## EE354L Quiz (~5%)

Open-book Open-notes Exam; No loose sheets
Calculators and Verilog Guides are allowed.

### Spring 2016
Instructor: Gandhi Puvvada
Wednesday, 2/24/2016
10:00 AM - 11:45 AM (1 Hour 45 min.)
Location: SLH100

Student’s Last Name: _______________________________________
Student’s First Name: _______________________________________
Student’s USC username: ____________________________ @usc.edu

<table>
<thead>
<tr>
<th>Ques#</th>
<th>Topic</th>
<th>Page#</th>
<th>Time</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EE101 review</td>
<td>2</td>
<td>20 min</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Counters, Synchronous and Asynchronous clear</td>
<td>3</td>
<td>10 min</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>State Diagram and RTL design</td>
<td>4-5</td>
<td>50 min</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SSD scanning control on Nexys-3 board - Intensity control</td>
<td>6</td>
<td>10 min</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Verilog</td>
<td>7</td>
<td>15 min</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>7</td>
<td>1H 45M</td>
<td>166</td>
<td></td>
</tr>
</tbody>
</table>

**Perfect Score**

|       |       |       |       | 155    |       |

---

Viterbi School of Engineering

University of Southern California
EE101 review:

1.1 Mr. Bruin and you (Mr. Trojan) went for an internship interview and you were asked to implement combinational logic to produce \( Y = X + 24_{10} \) \( X \) is a 4-bit unsigned number \( X[3:0] \). Since the range of values of \( X \) is 0-15\(_{10}\) (0000\(_2\)-1111\(_2\)), the range of values of \( Y \) is (24\(_{10}\)-49\(_{10}\)). Hence \( Y \) is a 6-bit unsigned number. Mr. Bruin’s incomplete design is shown on the left below. Help him to complete his design. Then show a much simpler design at gate-level on the right and get the internship. You show to the interviewer how a Half-Adder adding \( X_3 \) to 1 to produce \( Y_3 \) and a carry can be simplified and how this simplification can be used twice, once to produce \( Y_3 \) and a second time to produce \( Y_4 \) and \( Y_5 \).

\[
\begin{array}{c|cccc}
X & X_3 & X_2 & X_1 & X_0 \\
\hline
24 & 1 & 1 & 0 & 0 \\
\end{array}
\]

\[
X + 24 = Y_5 \quad Y_4 \quad Y_3 \quad Y_2 \quad Y_1 \quad Y_0
\]

1.2 _________ (While/Since) PROM can be used for implementing _________ (any/small) combinational logic, you _________ (would/wouldn’t) use a PROM to implement arithmetic circuits such as adder, comparator, etc. because ____________________________________________________________.

1.3 Generate the outgoing carry Cout for the Full-adder. Complete the AND-OR and the OR-AND implementations.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{in} )</td>
<td>( Y )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
Cout = X.Y + C_{in}.X + C_{in}.Y
\]
Given below are two counters, 74LS162A and 74LS163A, TI (Texas Instruments).

From the diagrams below, can you quickly identify which one has a synchronous clear control and which has an asynchronous clear control.

The one with an asynchronous CLR input pins is ________ (74LS162A/74LS163A)

The one with an asynchronous CLR input pins is ________ (74LS162A/74LS163A)

You could say so quickly because ________________________________________________
____________________________________________________________________________
____________________________________________________________________________

And say in a state S4 you need to clear the counter as shown on the side.
Would you prefer to use A or B to clear I? _____________ (A/B)
A. the CLR input pin on a counter with an asynchronous clear
B. the CLR input pin on a counter with a synchronous clear input

Explain why? ________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

The advantages of output coding of the symbolic states in the state machine implementation (example: our PB debouncer state machine implementation) is _________________________
____________________________________________________________________________
____________________________________________________________________________

Output coding is same as _____ (A/B/C) A. One-Hot coding B. Encoded state assignment C. Neither
This is a voting machine inspecting 5 votes serially. Here the first vote (V0) and the last vote (V4) are double-weight votes where as the middle three votes are single weight votes. So in a way it is like 7 votes (2*2 + 3*1 = 7). So weights-wise, you need to get 4 to win or lose 4 to lose overall. So the C2N is not necessarily two NO votes as the first double-weight vote can bring you there if it a NO vote. So if V0, V1, V2, V3, V4 are 0, 0, 1, 1, 1, then you pass through C, C2N, C3N, C3N, C3N, WON. All State transition arrows are in place. Complete the state transition conditions.

As per the above state diagram, there are 4 ways to win (corresponding to the 4 arrows arriving at the WON state) and two ways to lose (corresponding to the 2 arrows arriving at the LOST state). I gave an example for one winning situation. List examples for the other 5 situations and also list the sequence of significant states it traverses through.

<table>
<thead>
<tr>
<th>Winning or losing arrow</th>
<th>V0, V1, V2, V3, V4</th>
<th>State Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winning, C =&gt; WON arrow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Winning, C1N =&gt; WON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Winning, C2N =&gt; WON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Winning, C3N =&gt; WON</td>
<td>0 0 1 1 1</td>
<td>C, C2N, C3N, C3N, C3N, WON</td>
</tr>
<tr>
<td>Losing, C2N =&gt; LOST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Losing, C3N =&gt; LOST</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Here again we do Inches to Yard-Feet-Inches conversion, but we extract Yards first from Inches by subtracting 36 inches from inches and incrementing Yards. After extracting yards, if there are enough leftover inches, then we will convert them to feet. Complete the Moore machine below. Notice that there is no state transition from C_YARDS to C_FEET because if inches are exactly divisible by 36, that also means that there aren’t any left-over inches to convert to feet. Please read Q3.2.1 below first before completing the state transition conditions below.

3.2.1 (Mr. Bruin/Mr. Trojan) says that, unlike in the original Inches to Yard-Feet_Inches conversion (where the Inches variable is an unsigned number), here the Inches variable shall be a signed positive number to start with and may become negative later and you need a signed number comparator to compare the inches with a signed constant. 

Explain your choice with a fitting example: ______________________________________
___________________________________________________________________________
___________________________________________________________________________

3.2.2 Miss Trojan makes a recommendation regarding converting to feet (regarding C_FEET and U_FEET). She says that the feet can be at most 2. So, instead of possibly overshooting and then undoing costing 2 extra clocks, she recommends that we have C_FOOT1 then C_FOOT2 (with no loops looping back) as shown below. Complete the state transition conditions for the 3 state transition arrows.

(Mr. Trojan/Mr. Bruin) says, that if inches can be compared with 24 besides 12, then overshooting can be avoided in the original design also.
Reproduced below is a question and its solution from the Quiz of Spring 2015.

Scanning control alteration to adjust the intensity.

// The clock divider is just for your information
reg [26:0] divclk;
always @ (posedge board_clk, posedge reset)
begin
  if (reset)
    divclk <= 0;
  else
    divclk <= divclk + 1'b1;
end
assign sev_seg_clk = divclk[16:15];

// In the following four lines from the original design,
I have replaced An0 with An0_I (standing for Anode 0 Intermediate) and similarly the other three.

assign An0_I = ~(~(sev_seg_clk[1]) && ~(sev_seg_clk[0])); // when sev_seg_clk = 00
assign An1_I = ~(~(sev_seg_clk[1]) && (sev_seg_clk[0])); // when sev_seg_clk = 01
assign An2_I = ~( (sev_seg_clk[1]) && ~(sev_seg_clk[0])); // when sev_seg_clk = 10
assign An3_I = ~( (sev_seg_clk[1]) && (sev_seg_clk[0])); // when sev_seg_clk = 11

// In the following four lines, I have used divclk[17] and divclk[18] to keep a signal such as An0
// active or inactive (you figure it out) for extended length of time.
assign An0 = (An0_I) | (divclk[18]); // ORed with the (divclk[18])
assign An1 = (An1_I) | (divclk[18]); // ORed with the (divclk[18])
assign An2 = (An2_I) | (divclk[17]); // ORed with the (divclk[17])
assign An3 = (An3_I) | (divclk[17]); // ORed with the (divclk[17])

Since An0 and An1 were altered in one way and An2 and An3 were altered in a different way,
we expect SSD0 and SSD1 to glow with different intensity compared to SSD2 and SSD3.

Explain what behavior you expect to see and how you have arrived at your conclusion. Provide good
explanation. Since the Anode signals (which are passing through power transistors) are active
low, ORing with any divclk (does not matter [17] or [18]) which is slower than [16],
causes the Anode to be inactivated for exactly 50% of the time when the divclk is a 1.

So each of the 4 SSDs glow with HALF brightness and there is no difference between
two four intensities!
The solution says:

**ORing with any divclk (does not matter [17] or [18]) which is slower than [16].**

What happens if we OR with (divclk[24]) and (divclk[25]) in place of (divclk[17]) and (divclk[18]) respectively?

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

5 (10 + 5 + 16 = 31 points) 15 min.

![Diagram of state machine](image)

<table>
<thead>
<tr>
<th>cross off if not needed</th>
<th>Starting Value</th>
<th>Going to be</th>
</tr>
</thead>
<tbody>
<tr>
<td>in the first clock</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>in the second clock</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>in the third clock</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>

5.1 A student forgot that the above **RESET** is active low. Fix his reset on the side.

Four students wrote the case branch for the **COMP** state as shown below. Your comments?

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

5 pts

always @(posedge Clk, posedge Reset)
begin : q5_Verilog
if (Reset)
begin
state <= INIT;
I <= 4'bXXX;
J <= 4'bXXXX;
end
else

16 pts

COMP:
begin
if (I!=4'b1100)
state <= DONE;
I <= J * 2;
J <= J - 1;
end

COMP:
begin
if (I!=4'b1100)
state <= COMP;
Jnext = J - 1;
I <= Jnext * 2;
J <= Jnext;
end

COMP:
begin
if (I!=4'b1100)
state <= DONE;
Jnext = J - 1;
I <= Jnext * 2;
J <= Jnext;
end

COMP:
begin
if (I!=4'b1100)
state <= DONE;
Jnext = J - 1;
I <= Jnext * 2;
J <= Jnext;
end

By mistake, I used the VHDL notation :(