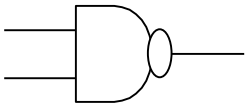
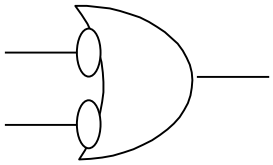
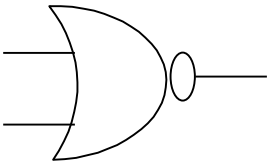
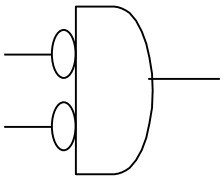
De Morgan's theorem

$$\left.\begin{aligned} \overline{A \,.\, B} &= \bar{A} + \bar{B} \\ \overline{A + B} &= \bar{A} \,.\, \bar{B} \end{aligned}\right\}$$

Thus,  **is equivalent to** 

Verify it using truth tables. Similarly,

 **is equivalent to** 

These can be generalized to more than two variables: to

$$\left.\begin{aligned} \overline{A.\,B.\,C} &= \bar{A} + \bar{B} + \bar{C} \\ \overline{A + B + C} &= \bar{A} \,.\, \bar{B} \,.\, \bar{C} \end{aligned}\right\}$$

Synthesis of logic circuits

Many problems of logic design can be specified using a truth table. Give such a table, can you design the logic circuit?

Design a logic circuit with three inputs A, B, C and one output F such that F=1 only when a majority of the inputs is equal to 1.

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

Sum of product form

$$F = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$

Draw a logic circuit to generate F

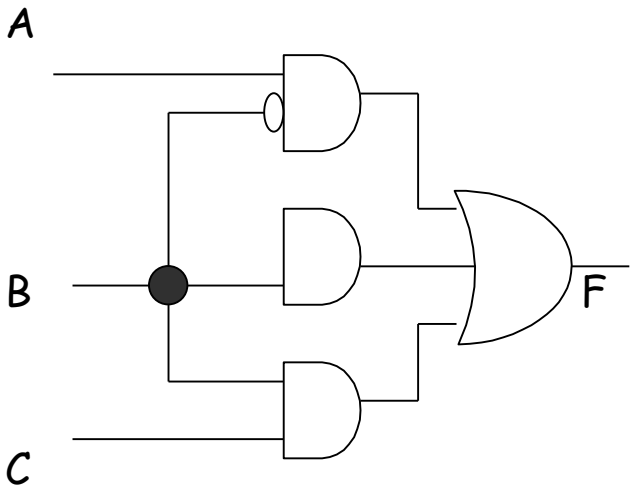
Simplification of Boolean functions

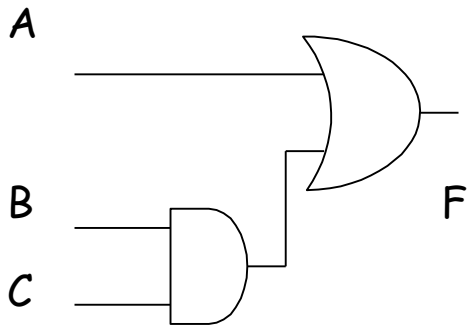
Using the theorems of Boolean Algebra, the algebraic forms of functions can often be simplified, which leads to simpler (and cheaper) implementations.

Example 1

$$
\begin{aligned}
F &= A.\overline{B} + A.B + B.C \\
&= A.(\overline{B} + B) + B.C \\
&= A.1 + B.C \\
&= A + B.C
\end{aligned}
$$

How many gates do you save from this simplification?





Example 2

$$F = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$

$$= \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C + A.B.C + A.B.C$$

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

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

$$= B.C + C.A + A.B$$

Example 3 Show that $A + A.B = A$

$$A + AB$$

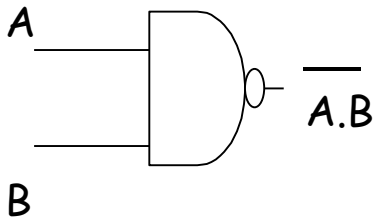
$$= A.1 + A.B$$

$$= A.\ (1 + B)$$

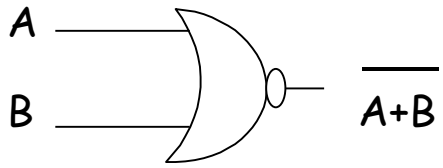
$$= A.\ 1$$

$$= A$$

Other types of gates

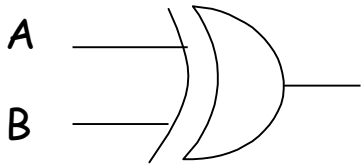


NAND gate



NOR gate

Be familiar with the truth tables of these gates.



$A \oplus B = \overline{A}.B + A.\overline{B}$

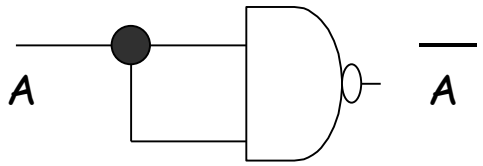
Exclusive OR (XOR) gate

NAND and NOR are universal gates

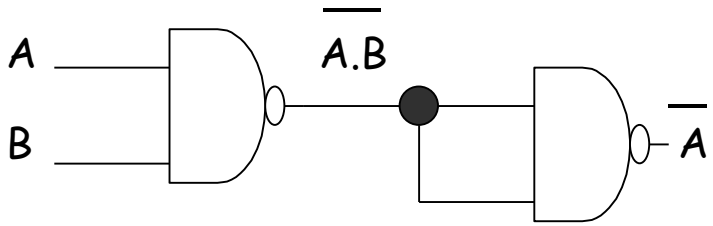
Any function can be implemented using **only NAND** or **only NOR** gates. How can we prove this?

(Proof for NAND gates) Any boolean function can be implemented using AND, OR and NOT gates. So if AND, OR and NOT gates can be implemented using NAND gates only, then we prove our point.

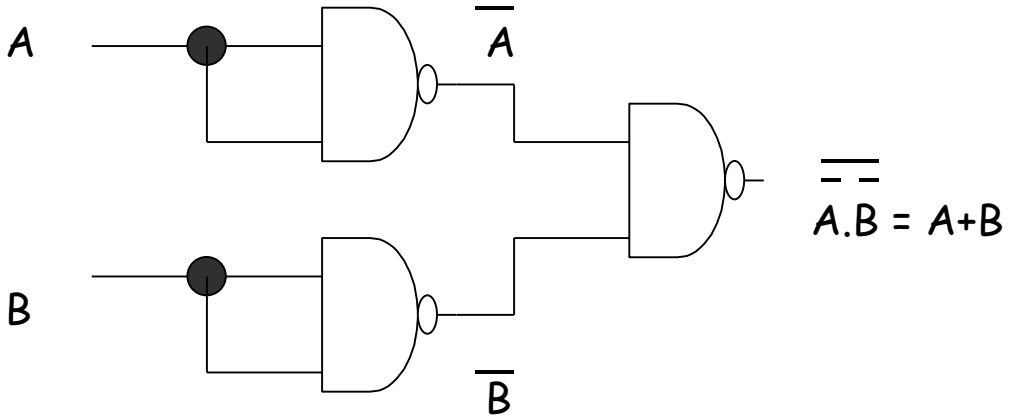
1. Implement NOT using NAND



2. Implementation of AND using NAND



1. Implementation of OR using NAND



Exercise. *Prove that NOR is a universal gate.*