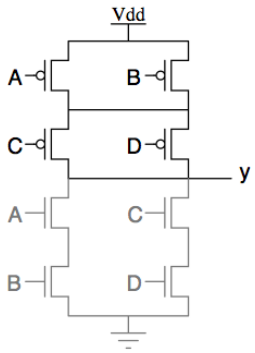
Problem 1.

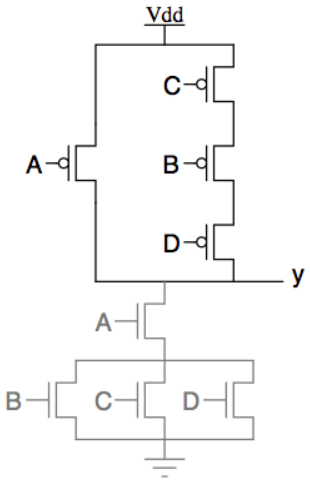
In the follows answers, the black section of the circuit is the new section, while the grey section is the section that was given in the problem.

a)



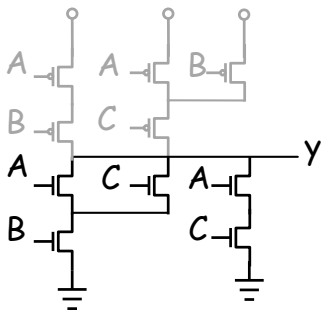
A	B	C	D	y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

b)



A	B	C	D	y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

c)



A	B	C	y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

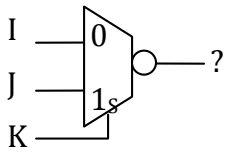
Problem 2.

K	J	I	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

The basic approach to solving logic problems is to apply a process of systematic elimination (i.e. eliminate impossible outputs for a given combination of inputs).

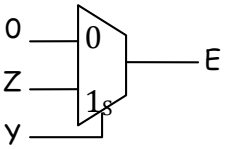
a) First consider those functions that can be implemented using only an inverting 2-input mux. The truth table for such a mux is shown on the right with the inputs labeled according to the given variables.

For example, in the first row the output of function E is 0. In order to generate this value, either one of I or J must be 1 (hence neither can be connected to X, Y, or Z) or input I must be tied to 1 if any of X, Y or Z are connected to K, or both J and K are connected to 1.
Likewise, the only way E could output a 1 when the 3 inputs are 1s (last row) is if J is tied to a 0.



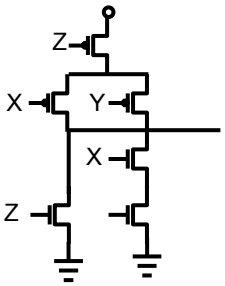
K=Z, I=X, J=Y yields function C

b) Using similar elimination method as in part a), it can be found the E is the only possible result.

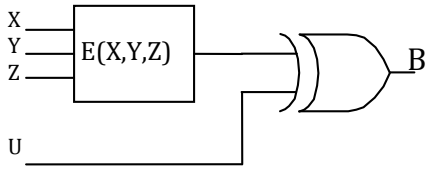


c) Only function A can be implemented as a single-level CMOS gate.

Function A, as follows:



d) An implementation of function E can be used to compute function B with the following circuit, if the unknown input, U, is properly connected.



What signal should be connected to U? X

Problem 3.

a) Give binary values for I_0, I_1, I_2, and I_3 which implement the following functions on the two inputs A and B: AND(A,B), OR(A,B), XOR(A,B), NAND(A,B), and NOR(A,B)
(2 pts).

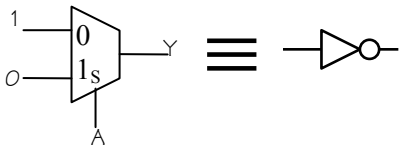
Gate	I_0	I_1	I_2	I_3
AND(A,B)	0	0	0	1
OR(A,B)	0	1	1	1
XOR(A,B)	0	1	1	0
NAND(A,B)	1	1	1	0
NOR(A,B)	1	0	0	0

b) Can every 2-input Boolean function be implemented using Ben's structure? Explain why or why not. (2 pts)

Every 2-input Boolean function can be implemented using a 4 input mux by connecting the two inputs to the select lines and copying the function's truth table values to their corresponding input.

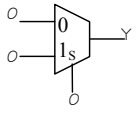
c) Show how to implement an inverter, as well as every 2-input gate using no more than 2-multiplexers to construct each one. (16 points)

An inverter can be implemented using a two input mux as follows:

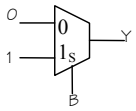


All other gates can be implemented using no more than 2 muxes as follows (using the inverter implementation shown above):

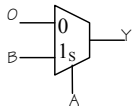
Zero		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	0



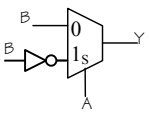
B		
A	B	Y
0	0	0
0	1	1
1	0	0
1	1	1



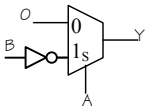
AND		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



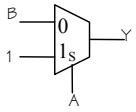
XOR		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



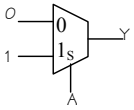
A > B		
A	B	Y
0	0	0
0	1	0
1	0	1
1	1	0



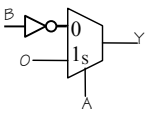
OR		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



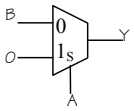
A		
A	B	Y
0	0	0
0	1	0
1	0	1
1	1	1



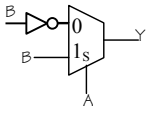
NOR		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



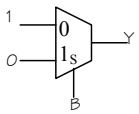
B > A		
A	B	Y
0	0	0
0	1	1
1	0	0
1	1	0



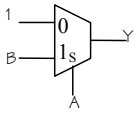
XNOR		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1



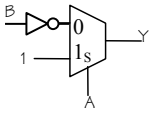
NOT B		
A	B	Y
0	0	1
0	1	0
1	0	1
1	1	0



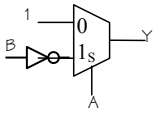
B ≥ A		
A	B	Y
0	0	1
0	1	1
1	0	0
1	1	1



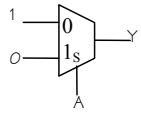
A ≥ B		
A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1



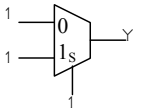
NAND		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



NOT A		
A	B	Y
0	0	1
0	1	1
1	0	0
1	1	0



ONE		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	1



d) Explain how every 4-input Boolean function can be implemented with an 8-input mux and a single inverter.

For the 4-input function f(A,B,C,D), we can choose A, B, and C to be the select inputs of the 8-input mux. With this arrangement, each of the 8 data inputs can be set to one of four values: 0, 1, D, and $\overline{D}$. (We can obtain the value of $\overline{D}$ using the single inverter given to us.) To determine which of the four values a data input should be attached to, one can examine the two corresponding rows for a given A, B, C combination in the truth table. If both values are 0, then the corresponding mux input is connected to 0. Similarly, if both values are 1 the mux input is set to one. If the two outputs for a given A, B, C combination are a 0 and a 1, then they must either follow D or $\overline{D}$, and the mux input should be connected accordingly.

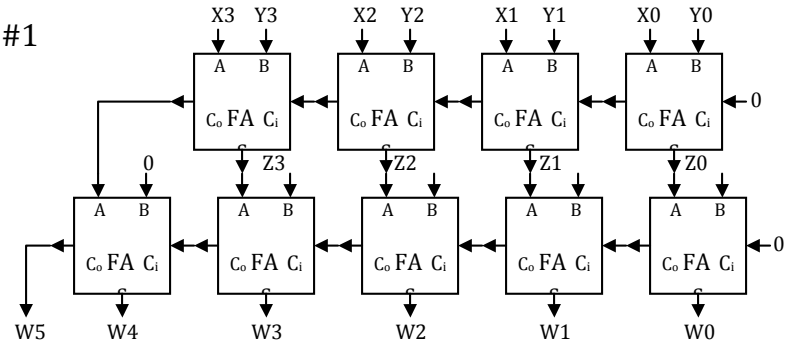
e) The TA explains to Lee that no more than 40% of all 4-input Boolean functions can be implemented with only an 8-input mux (i.e. no inverter). He jots down an upper bound on the number of such functions: $3^8/2^{14}$. Can you explain the TA's reasoning?

With no inverter, each of the mux's data inputs can be set to one of only three possible values: 0, 1, and D. As a result, there are 3^8 permutations of inputs to this mux, so only 3^8 Boolean functions can be implemented in this configuration. However, instead of using D as a possible data input, we can swap D with one of the three select inputs (A, B, or C). (This allows some functions that would have used $\overline{D}$ as a data input – functions where D=1 leads to f=0 and D=0 leads to f=1.) By choosing to use either A, B, C, or D as the data input, there are $3^8 * 4$ possible functions that can be implemented (Some of these functions are the same, so $3^8 * 4$ is an upper bound). Since there are a total of $2^{(2^4)} = 2^{16}$ possible 4-input Boolean functions, an upper bound on the percentage of 4-input functions that can be implemented using an 8-input mux without an inverter is $[(3^8 * 4) / 2^{16}]$, which is about 40%.

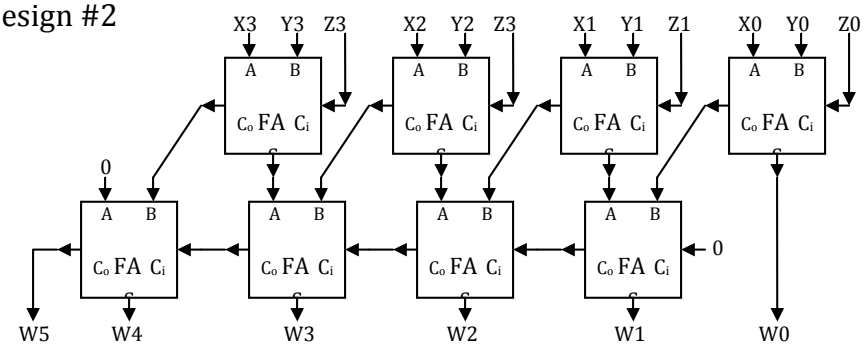
Problem 4)

Consider the follow two circuits for adding 3, 4-bit numbers (i.e. W = X + Y + Z):

Design #1



Design #2



a) Ignoring the speed of the calculation, do both 3-input adder designs compute the same result? If not, how do they differ? (5 pts)

Yes, they both compute the same result

b) Assume a unit delay for each full-adder (i.e. the output will become valid one time unit after all inputs are valid). Which design computes all bits of the sum fastest? Which design computes the first (least-significant) bit fastest? (5 pts)

By following the path of maximum delay in both circuits, one finds that Design #1 takes 6 unit delays in the worst case, and Design #2 takes 5 unit delays in the worst case. Also, Design #1 takes 2 unit delays to compute first bit, and Design #2 takes 1 unit delay to compute first bit.

c) Obviously, design #1 uses one more adder than design #2. Explain how design #1 can be modified slightly to use one fewer full-adder. After this change, which design computes all bits of its sum fastest? (5 pts)

Eliminate rightmost adder on bottom row of Design #1 and connect Z0 to the carry input of the rightmost adder on the top row, instead of 0.

d) Both designs could be improved by incorporating carry-lookahead logic in place of their carry-propagation chains. Which design would require less logic to implement carry lookahead? Explain. (5 pts)

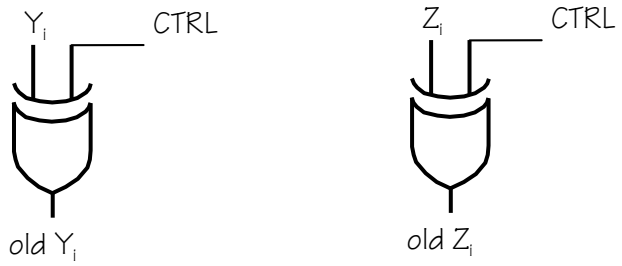
Compare number of carry-propagation chains. Since Design #1 has two and Design #2 has one, Design #2 would require less logic (lesser change) since only one chain has to be fixed.

e) Modify both designs to support the following operations:

$$W = X + Y + Z \quad \text{or} \quad W = X + Y - Z,$$

based on the value of a control signal (CTRL). (5 pts)

In each design, replace inputs connected to Z_i with:



Also, in the first design connect the carry-in of the upper right adder (currently connected to 0) to CTRL.

In the second design a half-adder need to be added, that adds CTRL to the output of the first adder (with output labeled WO). The sum of this adder will be the new "WO" and the carry will be fed into the first adder in the second row, which currently is connected to a '0'.