1.1 We said this a few times, "Later assignments override earlier assignments in HDL coding".

<table>
<thead>
<tr>
<th>Code</th>
<th>Assignments</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>$A = 2$; $B &lt;= A$; $A = 3$; $C &lt;= A$;</td>
<td>______;</td>
<td>______;</td>
</tr>
<tr>
<td>#2</td>
<td>$A &lt;= 2$; $B &lt;= A$; $A &lt;= 3$; $C &lt;= A$;</td>
<td>______;</td>
<td>______;</td>
</tr>
<tr>
<td>#3</td>
<td>$A = 2$; $B &lt;= A$; $A = 3$; $C &lt;= A$;</td>
<td>______;</td>
<td>______;</td>
</tr>
<tr>
<td>#4</td>
<td>$A &lt;= 2$; $B &lt;= A$; $A &lt;= 3$; $C &lt;= A$;</td>
<td>______;</td>
<td>______;</td>
</tr>
<tr>
<td>#5</td>
<td>$sel ? A = 2 : A = 4'bZ$; $assign B = A$; $assign sel ? A = 4'bZ : A = 3$; $assign C = A$;</td>
<td>______;</td>
<td>______;</td>
</tr>
</tbody>
</table>

Do you call the assignments to $A$ in code #5 blocking or non-blocking or neither? _______________________

Are the four assign statements considered to be concurrent or sequential? _______________________

What would $B$ and $C$ become in each of the 5 separate codes?

#1 $B = ______; C = ______;
#2 $B = ______; C = ______;
#3 $B = ______; C = ______;
#4 $B = ______; C = ______;
#5 Assume select is true. $B = ______; C = ______;
#5 Assume select is false. $B = ______; C = ______;

1.2 Which of the two pieces of code on the right will produce a syntax error and why? ________________

Draw the hardware implied by the good code.

Complete the equivalent two assign statements:
assign $X = sel? ;$
assign $Y = sel? ;$
1.3  Schematic #1 (Sch#1) is what we wanted to design. A student drew Sch#2 which is clearly different from Sch#1.

Sch#3 is a copy of Sch#2 with an incomplete fix in the control lines of LOAD and EN for you to complete to make it functionally equivalent to Sch#1.

Two other students wrote verilog codes Code#4 and Code#5. Read the two codes and find if they represent Sch#1 or Sch#2 or neither. If it is neither, then you draw (on the back of the opposite page to show what that code represents.

Code#4 represents ______________ (Sch#1 / Sch#2 / neither)
Code#5 represents ______________ (Sch#1 / Sch#2 / neither)

1.4  Cascading two 3-bit counters to build a 6-bit counter: Complete the design below by adding overall enable EN6 for the 6-bit counter.

1.4.1 Given on the next page is a verilog code with two always blocks for the two 3-bit counters. Students are required to use two always block for the two pieces of the counter and thoughtfully add the cascading logic. Four students added cascade logic in four different ways. Some of them may be right. You find out. If all are wrong, you can fix any one of them to make it work.
Your conclusion: Write "works" or "doesn’t work" and write a short explanation.

<table>
<thead>
<tr>
<th>Student #1’s code</th>
<th>_______________</th>
<th>___________________________________________</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student #2’s code</td>
<td>_______________</td>
<td>___________________________________________</td>
</tr>
<tr>
<td>Student #3’s code</td>
<td>_______________</td>
<td>___________________________________________</td>
</tr>
<tr>
<td>Student #4’s code</td>
<td>_______________</td>
<td>___________________________________________</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Given</th>
<th>Student #1</th>
<th>Student #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>always @(posedge CLK) begin if (EN) Q[2:0]&lt;=Q[2:0]+1; end</td>
<td>always @(posedge CLK) begin if (EN) Q[5:3]&lt;=Q[5:3]+1; end</td>
<td>always @(posedge CLK) begin if (EN) Q[2:0]&lt;=Q[2:0]+1; end</td>
</tr>
<tr>
<td>always @(posedge CLK) begin if (EN6) Q[2:0]&lt;=Q[2:0]+1; if (EN6 &amp; (Q[2:0]==3’b111)) EN6Upper &lt;= 1’b1; else EN6Upper &lt;= 1’b0; end</td>
<td>always @(posedge CLK) begin if (EN6Upper) Q[5:3]&lt;=Q[5:3]+1; end</td>
<td>always @(posedge CLK) begin if (EN6) Q[2:0]&lt;=Q[2:0]+1; end</td>
</tr>
<tr>
<td>always @(posedge CLK) begin if (EN6) Q[2:0]&lt;=Q[2:0]+1; end</td>
<td>always @(posedge CLK) begin if (EN6 &amp; (Q[2:0]==3’b111)) EN6Upper &lt;= 1’b1; else EN6Upper &lt;= 1’b0; if (EN6Upper) Q[5:3]&lt;=Q[5:3]+1; end</td>
<td>always @(posedge CLK) begin if (EN6 &amp; (Q[2:0]==3’b111)) EN6Upper = 1’b1; else EN6Upper = 1’b0; if (EN6Upper) Q[5:3]&lt;=Q[5:3]+1; end</td>
</tr>
<tr>
<td>always @(posedge CLK) begin if (EN6) Q[2:0]&lt;=Q[2:0]+1; end</td>
<td>always @(posedge CLK) begin if (EN6 &amp; (Q[2:0]==3’b111)) EN6Upper &lt;= 1’b1; else EN6Upper &lt;= 1’b0; if (EN6Upper) Q[5:3]&lt;=Q[5:3]+1; end</td>
<td>always @(posedge CLK) begin if (EN6 &amp; (Q[2:0]==3’b111)) EN6Upper = 1’b1; else EN6Upper = 1’b0; if (EN6Upper) Q[5:3]&lt;=Q[5:3]+1; end</td>
</tr>
</tbody>
</table>
2 ( 23 points) 15 min. Fixed Priority Resolver, Daisy chaining

2.1 Understand the top-left 2-input 2-output FPR, and complete the remaining three.

<table>
<thead>
<tr>
<th>High-active inputs and high-active outputs</th>
<th>High-active inputs and low-active outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

2.2 To make the above "cascadable" in a daisy chain fashion, let us add a "PRN" (priority in) input and a "PRO" (priority out) outputs. On the left below, a high-active cascadable building block was provided in *a nearly completed* fashion. Two such pieces were cascaded below in a daisy chain fashion. Complete that active-high building block and also build a similar building block on upper right side below with low-active inputs and low-active outputs.

![Diagram](image3.png)

(Mr. Bruin/Mr. Trojan) says that the active-high and active-low building blocks are interchangeable.

If you have 32 requests (and 32 grants) (all active high), how many active-high building blocks are needed in the daisy chain? ________

If inverters have 5 ps delay and other multi-input gates have 10 ps delay, which of the 16 grants (G0 to G15) will come out last and what is that longest delay? __________

A pico second is equal to ________ (10^{-3} / 10^{-6} / 10^{-9} / 10^{-12} / 10^{-15} ) seconds
3 (28 points) 15 min. Memory

Memory width and depth expansion: We needed an 8Kx8 ROM memory (with an overall CE chip-enable) using the 4 smaller ROM chips. Miss Bruin was not taught about natural boundaries of address ranges and has arranged the four chips as shown on the side. Since ROMs were already programmed with the needed content, we cannot change the map. So manage to complete the design below to suit.

The 8Kx8 ROM has _____ address pins and _______ data pins.

Complete the design below. Add missing labels, wires, and gates.

An 8K range in hex is from 0000 Hex to __________ hex.

Tell Miss Bruin the range of address implied by 4K placement in the map: _____ H to _____ H

3.1 Given below is the address map of a 8088 processor in a PC-XT computer (of 1980’s). You know that the 8088 processor has 1 Megabyte address space (address range: 00000 H - FFFFF H).

Find the size and address range of the shadowed area in the map below.

Size: _____ KB = _______ Bytes

=> Address pins A[ :0] change from 0’s to 1’s

© Copyright 2013 Gandhi Puvvada
4 (27 points) 15 min.

Barrel Shifters and Rotating Prioritizers:

4.1 2-input 2-output rotating prioritizer design: (a) complete the 2-input 2-output barrel shifter (cross switch on the left-end below) by labeling the internal I inputs appropriately. (b) then complete the rotating prioritizer design on the right by placing an inverter or just a wire in the box below. You need to explain adequately (why you chose to put an inverter or a wire) to gain points on this.

Explain:

4.2 In our class we talked about Barrel shifters of size 2, 4, 8, 16 (all powers of 2) and discussed how we can reduce the cost by using logarithmic barrel shifters. But if we need to implement a rotating prioritizer for 3 requests and 3 grants, we need two 3-input 3-output barrel shifters and a fixed priority resolver.

Design a 3-input 3-output barrel shifter (a) by using three of 3-to-1 muxes (b) by replacing each of the 3-to-1 muxes with two of 2-to-1 muxes.
State diagram design:

You are given a sorted array $M$ of ten 4-bit unsigned numbers. $M[0]$ is the smallest and $M[9]$ is the largest. We need to copy the elements of array $M$ to array $N$. However, while in array $N$, every number is treated as a signed number represented in Sign-Magnitude system. You know that sorting will be different for signed numbers.

So, if you consider the original array $M$ of unsigned numbers as made up of two separate chunks of sorted numbers, first chunk of numbers starting with "0" in MSB and the second chunk of numbers starting with "1" in MSB, then to resort and arrange them in array $N$, we need to simply copy the second chunk (of array $M$) as the first chunk (of array $N$) in reverse order and then copy the first chunk (of array $M$) as the second chunk (of array $N$) in original order as shown in the example below.

Note: the array $M$ can have only one chunk (i.e. all numbers starting with either "0" or "1").

Implementation #1: In order to reduce the wastage of clocks, we start "I", the index into memory array $M$ at 9. The memory array $N$ is dual-ported and the two indexes "J" and "K" are initialized to 0 and 9 respectively as shown. So, for the above example, we start with $N[J] <= M[I]$, decrementing $I$ and incrementing $J$. Once we finish the chunk #2, we start doing $N[K] <= M[I]$, decrementing $I$ and decrementing $K$. All this is done in the CBC "copy both chunks" state.
5.2 Implementation #2:
Here, we reverted the memory N back to single-ported. Only single index J is used to write to N. It is initialized to 0. For the data given in the example, I is decremented to process chunk #2 in M in reverse direction and then I jumps and processes the chunk #1 in M in forward direction (I is then incremented). We do not want to lose a clock when I is just jumping to the beginning of the M array. So we suggest that when needed you can apply a constant address say "4'b0000" in place of I to M. If you do so, do you want I to still jump to 0 (4'b0000) or 1 (4'b0001)? _________________

C221 Copy 2 to 1 (= Copy Chunk 2 of M to Chunk 1 of N)  
C122 Copy 1 to 2 (= Copy Chunk 1 of M to Chunk 2 of N)

After copying one chunk, "I" has to jump. But "J" goes on! So, instead of worrying about what value of "I" indicates the end of copying the second chunk, one can look at the terminal value of "J". In fact, if J is at its terminal value, can we go to DONE state, whether we are in C221 or C122? Yes / No.

If you are not wasting even one clock during transition from chunk #2 to chunk #1 of array M, are the total number of clocks (spent in C221 and C122 together) fixed or is it data-dependent? Explain: _____________________________________________________________________  
____________________________________________________________________________

As per your design, what kind of data will not make I increment at all? ___________________
____________________________________________________________________________

As per your design, what kind of data will not make I decrement at all? ___________________
____________________________________________________________________________

Complete the state diagram below.
5.2 Implementation #2:
Here, we reverted the memory N back to single-ported. Only single index J is used to write to N. It is initialized to 0. For the data given in the example, I is decremented to process chunk #2 in M in reverse direction and then I jumps and processes the chunk #1 in M in forward direction (I is then incremented). We do not want to lose a clock when I is just jumping to the beginning of the M array. So we suggest that when needed you can apply a constant address say "4'b0000" in place of I to M. If you do so, do you want I to still jump to 0 (4'b0000) or 1 (4'b0001)?

C221  Copy 2 to 1 (= Copy Chunk 2 of M to Chunk 1 of N)
C122  Copy 1 to 2 (= Copy Chunk 1 of M to Chunk 2 of N)

After copying one chunk, "I" has to jump. But "J" goes on! So, instead of worrying about what value of "I" indicates the end of copying the second chunk, one can look at the terminal value of "J". In fact, if J is at its terminal value, can we go to DONE state, whether we are in C221 or C122? Yes / No.

Since we can not help wasting one clock during transition from chunk #2 to chunk #1 of array M, are the total number of clocks (spent in C221 and C122 together) fixed or is it data-dependent? Explain:

As per your design, what kind of data will not make I increment at all?

As per your design, what kind of data will not make I decrement at all?

Complete the state diagram below.
5.3 Design the "I" counter data path for this implementation #2 in two ways. First figure out how many different values (either constants or variables (variable could be an expression like I+1) and list them below. We do not mind if I goes to Ten as we arrive in the DONE state but I should not change in the DONE state.

In INI state, I is initialized to _______. In C221 state, I may be initially ___________________ (incrementing / decrementing) then jumping to ___________________.

In C122 state I is __________. In Done state I should ___________________________.

All together _________ different items are trying to get into I.

Method #1 Complete the data path using muxes. Add as many 2-to-1 muxes as needed. Produce the control lines to control the select lines. Note that you need only 6 2-to-1 muxes to select one of a set of 7! If 7 is the number you need to focus on how to get 6 of the 7 to the input of the I register, If the situation is such that none of these 6 is supposed to go into the I register, by elimination, the 7th value arrives at the I input.

Redesign the data path using tristate buffers which allow us to focus on one item at a time. Use as many (4-bit) tristate buffers as needed to bring different values for I on to the tristate bus and convey it to the I register. Produce EN (enable) logic for each tristate buffer. I have randomly added 6 tristate buffers. You can add more or delete excess as needed.
6 ( 31 points) 20 min. FIFOs

6.1 A change in ___________ (depth / width) of the FIFO causes a change in the pinout of the FIFO because __________________________________________________________
____________________________________________________________________________

6.2 We have a 16-loc. 2-clock FIFO with 5-bit (not 4-bit) pointers (counters) for WP and RP. As you know a 5-bit counter goes from 00000 to 11111 and then rolls over to 00000. WP moves and RP follows. WP - RP also produces a 5-bit difference. While a 5-bit variable can have 32 values 00000 to 11111 (both inclusive), only ___________ (state a number) values of these 32 are legal and the remaining ___________ (state a number) values are illegal.

We ________ (can / can’t) use less than 5-bit pointers ________ (but / and also) we ________ (can / can’t) use more than 5-bit pointers.

Is it true that though WP can go through 32 different values, since there are only 16 locations in the FIFO, a WP of 00001 and a WP of 10001 both point to the same physical location 0001 in the 16 location FIFO? True / False

If WP is 10001 and RP is 00001, do you agree that the physical location 0001 is filled? ________
Yes / No

Do you wish to say that (a) the logical location 00001 (RP pointer value) is filled and the logical location 10001 (WP pointer value) is not filled or (b) the other way around? ________ [(a) / (b)]

Starting from reset (when WP = RP = 00000), if the producer has deposited 86 (decimal 86) items and the consumer should have at least consumed ____________ values and at most consumed ____________ values. Based on this, arrive at the 5-bit value for the WP and the 5-bit range of values for RP.

5-bit value for the WP ____________________
5-bit range of values for RP _____________________________

6.3 Gray code counters are needed in _______________ (single-clock / two-clock / both / neither) FIFO.
The 16-bit Gray code 1010_1010_0100_1111 represents an _______ (odd / even) decimal number.
The 16-bit Binary code 1010_1010_0100_1111 represents an _______ (odd / even) decimal number.

We enjoyed teaching this course! Hope you liked it! Hope to see some of you in EE457. Grades will be out in a week. Enjoy your winter break!
Happy Holidays!!! - Gandhi, Brandon, Chris, Jizhe, Vishal, Feng