EE 201 Problem Set 2 solutions

After solving the problems, look at the solutions posted on the course website and categorize your work for each problem on the following scale:

- ● Completely correct
- ◔ Nearly correct, but made a small mathematical or copying error
- ◒ Solved part of the problem correctly
- ◶ Started some work in the right direction
- ○ Incorrect, or didn't even know where to start on the problem
- Include a question mark (?) in addition to one of the above symbols if you don't feel like you understand the question or the solution well enough to make a definite judgement.

Problems with an asterisk (*) are optional.

1.a	1.b	1.c*	1.d	1.e*	2	3.a	3.b	3.c*	3.d*	4.a	4.b*	4.c

What questions do you have about these concepts and skills?

What things are you uncertain about (even if you don't have a specific question)?

Approximately how long did it take you to complete the homework?

How long did you take going over the solutions and writing this reflection?

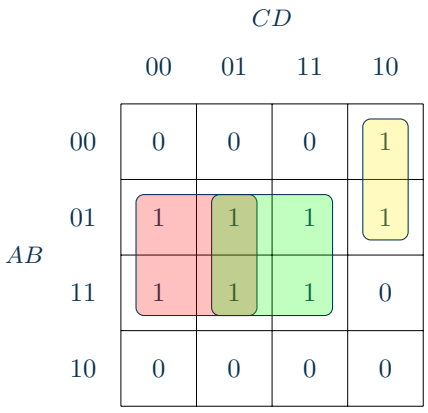
Turn in this self-assessment sheet on Gradescope. You do not need to turn in anything else, although we're happy to look at your work if you have questions!

Problem 1: Logic minimization

Minimize the truth tables and logic functions below.

(a) $\overline{A}BC + \overline{A}B\overline{C} + \overline{A}C\overline{D} + AB\overline{C} + BCD$

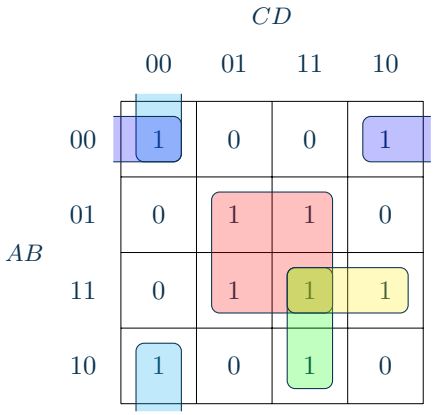
Solution:



$B\overline{C} + BD + \overline{A}C\overline{D}$

(b) $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}C\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}CD + AB\overline{C}D + ABC\overline{D} + ABCD$

Solution:



$BD + \overline{A}\,\overline{B}\,\overline{D} + \overline{B}\,\overline{C}\,\overline{D} + ABC + ACD$

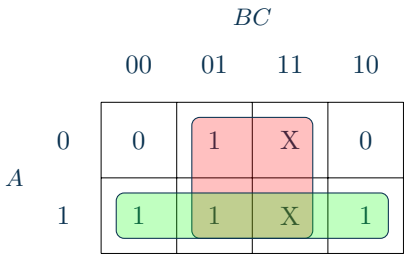
We made some progress, but the final answer is still gross. This is the sort of thing where a multiplexer or LUT really shines, because it can implement *any* 4-input truth table with the same logic. Trying to do this with individual gates would be frustrating.

(c)

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	X
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	X

Solution:

In this case, we include the X's because they allow us to draw a larger circle and eliminate several variables:



$A + C$

(d)

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

Solution:

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

$\overline{A}\,\overline{C} + ABC + AC\overline{D}$

(e)

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

Solution:

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

$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}BD + A\overline{B}D + ABC\overline{D}$

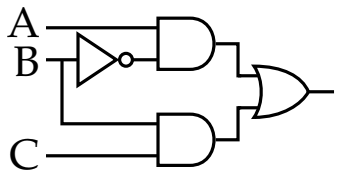
Interestingly, the X's don't allow any additional reduction of the equation!

Problem 2: Gate implementation

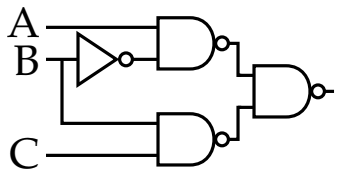
Implement the equation $Y = A\overline{B} + BC$ using only NAND gates (no inverters, just NANDs):

Solution:

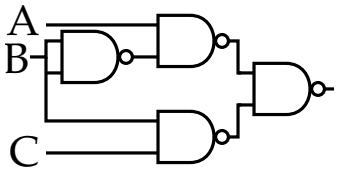
One way to accomplish this is with bubble pushing. Start by drawing the logic diagram as written:



Then we can use involution and bubble-pushing to switch the gates to NANDs:



Finally we need to generate $\overline{B}$, which we can do by using B as both inputs to a NAND gate:



Remember, the whole point of this is that inverting logic (NAND/NOR/NOT) is faster (lower propagation delay), smaller (fewer transistors) and uses less power than non-inverting logic. This actually has real utility when you're designing logic at the lowest level.

Problem 3: Multiplexers

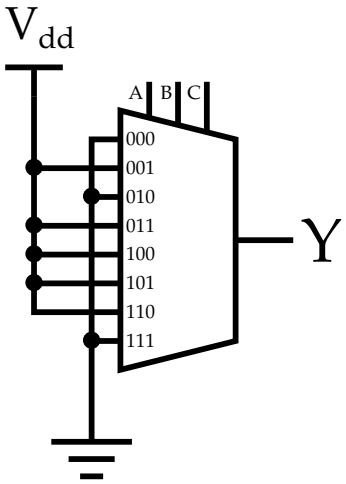
(a) Implement the equation $Y = A\overline{C} + \overline{A}C + \overline{B}C$ using an 8:1 multiplexer.

Solution:

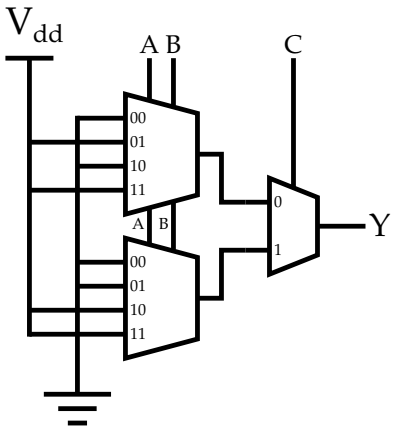
This equation isn't in canonical form, so each term represents more than one row of the truth table (and therefore more than one input to the mux.) Start by writing out the truth table.

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

From here, the multiplexer implementation is straightforward:



(b) Write a truth table for this circuit, showing the output Y as a function of the inputs A, B, and C.



Solution:

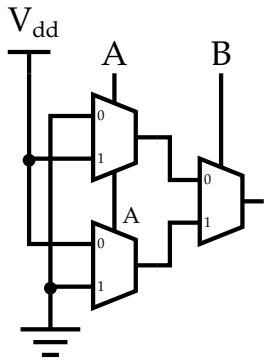
Be careful of the ordering of A, B, and C! If you draw them in the conventional order (as below), note that you'll need to read alternately from the top and bottom 4:1 multiplexers.

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

(c) Draw a logic diagram showing how you could implement the equation $Y = A \oplus B$ using only 2:1 multiplexers. *Hint: It might help to first draw how you could implement this with a 4:1 multiplexer.*

Solution:

One solution is to just build a 4:1 mux with three 2:1 multiplexers, and wire up the inputs to match the truth table of an XOR:



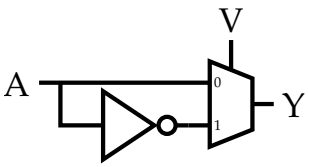
(d) Draw a logic diagram for a circuit with the following behavior:

- When V is low, the output signal is simply the input: $Y = A$.
- When V is high, the output signal is inverted : $Y = \overline{A}$.

Write a truth table for Y in terms of A and V. What do you notice?

Solution:

Here is one possible logic implementation:



And here's the truth table:

V	A	Y
0	0	0
0	1	1
1	0	1
1	1	0

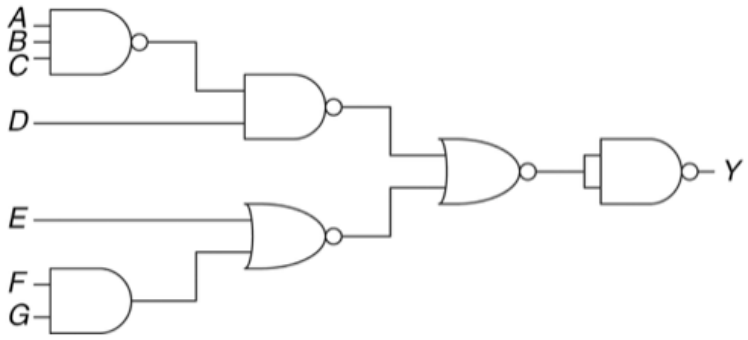
Hopefully you don't have to look at this too long to notice that this is just another XOR gate!

This is actually a helpful and common way to think about XOR: it allows one signal to control whether another is inverted or not.

Problem 4: Timing

Gate	t_{pd} (ps)	t_{cd} (ps)
NOT	15	10
2-input NAND	20	15
3-input NAND	30	25
2-input NOR	30	25
3-input NOR	45	35
2-input AND	30	25
3-input AND	40	30
2-input OR	40	30
3-input OR	55	45
2-input XOR	60	40

(a) Determine the propagation delay and contamination delay of the circuit in the figure below. Use the gate delays given in the table above.

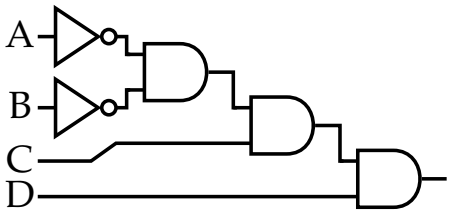


Solution:

The propagation delay on the path from $A/B/C$ to the output is $30 + 20 + 30 + 20 = 100$ps. The propagation delay on the path from F/G to the output is $30 + 30 + 30 + 20 = 110$ps. So this is the critical path, and the overall propagation delay is 110 ps.

The short path will be from D to the output, with a contamination delay of $15 + 25 + 15 = 55$ps.

(b) Determine the propagation delay and contamination delay of the circuit in the figure below. Use the gate delays given in the table above.



Redesign the circuit so that it runs faster (i.e., lower propagation delay), using the gates in the table above. There are multiple ways to speed it up!

Solution:

The critical path is from A or B to the output:

$$t_{pd} = 15 + 30 + 30 + 30 = 105\text{ps}$$

The short path will be from D to the output:

$$t_{cd} = 25\text{ps}$$

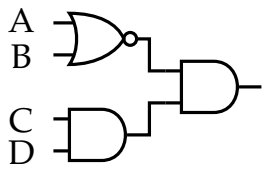
There are several things we could do to speed this up:

- Use fewer gates
- Use faster gates
- Rearrange the gates to shorten the critical path while performing the same logical function

In this case, A and B compute $\overline{A}\,\overline{B} = \overline{A + B}$ (by DeMorgan's theorem), so we can replace the NOT/AND gates with a single NOR.

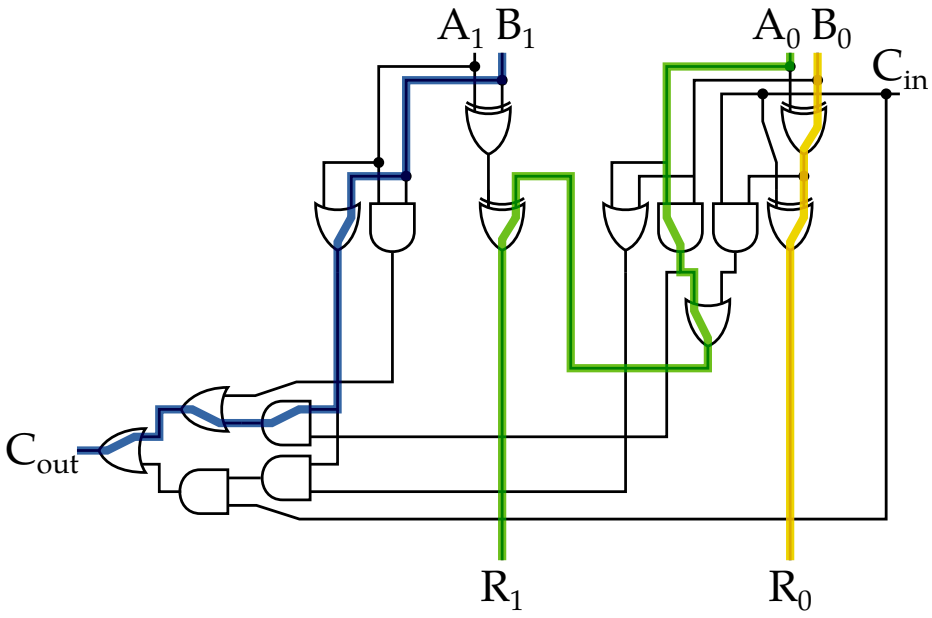
The two remaining AND gates can be combined into a 3-input AND gate, giving an overall delay of $30 + 40 = 70$ps.

For these particular gate delay numbers, we can actually do slightly better by putting the operation on C and D in parallel with A and B:



This has an overall propagation delay of $30 + 30 = 60$ps.

(c) Below is a section of a carry-lookahead adder with a block size of 2. For each of the three outputs (R_0, R_1 and C_{cout}), highlight the critical path. Use the gate delays given in the table above.



Solution:

One set of critical paths is shown above. Note that there are several equivalent paths for each output (e.g., A_0 to R_0 is the same delay as B_0 to R_0.)

Practice Problems - For review

These are selected problems from the textbook (at the end of each chapter) which may be helpful for practice and review. The answers to these problems are online at `https://booksite.elsevier.com/9780128000564/solutions.php`.

- 2.17 (simplifying equations, drawing circuits)
- 2.27 (bubble pushing)
- 2.31 (minimizing with don't-cares)
- 2.35 (writing and minimizing equations, drawing circuits)