CSE370 HW3 Solutions (Winter 2010)

1. **CLD2e, 4.9**
   We are asked to implement the function f(A,B,C,D,E) = A + C'D + BD' + B'D + B'CE using the smallest possible multiplexer. We can't use any extra gates or the complement of a variable.

   Looking at the function it appears that A and E are good potential candidates for eliminating since they only appear in one term each and their complements are not used in any of the terms.

   If we build the truth table for the case where we try to get rid of E (left) or A (right), we get the following tables:

A	B	C	D	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	E
0	0	1	1	1
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	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

B	C	D	E	f
0	0	0	0	A
0	0	0	1	A
0	0	1	0	1
0	0	1	1	1
0	1	0	0	A
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	A
1	1	1	1	A

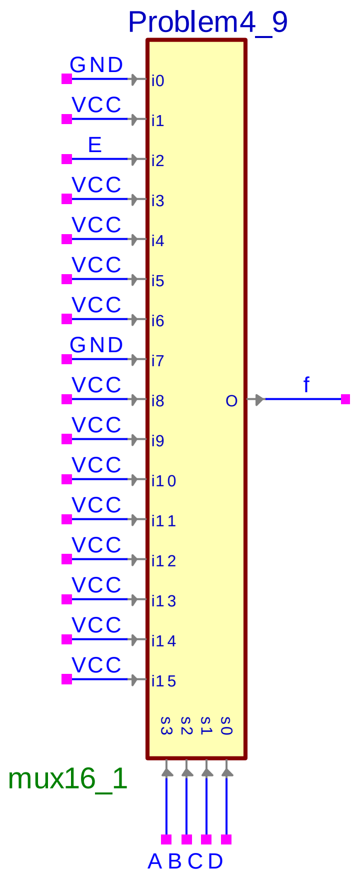
   Note: To fill in the truth tables I go through each term in the equation and fill in the rows that must always be 1's. Then for the remaining blanks I look first to see if any term could make it true, if not it gets a zero. If one of the terms could make it true I check if it's true when the variable we eliminated is true or not. This gives us whether we should use the complement or not.

   Both work, but now we need to check if we can eliminate both A and E to use an even smaller MUX:

B	C	D	F
0	0	0	A
0	0	1	1
0	1	0	A+E
0	1	1	1
1	0	0	1
1	0	1	1

1	1	0	1
1	1	1	A

In this case we would need an OR gate to use a smaller MUX, so the previous solution(s) are the best we can do. Here is the resulting circuit:



2. **CLD2e, 4.12 part a**
We have the function f(A,B,C,D,E,F) = ∑ m(3, 7, 12, 14, 15, 19, 23, 27, 28, 29, 31, 35, 39, 44, 45, 46, 48, 49, 50, 52, 53, 55, 56, 57, 59).

I am first going to show how we can construct this using a 32:1 MUX. We can build a 32 row truth table and use the variable F as the input variable. I chose F because it is the lowest order bit, so it lets me use a trick to quickly fill in the truth table.

Also, to save space I am going to write the 32 row truth table as a 16 row truth tables with two outputs, the first set of outputs are with the high order bit A is 0 and the second set it when it's 1. For illustrative purposes I have also included the minterms that affect each row of the output.

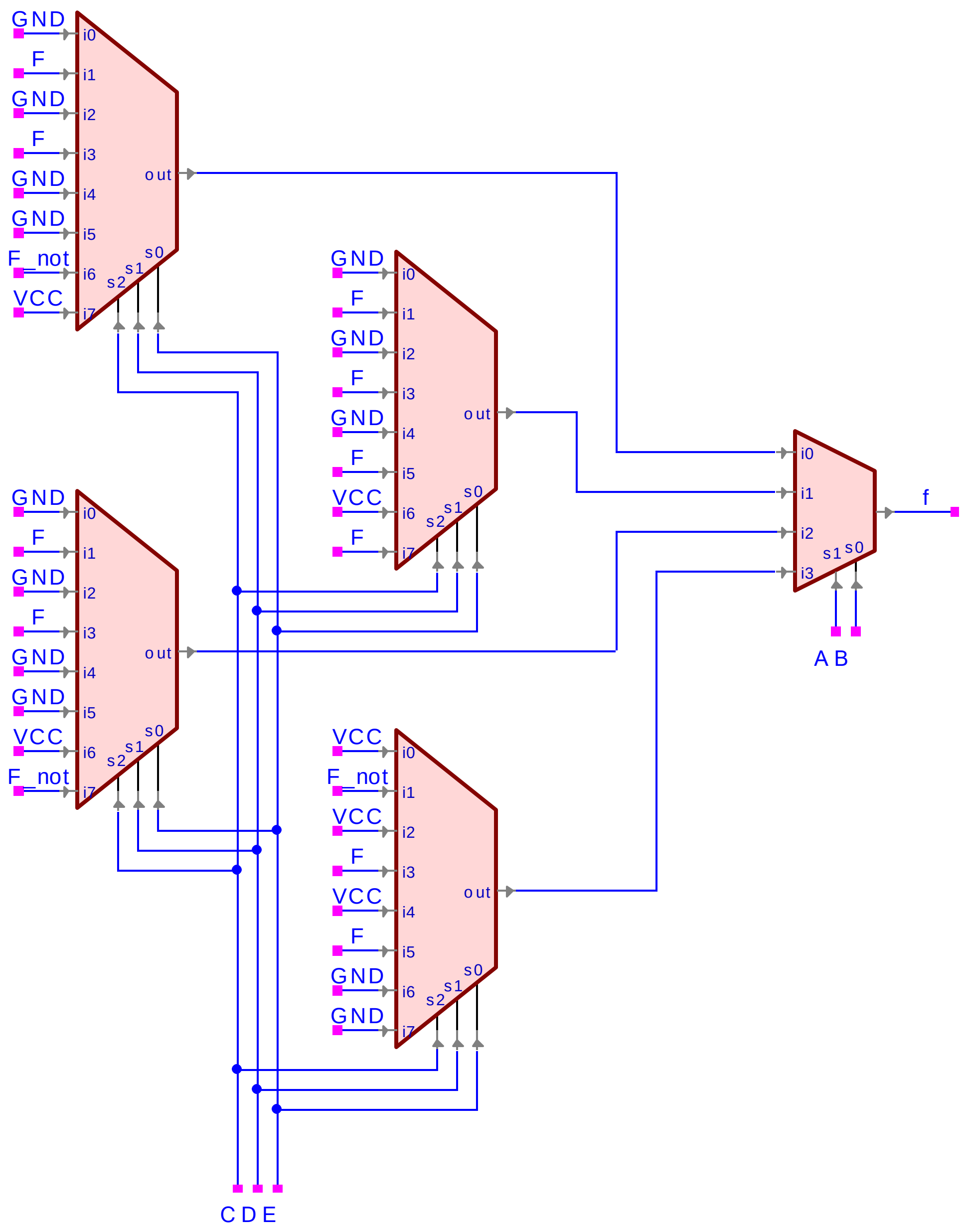
				A=0		A=1	
B	C	D	E	m’s	f	m’s	f
0	0	0	0	m0,m1	0	m32,m33	0
0	0	0	1	m2,m3	F	…	F
0	0	1	0	m4,m5	0	…	0
0	0	1	1	m6,m7	F	…	F
0	1	0	0	m8,m9	0	…	0
0	1	0	1	…	0	…	0
0	1	1	0	…	F’	…	1
0	1	1	1	…	1	…	F’
1	0	0	0	…	0	…	1
1	0	0	1	…	F	…	F’
1	0	1	0	…	0	…	1
1	0	1	1	…	F	…	F
1	1	0	0	…	0	…	1
1	1	0	1	…	F	…	F
1	1	1	0	…	1	…	0
1	1	1	1	m30,m31	F	m62,m63	0

To fill in this table, we now can just look at all the minterms in the equation that correspond to each row. If both are there, then output 1. If neither is there, then output 0. If the first is there and not the second, then output F’. Otherwise, output F.

Once we have the truth table we just need to figure out how to realize the circuit using the MUXes we have. We need 32 inputs so let’s use 4 8:1 MUXes and select using the low order bits (C,D,E), then route these 4 outputs into a 4:1 MUX and select the final result using the two high order bits (A,B).

Note: Some people didn’t go ahead and draw this circuit. It’s tedious, but good practice to make sure you understand how everything hooks together.

We get the resulting circuit diagram:



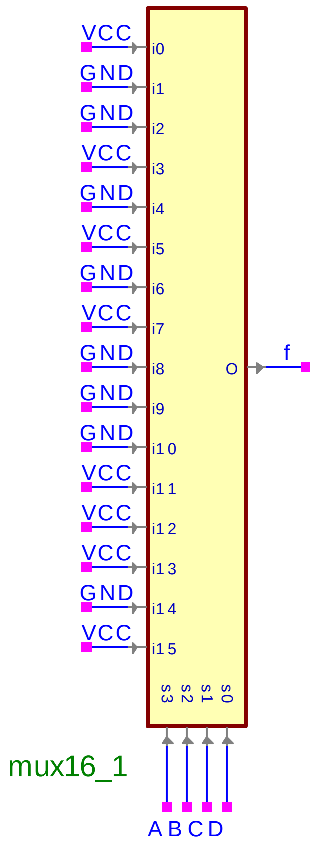
This used 4 8:1 MUXes and a single 4:1 MUX, for a total of 5 chips!

3. CLD2e, 4.19

Consider f(A,B,C,D) = ∑ m(0, 3, 5, 7, 11, 12, 13, 15)

a. Part a

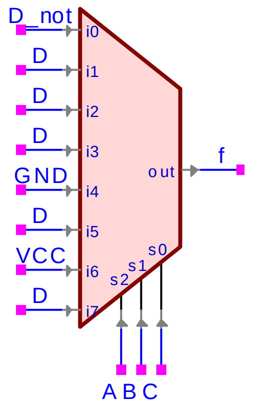
This first part is trivial since we are given the minterms we just set each input corresponding to the true minterms equal to 1 and the others equal to 0. This gives use the resulting 16:1 MUX.



b. Part b

Now to use only an 8:1 MUX we need to eliminate one of the variables. Let's choose D for simplicity. The truth table for the function and the corresponding circuit become:

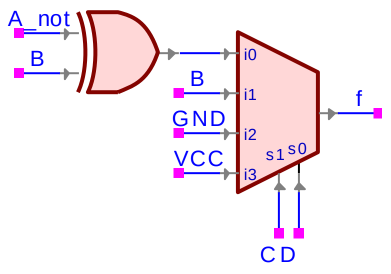
A	B	C	D	f	
0	0	0	0	1	D'
0	0	0	1	0	
0	0	1	0	0	D
0	0	1	1	1	
0	1	0	0	0	D
0	1	0	1	1	
0	1	1	0	0	D
0	1	1	1	1	
1	0	0	0	0	0
1	0	0	1	0	
1	0	1	0	0	D
1	0	1	1	1	
1	1	0	0	1	1
1	1	0	1	1	
1	1	1	0	0	D
1	1	1	1	1	



c. Part c

As was pointed out, we can't use a single gate if we use A and B as the select inputs. So instead we will use C and D as the select inputs and come up with functions in terms of A and B. This gives the following truth table and corresponding circuit:

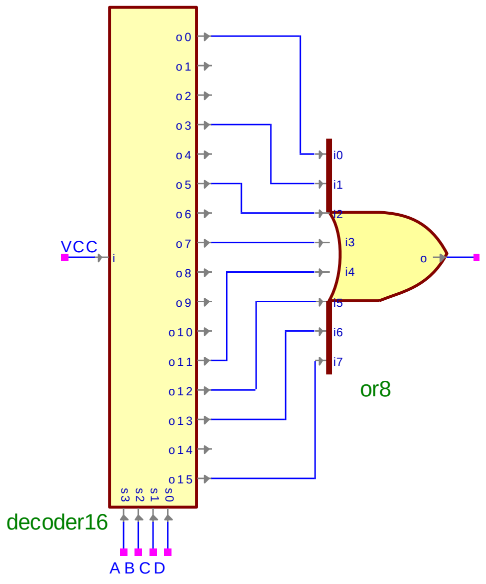
C D	A	B	F	
00	0	0	1	A' XOR B
	0	1	0	
	1	0	0	
	1	1	1	
01	0	0	0	B
	0	1	1	
	1	0	0	
	1	1	1	
10	0	0	0	0
	0	1	0	
	1	0	0	
	1	1	0	
11	0	0	1	1
	0	1	1	
	1	0	1	
	1	1	1	



If you instead used A and B as the select input you would get the following as inputs to your 4:1 MUX: C' XOR D, D, C & D, and C' + D, which takes several gates more than the solution above.

d. Part d

Following the procedure that we learned in class to use a 4:16 decoder we can OR all the outputs that correspond to the true minterms. This givens the following circuit:



4. CLD2e, 3.19

Let M be the value after the leftmost XOR gate and let N be the value after the leftmost NAND gate. We then get the following truth table for the circuit:

		M		N		F	G
A	B	A XOR B	AB	(AB)’	AM	(AM)’	N XOR M
0	0	0	0	1	0	1	1
0	1	1	0	1	0	1	0
1	0	1	0	1	1	0	0
1	1	0	1	0	0	1	0

This gives us the truth tables:

F	A	
B	1	0
	1	1

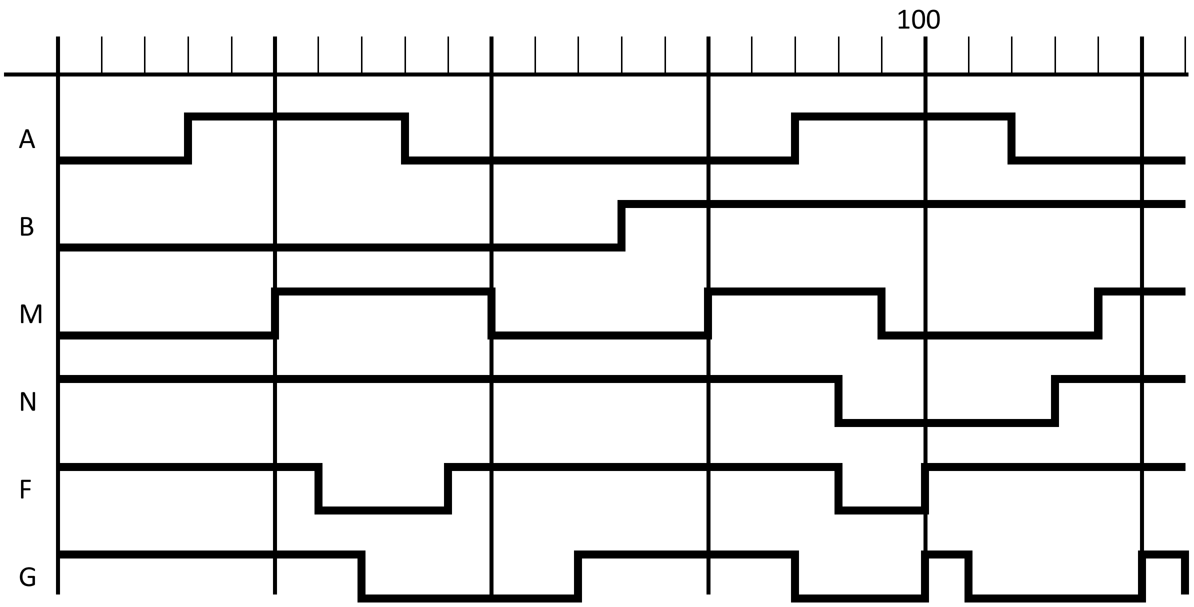
G	A	
B	1	0
	0	0

Which gives the reduced equations:

F = A’ + B

G = A’B’

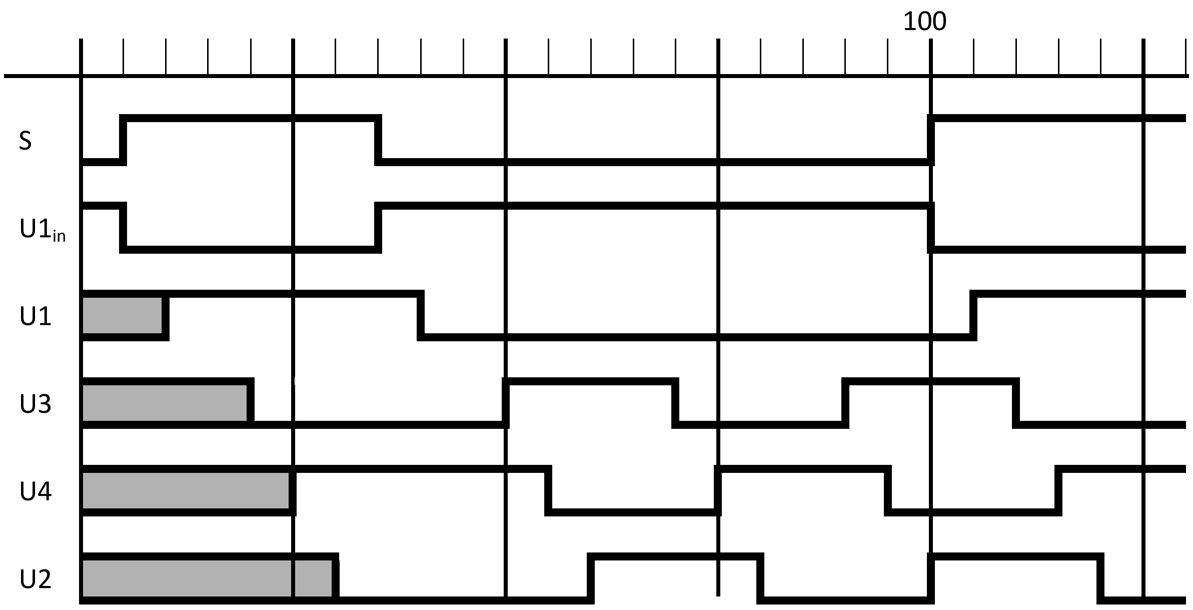
Assuming a delay of 5 for the NAND and a delay of 10 for the XOR:



5. CLD2e, 3.21

a. Part a

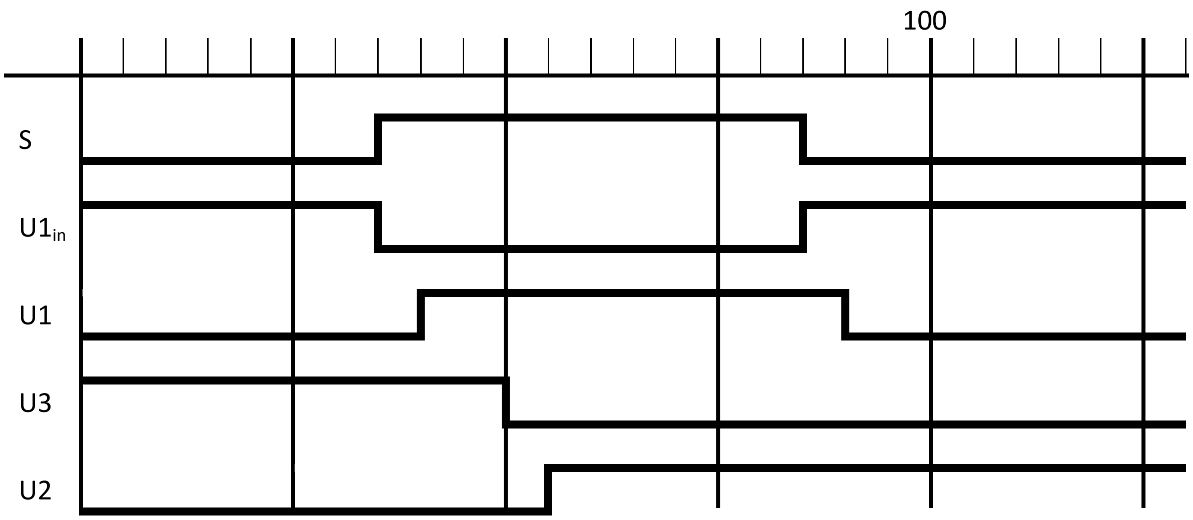
For this circuit I am assuming that the inverters have 5 delay and the NOR gate has a delay of 10. This gives us the following timing diagram:



We can see the when the switch is closed the circuit oscillates between 0 and 1 with the delay of the OR and two inverters.

Note: A lot of people showed how it oscillated but didn't show how it stops once the switch closes.

b. **Part b**
For this circuit I am assuming that the inverters have 5 delay and the NOR gate has a delay of 10. This gives us the following timing diagram:



The circuit does not oscillate, in fact the output of U3 is 1 until the switch is closed and then remains 0 forever because the input U2 becomes high.

Note: Most people said that it just stays 0 all the time. What is really going on though is that it could start out at 1 (when the switch is open), and the first time the switch closes it will go to 0 and will remain zero no matter what happens after that point.

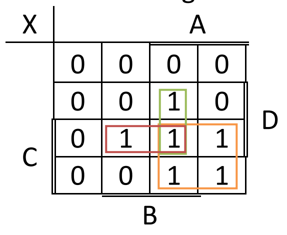
Note: A couple people incorrectly asserted that it doesn't oscillate because there is less delay in the circuit. This is not the case, the reason it doesn't oscillate is because the output of U3 is only inverted once, and so (coupled with the NOR gate) as soon as the output becomes 0 the input will always have a 1 on it.

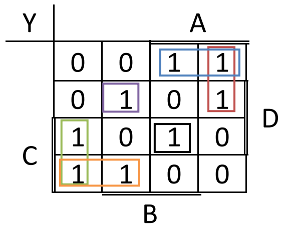
6. **CLD2e, 3.26**
Consider an adder of 2 2-bit numbers.

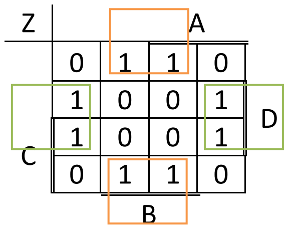
a. **Part a**
The truth table for this circuit is as follows:

A	B	C	D	X	Y	Z
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

b. **Part b**
We get the following three k-maps for X, Y, and Z:







This gives us the following equations:

X = AC + ABD + BCD
Y = AC'D' + AB'C' + A'B'C + A'CD' + ABCD + A'BC'D
Z = BD' + B'D = B XOR D

c. **Part c**
From looking at Figure 3.32 we can see that the delay for chaining two full adders will be 4 levels of logic (ignoring inverters and ignoring the fact that there is 4-input OR gate).

On the other hand the formulae derived above are all 2 level logic. So it is approximately twice as fast.

Note: Some people were confusing size of the circuit with the delay. What this question was hoping people would realize is that the circuit we designed is bigger (more total gates), but faster because we have fewer levels of logic. The calculations aren't precise because we ignore things like inverters and # of inputs to the gates, but "levels of logic" is a good approximation in many cases. We see this same space/speed tradeoff in the Adders section of the book.

7. **CLD2e, 3.32 (follow same instructions as 3.28)**
The point of this problem was to give you practice using Verilog and start thinking about how you can represent things in multiple ways using Verilog.

Using the Boolean expressions from 3.26 we get the Verilog:

```
module adder2x2bit(A, B, C, D, X, Y, Z);
     input A, B, C, D;
     output X, Y, Z;

     assign X = (A & C) | (B & C & D) | (A & B & D);
     assign Y = (A & ~C & ~D) | (A & ~C & ~B) |
                (~A & ~B & C) | (~A & C & ~D) |
                (~A & B & ~C & D) | (A & B & C & D);
     assign Z = B ^ D;
endmodule
```

We could have also written this more structurally (using "and" and "or" modules).

The second part of the problem asks us if there is another way to write it that describes the behavior instead of the logic. Here they were trying to get us to realize that since we are doing addition, we can use the "+" operator.

```
module adder2x2bit(A, B, C, D, X, Y, Z);
     input A, B, C, D;
     output X, Y, Z;

     assign {X,Y,Z} = {A,B} + {C,D};
endmodule
```

Note: A lot of people didn't read the question carefully and just did 3.32. By following the instructions of 3.28, the idea was to let you see the difference between describing the logic of a circuit using Boolean Logic, versus expressing directly what the circuit is trying to calculate.

Note: Also a lot of people added a line: "reg X, Y, Z;". I think lecture slide 9 had a slight error. In general you need to use the "reg" keyword for assignments done in an "always" block, but not with "assign" statements outside an "always" block. Active HDL may let you get away with it though.