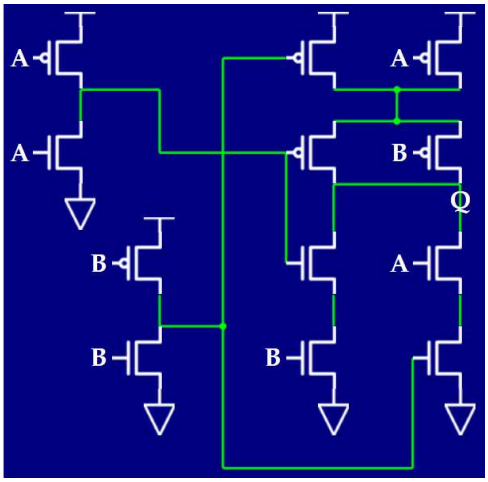
ECE 120: Introduction to Computing Spring 2019 ZJUI

due at start of section on Friday 22 March 2019

Homework 4: Logic and Minimization

In this assignment, minimal SOP and POS refer to the K-map solutions with minimal area heuristic discussed in class (literals + operations, not counting complements on literals). When we ask for SOP and POS expressions, you need only consider expressions in these forms, and solving the K-map as shown in class produces the correct answer. The online exercises will help you to practice these skills.

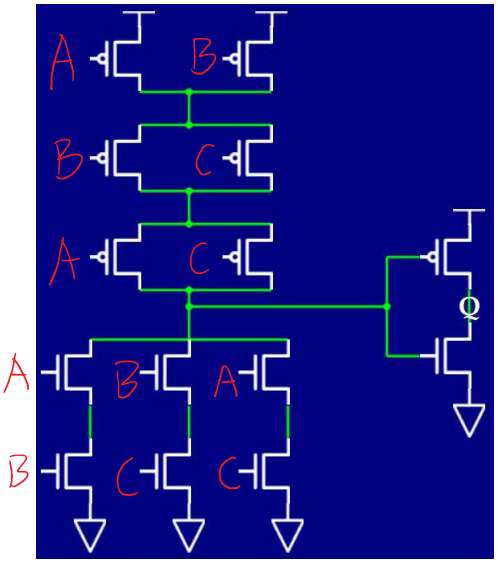
1. Building Gates from Transistors



a. Analyze the circuit shown to the right. What is the output Q in terms of inputs A and B? For credit, write your answer as simply as possible.
[(AB') + (A'B)]' = (A ⊕ B)'

b. Label the 12 transistor gate inputs on the diagram shown below to implement the carry out signal Q for a three-bit addition of inputs A, B, and C. (The carry out signal is equal to 1 iff at least two of the inputs are 1.)

AB + BC + AC = [(AB + BC + AC)']'
One possible labeling is shown below.



2. Simplifying with K-maps

The function F(A,B,C,D) is specified by the Karnaugh map to the right.

F

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

a. List all prime implicants of F(A,B,C,D).
AC', C'D, AB'D', B'CD'

b. Find a minimal SOP expression for F(A,B,C,D).
AC' + C'D + B'CD'

c. Find a minimal POS expression for F(A,B,C,D).
(C' + D')(B' + C')(A + C + D)

3. Two-level Circuits

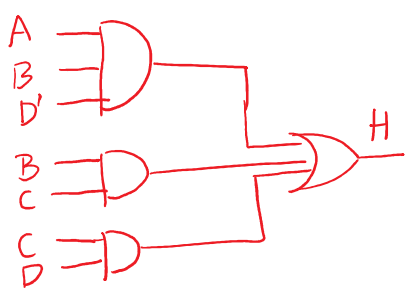
Let H(A,B,C,D) = (AB + A'CD + BD')(A' + B'C + D)' + C(BD' + CD)

a. Using a K-map, derive a minimal SOP expression for H. Show your work.
b. Draw an AND-to-OR circuit for H.
c. Using a K-map, derive a minimal POS expression for H. Show your work.
d. Draw an OR-to-AND circuit for H.

Note: When drawing circuits, assume that complemented inputs are available.

a. H = CD + BC + ABD'

b.

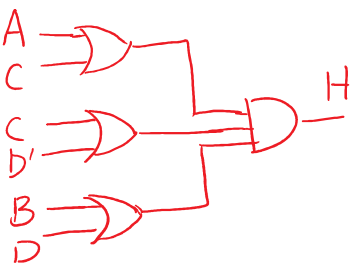


H

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

c. H = (A + C)(C + D')(B + D)

d.



H

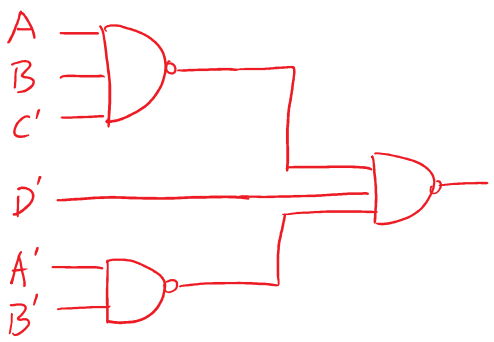
CD \ AB	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	1	1	1	1
10	0	1	1	0

4. NAND/NOR Circuits

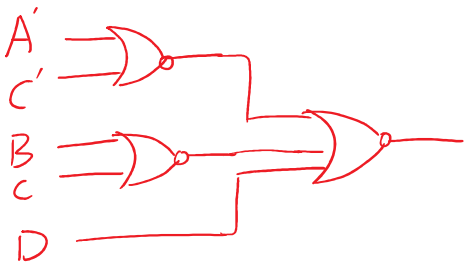
a. Implement ABC' + D + A'B' using only NAND gates. Do not simplify the expressions.
b. Implement (A' + C')(B + C)D' using only NOR gates. Do not simplify the expressions.

Note: When drawing circuits, assume that complemented inputs are available.

a.



b.

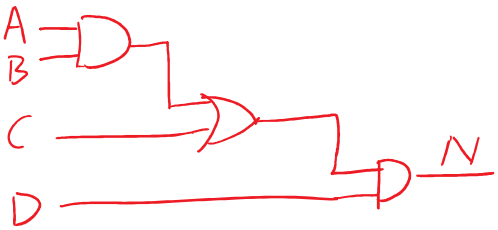


5. Smaller or Faster?

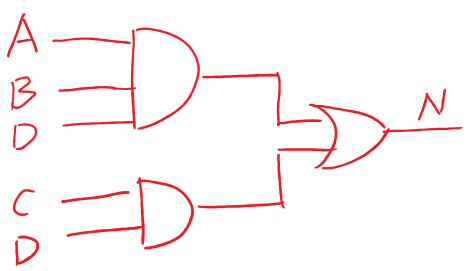
Consider the function N(A,B,C,D) = (AB + C)D. *Hint: all four implementations below require exactly three gates, assuming that complemented inputs are available.*

a. Based on the given functional form, implement N using AND and OR gates.
b. Now re-implement N as two-level logic.
c. Next, redraw your circuit from **part (b)** using only NAND and NOR gates.
d. Now use deMorgan's law to redraw your circuit from **part (a)** using only NAND and NOR gates.
e. Finally, using the heuristics given in class and in the notes, calculate the delay and area for the implementations in **parts (c)** and **(d)**. Which one is better?

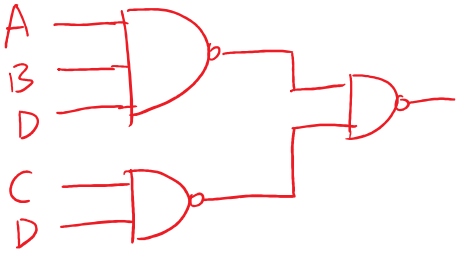
a.



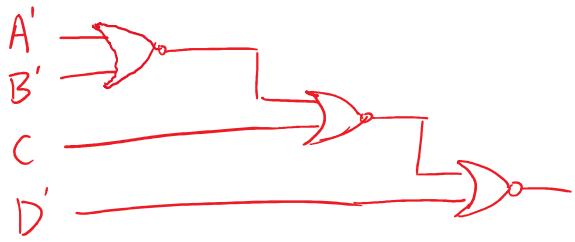
b.



c.



d. (AB + C)D = ((AB + C)' + D')'
AB = (A' + B')'



e. Delay: part (c) = 2
part (d) = 3

Area: part (c) = 8 (5 literals + 3 gates)
part (d) = 7 (4 literals + 3 gates)

The implementation in part (c) is better in delay, while that in part (d) is better in area.

6. Logic Design Word Problem

As the only ZJUI student in a group working to produce an improved microwave oven, you have been selected to develop the logic that controls the interior light L (L=1 to turn the light on, and L=0 to turn it off) and the microwave generation G (G=1 to generate microwave radiation, and G=0 to stop generating radiation).

You must generate the L and G output signals based on the following input signals:

- Door status D: when the door is open, D=1; when the door is closed, D=0.

- Timer value $C=C_2C_1C_0$: a 3-bit unsigned timer value used to time cooking.
- A steady power signal P: this signal provides an early warning of voltage drops; P=1 means a steady power supply, and P=0 indicates variability.

Your implementation must behave as follows. First, the light should turn on whenever the door is open, and should also turn on whenever the microwave generation is turned on. Second, the microwave generation should never be turned on when the door is open, as the radiation could harm the user. Third, the microwave generation should be turned off whenever the power supply becomes variable. Finally, the microwave generation should only be turned on when the timer is non-zero.

Based on this description, you must design a circuit for the new microwave. Start by deriving minimal SOP forms for both L and G (you may want to think about how you can reduce the number of input variables to avoid using a 5-variable K-map). Then draw a circuit diagram for your design, with inputs D, C_2, C_1, C_0, and P on the left, and outputs L and G on the right. Use two-level logic in your design, with a total of five gates. Assume that complemented inputs are available.

Translating the requirements in order, we have:

$D = 1 \rightarrow L = 1$ and $G = 1 \rightarrow L = 1$

$D = 1 \rightarrow G = 0$

$P = 0 \rightarrow G = 0$

$C_2 = 0$ and $C_1 = 0$ and $C_0 = 0 \rightarrow G = 0$

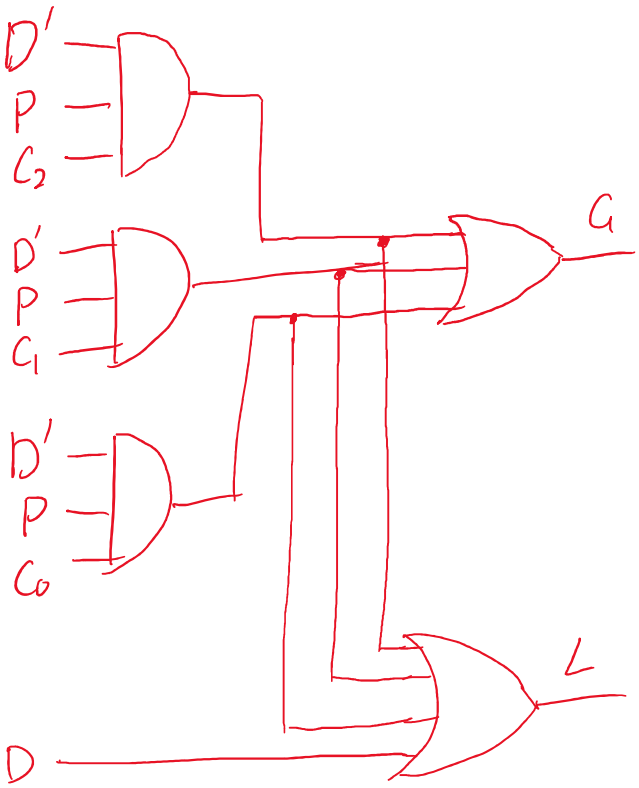
The timer is either zero or non-zero, so let's define $C = C_2 + C_1 + C_0$.
We can then draw a K-Map for G:

G		DP			
		00	01	11	10
C	0	0	0	0	0
	1	0	1	0	0

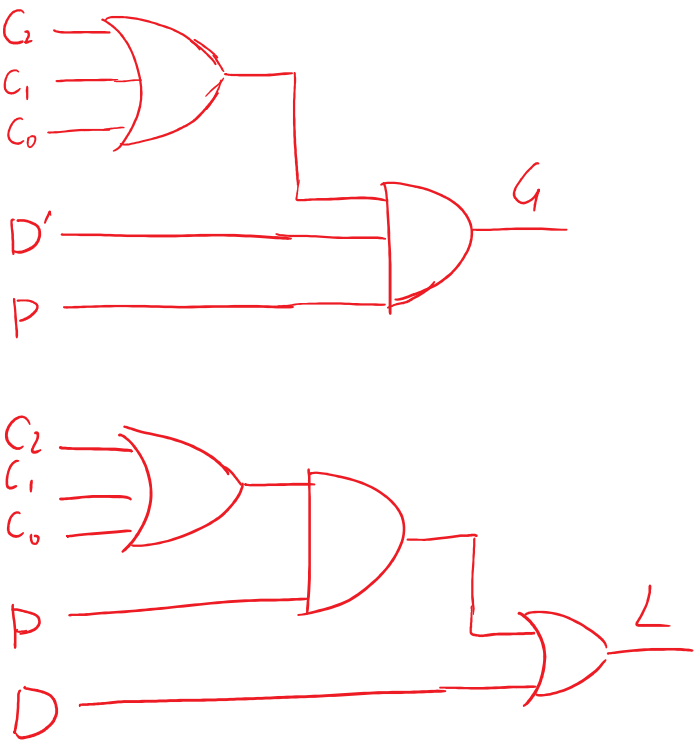
So $G = D'PC$, which we can write as $G = D'PC_2 + D'PC_1 + D'PC_0$.
Recalling that $G \rightarrow L$, we can then write a K-Map for L:

L		DP			
		00	01	11	10
C	0	0	0	1	1
	1	0	1	1	1

Notice that L is simply D + G. With G in SOP form, we can reuse the AND gates from G and add one more OR gate for L.



One can also implement using fewer, simpler gates (and thus less area) if one is willing to tolerate a little more delay (but the problem specifically required two-level logic):



7. Automatic Umbrellas: Another Word Problem

Prof. Lumetta has noticed that there are umbrellas outside of the café north of the Canteen. Unfortunately, those umbrellas require human operators, so sometimes they are in the wrong position (open or closed). Fortunately, we have many eager ECE students on campus, so … guess what.

Calculate the umbrella opening function U, which should be 1 to open an umbrella, or 0 to close an umbrella, as a function of the four variables described in the following requirements.

1. The umbrellas should be open when it's raining (R = 1).
2. Otherwise, if it's night time (D = 0), the umbrellas should be closed.
3. Otherwise, if it's sunny (S = 1), the umbrellas should be open.
4. Otherwise, if it's cool outside (C = 1), the umbrellas should be closed.
5. Otherwise, the umbrellas should be open.

a. Draw a K-map for the function U(R,D,S,C).
b. Calculate a minimal SOP expression for U.

a.

U	RD			
	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	1	1	1
10	0	1	1	1

SC

b. U = R + DC' + DS

8. Comparing Functions with a Truth Table

Update your Subversion directory to obtain the subdirectory **`hw4`**, which contains the program **`logical.c`**. Compile and execute the program as given. The program prints a truth table for the functions Q(A,B,C) = (AB) ⊕ (AC') ⊕ (B'C') and R(A,B,C), which is initially set directly equal to Q(A,B,C).

a. Look at the code. Notice that each logic calculation uses bitwise operators (~, &, and |), and that the last operation uses &1 to discard the other bits. Edit the code and remove the &1 operation from the end of the assignment to **Q**. Then recompile and re-execute the code. Explain why the Q values sometimes appear as negative values in the resulting output. When you are done, undo your changes/revert to the original code.

When A, B or C are assigned 0 or 1, the high bits are 0s. These bits will be changed into 1s when doing the NOT operation. If these 1s are kept to the end of logic calculation, that leads to printing a negative value.

b. Rewrite the expression for Q using only the input variables (A, B, and C) and AND, OR, and NOT functions. *Hint: You may want to look at the printed truth table to do so rather than manipulating the formula algebraically!* Turn in a copy of your modified code along with a copy of the output showing that the functions Q and R remain equivalent after your changes.

R = ((A & B & C) | ((~A) & (~B) & (~C)) &1);

Output:

```
[student@localhost hw4]$ gcc logical.c -o logical && ./logical
 A B C | Q R
-------+-----
 0 0 0 | 1 1
 0 0 1 | 0 0
 0 1 0 | 0 0
 0 1 1 | 0 0
 1 0 0 | 0 0
 1 0 1 | 0 0
 1 1 0 | 0 0
 1 1 1 | 1 1
[student@localhost hw4]$
```