in your head) What is the value of rand 2 clocks after reset de-asserts? (assuming ***next*** is high)

We graded very lenient on this. Answer is 8'h05, but also accepted some other values.

11) (2.5pts) Consider the test bench shown:

```
module my_tb();
reg a,b,c,d;

MyDesign iDUT( .a(a),.b(b),.c(c) );

intial  begin
 a= 1; b=0; c=0;
 #3 a=0;
end

intial
 a=b;

endmodule
```

Circle the correct statement(s).

a. This code has a race condition on ***a***
b. This code implies a short in hardware
c. This c
d. This will not compile due to assigns to ***a*** from two **initial** blocks.

This is a test bench…there is no hardware inferred

12) (2.5pts) To make state machine code more readable one can use **localparam** to give the states meaningful names. In system Verilog one can use enumerated types. What advantage does use of enumerated types have over **localparam**?

The state names show up in the simulation waveforms.

13) (8pts) Complete the Verilog to produce a multiply accumulate unit that multiplies two signed 8-bit signed numbers, and adds that product to ***accum*** to produce ***sum***. The add should be saturating. If the value was to exceed 0x7fff it should be pinned at 0x7fff, or if negative if it was going to be more negative than 0x8000 then it should be pinned at 0x8000.

```
module MAC(input signed [7:0] A,B; input  [15:0] accum; output [15:0] result);

wire  signed          [15:0] product;
wire [15:0] unsat_sum;
wire sat_pos,sat_neg;
///// Implement multiplier /////
assign product = A * B

///// Implement addition /////
assign unsat_sum = product + accum;

//// determine if saturated positive or negative ////
assign sat_pos = (~accum[15] & ~product[15]) ? unsat_sum[15] : 1'b0; // positive overflow

assign sat_neg = (accum[15] & product[15]) ? ~unsat_sum[15] : 1'b0;

assign result = (sat_pos) ? 16'h7fff :
                (sat_neg) ? 16'h8000;
                unsat_sum;

endmodule
```

14) **(9pts)** Write a test bench for the MAC unit of the prior problem. The test bench should test a couple of values and stop if it fails, and display a happy message if it passes. ***(don't worry about the values used for testing, judging this on structure/format)***

Was looking for:

Proper syntax

Stimulus to be defined as type reg, and output to be of type wire

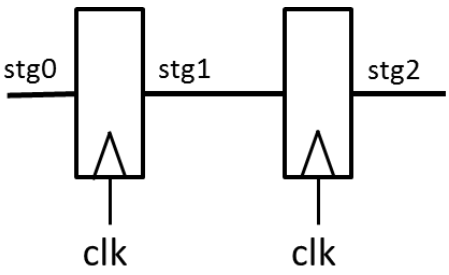
Proper instantiation of the DUT

Stimulus to be applied

Some attempt at self checking

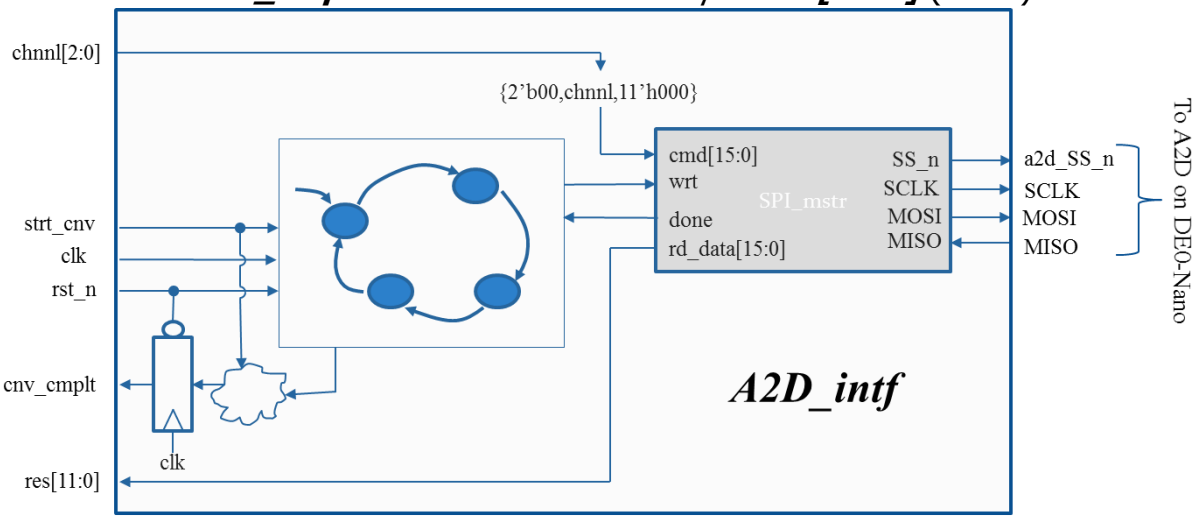
15) **(3pts)** Your design has a control signal that needs to be pipelined. The implementation is shown below. Synopsys timing analysis had determined there is a hold time problem. What will Synopsys do to fix the hold time issue?

It would add a buffer between the two flops to slow the signal down.



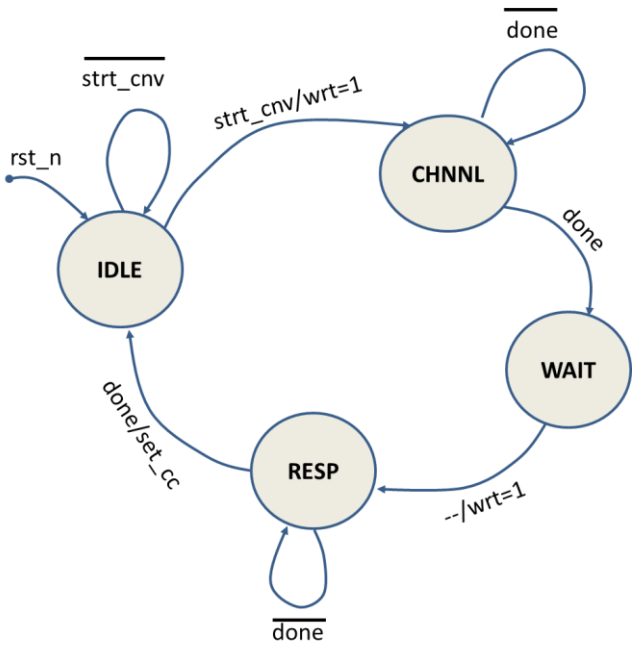
16) (18 pts) In HW3 you created a SPI master. We are now going to use that SPI master to create an interface to the A2D on the DE0-Nano board. The basic steps are:

a. When ***strt_cnv*** is asserted initiate a SPI transaction to inform the A2D what channel to convert (A2D returns garbage during this transaction).
b. Wait one clock cycle after the SPI transaction has completed.
c. Initiate a SPI transaction to grab the data returned from the A2D (not so important what command was sent to A2D during this transaction).
d. Assert ***conv_cmplt*** when A2D data is ready on ***res[11:0]*** (result)



SM Function: Send command to A2D via SPI to ask for conversion on channel. Once that transaction completes wait one clock cycle. Then start new transaction to read the result of the A2D conversion back.

Draw a bubble diagram for the state machine:



Finish the code to implement A2D_intf including SM functionality.

```
module A2D_intf(input clk,rst_n,strt_cnv; input [2:0] chnnl; output reg cnv_cmplt;
                output [11:0] res; output a2d_SS_n,SCLK,MOSI input MISO);
  wire [15:0] cmd;
  wire done;
  ///// define outputs of SM as type logic /////
  logic wrt, set_cc;
  reg cnv_cmplt;
  ///// defines for states for SM below //////
  typedef enum reg[1:0] {IDLE,CHNNL,WAIT,RESP} state_t;
  state_t state, nxt_state;
  ///// Instantiate SPI master for A2D interface /////
  SPI_mstr16 iSPI(.clk(clk),.rst_n(rst_n),.SS_n(a2d_SS_n),.SCLK(SCLK),.MISO(MISO),
                  .MOSI(MOSI),.wrt(wrt),.done(done),.rd_data(res),.cmd(cmd));
   always_ff @(posedge clk,negedge rst_n)
     if (!rst_n)
       state <= state;
    else
       state <= nxt_state;
   always_ff @(posedge clk, negedge rst_n)
      if (!rst_n)
        cnv_cmplt <= 1'b0;
     else if (strt_cnv)
        cnv_cmplt <= 1'b0;
     else if (set_cc)
       cnv_cmpl <= 1'b1;
   assign cmd = {2'b00,chnnl,11'b000};
   always_comb begin
     wrt = 0;
     set_cc = 0;
     nxt_state <= IDLE;
     case (state)
       IDLE : if (strt_cnv) begin
         wrt = 1;
         nxt_state = CHNNL;
       end
       CHNNL : if (done)
         nxt_state = WAIT;
       WAIT : begin
         nxt_state = RESP;
         wrt = 1;
       end
       RESP : if (done) begin
         nxt_state = IDLE;
         set_cc = 1;
       end
     endcase
   end
endmodule
```