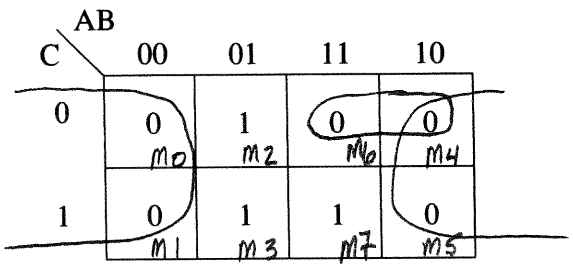
EECS 140/141 Introduction to Digital Logic Design
Fall Semester 2019
Exam #1 Date: 7 October 2019

NAME: ______________________________ KUID: KEY

General Instructions

1. This exam is closed-book. You are allowed a non-communicating calculator and one side of one page (8.5" X 11") of notes.
2. A reference sheet of the Boolean Algebra axioms and properties is supplied for your use.
3. Put your KUID on *each* page, in case the exam pages get separated.
4. There are 100 points possible on this exam.
5. When logic network diagrams are requested, you *must* draw them using your logic template. Failure to do this will be penalized.
6. Clearly indicate your final answer to each problem by putting a box around it (not necessary for logic network diagrams).
7. Show your work:
   a. If your answer is incorrect, partial credit may be awarded based on the work shown. Even if your answer is correct, you will *not* receive full credit unless you have shown some work on the exam pages.
   b. In particular, if you are asked to prove a relationship using algebraic manipulations from Boolean Algebra, you should *justify* each step with a property number from the Boolean Algebra reference sheet.
   c. Show all work on the exam pages. I have tried to leave lots of room for you to show your work, and the back of the last page is blank for you to continue work for some problem if you need to. Please do not use any other pages unless absolutely necessary. If you do use the back of the last page or other paper, *clearly* indicate on the *problem* page that there is additional work elsewhere (and where that additional work is).
   d. You may use the logic network diagrams and K-maps printed on the exam pages to show some of your work -- you do not need to re-copy them unless you wish to.
   e. The bottom line is this: the easier it is for me to figure out what you are trying to do, the more likely I will be to award partial credit.
8. Stay calm. If you are having trouble with one problem (or part of a problem), leave it and go on. Even if you are not able to work one part of a problem, you may still be able to work subsequent parts.

1. Consider the following K-Map for function H.



a. (3 points) Give a sum-of-minterms expression for this function H. That is, write it in the form of:

$H(A, B, C) = \Sigma m(\ldots\ldots)$

where the $H = 1$ minterm numbers are listed inside the parentheses.

$H(A,B,C) = \Sigma m(2,3,7)$ (see cell labels above).

b. (5 points) Find the Canonical Sum of Products (CSoP) expression for H *and determine the cost (as we have defined it in this class)* of this synthesis.

$H = \bar{A}\cdot B\cdot \bar{C} + \bar{A}\cdot B\cdot C + A\cdot B\cdot C$

$\text{cost} = (i+2)m+1 = (3+2)(3)+1 = \underline{16}$

c. (5 points) Find the Canonical Product of Sums (CPoS) expression for H *and determine the cost of this synthesis.*

$H = (A+B+C)\cdot(A+B+\bar{C})\cdot(\bar{A}+\bar{B}+C)\cdot(\bar{A}+B+C)\cdot(\bar{A}+B+\bar{C})$

$\text{cost} = (i+2)M+1 = (3+2)(5)+1 = \underline{\underline{26}}$

d. (7 points) Find the minimum-cost Product of Sums (PoS) expression for H *and determine the cost of this synthesis.*

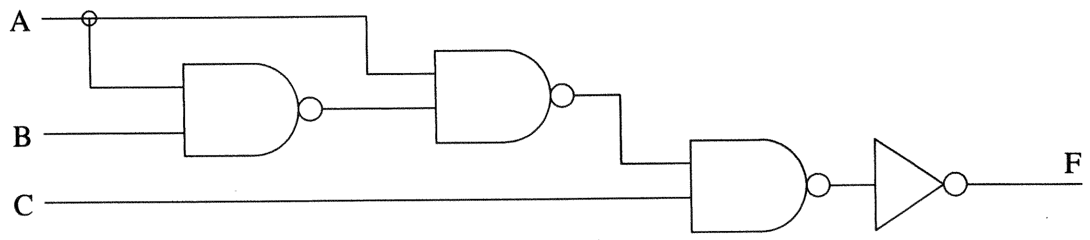
See K-map markings:

$H = B\cdot(\bar{A}+C)$

cost: 2 gates, 2 inputs to each gate:

$2+(2)(2) = \underline{\underline{6}}$

2. Consider the logic network diagram given below.



a. (6 points) If each gate as shown in this network was implemented using CMOS technology, how many total transistors would be required for the network?

NMOS : NAND ⇒ # inputs NOT ⇒ 1 NMOS = (3)(2) + 1 = 7

CMOS = 2 · NMOS ⇒ $\boxed{\text{CMOS: } 7(2) = 14 \text{ transistors}}$

b. (7 points) Determine and draw the Truth Table for the function $F(A, B, C)$.

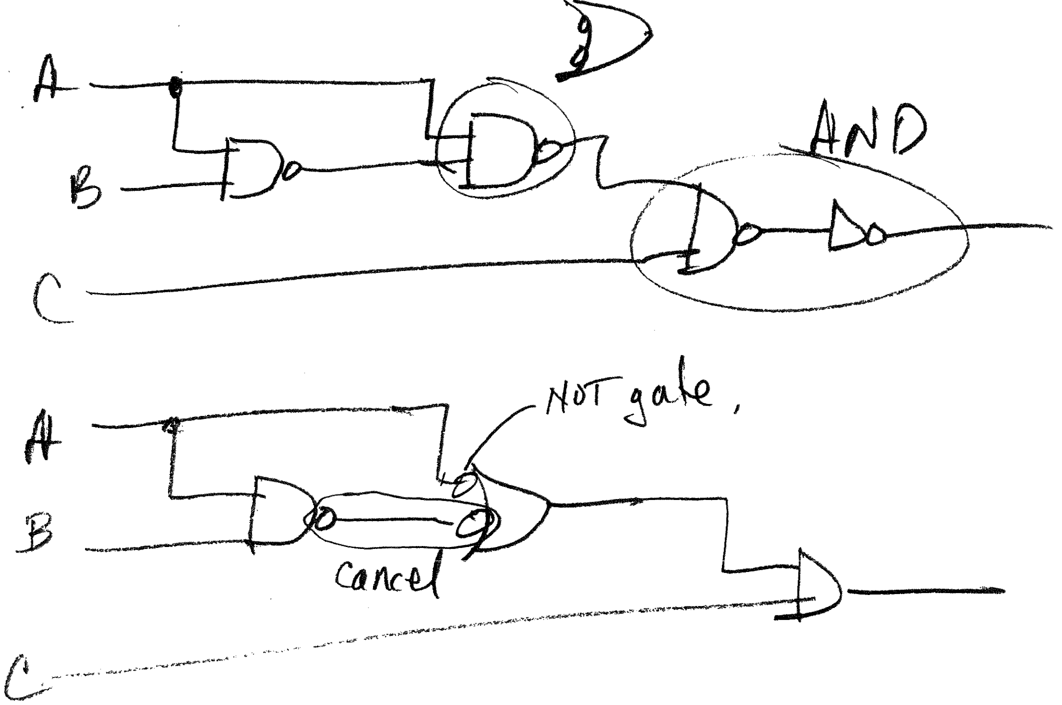
If C=0, then F=0

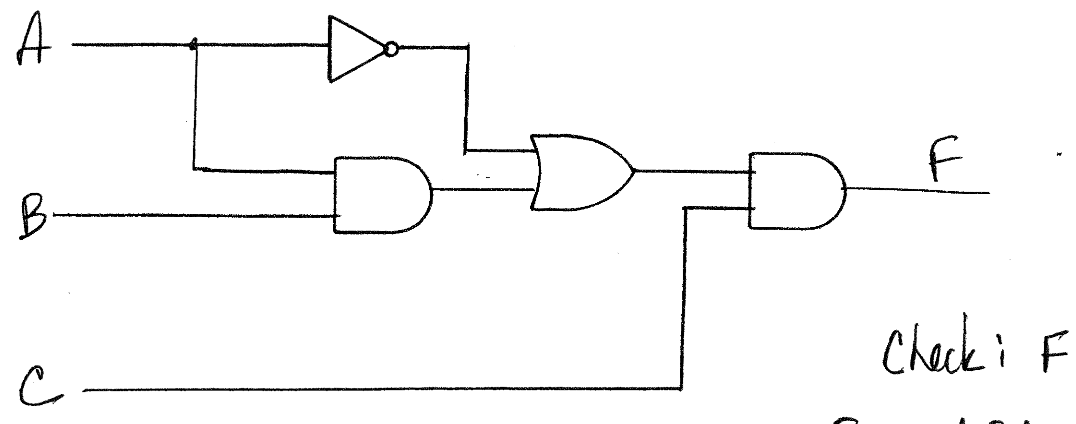
If C=1 and A=0, then F=1

If C=1 and A=1 and B=1, then F=1

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

c. (7 points) *Starting from the logic network diagram on the previous page,* convert it into a logic network that uses AND, OR, and NOT gates (no other types of gates allowed). You must do this by converting each NAND gate into and AND or OR gate, using what you know about the definition of NAND and equivalent forms of NAND gates. Your answer *MUST NOT* be in Sum of Products or Product of Sums form and it *MUST* include at least one OR gate and at least one AND gate. You may mark on the original logic network diagram, and you may draw intermediate logic networks without your logic template, but your final logic network diagram must be drawn by you using your logic template.



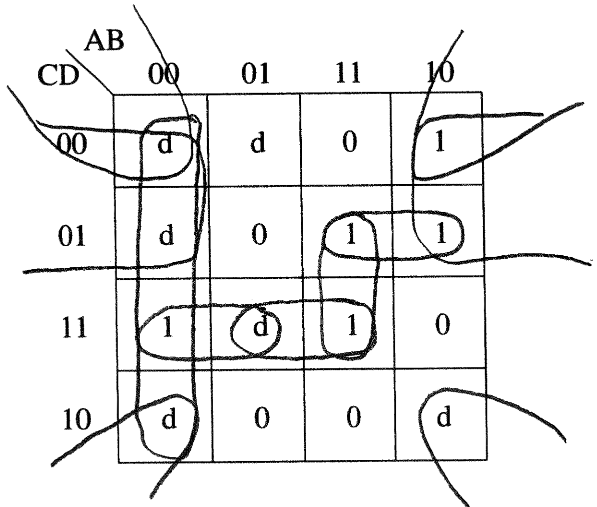


Check: $F = (AB + \bar{A}) \cdot C$

$F = ABC + \bar{A}C$

matches T.T.

3. Consider the following K-Map for function J, which we will be implementing with a Sum of Products (SoP) form. Each d in the K-Map indicates a *don't – care* output.



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

a. (6 points) Identify *all* of the Prime Implicants (PIs) for J, *writing a logic expression for each one.*

$\overline{B}\,\overline{C}$, $\overline{B}\,\overline{D}$, $\overline{A}\,\overline{B}$

$\overline{A}CD$, BCD, ABD, $A\overline{C}D$

b. (6 points) Identify *all* of the Essential Prime Implicants (EPIs) for J, *writing a logic expression for each one.*

None is essential. Each $J=1$ minterm is included in at least 2 PIs.

c. (8 points) Find a minimum-cost (as we have defined cost in this class) SoP synthesis for J.

Choose $\overline{B}\,\overline{C}$: 2-literal PI that adds 2 to cover
Then Choose $\overline{A}\,\overline{B}$: " " " " 1 " "
Then choose ABD to complete cover.

$$J = \overline{B}\,\overline{C} + \overline{A}\,\overline{B} + ABD$$

4. (15 points) Prove the following Boolean equality using Boolean algebraic manipulation. You may *not* use a Truth Table or K-map to show the equality. You *must* justify each step of your manipulation with the Boolean Algebra property number or numbers used to obtain that step.

$$AB + BC + \bar{A}C + \bar{A}BC = AB + \bar{A}C$$

$AB + BC + \underbrace{\bar{A}C + \bar{A}BC}$ Left side.

$AB + BC + \bar{A}C$ 13a where $X = \bar{A}C$ and $y = B$. (also 10a)

$AB + \bar{A}C$ 17a where $X = A$ $y = B$ $z = C$.

↑ RHS. Done.

OR

$AB + BC + \bar{A}C + \bar{A}BC$ Left side.

$\underbrace{AB + \bar{A}BC} + BC + \bar{A}C$ 10b (repeated)

$(A + \bar{A}C)B + BC + \bar{A}C$ 12a $X = B$ $y = A$ $z = \bar{A}C$ (and 10a.)

$(A + C)B + BC + \bar{A}C$ 16a where $X = A$, $y = C$

$AB + BC + BC + \bar{A}C$ 12a (and 10a)

$AB + BC + \bar{A}C$ 7b $X = BC$

$AB + \bar{A}C$ 17a Done.

OR

$AB + BC + \bar{A}C + \bar{A}BC$ Left Side.

$AB + \underbrace{BC + \bar{A}BC} + \bar{A}C$ 10b

$AB + BC + \bar{A}C$ 13a where $X = BC$ and $y = \bar{A}$ also 10a.

$AB + \bar{A}C$ 17a Done.

5. A certain vending machine dispenses only one kind of candy that has a price of 15 cents. The machine has 3 slots for coins: two for dimes (10 cents each), and one for a nickel (5 cents). The machine has sensors to determine whether or not each slot has a coin in it. A customer places coins in the slots and then presses a "dispense" button to dispense candy. The machine also has a "change bin" for returning excess money to the customer, as needed. The exact operation of the machine is as follows. If there is a total of 15 or more cents in the coin slots when the "dispense" button is pressed, all of the coins are taken into the machine, a 15-cent candy is dispensed and any amount over 15 cents is returned in the "change bin". If there is a total of less than 15 cents in the coin slots when the "dispense" button is pressed, no candy is dispensed and all of the money in the coin slots is returned in the "change bin".

The design project is to design a 3-input, 3-output digital logic circuit to be used in controlling the dispensing of candy and returning money in the "change bin". HOWEVER, we will only do the first steps here: defining variables, assigning values, and specifying a truth table.

You must define the 3 output variables as follows. Output f_1 should have value 1 if and only if a candy bar should be dispensed if the "dispense" button is pressed with the current combination of coins in the coin slots. Then, output f_2 should have value 1 if and only if exactly 5 cents should be returned in the "change bin" with the current combination of coins in the coin slots. Finally, output f_3 should have value 1 if and only if exactly 10 cents should be returned in the "change bin" with the current combination of coins in the coin slots.

You are to do the following:

a. (3 points) *Define* the 3 input variables AND *assign* binary values to the 3 input variables. That is, give each input a symbol and describe what logic 0 and logic 1 indicate. Note that I have done this for the 3 outputs above.

d_1: dime in first dime slot:
1 if coin in slot, 0 if not

d_2: dime in 2nd dime slot:
1 if coin in slot, 0 if not

n: nickel in nickel slot:
1 if coin in slot, 0 if not

b. (7 points) Draw the 3-input, 3-output truth table for this digital logic circuit. This will be like a regular 3-input, 1-output truth table, except there will be 3 output columns (instead of just 1), one each for f_1, f_2, and f_3. So, there will be a total of 6 columns in your truth table, 3 for the inputs and 3 for the outputs. Draw the table neatly and carefully, so that it is easy for me to see the individual rows and columns.

d_2	d_1	n	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	1

or

n	d_1	d_2	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	1

6. Below are K-maps for two 4-input functions L and M. Each function has the same 4 inputs: A, B, C, and D.

Function L

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

Function M

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

a. (3 points) Considering function L *by itself*, find ALL (there may only be one) minimum-cost sum of products (SoP) syntheses (implementations) for L, AND give the minimum cost (as we have defined it in this class). Each answer should be a SoP logic expression. You need not explicitly write down all Prime Implicants (PIs) and Essential Prime Implicants (EPIs), even though you will need to determine them in order to find the minimum-cost SoP implementation(s).

$L = \underbrace{\bar{A}\bar{B}\bar{D} + ABD}_{\text{EPIs}} + \underbrace{\bar{A}B\bar{C}}_{\text{Completes Cover.}}$ $\boxed{\text{Cost} = 16}$

b. (3 points) Repeat part (a) for function M.

$M = \underbrace{\bar{A}\bar{C}\bar{D} + ABD}_{\text{EPIs}} + \underbrace{B\bar{C}D}_{\text{Completes Cover}}$ $\boxed{\text{or}}$ $M = \underbrace{\bar{A}\bar{C}\bar{D} + ABD}_{\text{EPIs}} + \underbrace{\bar{A}B\bar{C}}_{\text{Completes Cover.}}$

$\boxed{\text{cost} = 16}$

c. (6 points) Now try to find the minimum-cost synthesis for the *joint* 4-input, 2-output (L and M) function. *Specifically*, your answer should be the *jointly – minimized* SoP logic expressions for L and M. If you intend to share any of the product terms between the two output functions, circle those product terms. There is more space for this part on the next page.

Could use 2nd synthesis for M and share ABD and $\bar{A}B\bar{C}$ terms. Cost: $\underset{\text{(for L)}}{16} + \underset{\text{(additional for M)}}{8} = 24$

More space for the part on the previous page.

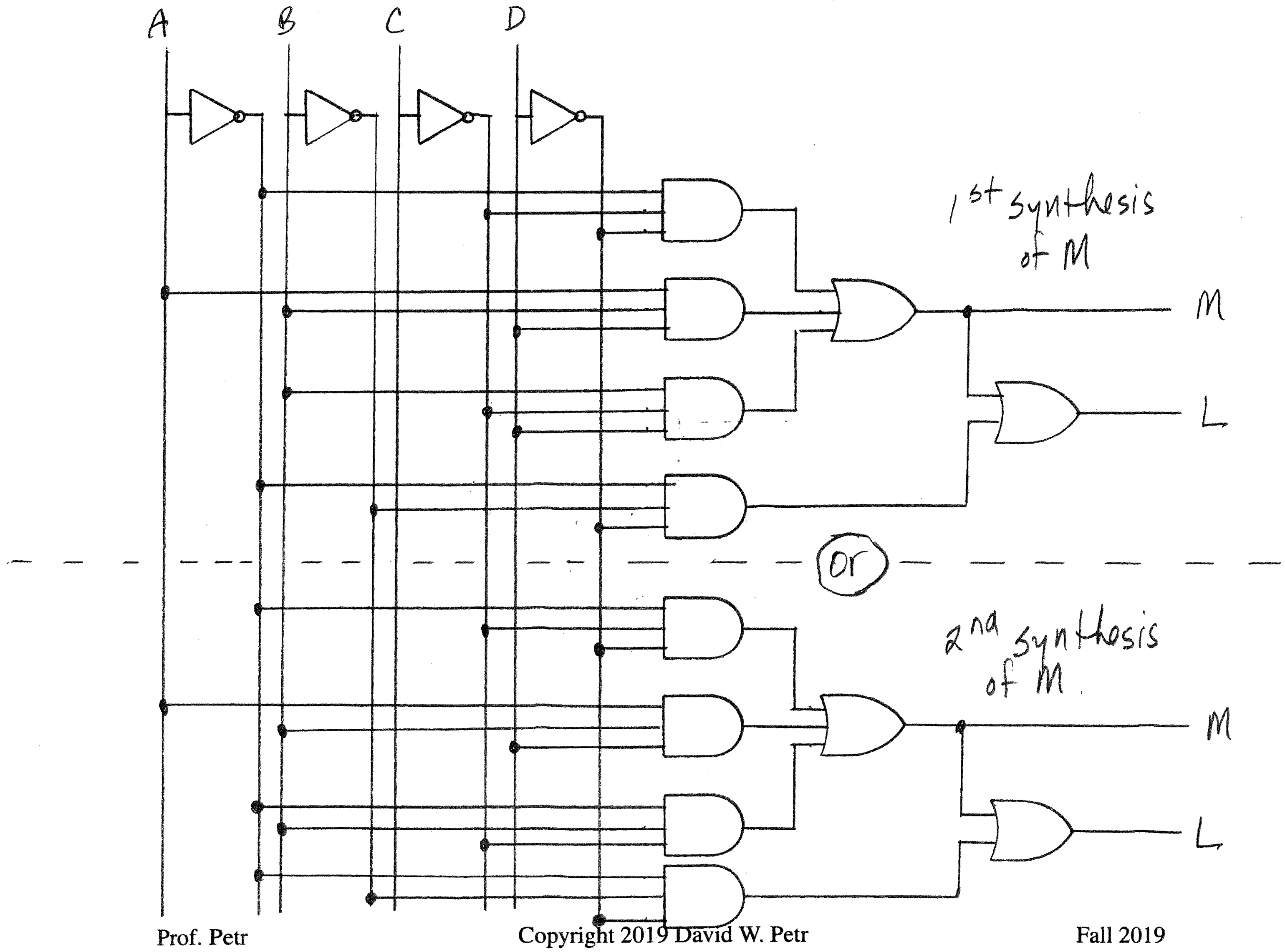
But better to use $\underline{\text{either}}$ synthesis for M and then

$L = M + \bar{A}\bar{B}\bar{D}$ Cost = $\underset{(\text{for } M)}{16} + \underset{(\text{additional for } L)}{7} = 23$.

$M = \bar{A}\bar{C}\bar{D} + ABD + B\bar{C}D$ [✓ or] $M = \bar{A}\bar{C}\bar{D} + ABD + \bar{A}B\bar{C}$

$L = M + \bar{A}\bar{B}\bar{D}$

d. (3 points) Draw *(using your logic template)* the *jointly – minimized* 4-input, 2-output logic circuit that implements the logic expressions (including any sharing) from the previous part, using only AND, OR, and NOT gates.



More space for any problem.