<u>Boolean Algebra</u>

A + O = A	A + A' = 1
A . 1 = A	A. A' = 0

 $\begin{array}{c}
1 + A = 1 \\
0. A = 0
\end{array}$ $\begin{array}{c}
A + B = B + A \\
A \cdot B = B \cdot A
\end{array}$

A + (B + C) = (A + B) + CA. (B. C) = (A. B). C

$$A + A = A$$
$$A \cdot A = A$$

A. (B + C) = A.B + A.CA + B.C = (A+B). (A+C)

Distributive Law

$$A \cdot B = A + B$$

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$
De Morgan's theorem

De Morgan's theorem

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

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



Verify it using truth tables. Similarly,



These can be generalized to more than two variables: to

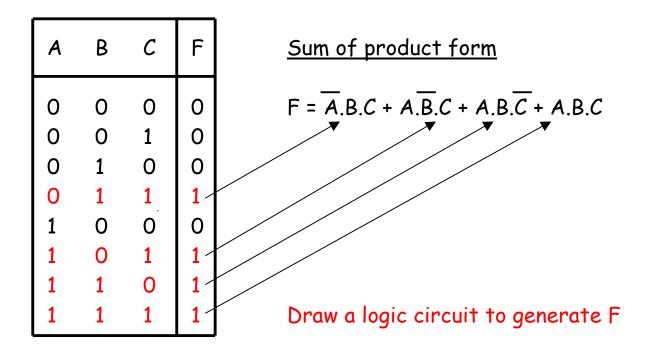
$$\overline{A. B. C} = \overline{A + B + C}$$

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

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.



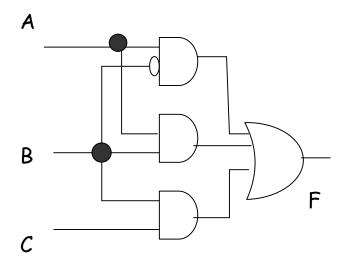
Simplification of Boolean functions

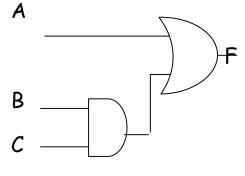
Using the theorems of Boolean Algebra, the algebraic forms of functions can often be simplified, which leads to simpler (and cheaper) implementations.

Example 1

F	=	A.B + A.B + B.C
	=	A. (B + B) + B.C
	=	A.1 + B.C
	=	A + B.C

How many gates do you save from this simplification?





Example 2

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

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

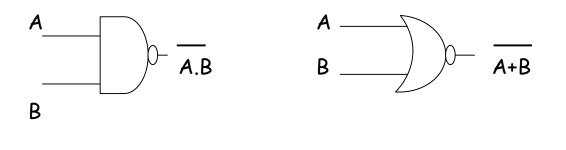
$$= (\overline{A.B.C + A.B.C}) + (\overline{A.B.C + A.B.C}) + (\overline{A.B.C + A.B.C})$$

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

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

- Example 3 Show that A + A.B = A
 - A + AB
- = A.1 + A.B
- = A. (1 + B)
- = A.1
- = A

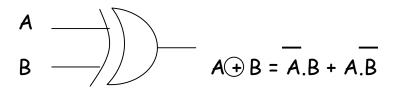
Other types of gates



NAND gate

NOR gate

Be familiar with the truth tables of these gates.



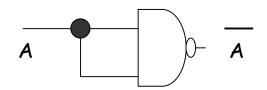
Exclusive OR (XOR) gate

NAND and NOR are universal gates

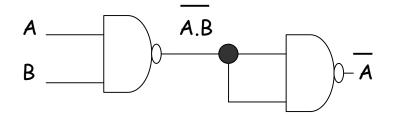
Any function can be implemented using only NAND or only NOR gates. How can we prove this?

(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.

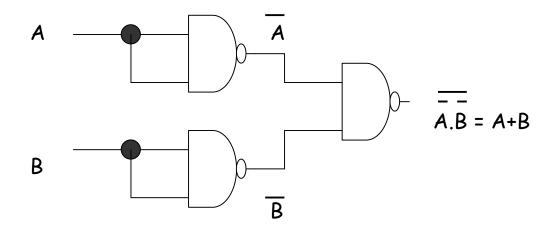
1. Implement NOT using NAND



2. Implementation of AND using NAND



1. Implementation of OR using NAND



Exercise. Prove that NOR is a universal gate.