

# Logic Gate



College of  
Engineering & Technology



Al-Mustaqbal  
University

Level 1 , Semester 1  
@ Department of prosthetic and orthotic Engineering

Prepared by  
Dr. Samir Badrawi  
2024-2025

## Arithmetic Operations & Boolean Algebra

*The majority of this course material is based on text and presentations of :*

*Floyd, Digital Fundamentals, 10<sup>th</sup> ed., © 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved*

## Octal Numbers

Octal uses eight characters the numbers 0 through 7 to represent numbers. There is no 8 or 9 character in octal.

Binary number can easily be converted to octal **by grouping bits 3 at a time** and writing the equivalent octal character for each group.

**Example**

Express  $1\ 001\ 011\ 000\ 001\ 110_2$  in octal:

**Solution**

Group the binary number by 3-bits starting from the right. Thus,  **$113016_8$**

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

## Octal Numbers

Octal is also a weighted number system. The column weights are powers of 8, which increase from right to left.

Column weights  $\left\{ \begin{array}{cccc} 8^3 & 8^2 & 8^1 & 8^0 \\ 512 & 64 & 8 & 1 \end{array} \right.$

**Example** Express  $3702_8$  in decimal.

**Solution** Start by writing the column weights:

512 64 8 1  
3 7 0  $2_8$

$$3(512) + 7(64) + 0(8) + 2(1) = 1986_{10}$$

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

## Hexadecimal Numbers

Hexadecimal uses sixteen characters to represent numbers: the numbers 0 through 9 and the alphabetic characters A through F.

Large binary number can easily be converted to hexadecimal by grouping bits 4 at a time and writing the equivalent hexadecimal character.

**Example** Express  $1001\ 0110\ 0000\ 1110_2$  in hexadecimal:

**Solution** Group the binary number by 4-bits starting from the right. Thus, **960E**

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

## Hexadecimal Numbers

Hexadecimal is a weighted number system. The column weights are powers of 16, which increase from right to left.

$$\text{Column weights } \left\{ \begin{array}{cccc} 16^3 & 16^2 & 16^1 & 16^0 \\ 4096 & 256 & 16 & 1 \end{array} \right.$$

**Example** Express  $1A2F_{16}$  in decimal.

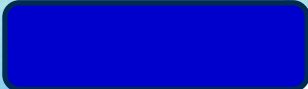
**Solution** Start by writing the column weights:







4096	256	16	1
------	-----	----	---

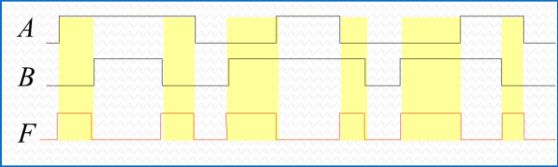
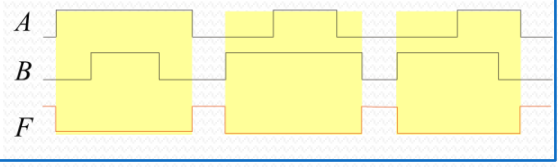
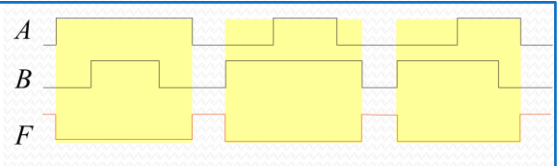
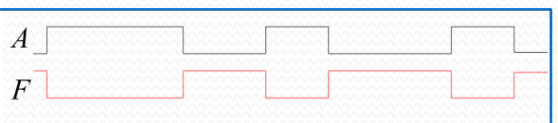
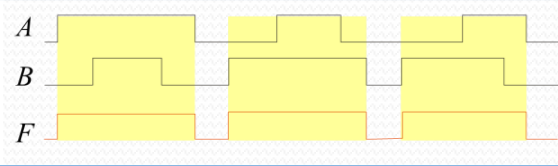
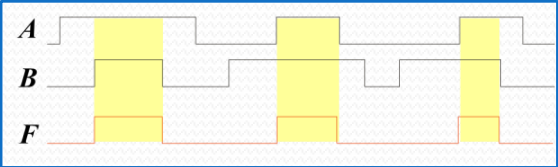
1	A	2	$F_{16}$
---	---	---	----------

$$1(4096) + 10(256) + 2(16) + 15(1) = 6703_{10}$$

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111



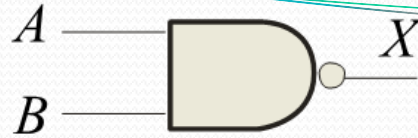
Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																



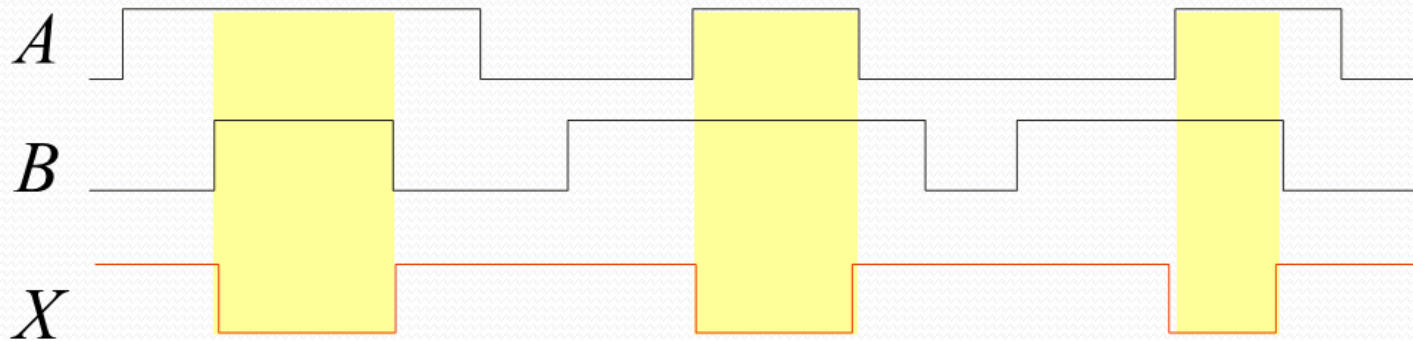
# Basic Logic Gated

**F : Output Waveforms**

## The NAND Gate



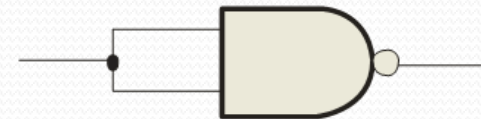
Example waveforms:



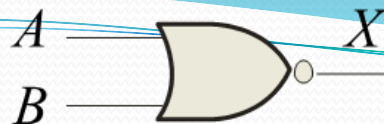
The NAND gate is particularly useful because it is a “universal” gate – all other basic gates can be constructed from NAND gates.

### Question

How would you connect a 2-input NAND gate to form a basic inverter?



## The NOR Gate



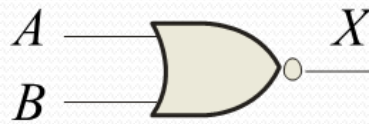
The **NOR** gate produces a LOW output if any input is HIGH; if all inputs are HIGH, the output is LOW. For a 2-input gate, the truth table is

Inputs		Output
<i>A</i>	<i>B</i>	<i>X</i>
0	0	1
0	1	0
1	0	0
1	1	0

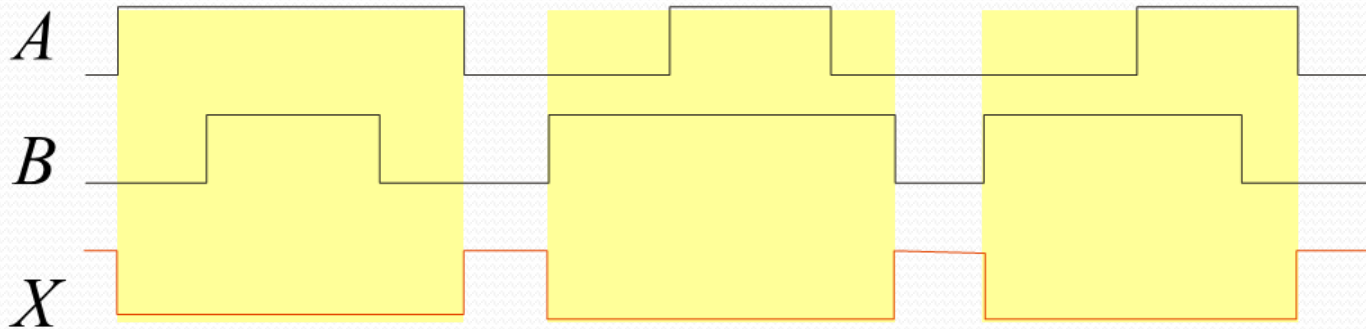
The **NOR** operation is shown with a plus sign (+) between the variables and an overbar covering them. Thus, the NOR operation is written as  $X = \overline{A + B}$ .



## The NOR Gate



Example waveforms:



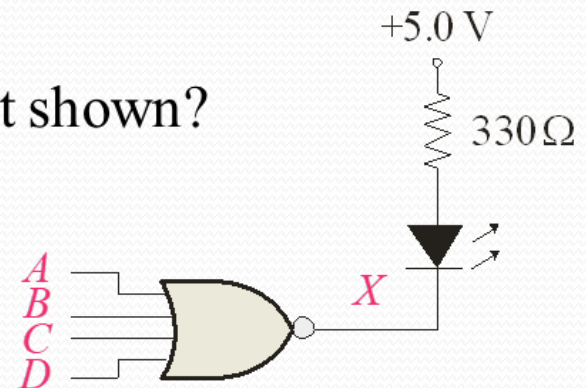
The NOR operation will produce a LOW if any input is HIGH.

**Example**

When is the LED is ON for the circuit shown?

**Solution**

The LED will be on when any of the four inputs are HIGH.

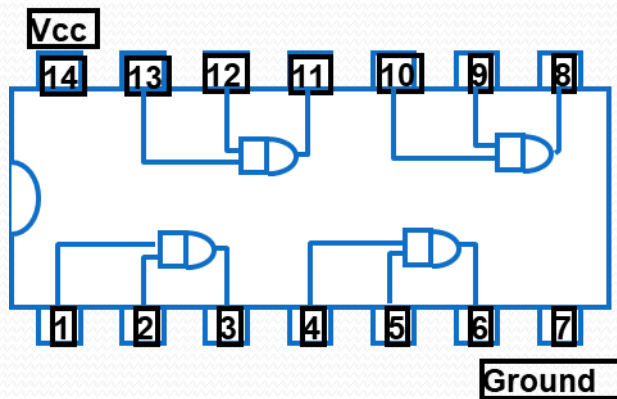


# Combinational Logic circuits

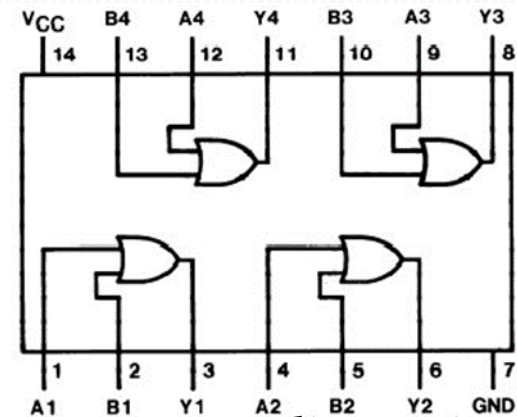
The AND, OR, NAND, NOR and NOT are the basic building blocks of digital systems.

The art of digital system design is to link these building blocks together in the most effective manner to construct what is called *Combinational Logic circuits*.

These are called *combinational circuits* because the *output of a circuit at any given moment depends on the combination of input signals present*.

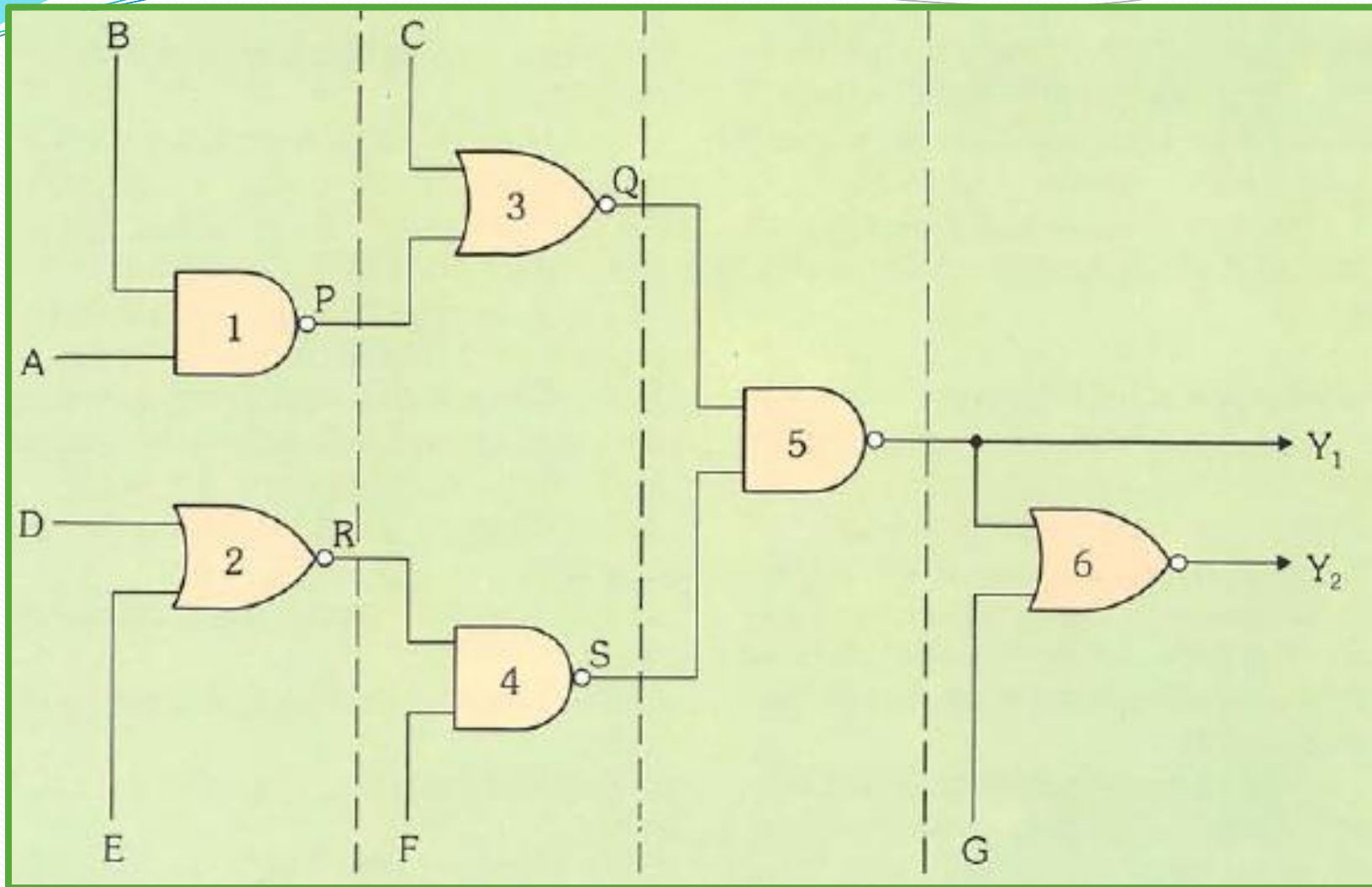


74LS08 Quad 2-input  
AND Gate IC Package



74LS08 Quad 2-input  
OR Gate IC Package

# *combinational circuits*



**How many inputs (i/p) and how many outputs (o/p) in the above circuit ?**

# Algebraic Function

## Boolean Function Representation

The use of switching devices like transistors give rise to use a special mathematics case called the *Boolean algebra*.

The principles of logic were developed by *George Boole* (1815-1884) who, along with *Augustus De Morgan*, formulated a basic set of rules that govern the relationship between the true - false statements of logic. Boole's work laid the foundation to what later became known as *Boolean Algebra*.

Nearly one hundred years later, *Claude Shannon*, an American postgraduate at Massachusetts Institute of Technology (MIT), realized that Boole's work was relevant in particular to the analysis of *switching circuits in telephone exchanges* and, more generally, formed a mathematical basis for the electronic processing of binary information.

## Boolean Addition

In Boolean algebra, a **variable** is a symbol used to represent an action, a condition, or data. A single variable can only have a value of 1 or 0.

The **complement** represents the inverse of a variable and is indicated with an overbar. Thus, the complement of  $A$  is  $\overline{A}$ .

A **literal** is a variable or its complement.

Addition is equivalent to the OR operation. The sum term is 1 if one or more of the literals are 1. The sum term is zero only if each literal is 0.

**Example** Determine the values of  $A$ ,  $B$ , and  $C$  that make the sum term of the expression  $\overline{A} + B + \overline{C} = 0$ ?

**Solution** Each literal must = 0; therefore  $A = 1$ ,  $B = 0$  and  $C = 1$ .

## Boolean Multiplication

In Boolean algebra, multiplication is equivalent to the AND operation. The product of literals forms a product term. The product term will be 1 only if all of the literals are 1.

**Example** What are the values of the  $A$ ,  $B$  and  $C$  if the product term of  $A \cdot \overline{B} \cdot \overline{C} = 1$ ?

**Solution** Each literal must = 1; therefore  $A = 1$ ,  $B = 0$  and  $C = 0$ .

## Commutative Laws

The **commutative laws** are applied to addition and multiplication. For addition, the commutative law states

**In terms of the result, the order in which variables are ORed makes no difference.**

$$A + B = B + A$$

For multiplication, the commutative law states

**In terms of the result, the order in which variables are ANDed makes no difference.**

$$AB = BA$$

## Associative Laws

The **associative laws** are also applied to addition and multiplication. For addition, the associative law states

**When ORing more than two variables, the result is the same regardless of the grouping of the variables.**

$$A + (B + C) = (A + B) + C$$

For multiplication, the associative law states

**When ANDing more than two variables, the result is the same regardless of the grouping of the variables.**

$$A(BC) = (AB)C$$

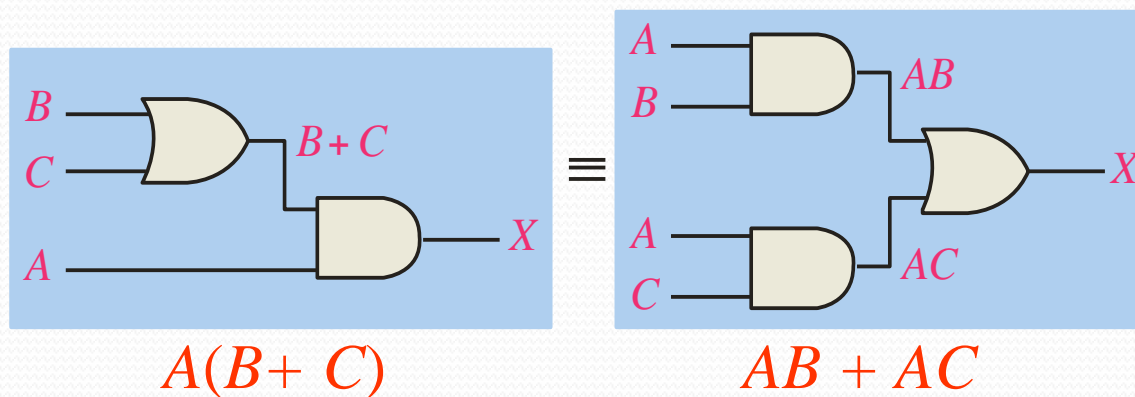


## Distributive Law

The **distributive law** is the factoring law. A common variable can be factored from an expression just as in ordinary algebra. That is

$$AB + AC = A(B + C)$$

The distributive law can be illustrated with equivalent circuits:



## Rules of Boolean Algebra

$$1. A + 0 = A$$

$$2. A + 1 = 1$$

$$3. A \cdot 0 = 0$$

$$4. A \cdot 1 = A$$

$$5. A + A = A$$

$$6. A + \bar{A} = 1$$

$$7. A \cdot A = A$$

$$8. A \cdot \bar{A} = 0$$

$$9. \bar{\bar{A}} = A$$

$$10. A + AB = A$$

$$11. A + \bar{A}B = A + B$$

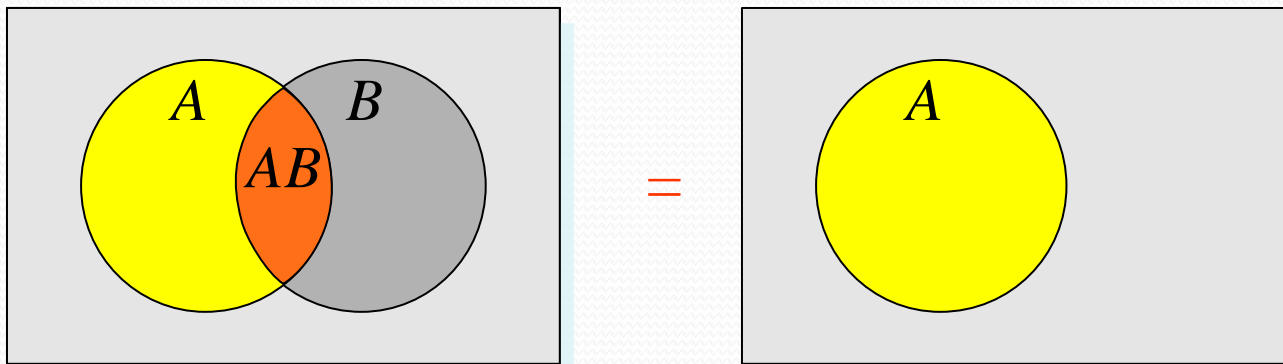
$$12. (A + B)(A + C) = A + BC$$

## Rules of Boolean Algebra

Rules of Boolean algebra can be illustrated with *Venn* diagrams. The variable  $A$  is shown as an area.

The rule  $A + AB = A$  can be illustrated easily with a diagram. Add an overlapping area to represent the variable  $B$ .

The overlap region between  $A$  and  $B$  represents  $AB$ .



The diagram visually shows that  $A + AB = A$ . Other rules can be illustrated with the diagrams as well.

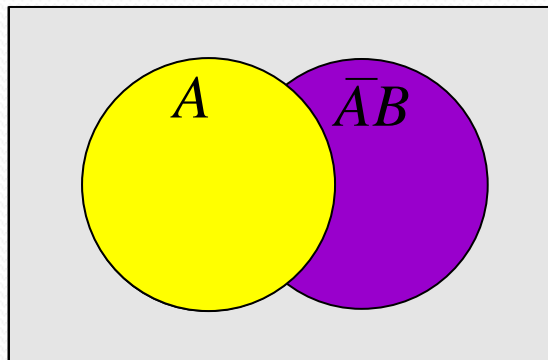
## Rules of Boolean Algebra

**Example** Illustrate the rule  $A + \bar{A}B = A + B$  with a Venn diagram. (Rule 11)

**Solution**

This time,  $A$  is represented by the blue area and  $B$  again by the red circle.

The intersection represents  $\bar{A}B$ . Notice that  $A + \bar{A}B = A + B$



## Rules of Boolean Algebra

**Rule 12**, which states that  $(A + B)(A + C) = A + BC$ , can be proven by applying earlier rules as follows:

$$\begin{aligned}(A + B)(A + C) &= AA + AC + AB + BC \\&= A + AC + AB + BC \\&= A(1 + C + B) + BC \\&= A \cdot 1 + BC \\&= A + BC\end{aligned}$$

This rule is a little more complicated, but it can also be shown with a Venn diagram, as given on the following slide...

## Rule 12 :

$(A + B)(A + C) = A + BC$  can be proven by Venn Diagram :-

Three areas represent the variables  $A$ ,  $B$ , and  $C$ .

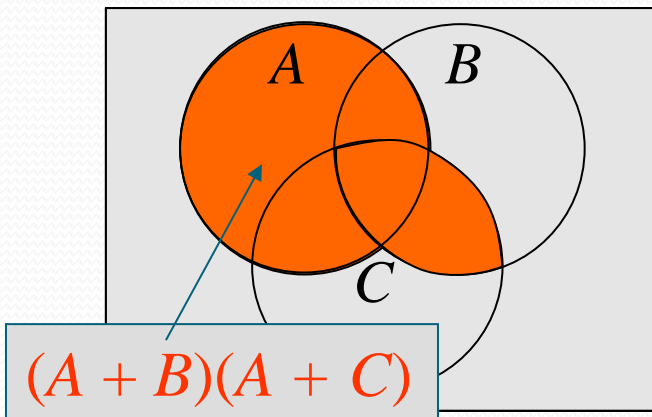
The area representing  $A + B$  is shown in yellow.

The area representing  $A + C$  is shown in red.

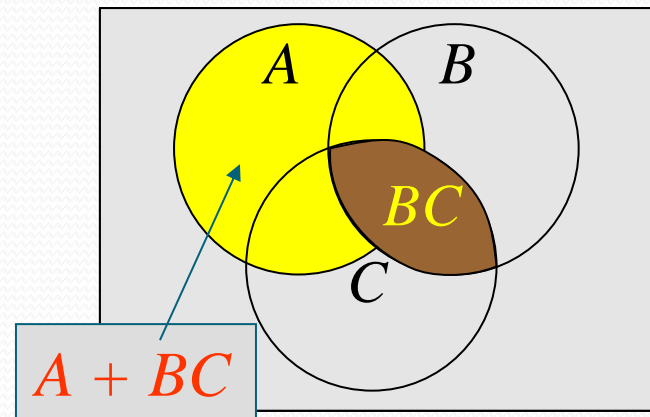
The overlap of red and yellow is shown in orange.

The overlapping area between  $B$  and  $C$  represents  $BC$ .

ORing with  $A$  gives the same area as before.



=



## Homework (team/group work)

*using Boolean algebra Rules,*

*1. prove :*

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

*2. Show that the function:*

$$F = A B C + A B C + A B C + A B C$$

*equals 1 when A equals 1.*

# DeMorgan's Theorems

## DeMorgan's 1<sup>st</sup> Theorem

The complement of a product of variables is equal to the sum of the complemented variables.

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

## DeMorgan's 2<sup>nd</sup> Theorem

The complement of a sum of variables is equal to the product of the complemented variables.

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

To apply DeMorgan's theorem to the expression, you can **break** the overbar covering both terms and **change** the sign between the terms.