Boolean Logic

Boolean logic lives in a black-and-white world where everything is either true (logic 1) or false (logic 0). There are other forms of logic, such as fuzzy logic, where things can be grey. That's for another course, however.

Standard Boolean functions are as follows:

AND: Y is true of A and B are both true. It's false otherwise.

- Symbol: ·
- Truth Table:

A	B	$Y = AB$	$Y = \overline{AB}$
		AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

OR: Y is true if either A or B is true. Note that logical OR is different than the word 'or' in English, which usually means A or B but not both. ("Either you or I are going to the store" means that we're not both going to go.) Logical OR is more akin to the English term 'and/or'.

- Symbol: +
- Truth Table:

A	B	$Y = A + B$	$Y = \overline{A + B}$
		OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

NOT: Y = NOT A means that whatever A is, Y is the opposite.

Symbols: Not A is written as

- $\overline{A}$
- $\sim A$
- A'

There are a few other variations, such as

NOR: Not OR

A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

DeMorgan's Theorem:

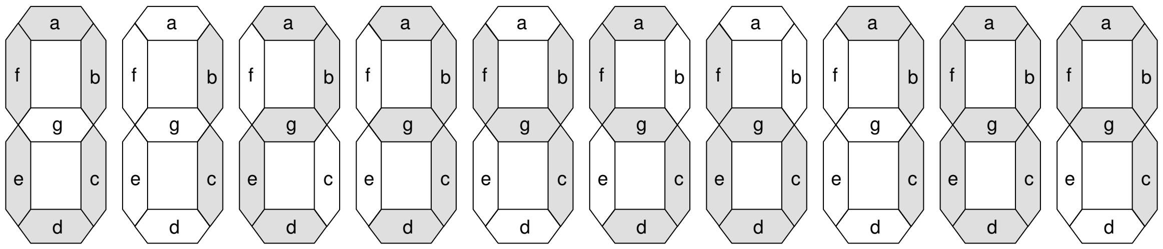
$$\overline{AB} = \overline{A} + \overline{B}$$

$$\overline{A+B} = \overline{A} \cdot \overline{B}$$

Implementing Logic Using NAND Gates

To implement logic using NAND gates, you circle the ones.

For example, suppose you want to design a circuit to light up LED (a) for a 7-segment display



- Input = 4 digital signals (ABCD)
- Output: 0 = light off, 1 = light on
- Relationship:

If you represent the number as ABCD, then the value for ABCD on a Karnaugh map would be

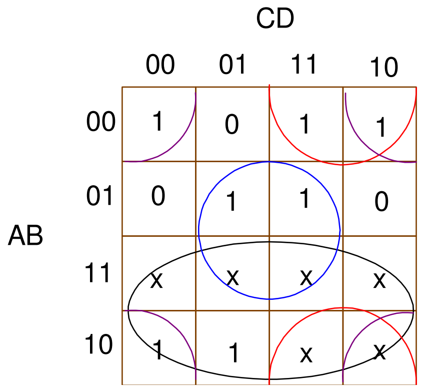
Value of ABCD	CD 00	01	11	10
AB 00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Filling in each entry with whether LED (a) is on to represent that number gives the following Karnough map:

Ya(ABCD) CD

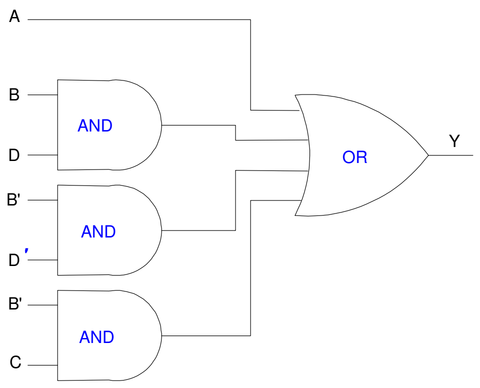
AB	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	x	x	x	x
10	1	1	x	x

Circle the ones to generate Ya:

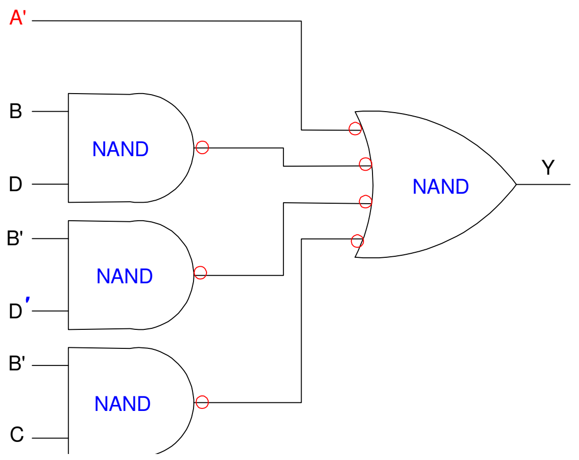


$$Y = A + BD + \overline{B}\overline{D} + \overline{B}C$$

Implement this using AND and OR gates



To convert to NAND gates, add in a double-negative

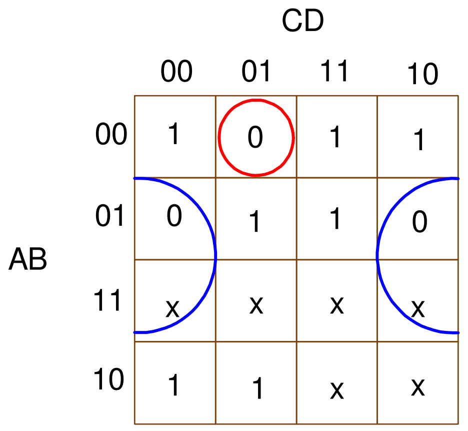


Implementation of Ya using NAND gates

Implementation using NOR Gates

To use NOR gates, you circle the zeros. For example, repeat the previous design:

First, you circle the zeros to generate not Y



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

$$\overline{Y} = \overline{A}\,\overline{B}\,\overline{C}D + B\overline{D}$$

Negate both sides to find Y

$$Y = \overline{\overline{A}\,\overline{B}\,\overline{C}D + B\overline{D}}$$

Use DeMorgan's theorem:

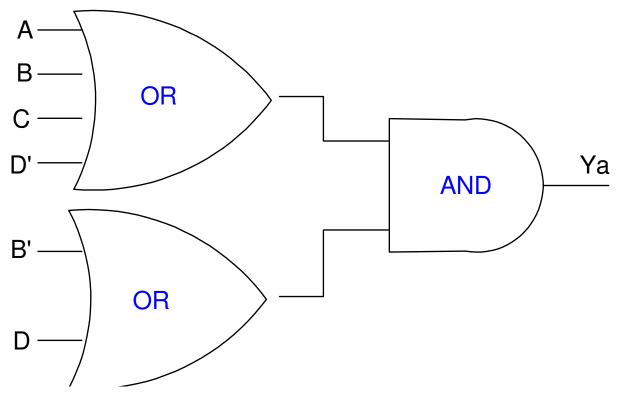
$$\overline{AB} = \overline{A} + \overline{B}$$

$$\overline{A+B} = \overline{A}\,\overline{B}$$

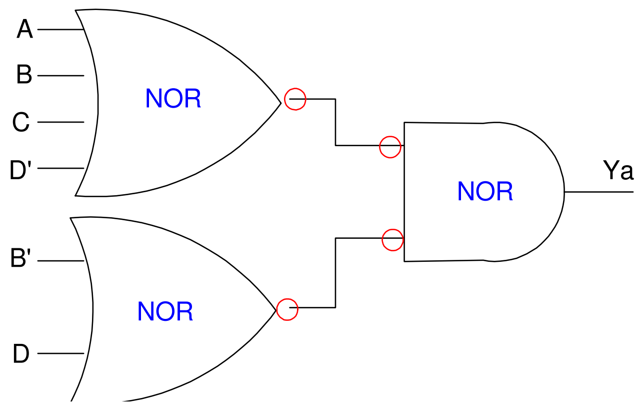
to give

$$Y = (A + B + C + \overline{D})(\overline{B} + D)$$

Implement using OR and AND gates



Add in a double-negative to turn these into NOR gates



Implementation of Ya using NOR gates

Summary:

You can implement any logic using NAND or NOR gates. The only difference is if you prefer circling the ones (NAND) or zeros (NOR).