6- Logic circuits (networks).

Logic gates can be combined together to produce more complex logic circuits (networks).

Note: The output from a logic circuit (network) is checked by producing a truth table.

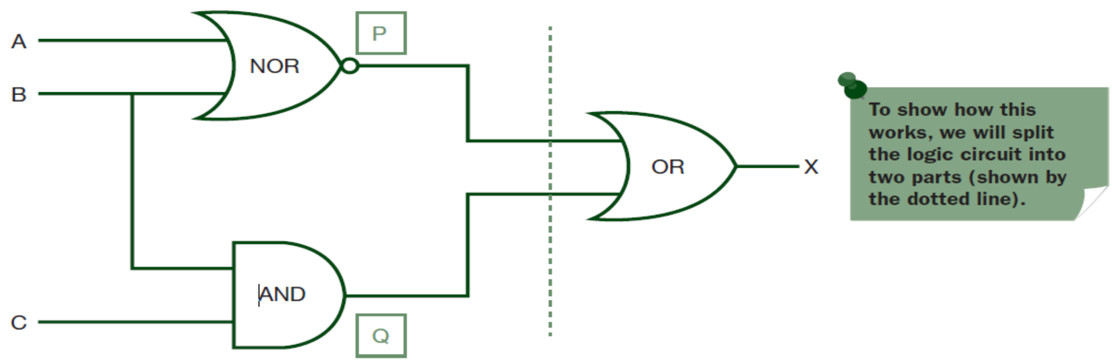
Two different types of problem are considered here:

– drawing the truth table from a given logic circuit (network)

– designing a logic circuit (ne twork) from a given problem and testing it by
also drawing a truth table.

Example 1

Produce a truth table from the following logic circuit (network).



-First part

There are 3 inputs; thus we must have 23 (i.e. 8) possible combinations of 1s
and 0s. To find the values (outputs) at points **P** and **Q**, it is necessary to consider the truth tables for the NOR gate (output P) and the AND gate (output Q) i.e.

P = A **NOR** B

Q = B **AND** C

We thus get:

INPUT A	INPUT B	INPUT C	OUTPUT P	OUTPUT Q
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

-Second part

There are 8 values from **P** and **Q** which form the inputs to the last **OR** gate.

Hence we get X = P **OR** Q which gives the following truth table:

INPUT P	INPUT Q	OUTPUT X
1	0	1
1	0	1
0	0	0
0	1	1
0	0	0
0	0	0
0	0	0
0	1	1

Which now gives us the final truth table for the logic circuit given at the start of the example:

INPUT A	INPUT B	INPUT C	OUTPUT X
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Example 2

Consider the following problem.

A system used 3 switches A, B and C; a combination of switches determines
whether an alarm, X, sounds:

If switch A or switch B are in the ON position and if switch C is in the OFF position then a signal to sound an alarm, X is produced. It is possible to convert this problem into a logic statement.

So we get:

If (A = 1 **OR** B = 1) **AND** (C = **NOT** 1) then X = 1

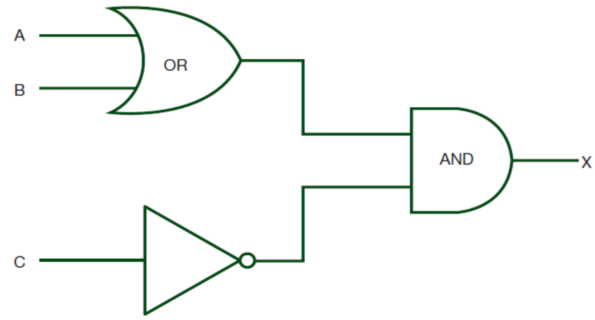
The first part is two inputs (A and B) joined by an **OR** gate

The output from the first part and the third part are joined by an **AND** gate

The third part is one input (C) which is put through a **NOT** gate

So we get the following logic circuit (network):

Remembering that ON = 1 and OFF = 0; also remember that we write 0 as NOT 1.



This gives the following truth table:

INPUT A	INPUT B	INPUT C	OUTPUT X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Home works :

***Qutation1*: **A manufacturing process is controlled by a built in logic circuit which is made up of AND, OR and NOT gates only. The process receives a STOP signal (i.e. X = 1) depending on certain conditions, shown in the following table:

INPUTS	BINARY VALUES	CONDITION IN PROCESS
V	1	Volume > 1000 litres
	0	Volume <= 1000 litres
T	1	Temperature > 750°C
	0	Temperature <= 750°C
S	1	Speed > 15 metres/second (m/s)
	0	Speed <= 15 metres/ second (m/s)

A stop signal (X = 1) occurs when:

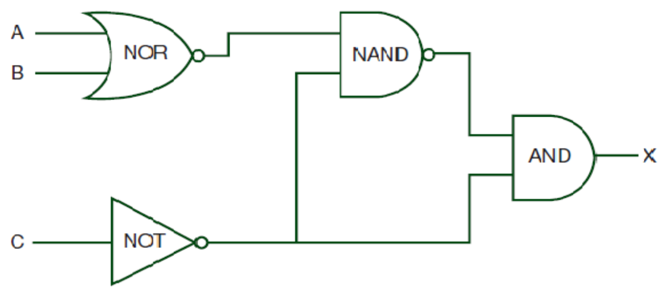
either Volume, V > 1000 litres and Speed, S <= 15 m/s
or Temperature, T <= 750°C and Speed, S > 15 m/s

***Draw the logic circuit and truth table to show all the possible situations
when the stop signal could be received.***

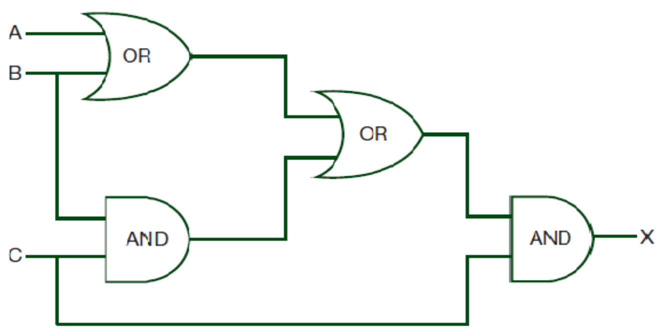
Qutation2:

produce truth tables from the given logic circuits (networks). Remember, if there are two inputs then there will be 4 possible outputs; if there are three inputs then there will be 8 possible outputs.

9.4.3



9.4.4



Important Notes:

1-XOR Notes:

- *The logic function implemented by a 2-input Ex-OR is given as either: "* $Q=(A\oplus B)=\overline{A}.B+A.\overline{B}$ *", which mean "A OR B but NOT both" will give an output at Q.*

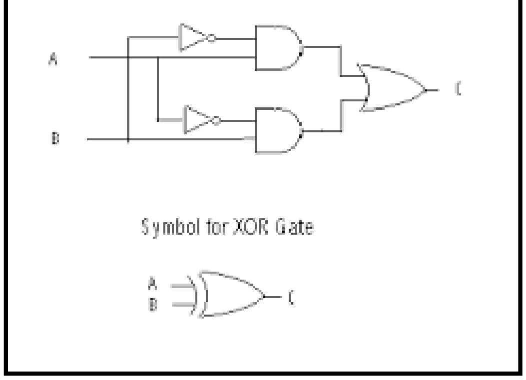
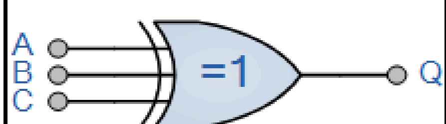


Figure 3. Composite Circuit for XOR Gate

- *in general, an Ex-OR gate will give an output value of logic "1" ONLY when there are an **ODD** number of 1's on the inputs to the gate, if the two numbers are equal, the output is "1".*

- *Then an Ex-OR function with more than two inputs is called an "odd function" or modulo-2-sum (Mod-2-SUM), not an Ex-OR.*

- *This description can be expanded to apply to any number of individual inputs as shown below for a 3-input Ex-OR gate.*

Symbol	Truth Table			
 **3-input Ex-OR Gate**	**C**	**B**	**A**	**Q**
	0	0	0	0
	0	0	1	1
	0	1	0	1
	0	1	1	0
	1	0	0	1
	1	0	1	0
	1	1	0	0
	1	1	1	1
Boolean Expression Q = A ⊕ B ⊕ C	**"Any ODD Number of Inputs" gives Q**			

2-XNOR Notes :

- *The logic function implemented by a 2-input Ex-NOR gate is given as* $Q=\overline{(A\oplus B)}=\overline{A.B}+A.B$ *,which mean "when both A AND B are the SAME" will give an output at Q.*
- *In general, an Exclusive-NOR gate will give an output value of logic "1" ONLY when there are an EVEN number of 1's on the inputs to the gate (the inverse of the Ex-OR gate) except when all its inputs are "LOW" or "0".*
- *Then an Ex-NOR function with more than two inputs is called an "even function" or modulo-2-sum (Mod-2-SUM), not an Ex-NOR.*

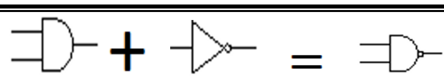
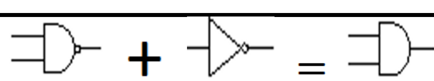
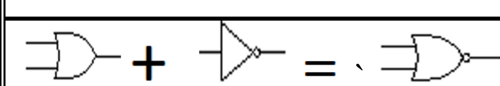
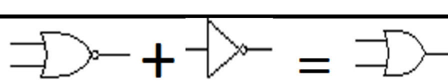
3-Digital Logic Gates Summary

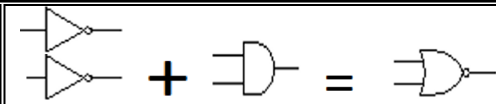
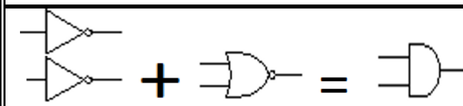
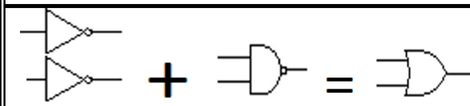
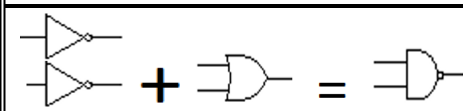
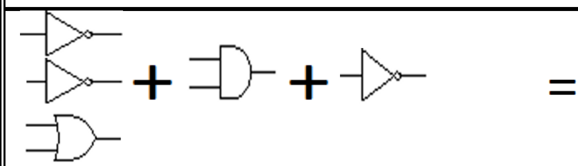
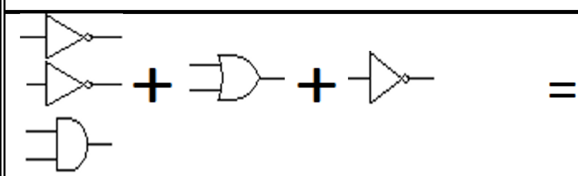
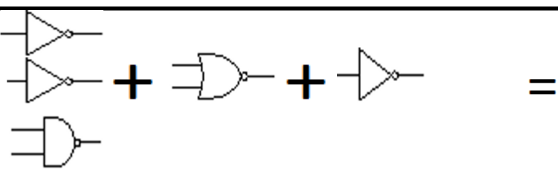
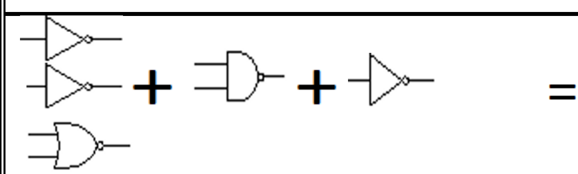
The following logic gates truth table compares the logical functions of the 2-input logic gates .

Inputs		Truth Table Outputs For Each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

3- Using inverters to Convert Gates

Frequently it is convenient to convert a basic gate such as an **AND** , **OR** , **NAND** , or **NOR** to another logic function . this can be done easily with the use of inverters (**NOT gate**). The following chart is a handy guide for converting any given gate to any other logic function.

Invert outputs		AND TO NAND
		NAND TO AND
		OR TO NOR
		NOR TO OR

Invert inputs		AND TO NOR
		NOR TO AND
		NAND TO OR
		OR TO NAND
Invert inputs and outputs		AND TO OR
		OR TO AND
		NOR TO NAND
		NAND TO NOR

✍**Home work**:

write the Truth table for each case in the table above.

6-Boolean Algebra

This section describes various mathematic laws of Boolean algebra . Boolean Algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as Binary Algebra or logical Algebra. Boolean algebra was invented by George Boole in 1854. *variable , complement* and *literal* are terms used in Boolean Algebra .

A **variable** : **is** a symbol used to represent an action , a condition , or data. Any single variable can have only a 1 or a 0 value.

Complement: is the inverse of a variable and is indicated by a bar over the variable (overbar) .for example , the complement of A is $\overline{A}$.

A Literal: is a variable or the complement of a variable.

+ Rule in Boolean algebra

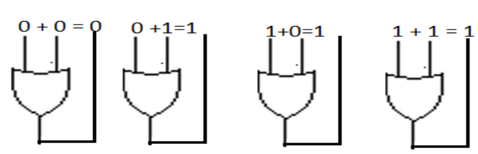
Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.

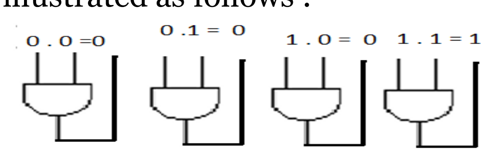
- Complement of a variable is represented by an over bar (-). Thus complement of variable B is represented as . Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.

- **ORing** of the variables is represented by a plus (+) sign between them. For example

ORing of **A, B, C** is represented as **A + B + C.** its also equivalent to the **OR** operation as illustrated as follows :



- Logical **ANDing** of the two or more variable is represented by writing a dot between them such as **A.B.C.** Sometime the dot may be omitted like **ABC.** its also equivalent to the **and** operation as illustrated as follows :



+Boolean Laws

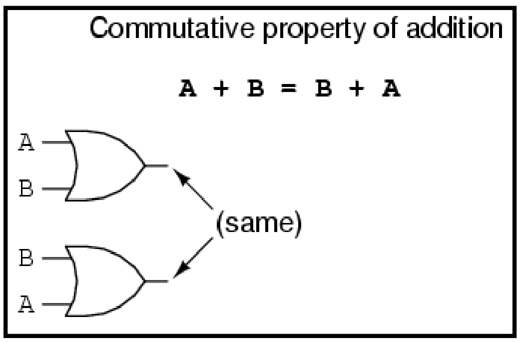
There are six types of Boolean Laws.

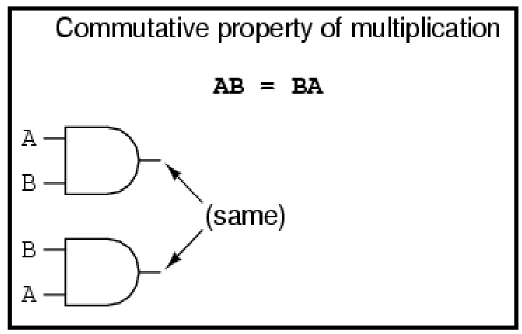
1- COMMUTATIVE LAW

Any binary operation which satisfies the following expression is referred to as commutative operation

$$\boxed{\text{(i) } A \cdot B = B \cdot A \qquad \text{(ii) } A + B = B + A}$$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit. Remember, Boolean Algebra as applied to logic circuits ,the commutative law can applied to OR and AND gate makes no difference , as show in next figures.





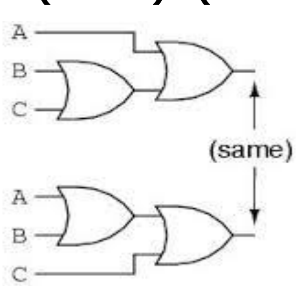
2- ASSOCIATIVE LAW

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

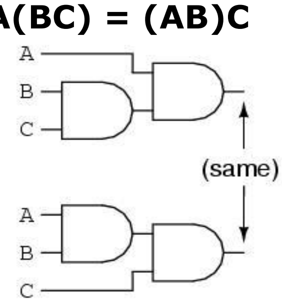
$$\boxed{\text{(i) } (A \cdot B) \cdot C = A \cdot (B \cdot C) \qquad \text{(ii) } (A + B) + C = A + (B + C)}$$

The follows figures show how to applied the associative low to 2-input OR gates and 2-input And gates.

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



A(BC) = (AB)C

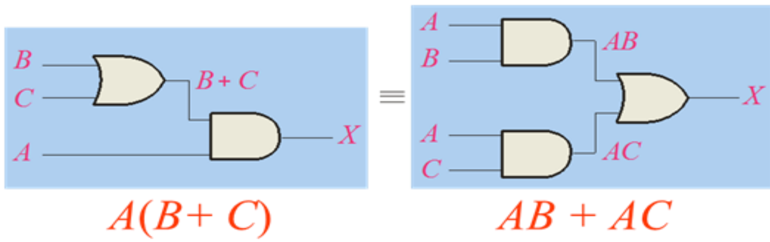


3- DISTRIBUTIVE LAW

Distributive law states the following condition

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

The follows figures show how to applied the distributive low to 2-input OR gates and 2-input And gates.



Where the symbol ≡ mean "equivalent to"

Rules of Boolean Algebra

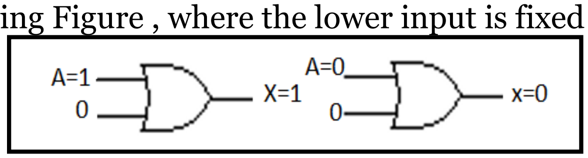
The following table lists the 12 basic rules that are useful in manipulating and simplifying **Boolean expressions** .Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in term of the simpler rules and law previously discussed.

No.	Rule	No.	Rule
1	$A + 0 = A$	7	$A . A = A$
2	$A + 1 = 1$	8	$A . \bar{A} = 0$
3	$A . 0 = 0$	9	$\bar{\bar{A}} = A$
4	$A . 1 = A$	10	$A + AB = A$
5	$A + A = A$	11	$A + \bar{A}B = A + B$
6	$A + \bar{A} = 1$	12	$(A+B)(A+C) = A+BC$

Notes: A ,B or C can represent a single variable or a combination of variables

Rule 1 : A + 0 = A (Identity Law)

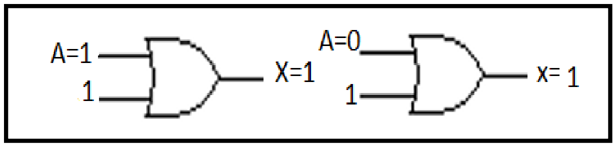
The variable ORed with 0 is always equal to the variable . This rule is illustrated in the following Figure , where the lower input is fixed at 0.



X=A+0=A

Rule 2 : A + 1 = 1 (NULL Elements Law)

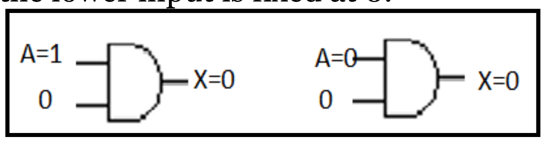
A variable ORed with 1 is always equal to 1. This rule is illustrated in the following Figure , where the lower input is fixed at 1.



X=A+1=1

Rule 3 : A . 0 = 0 (NULL Elements Law)

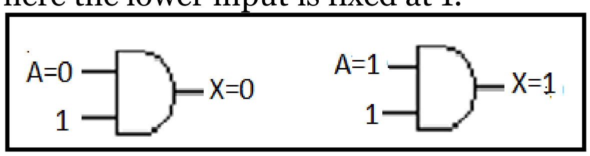
A variable ANDed with 0 is always equal to 0. This rule is illustrated in the following Figure , where the lower input is fixed at 0.



X=A . 0=0

Rule 4 : A .1 = A (Identity Law)

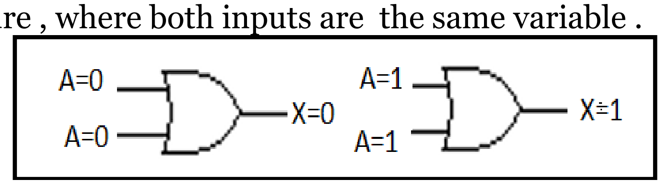
A variable ANDed with 1 is always equal to the variable . This rule is illustrated in the following Figure , where the lower input is fixed at 1.



X=A . 1=A

Rule 5 : A + A= A (Idempotent Law)

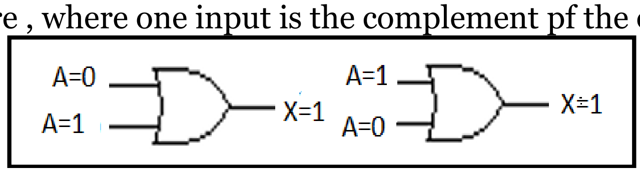
A variable ORed with itself is always equal to the variable . This rule is illustrated in the following Figure , where both inputs are the same variable .



X=A +A=A

Rule 6 : A + $\overline{A}$ = 1

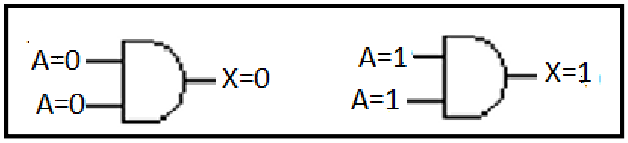
A variable ORed with its complement is always equal to 1. This rule is illustrated in the following Figure , where one input is the complement pf the other.



X=A + $\overline{A}$ =1

Rule 7: A .A = A (Idempotent Law)

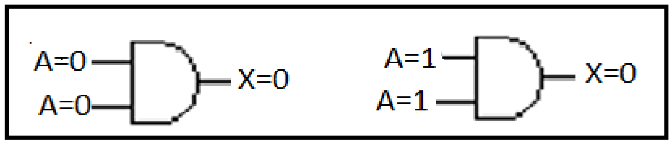
A variable Anded with itself is always equal to the variable . This rule is illustrated in the following Figure , where both inputs are the same variable .



X=A . A= A

Rule 8: A . $\overline{A}$ = 0 (Complement Law)

A variable ANDed with its complement is always equal to 0. This rule is illustrated in the following Figure.



X=A . $\overline{A}$ =0

Rule 9 : $\overline{\overline{A}}$ = A (Complement Law)

The double complement of a variable is always equal to the variable. This rule is illustrated in the following Figure using inverters .

$$\overline{\overline{A}} = A$$

Rule 10: A + AB = A

This rule can be proved by applying the distributive law , rule 2 and rule 4 as follows:

A + AB = A .1 + AB= A(1 + B)	factoring (distributive law)
=A . 1	Rule2 :(1 +b)=1
=A	Rule 4: A . 1 =A

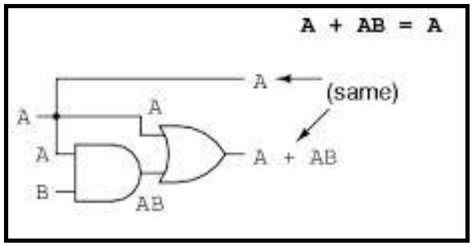
Note : the proof is shown in table bellow , which shows the troth table and the resulting

logic circuit simplification .

1- troth table

A	B	AB	A+AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

2- logic circuit



Rule 11: A + $\overline{A}$B= A + B

This rule can be proved as follows :

A + $\overline{A}$B=(A +AB) + $\overline{A}$**B**	Rule 10: A= A+ AB
= (AA +AB) + $\overline{A}$**B**	Rule 7 : A=AA
=AA +AB +A$\overline{A}$ + $\overline{A}$**B**	Rule 8:adding A$\overline{A}$**=0**
=(A + $\overline{A}$)(A + B)	factoring
= 1 . (A + B)	Rule 6: A + $\overline{A}$ =1
= A + B	Rule4 :drop the 1

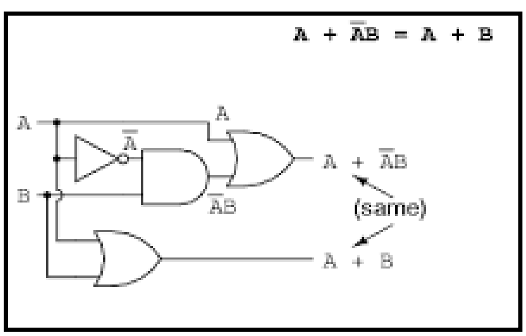
Note : the proof is shown in table bellow , which shows the troth table and the resulting

logic circuit simplification

1- troth table

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

2- logic circuit



Rule 12 : (A + B)(A + C) = A+BC

This rule can be proved as follows :

(A + B)(A+ C) = AA +AC +AB +BC

=A +AC +AB +BC

=A(1+C) +AB +BC

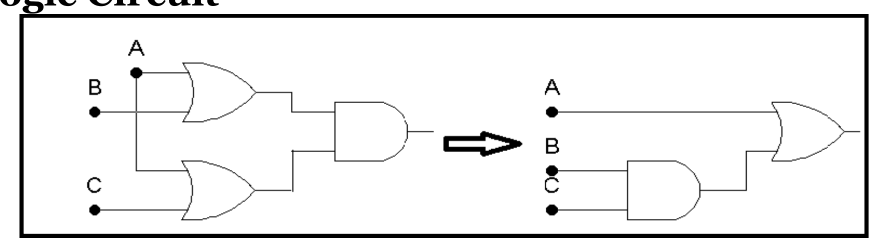
$= A \cdot 1 + AB + BC$
$= A(1 + B) + BC$
$= A + BC$

Note : the proof is shown in table bellow , which shows the troth table and the resulting
logic circuit simplification

1- Troth Table

A	B	C	A+B	A+C	(A+B) (A +C)	BC	A+BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

1- Logic Circuit



+Important Boolean Theorems

Following are few important Boolean functions and theorems.

♦ Boolean Expression/Function

Boolean algebra deals with binary variables and logic operation. A **Boolean Function** is described by an algebraic expression called **Boolean expression** which consists of binary variables, the constants 0 and 1 and the logic operation symbols. Consider the following example

F (A, B, C, D)	=	$A + \overline{BC} + ADC$	Equation No. 1
Boolean Function		Boolean Expression	

Here the left side of the equation represents the output Y. So we can state equation no. 1

$$Y = A + \overline{BC} + ADC$$

♦Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result. It is possible to convert the switching equation into a truth table. For example consider the following switching equation.

$$F(A, B, C) = A + BC$$

The output will be high (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is shown by Table (a). The number of rows in the truth table is 2n where n is the number of input variables (n=3 for the given equation). Hence there are 23 = 8 possible input combination of inputs.

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

♦De Morgan's Theorems

The two theorems suggested by De-Morgan which are extremely useful in Boolean Algebra are as following.

+Theorem 1

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

NAND = Bubbled OR

- The left hand side (LHS) of this theorem represents a **NAND** gate with input A and B where the right hand side (RHS) of the theorem represents an **OR** gate with inverted inputs.
- This **OR** gate is called as **Bubbled OR**.

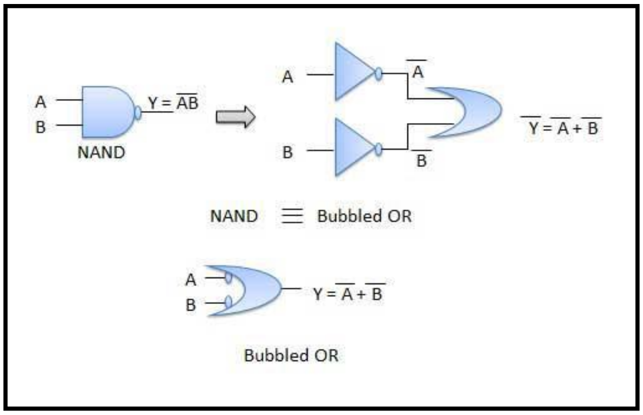


Table showing verification of the De-Morgan's first theorem

A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

+Theorem 2

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

NOR = Bubbled AND

- The LHS of this theorem represented a **NOR** gate with input A and B whereas the RHS represented an **AND** gate with inverted inputs.
- This **AND** gate is called as **Bubbled AND**.

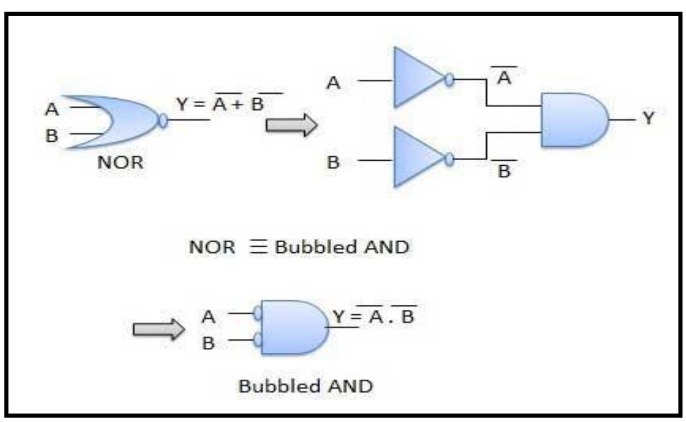


Table showing verification of the De-Morgan's second theorem

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Example : Apply DemMorgan's theorems to the following expression :

$$1-\overline{XYZ}=\overline{X}+\overline{Y}+\overline{Z}$$
$$2-\overline{X+Y+Z}=\overline{X}\ \overline{Y}\,\overline{Z}$$

That mean :

$$1-\overline{(A\,.\,B.\,Z.\ \ldots\ldots\ldots)}=\overline{X}+\overline{Y}+\overline{Z}+\ldots\ldots\ldots$$
$$2-\overline{(X+Y+Z+\ldots\ldots\ldots)}=\overline{X}\ .\,\overline{Y}\,.\overline{Z}.\ \ldots\ldots\ldots$$

Simplification Using Boolean Algebra

Many times in the application of Boolean algebra , you have to reduce a particular expression to its simplest form or change its form to a more convenient one to implement the expression most efficiently .

the approach taken un this section is to use the basic laws , and theorems of Boolean algebra to manipulate and simplify an expression .

This method depends on a thorough knowledge of Boolean algebra and considerable practice in its application , not to mention a little ingenuity and cleverness.