Name: Gurbir Arora

UID: 105178554

Part 1: Syntax Review and Debugging

1. You see the following code in a module:

$16^1 h3EC2$
$16^3 16^2 16^1 16^0$

```
assign a = 16'h3EC2;
```

   a. What is the signal type of a ? (Choose the correct option)
      i. Wire
      ii. Reg
   b. What is the value of a in decimal? 16,066

2. What is wrong with the following code? (This is a common mistake for beginners)
   Circle the line(s) that have an error and propose the correct solution.
   Hint: the error is a syntax error, not a logical one.

```
module my_module
{
   input [2:0] a,
   input [2:0] b,
   output [2:0] c
};

     always @ (*)
     begin
       assign c = a & b;
     end
endmodule
```

c <= a & b;

3. Your friend finds that his state machine code which should trigger once every clock cycle instead triggers multiple times per clock cycle.

```
always @ (clk)
begin
     … some state machine code …
end
```

   Answer the following:
   a. How many times does this code trigger per clock cycle? 2
   b. What change should be made to trigger only once per cycle? (There are several correct answers, try to choose the simplest one)

always @ (posedge clk)

4. Fill in the blanks:

```
module mod_a
(
   input a_en,
   input clk,
   input wire ascii_in,
   output reg [25:0] decoded_letter,
   output reg decode_error
);
       …
       always @(posedge clk)
       begin
              if (a_en == 1'b1)
              begin
                     case (ascii_in)
                            8'h41:
                            begin
                                   decoded_letter <= 1'd1;
                                   decode_error <= 1'b0;
                            end
                     …
              end
       end
       …
endmodule

module mod_b
(
   input clk,
   input [7:0] switches,
   input sys_enable,
   …
);
       …
       wire decode_enable;
       wire [25:0] decode_output;
       wire error_check;
       mod_a my_instance
       (
              .a_en(decode_enable),
              .clk(clk),
              .ascii_in(switches),
              .decoded_letter(decode_output),
              .decode_error(error_check)
       );

       always @ (posedge clk) //enable clock synchronization
       begin
              if (sys_enable == 1'b1)
              begin
                     decode_enable <= 1'b1;
              end
              else if (sys_enable == 1'b0)
              begin
                     decode_enable <= 1'b0;
              end
       end
       …
endmodule
```

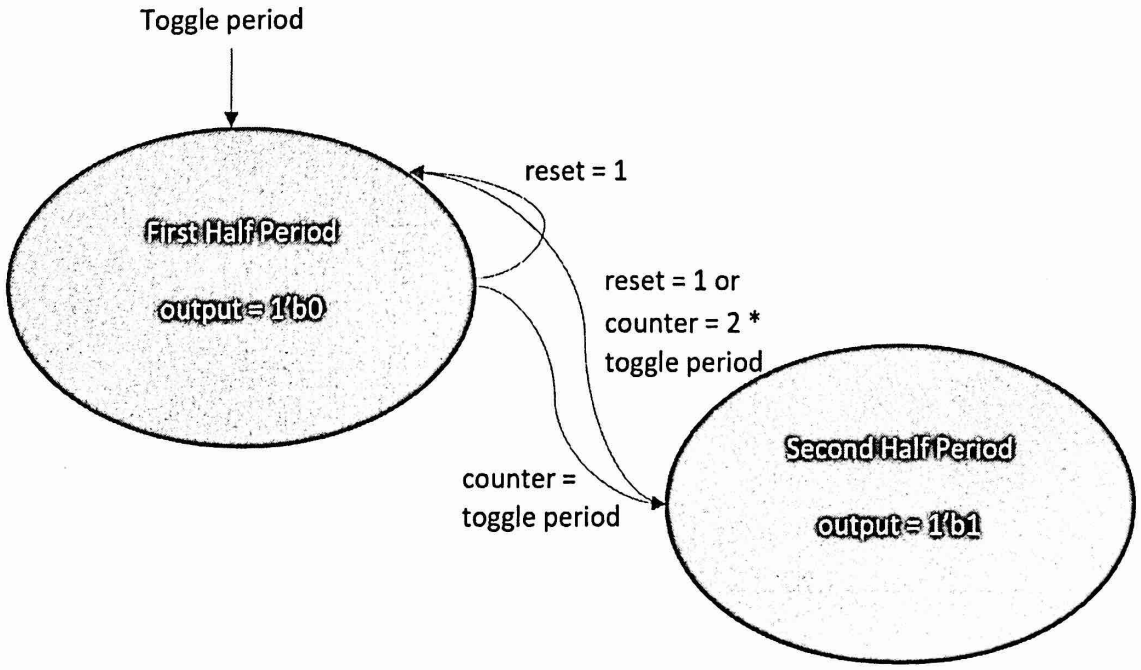
Part 2: Design Task (Clock Divider)

Your goal is to synthesize a slower clock signal given the system's hardware clock. Your design shall contain the following:

1. A clock division module that computes the slower clock signal's value depending on the toggle period
    a. We define the toggle period as the number of hardware clock periods that fit into a half-period of the slower clock
    b. Inputs to this module:
        i. Hardware clock `hw_clk`
        ii. Toggle period `tp`
            1. Assume maximum value of 1000
        iii. Reset signal `rst`
            1. We'll accept either asynchronous or synchronous reset
    c. Outputs from this module:
        i. Clock output `clk_out`
    d. `clk_out` is computed via a state machine implementation
        i. The two states correspond to each of the half periods of `clk_out`
    e. An internal signal `counter` that tracks the elapsed number of `hw_clk` periods since entry into the current state

Fill out the module description in the code template text file provided. Attach the file in your project submission!

To guide your design, we give you the following high-level state diagram. You'll be implementing this state machine with a simple counter signal.



(IMPORTANT: Keep this page in submission even if left unused)

```
module clock_divider
{
      //TODO: fill out the inputs and outputs

};

//TODO: declare a counter signal to count elapsed hardware clock periods
        (assume that the counter signal is initialized to 0)


always @ ( //TODO: fill the sensitivity list )
begin
      //TODO: write code to increment the counter
end

always @ ( //TODO: fill the sensitivity list )
begin
      //TODO: write the state machine body
end


endmodule
```

posedge hw_clk

counter++;

negedge hw_clk

if(counter == tp)

note: compared counter to tp-1 bc counter begins @ 0 & tp begins @ 1

(IMPORTANT: Keep this page in submission even if left unused)

```
module clock_divider(
    input hw_clk,
    input rst,
    input [9:0] tp,
    output clk_out
);
reg [10:0] counter; // length of 11 since ⌈log2(2000)⌉ = 11 & max val is 2000
always @(posedge hw_clk or posedge rst)
    begin
        if(rst)
            counter <= 11'd0;
        else if((counter==(tp-1'b1))||(counter==(2'd2*(tp-1'b1))))
            counter <= 11'd0;
        else
            counter <= counter+1'b1;
    end
always @(posedge hw_clk or posedge rst)
    begin
        if (rst)
            clk_out <= 1'b0;
        else if((counter==(tp-1'b1))||(counter==(2'd2*(tp-1'b1))))
            begin
                if (clk_out==1'b0)
                    clk_out <= 1'b1;
                else
                    clk_out <= 1'b0;
            end
    end
assign clk_out = (clk_out==1'b1);
endmodule
```

(IMPORTANT: Keep this page in submission even if left unused)