Homework #2 (v20170501)

(Deadline: 11:59PM PDT, May 5, 2017)

Name (Last, First): Miller, Joseph

Student Id #: 504-744-848

INSTRUCTIONS

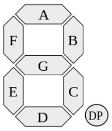
This homework is to be done individually. You may use any tools or refer to published papers or books, but may not seek help from any other person or consult solutions to prior exams or homeworks from this or other courses (including those outside UCLA). You're allowed to make use of tools such as Logisim, WolframAlpha (which has terrific support for boolean logic) etc.

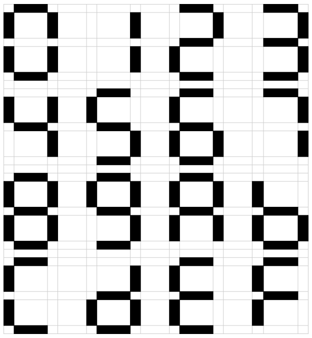
You must submit all sheets in this file based on the procedure below. Because of the grading methodology, it is much easier if you print the document and answer your questions in the space provided in this problem set. It can be even easier if you answer in electronic form and then download the PDF. Answers written on sheets other that the provided space will not be looked at or graded. Please write clearly and neatly - if we cannot easily decipher what you have written, you will get zero credit

SUBMISSION PROCEDURE: You need to submit your solution online at Gradescope (https://gradescope.com/). Please see the following guide from Gradescope for submitting homework. You'd need to upload a PDF and mark where each question is answered.
http://gradescope-static-assets.s3-us-west-2.amazonaws.com/help/submitting_hw_guide.pdf

Problem #1

A 7-segment display can be used not only to display numbers from 0-9 but can be adapted to display hexadecimal digits as well up to F allowing one to fully use the 4-bits encoding (see figure on the right below for the characters from 0-F). Determine the minimum logic by using AND, and OR gates (you may assume inverted inputs are also available). Note that your implementation should result in the least number of inputs to the AND, OR gates you use.





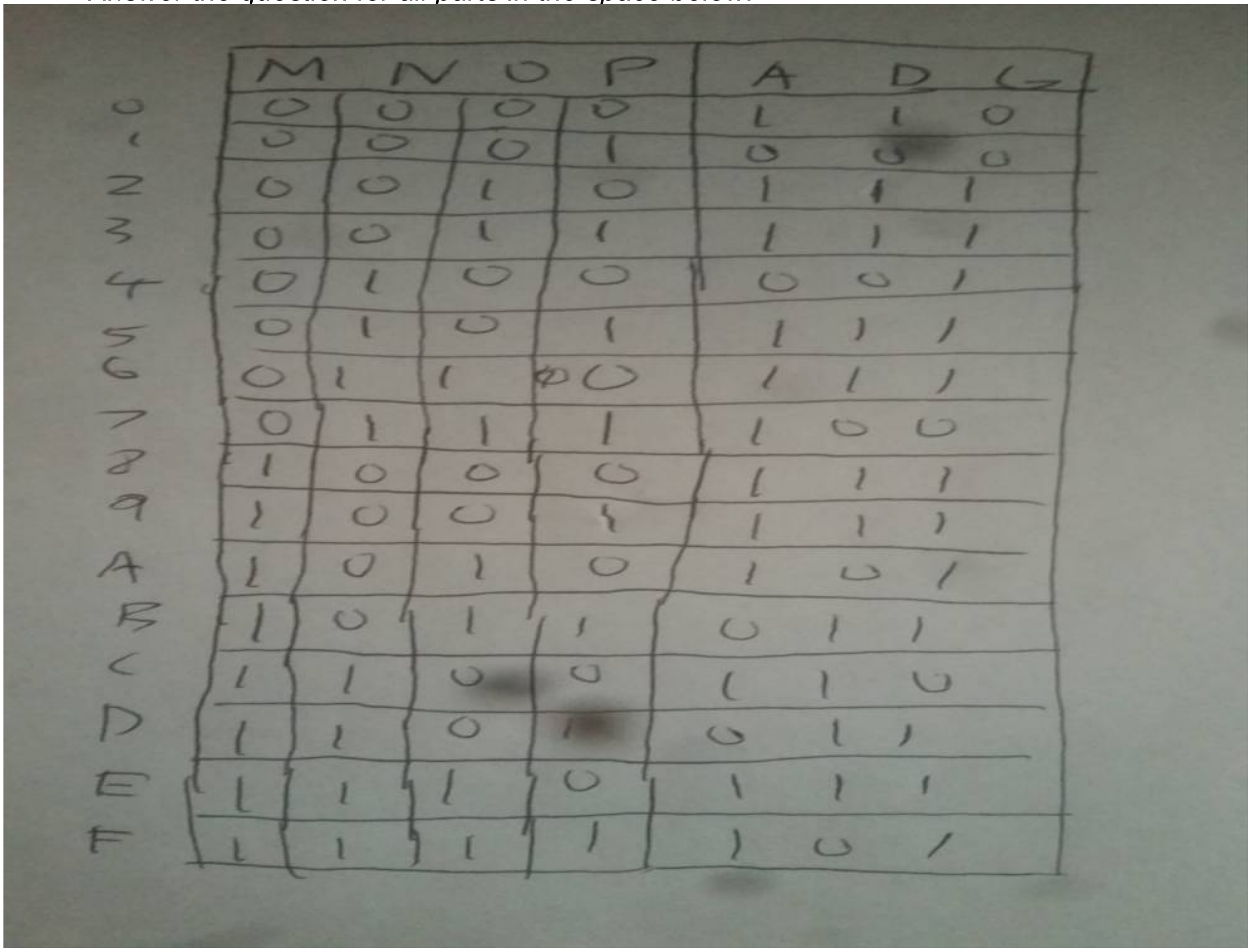
(a) Write a single truth table (4 inputs, d[3:0])for the segments A, D, and G (3 outputs) and implement Segment A

(b) Implement Segment D

(c) Implement Segment G

(d) If all three outputs are to be simultaneously implemented, show how the amount of logic can be reduced.

Answer the question for all parts in the space below.

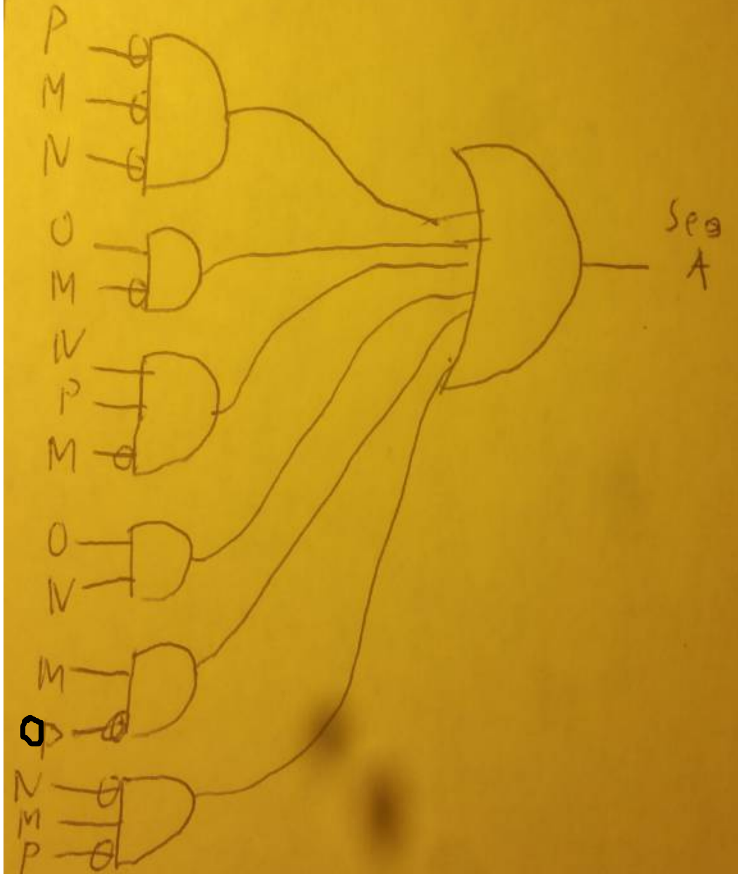


	M	N	O	P	A	D	G
0	0	0	0	0	1	1	0
1	0	0	0	1	0	0	0
2	0	0	1	0	1	1	1
3	0	0	1	1	1	1	1
4	0	1	0	0	0	0	1
5	0	1	0	1	1	1	1
6	0	1	1	0	1	1	1
7	0	1	1	1	1	0	0
8	1	0	0	0	1	1	1
9	1	0	0	1	1	1	1
A	1	0	1	0	1	0	1
B	1	0	1	1	0	1	1
C	1	1	0	0	1	1	0
D	1	1	0	1	0	1	1
E	1	1	1	0	1	1	1
F	1	1	1	1	1	0	1

Segment A

MN \ OP	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	0	1	1
10	1	1	0	1

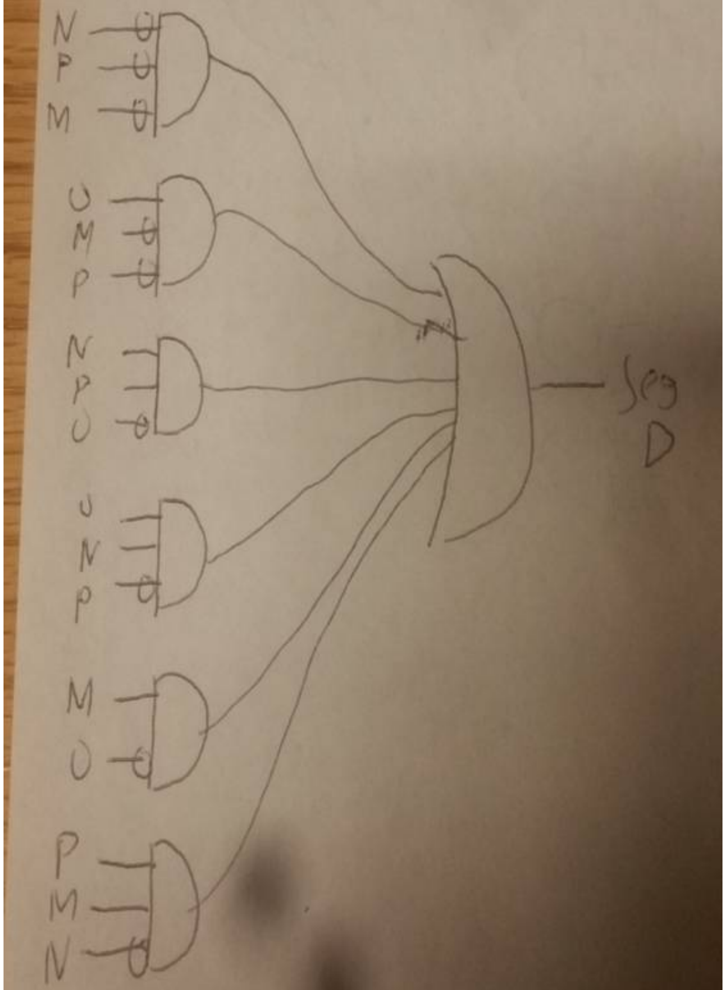
logic: $(\neg P \wedge \neg M \wedge \neg N) \vee (O \wedge \neg M) \vee (N \wedge P \wedge \neg M) \vee (O \wedge N) \vee (M \wedge \neg P) \vee (\neg N \wedge M \wedge \neg O)$



Segment D

MN \ OP	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	1	1	0	1
10	1	1	1	0

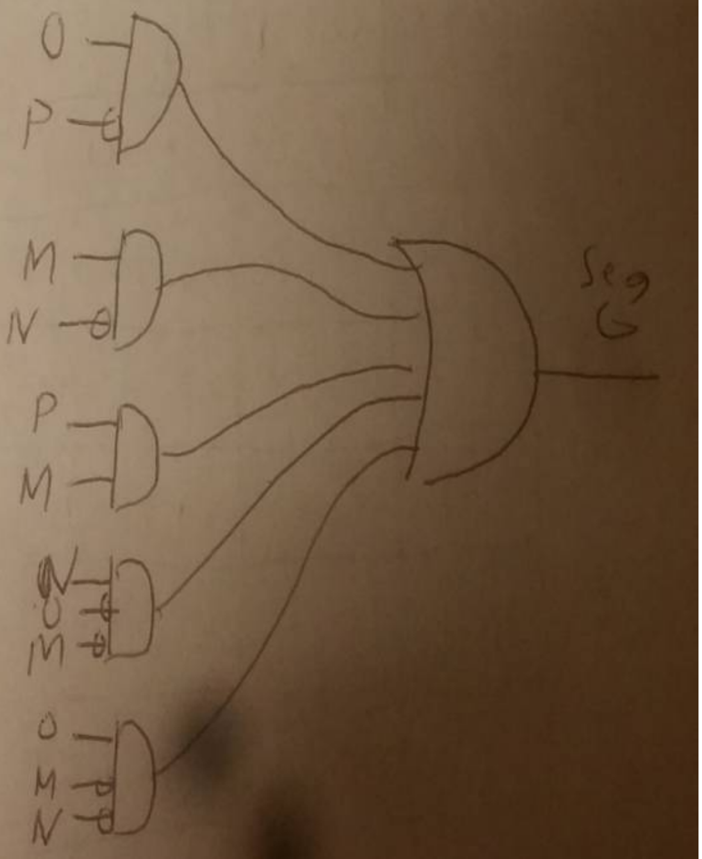
logic: $(\neg N \wedge \neg P \wedge \neg M) \vee (O \wedge \neg M \wedge \neg N) \vee (N \wedge P \wedge \neg O) \vee (O \wedge N \wedge \neg P) \vee (M \wedge \neg O) \vee (P \wedge M \wedge \neg N)$



Segment G

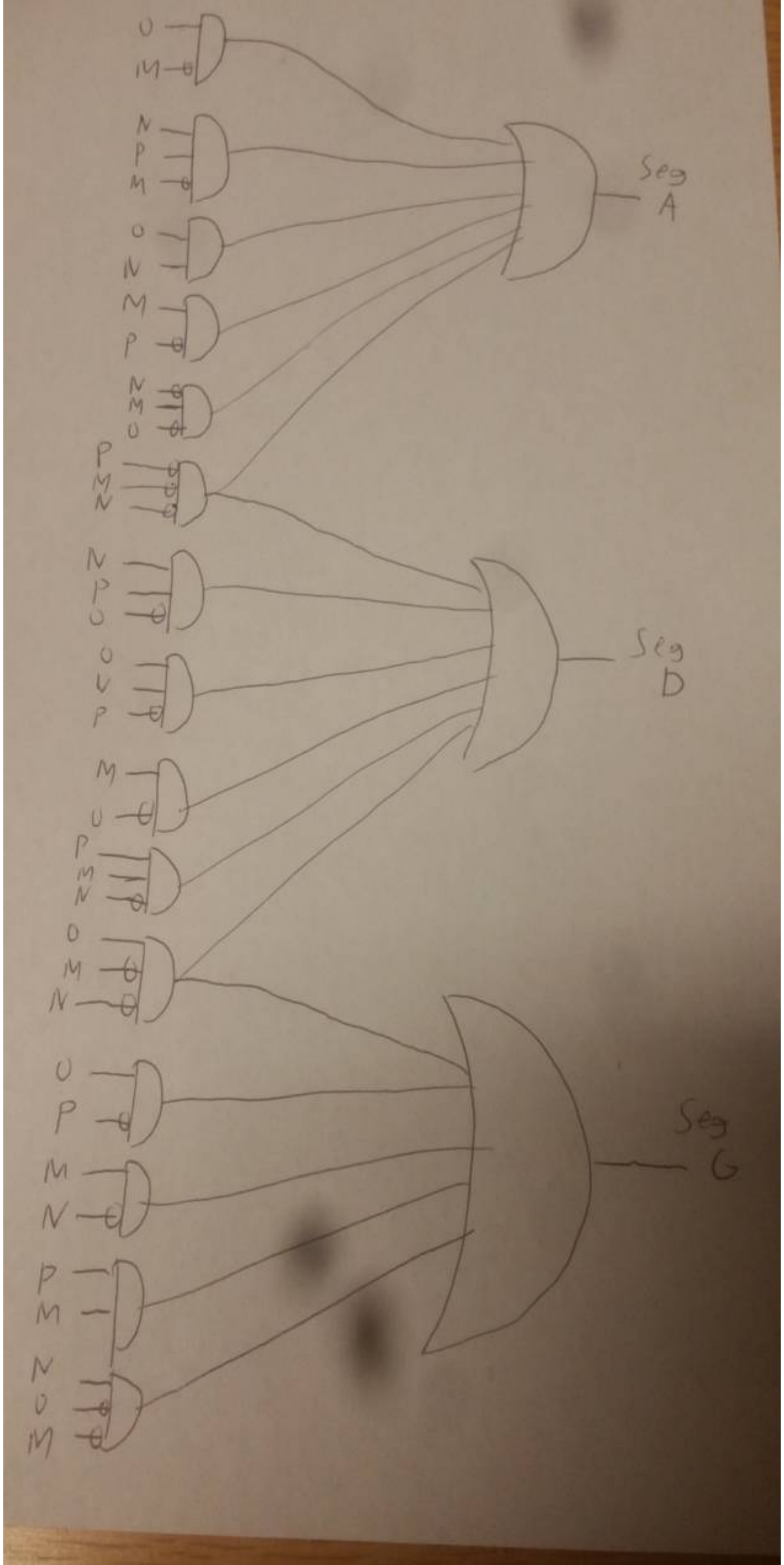
MN \ OP	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	1

logic: $(O \wedge \neg P) \vee (M \wedge \neg N) \vee (P \wedge M) \vee (N \wedge \neg O \wedge \neg M) \vee (O \wedge \neg M \wedge \neg N)$



or

All implemented together



Problem #2

The same idea of 2's complement can be applied to decimal digits, 10's complement.

(a) With 4 decimal digits, show how you would complement a number (make the number negative) by writing the function for complementing an integer, *N*. As an example, express the number -2144.

(b) What is the largest and smallest decimal number that can be represented with 4 decimal digitals?

(c) If each decimal digit is represented using Binary Coded Decimal (BCD - https://en.wikipedia.org/wiki/Binary-coded_decimal), where decimal digits are represented with 4 binary bits: 0=4'b0000, 1=4'b0001, 9=4'b1001. Implement a 1 decimal digital adder. Use as many 1-bit Full-Adder (FA), Half-Adder (HA), AND, OR, INV logic gates as you need.

Answer the question for all parts in the space below.

a) use 9,999 - |n| + 1 to find the tens compliment of a four digit decimal number. The sign on the answer is the opposite of what the starting sign was.

-2144 = 9,999 - |-2,144| + 1 = 7,856

to verify, 7,857 = 9,999 - 7,856 + 1 = -2,144

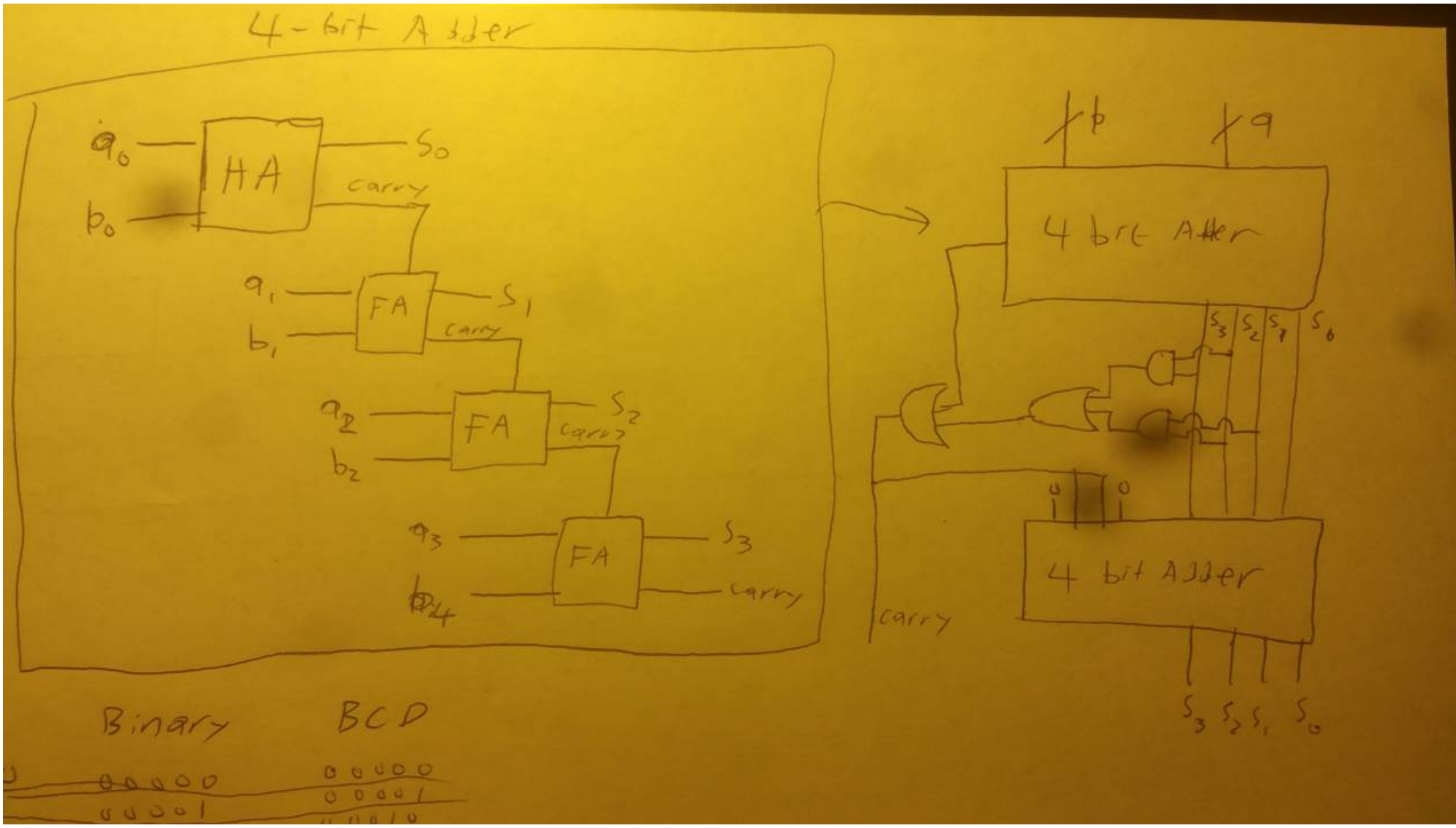
b) Since we have unsigned numbers from 0000 to 9999 the maximum range for a tens compliment would have to be from either -5000 to 4999.

c) when we add two bcd numbers we end up with a decimal number. This is an issue since bcd ranges from 0000 to 1001. To account for error when the sum of the two bcds is greater than 9, we need to come up with a relation between 10-18 in binary and 10-18 in bcd. By writing down a chart of binary and bcd representations of 0 - 18 I found that all bcd numbers after 9 are their binary equivalent plus 0110. The final design needs to optionally add 0110 to make this adder work. By consulting the chart, I found a few equations that tell me when I should add this six. For 16-18 the carry bit is a one. for 12-15 I found when s3 is a one and when s1 or s2 are ones. for 10 and 11 I found when s1 and s3 are both one. Using this logic, After the first sum I should have a circuit that follows the following logic to determine when to add a 6.

$$C \vee (S_3 \wedge S_2) \vee (S_1 \wedge \underline{S_Z})$$

By the result of this will be the carry bit for the final answer and will be used to create the 0110 needed to convert the binary to bcd. when carry is 0 the final addition should be the first number with 0 and when carry is one, the number should be 0110. To implement this I connected the first carry to b1 and b2 and connected b0 and b3 with a 0. This optionally adds a 0110 and will give the final answer as a bcd with one carry bit.

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



Problem #3

GPS often uses a longitude coordinate system (latitude as well) based on Degree [0,359], Minutes [0,59], and Seconds [0,59].

(a) If this was represented using BCD, how many bits of are needed to represent the information for either latitude or longitude.

(b) If each of the pieces of the information is represented individually (Degree, Minutes, Seconds), how many bits are needed?

(c) If the entire longitude is represented as a single binary number, how many bits are needed?

(d) Are there any advantages for representing in (c) rather than (b)?

(e) Would it be a good idea to represent this in floating point? Explain why or why not.

Answer the question for all parts in the space below.

a) 12 for the degrees + 8 for minutes + 8 for seconds = 28 bits.

b) 101101000 is 360 in binary and is 9 bits long.
111100 is 60 in binary and is 6 bits long.
9 + 6 + 6 = 21 bits needed.

c) 360 degrees * 60 minutes * 60 seconds
+ 60 minutes * 60 seconds + 60 seconds = 1299660
seconds
This number is 100111101010011001100 in binary so we need
21 bits.

d) With the representation of c, doing addition and subtraction on the latitude and longitude is simpler to implement since you can add the specific amount of seconds the values are changing, rather than changing degrees, minutes, and seconds individually.

e) It would be a bad idea to implement this as a floating point number since a gps system would need to do lots of additions and subtractions to update coordinates. Floating point sacrifices precision so the values for the gps wouldn't be as quite as accurate as they would be in other representations.

Problem #4

A shifter is a block that takes an n-bit word and changes the significance of each bit position of the word.

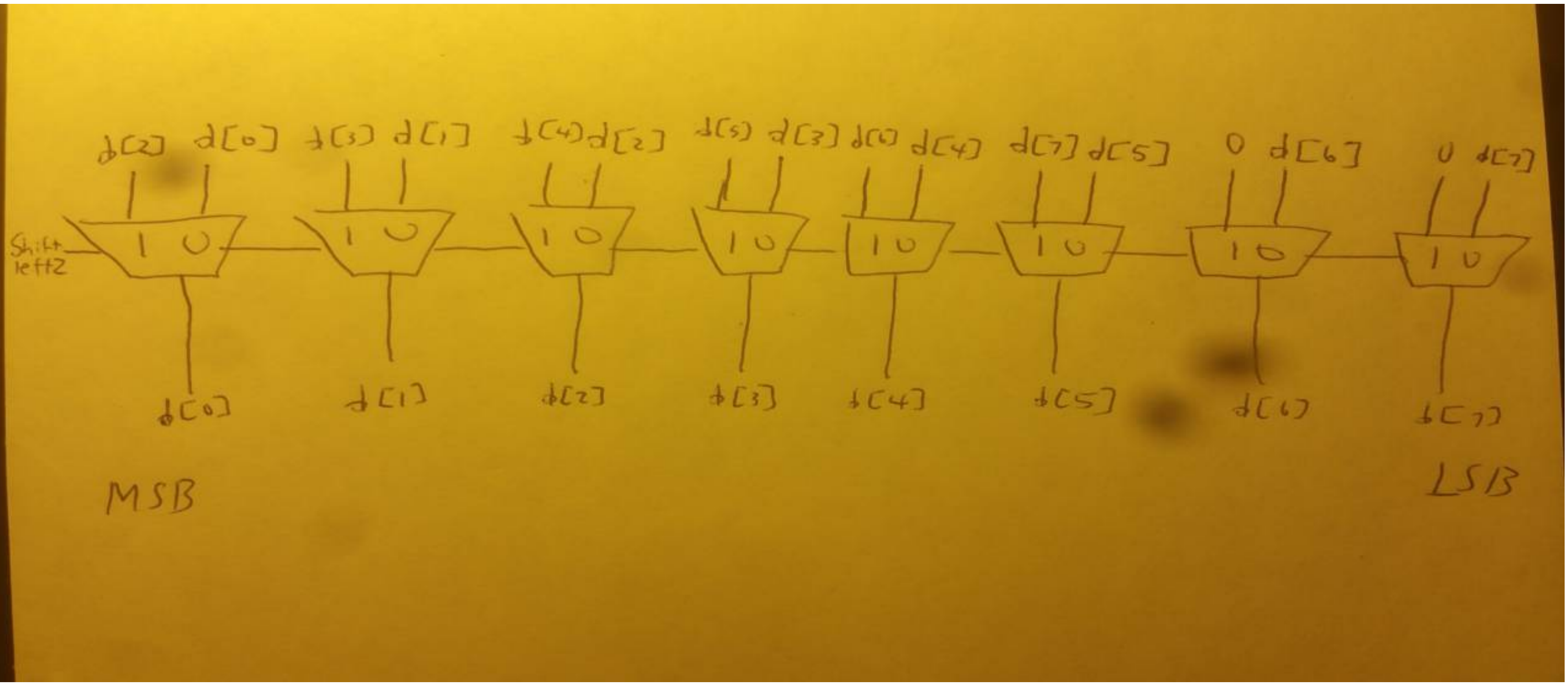
(a) Draw the design of a block that optionally shifts an 8-bit 2's complement word by 2 bit positions to the left (toward the MSB) when a *shift_left2* =1'b1 and does not shift when *shift_left2* =1'b0. How does this change the value of the word? Build this with 2:1 multiplexers.

(b) If the design in (a) is instead optionally shifting to the right (toward the LSB) with a *shift_right* =1'b1, what change would be needed to the design? How does this change the value of the word? You can choose to do either logical or arithmetic shift but you need to specify this clearly.

(c) Draw the design of a variable shifter that can arbitrarily shift in either direction based on a 1-bit input, *dir*, for direction (*dir*=1'b1 for left shift and *dir*=1'b0 for right shift) and 3-bit binary input, *shift*[2:0], indicating the amount of shift (up to 7). Try to use the least number of 3:1 multiplexers. Each 3-input multiplexers accepts 3-inputs {top, mid, bot} and 3 select signals (1-hot) {selt, selm, selb} to select each of those inputs.

(d) What is a barrel-shifter? How would you change the design in (c) to implement this.

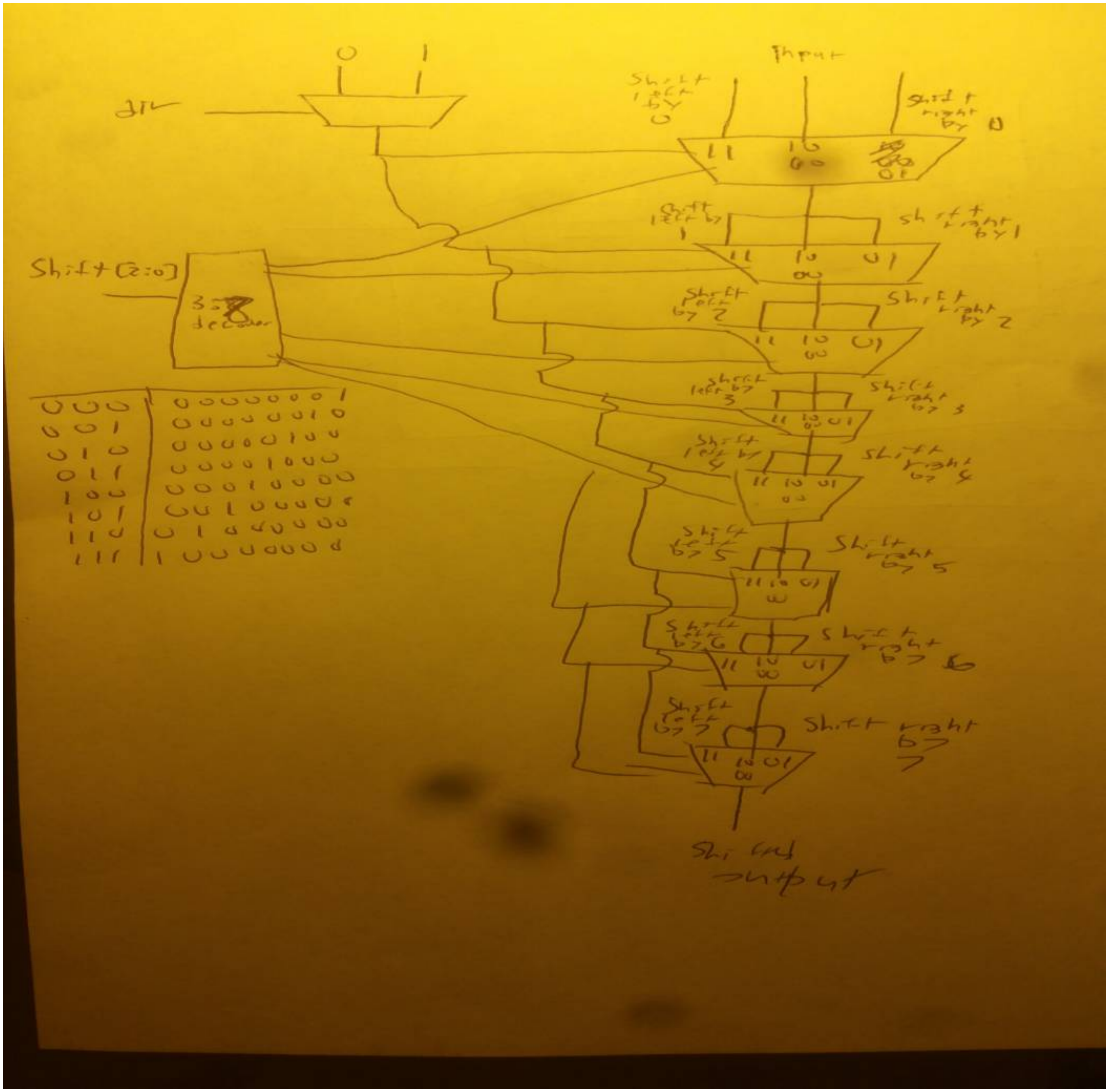
Answer the question for all parts in the space below.



This changes the value of the word by multiplying its value by 2^2.

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

b) To implement a logical right shift, make the left two inputs on the multiplexer be 0 then make the next 5 inputs be d[0] to d[5] for the right shift select. This change would cause the value of the word to be divided by 2^2.



0 0 0	0 0 0 0 0 0 0 1
0 0 1	0 0 0 0 0 0 1 0
0 1 0	0 0 0 0 0 1 0 0
0 1 1	0 0 0 0 1 0 0 0
1 0 0	0 0 0 1 0 0 0 0
1 0 1	0 0 1 0 0 0 0 0
1 1 0	0 1 0 0 0 0 0 0
1 1 1	1 0 0 0 0 0 0 0

d) a barrel shifter is a shifter with cyclic properties. If we shifted ABCD left by 2 it would be CDAB rather than CD00. to implement it in the prior design, whe doing the shifts using the circuit in part a, rather than having 0 be inputed, loop the value that would be cut off back around to the opposite multiplexer in the chain (e.g. left shifting 4 bits would have 4 muxes. instead of putting a 0 on the far right mux, link it to the most significant bit, the value of the far right mux when not shifting.)

Problem #5

A 16-bit half-precision floating point format is introduced where the exponent is 5 bits and the fraction is 10 bits.

(a) What is the expected bias?
(b) What is the relative accuracy?
(c) What is the span of the numbers? Assume that the maximum E=5'b11110, and the minimum E=5'b00001
(d) How would you represent a number closest to 1002.5?

Answer the question for all parts in the space below.

a) 2^4 - 1 = 15

b) (2^-14)/2 = (3.0518 * 10^-5)

c)

2^(11110 - 1111) * 1.1111111111

= 2^15 * 1.9990234375 = 65504

so max number is 65504.

2^(00001 - 1111) * 1.0000000000

= 2^(-14) * 1 = 6.1035 * 10^-5
is the smallest possible positive value.

d) sign bit = 0 (positive)

1002 = 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 + 2

.5 = 2^-1

1111101010.1 (shift by 9)

1.1111010101

E - 15 = 9 -> E = 24

24 in binary is 11000

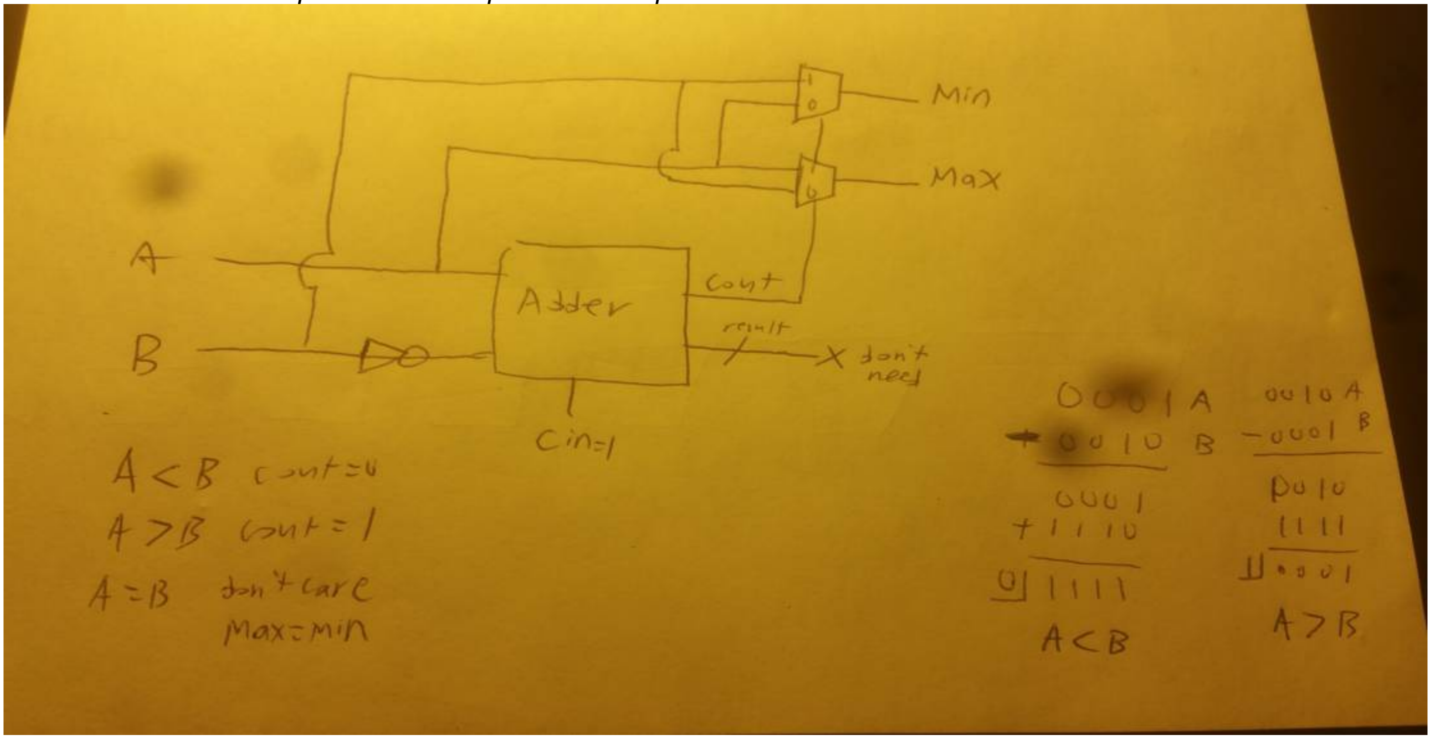
floating point representation:
0_11000_1111010101

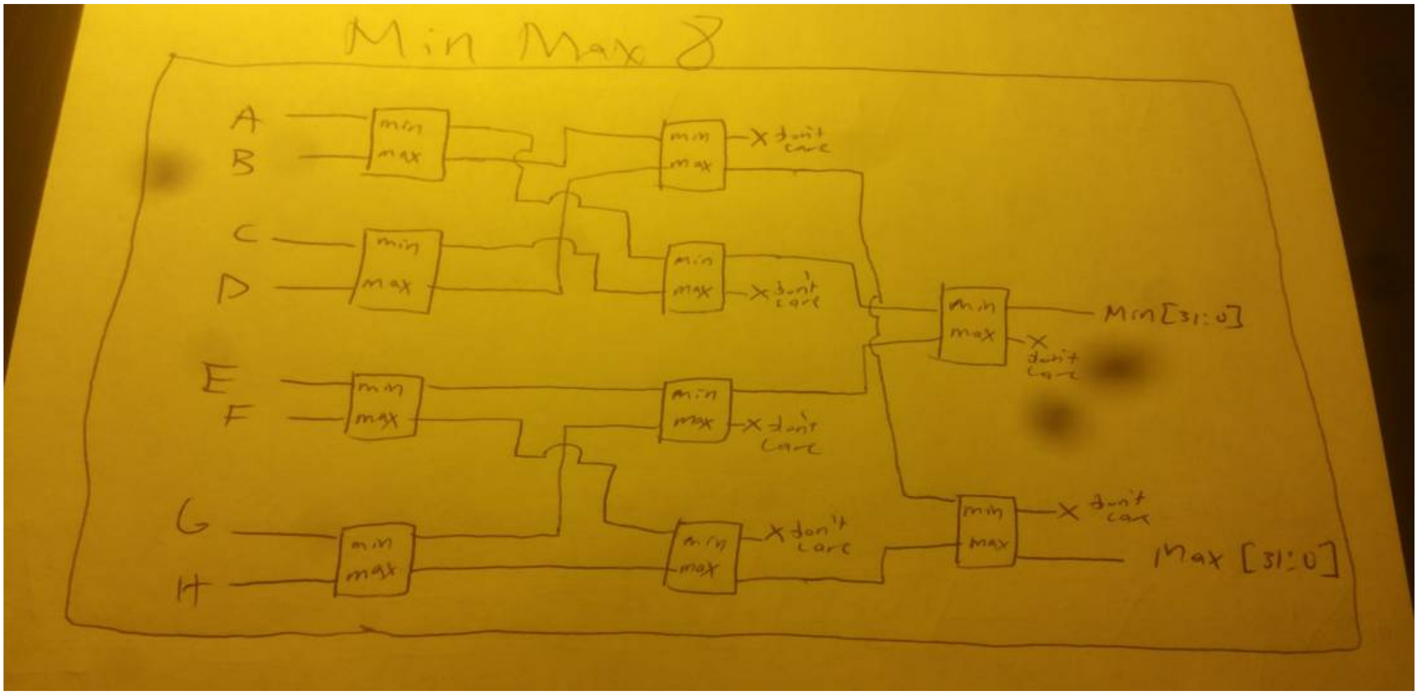
Problem #6

(a) A MinMax2 is a logic block that takes two 32-bit signed (2's complement) inputs (*inA*[31:0] and *inB*[31:0]) and produce two 32-bit outputs (*Min*[31:0] and *Max*[31:0]). Max takes the larger of the two inputs and vice versa for Min. Show the design of this block assuming that a 32-bit adder (ADD32) is available. Be sure to account for the overflow condition (Hint: when over/underflow occurs one of the numbers is still larger than the other).

(b) Use multiple MinMax2 block, implement a MinMax8 block that compares 8 32-bit inputs to find the 2 32-bit outputs (*Min*[31:0] and *Max*[31:0]) that correspond to the maximum and minimum inputs of the 8.

Answer the question for all parts in the space below.



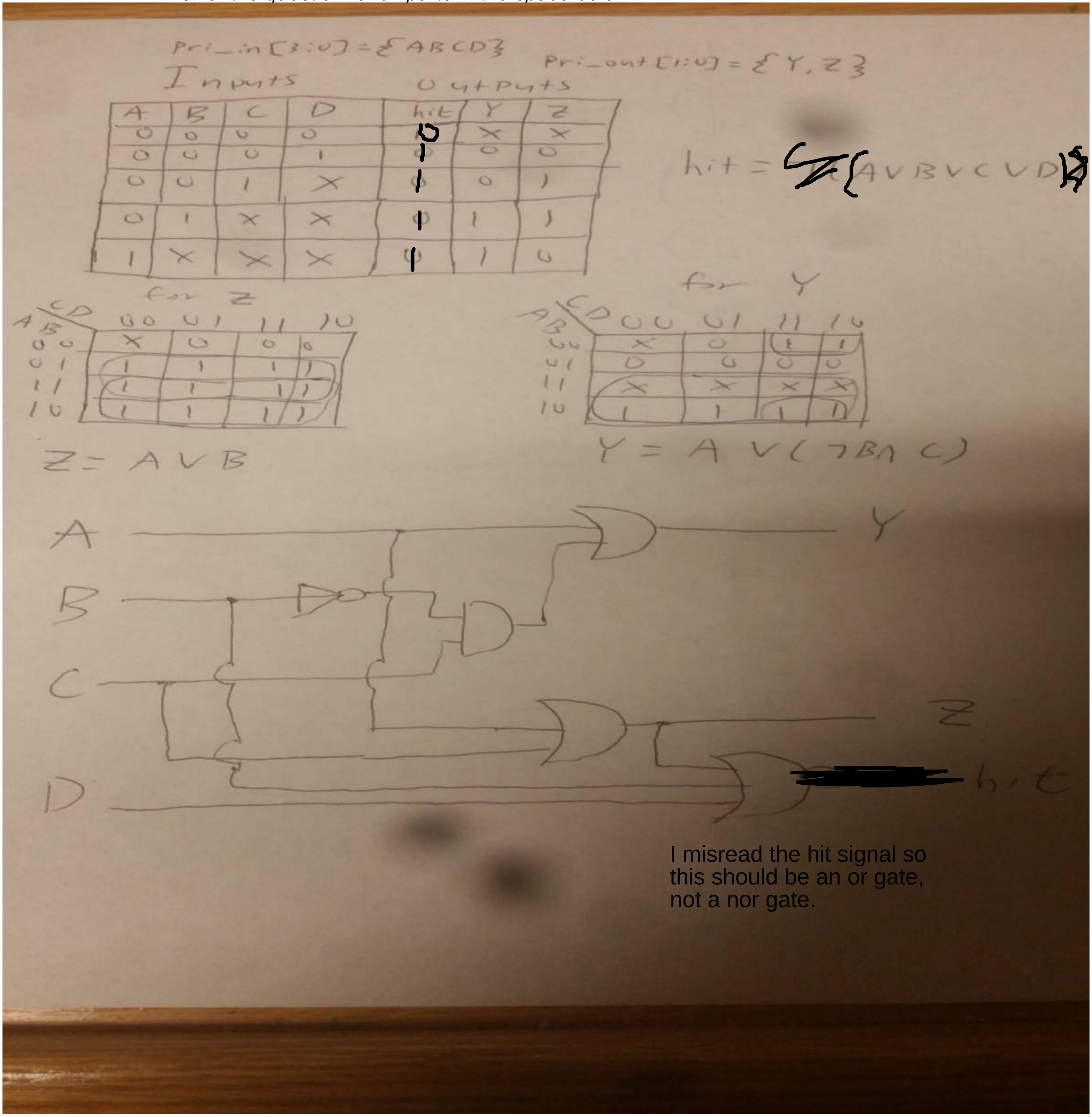


Problem #7

(a) Implement by drawing the logic for a 4:2 priority encoder. The inputs are *pri_in*[3:0], and the outputs are *pri_out*[1:0] and *hit* (1'b0 to correspond to when none of *pri_in*[3:0] being asserted). Build this using AND, OR, INV and MUX.

(b) Build a 16:4 priority encoder out of blocks of the 4:2 priority encoders in (a) and any necessary additional AND, OR, INV, or MUX. The inputs are *pri_in*[15:0], and the outputs are *pri_out*[3:0] and *hit*. Similar to the definitions in (a).

Answer the question for all parts in the space below.



Pri_in[3:0] = {ABCD} Pri_out[1:0] = {Y, Z}

Inputs				Outputs		
A	B	C	D	hit	Y	Z
0	0	0	0	0	X	X
0	0	0	1	1	0	0
0	0	1	X	1	0	1
0	1	X	X	1	1	1
1	X	X	X	1	1	0

hit = {A∨B∨C∨D}

for Z

AB \ CD	00	01	11	10
00	X	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

Z = A ∨ B

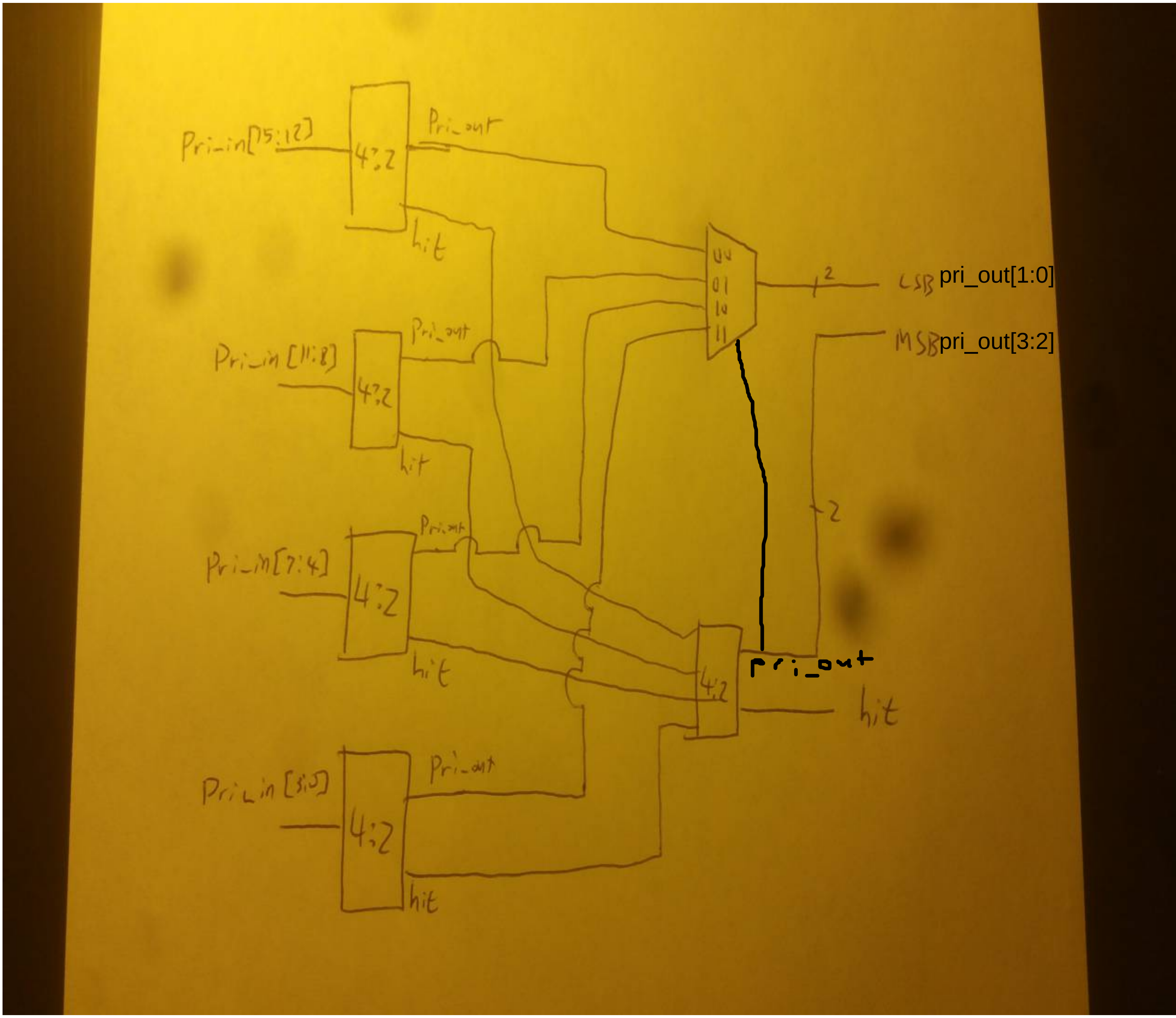
for Y

AB \ CD	00	01	11	10
00	X	0	1	1
01	0	0	0	0
11	X	X	X	X
10	1	1	1	1

Y = A ∨ (¬B ∧ C)

I misread the hit signal so this should be an or gate, not a nor gate.

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


Pri_in[15:12]
4:2
Pri_out
hit
Pri_in[11:8]
4:2
Pri_out
hit
Pri_in[7:4]
4:2
Pri_out
hit
Pri_in[3:0]
4:2
Pri_out
hit
00
01
10
11
2
LSB
pri_out[1:0]
MSB
pri_out[3:2]
2
4:2
pri_out
hit