UNIT – II

Objectives:
To get acquainted with the concepts of logic gates, Boolean algebra and minimization of logic functions.

Syllabus: NOT, AND, OR, Universal Gates, Ex-Or and Ex-Nor Gates, Boolean Theorems, Complement and Dual, SOP, POS, two level realization of logic functions using universal gates, minimizations of logic functions (POS and SOP) using Boolean theorems, K-map (upto four variables), don't care conditions.

Outcomes:
At the end of the unit, Students will be able to

- Realize the logic functions using basic logic gates and universal logic gates.
- Express a logic function in SOP and POS forms.
- Simplify the logic expressions using Boolean laws and Boolean theorems.
- Simplify the logic expressions using K-map.

Learning Material

Logic Gate:

- Logic gates are the basic building blocks of a computer.
- Gates are basic circuits that have at least one (and usually more) input and exactly one output. Input and output values are the logical values TRUE and FALSE or ON or OFF.
- A useful way of describing the relationship between the inputs of logic gates and their output is the truth table.
- A table which lists all the possible combinations of input variables and the corresponding outputs is called a truth table.
- There are three basic kinds of gates-AND, OR and NOT (or inverter).

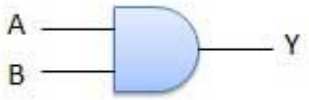
1. AND Gate

- A circuit which performs an AND operation is shown in figure. It has n inputs (n >= 2) and one output.

- AND gate output is 1 if and only if all its inputs are 1,hence And gate is also called an ***all*** or ***nothing*** gate.
- The symbol for the AND operation is **'.'** or we use no symbol at all.

Y = A AND B AND C ..... N
Y = A.B.C ..... N
Y = ABC ..... N

Logic symbol



Truth Table

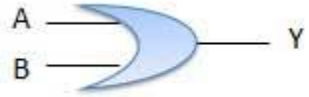
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

2. OR Gate

- A circuit which performs an OR operation is shown in figure. It has n inputs (n >= 2) and one output.
- Output is 1 even if one of its inputs is in logic 1 state.
- Output is 0 only when all the inputs are 0. Hence an OR gate is also called as an ***any*** or ***all*** gate.
- The symbol for OR operation is **'+'**.

Y = A OR B OR C ..... N
Y = A + B + C ..... N

Logic symbol



Truth Table

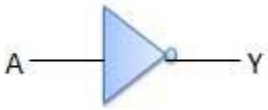
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

3. NOT Gate

- NOT gate is also known as **Inverter**. It has one input A and one output Y.
- Its output is always the complement of its input.
- If the input is 1 then output is 0 and vice versa.

Y = NOT A
Y = $\overline{A}$

Logic symbol



Truth Table

Inputs	Output
A	B
0	1
1	0

UNIVERSAL GATES:

- NAND and NOR are universal logic gates.
- Any logic circuit can be realized using universal logic gates.

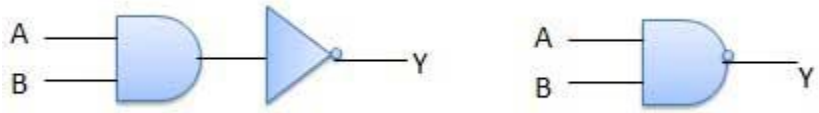
- Both NAND and NOR can perform all 3 logic functions(AND,OR&NOT).

NAND Gate

- A NOT-AND operation is known as NAND operation. It has n inputs (n >= 2) and one output.
- Output is 0 only when all the inputs are 1.
- NAND is a combination of AND gate and NOT gate.
- The expression for the output of NAND gate with two inputs is $(AB)^1$ or $\overline{AB}$.

Y = A NOT AND B NOT AND C ..... N
Y = A NAND B NAND C ..... N

Logic symbol



Truth Table

Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

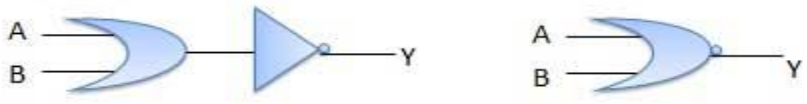
NOR Gate

- A NOT-OR operation is known as NOR operation. It has n inputs (n >= 2) and one output.
- Output is 1 only when all the inputs are 0.

- NOR is a combination of OR gate and NOT gate.
- The expression for the output of NAND gate with two inputs is $(A+B)^1$ or $\overline{A + B}$.

Y = A NOT OR B NOT OR C ..... N
Y = A NOR B NOR C ..... N

Logic symbol



Truth Table

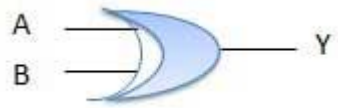
Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

EXCLUSIVE- OR (XOR) Gate:

- X-OR or Ex-OR gate is a special type of gate.
- It is used in the implementation of adders and subtractors.
- The exclusive-OR gate is abbreviated as EX-OR gate or X-OR gate.
- When the inputs are not equal, this produces output 1.
- It has n inputs (n >= 2) and one output.

Y = A XOR B XOR C ..... N
Y = $A \oplus B \oplus C$ ..... N
Y = $\overline{A}B + A\overline{B}$

Logic diagram



Truth Table

Inputs		Output
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

THE EXCLUSIVE-NOR (XNOR) Gate:

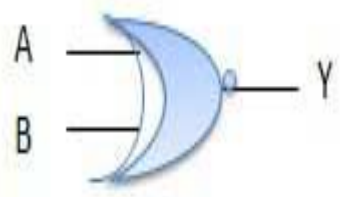
- XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor.
- The exclusive-NOR gate is abbreviated as EX-NOR gate or X-NOR gate.
- It has n input (n >= 2) and one output.
- When the inputs are not equal, this produces output 1.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$

$$Y = A \odot B \odot C \ldots\ldots N$$

$$Y = \bar{A}B + A\bar{B}$$

Logic diagram



Truth Table

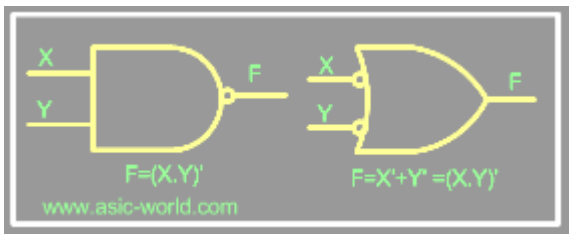
Inputs		Output
A	B	A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

Universal Gates

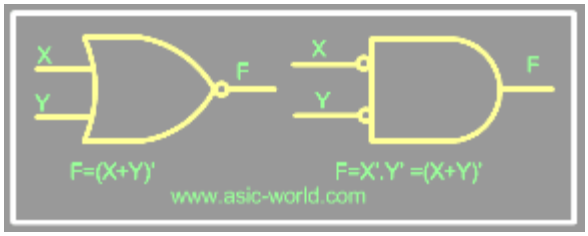
- Universal gates are the ones which can be used for implementing any gate like AND, OR and NOT, or any combination of these basic gates;
- NAND and NOR gates are universal gates. But there are some rules that need to be followed when implementing NAND or NOR based gates.

To facilitate the conversion to NAND and NOR logic, we have two new graphic symbols for these gates.

NAND Gate



NOR Gate



Realization of logic function using NAND gates

- Any logic function can be implemented using NAND

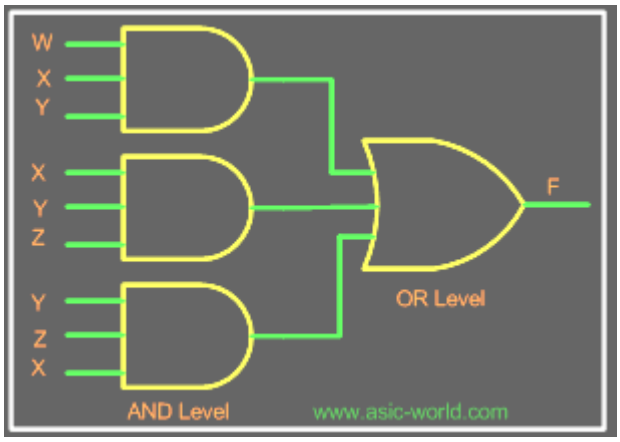
gates. To achieve this,

- First the logic function has to be written in Sum of Product (SOP) form.
- Once logic function is converted to SOP, then is very easy to implement using NAND gate.
- In other words any logic circuit with AND gates in first level and OR gates in second level can be converted into a NAND-NAND gate circuit.

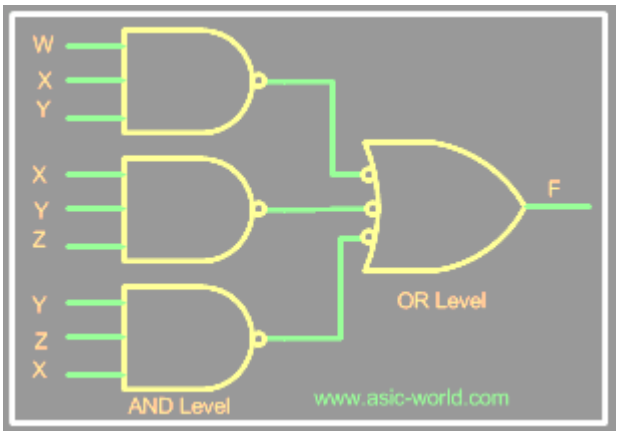
Consider the following SOP expression

F = W.X.Y + X.Y.Z + Y.Z.W

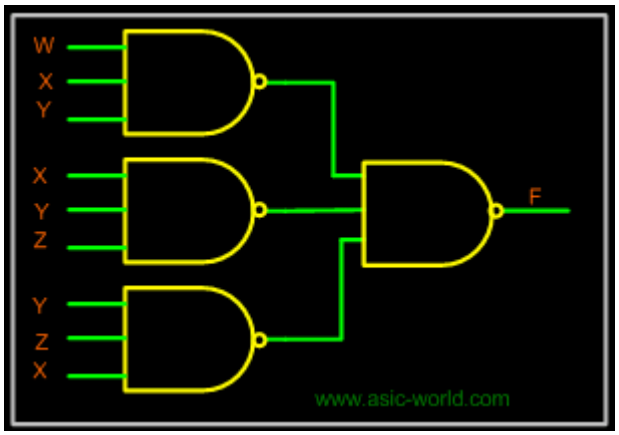
The above expression can be implemented with three AND gates in first stage and one OR gate in second stage as shown in figure.



If bubbles are introduced at AND gates output and OR gates inputs (the same for NOR gates), the above circuit becomes as shown in figure.



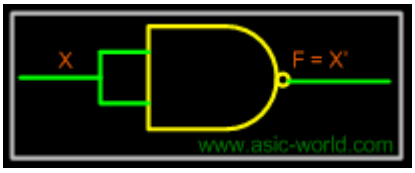
Now replace OR gate with input bubble with the NAND gate. Now we have circuit which is fully implemented with just NAND gates.



Realization of logic gates using NAND gates

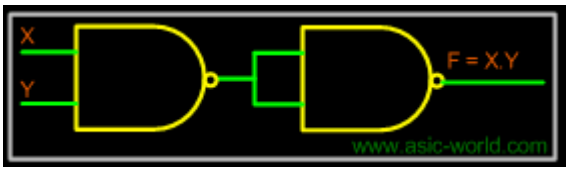
Implementing an inverter using NAND gate

Input	Output	Rule
(X.X)'	= X'	Idempotent



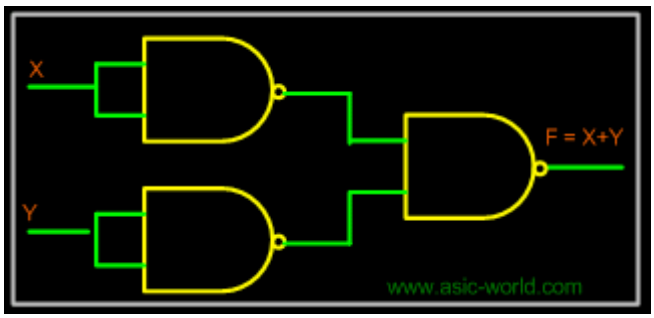
Implementing AND using NAND gates

Input	Output	Rule
((XY)'(XY)')'	= ((XY)')'	Idempotent
	= (XY)	Involution



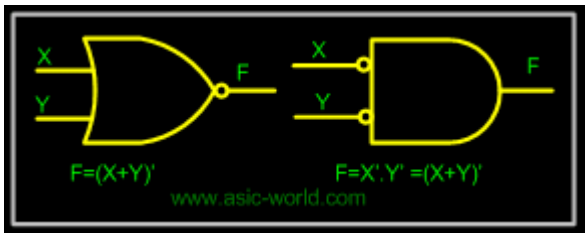
Implementing OR using NAND gates

Input	Output	Rule
((XX)'(YY)')'	= (X'Y')'	Idempotent
	= X''+Y''	DeMorgan
	= X+Y	Involution



Implementing NOR using NAND gates

Input	Output	Rule
((XX)'(YY)')'	=(X'Y')'	Idempotent
	=X''+Y''	DeMorgan
	=X+Y	Involution
	=(X+Y)'	Idempotent



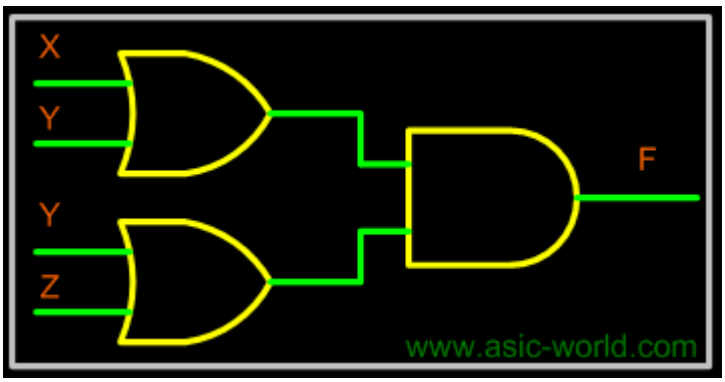
Realization of logic function using NOR gates

- Any logic function can be implemented by using NOR gates.
- To achieve this, first the logic function has to be written in Product of Sum (POS) form.
- Once it is converted to POS, then it's very easy to implement using NOR gate.
- In other words any logic circuit with OR gates in first level and AND gates in second level can be converted into a NOR-NOR gate circuit.

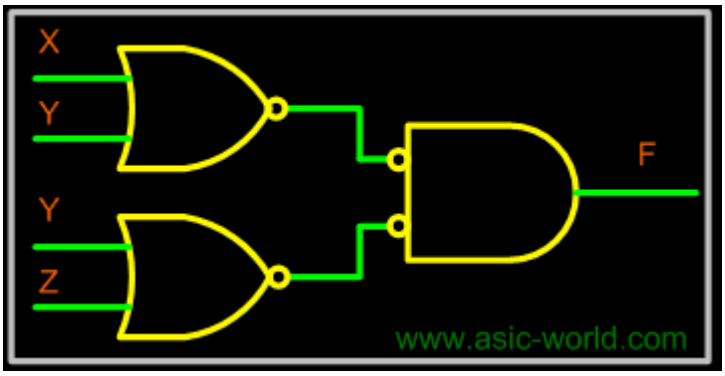
Consider the following POS expression

F = (X+Y) . (Y+Z)

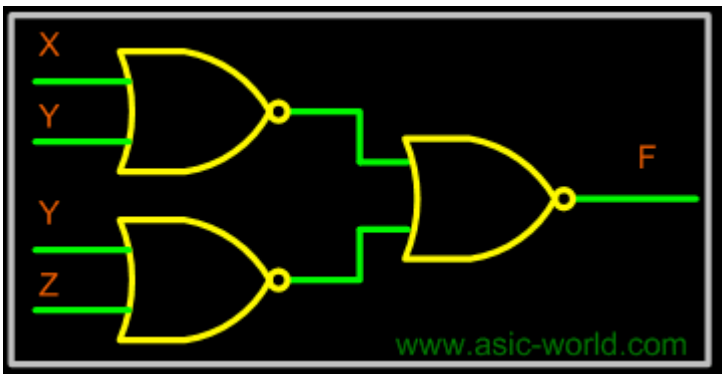
The above expression can be implemented with three OR gates in first stage and one AND gate in second stage as shown in figure.



If bubble are introduced at the output of the OR gates and the inputs of AND gate, the above circuit becomes as shown in figure.



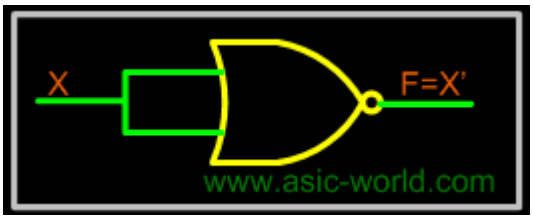
Now replace AND gate with input bubble with the NOR gate. Now we have circuit which is fully implemented with just NOR gates.



Realization of logic gates using NOR gates

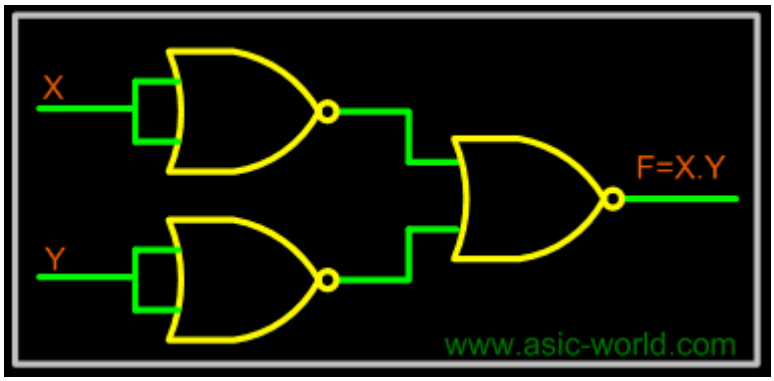
Implementing an inverter using NOR gate

Input	Output	Rule
(X+X)'	= X'	Idempotent



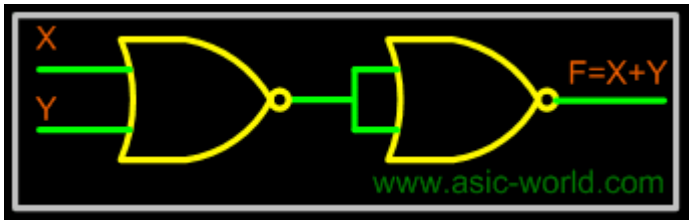
Implementing AND using NOR gates

Input	Output	Rule
((X+X)'+(Y+Y)')'	=(X'+Y')'	Idempotent
	= X''.Y''	DeMorgan
	= (X.Y)	Involution



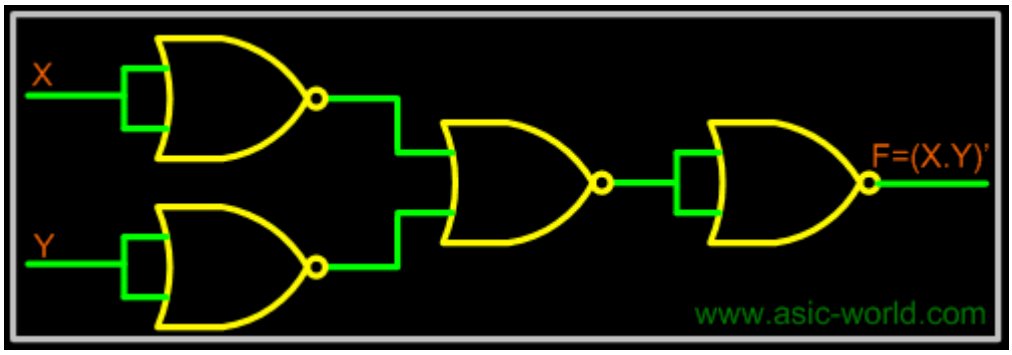
Implementing OR using NOR gates

Input	Output	Rule
((X+Y)'+(X+Y)')'	= ((X+Y)')'	Idempotent
	= X+Y	Involution



Implementing NAND using NOR gates

Input	Output	Rule
((X+Y)'+(X+Y)')'	= ((X+Y)')'	Idempotent
	= X+Y	Involution
	= (X+Y)'	Idempotent



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.
- Any complex logic statement can be expressed by a Boolean function.

Rules in Boolean Algebra

1. $A + 0 = A$	7. $A \cdot A = A$
2. $A + 1 = 1$	8. $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	9. $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	10. $A + AB = A$
5. $A + A = A$	11. $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	12. $(A + B)(A + C) = A + BC$

A, B, or *C* can represent a single variable or a combination of variables.

Boolean Laws

There are six types of Boolean Laws.

- Commutative law
  - Any binary operation which satisfies the following expression is referred to as commutative operation.

$$(i)\ A.B = B.A \qquad (ii)\ A + B = B + A$$

- Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

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

$$(i)\ (A.B).C = A.(B.C) \qquad (ii)\ (A + B) + C = A + (B + C)$$

- Distributive law
  - Distributive law states the following condition.

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

- AND law
  - These laws use the AND operation. Therefore they are called as **AND** laws.

$$(i)\ A.0 = 0 \qquad (ii)\ A.1 = A$$
$$(iii)\ A.A = A \qquad (iv)\ A.\overline{A} = 0$$

- OR law
  - These laws use the OR operation. Therefore they are called as **OR** laws.

$$(i)\ A + 0 = A \qquad (ii)\ A + 1 = 1$$
$$(iii)\ A + A = A \qquad (iv)\ A + \overline{A} = 1$$

- INVERSION law
  - This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

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

Basic Laws of Boolean Algebra

•Identity laws: A + 0 = A

A * 1 = A

- **Inverse laws: A + A = 1**

  A * A = 0

- **Zero and one laws: A + 1 = 1**

  A * 0 = 0

- **Commutative laws: A + B = B+A**

  A * B = B * A

- **Associative laws: A + (B + C) = (A + B) + C**

  A * (B * C) = (A * B) * C

- **Distributive laws: A * (B + C) = (A * B) + (A * C)**

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

- **DeMorgan's laws: (A + B) = A * B**

  (A * B) = A + B

Two Valued Boolean Algebra:

A two valued Boolean Algebra is defined on a set of two elements, B = { 0, 1 }, with rules for the two binary operators **+** and **.**

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

These rules are exactly same as the AND, OR, & NOT operations, respectively. Now show that the Huntington postulates are valid for the set B = { 0, 1 } and the two binary operators defined above.

1. Closure: The result of each operation is 1 or 0 and 1, 0 B.

2. From the table an **identity element** with respect to + is 0, since

0 + 0 = 0

0 + 1 = 1 + 0 = 1

An identity element with respect to . is 1,

since 1.1 = 1

1.0 = 0.1 = 0

3. Commutative law

0 + 1 = 1 + 0 = 1 (for +)

0 . 1 = 1 . 0 = 0 (for .)

4. Distributive law

a. . is distributive over + : x (y + z) = xy + xz

b. + is distributive over . : x + (y. z) = (x + y) . (x + z)

x	y	z	y+z	x.(y+z)	x.y	x.z	xy + xz
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

5. From the **complement** table,

a. x + x' = 1 since 0 + 0' = 0 + 1 = 1

1 + 1' = 1 + 0 = 1

b. x . x' = 0 since 0 . 0' = 0 . 1 = 0

1 . 1 ' = 1. 0 = 0

6. Two valued boolean algebra has two distinct elements 1 and 0.

Theorems and Properties of Boolean Algebra:

Theorem 1:

a. x + x = x

proof:

x + x = (x + x) . 1 by postulate (2b)

= (x + x) (x + x')
= (x + xx')
= x + 0
= x

b. x . x = x

proof:

x . x = x. x + 0
= x . x + x . x'
= x (x + x')
= x . 1
= x

Theorem 2:

a. x + 1 = 1

proof:

x + 1 = 1 . (x + 1)
= (x + x')(x + 1)
= x + x' . 1
= x + x'
= 1

b. x.0 = 0 by duality

Theorem 3:

(x')' = x

Theorem 4:

a. x + (y + z) = (x + y) + z associative
b. x (yz) = (xy) z associative

Theorem 5: (De Morgan)

a. (x + y)' = x' y'

b. (xy)' = x' + y'

proof:

x	y	x+y	(x+y)'	x'	y'	x'y'
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Theorem 6: (Absorption)

a. x + xy = x

proof:

x + xy = x.1 + xy by postulate 2 (b)
= x (1 + y) by postulate 4 (a)
= x (y + 1) by postulate 3 (a)
= x . 1 by theorem 2 (a)
= x by postulate 2(b)

b. x (x + y) = x
proof:

x	y	xy	x+xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Operator precedence for evaluating the boolean expression is:

1. First the expressions inside parentheses must be evaluated.
2. Complement (NOT)
3. AND
4. OR

Boolean Functions

A Boolean function is a special kind of mathematical function f: $X^n \rightarrow X$ of degree n, where X = {0, 1} is a Boolean domain and n is a non-negative integer. It describes the way how to derive Boolean output from Boolean inputs.

Example – Let, F(A, B) = A^1B^1. This is a function of degree 2 from the set of ordered pairs of Boolean variables to the set {0, 1} where F(0, 0) = 1, F(0, 1) = 0, F(1, 0) = 0 and F(1, 1) = 0

- Simplify the boolean functions to a minimum number of literals.

1. x + x'y = (x + x') (x + y) = 1.(x + y) = x + y

2. x (x' + y) = xx' + xy = 0 + xy = xy

3. x'y'z + x'yz + xy' = x'z (y' + y) + xy' = x'z + xy'

4. xy + x'z + yz = xy + x'z + yz (x + x')
= xy + x'z + xyz + x'yz
= xy (1 + z) + x'z (1 + y)
= xy + x'z

5. (x + y) (x'z) (y + z) = (x + y) (x' + z)

Complement of a Function:

(A + B + C)'	=	(A + X)'	let B + C = X
	=	A' X '	De Morgan thm 5a
	=	A' (B + C)'	
	=	A' . (B' C')	De Morgan thm. 5a
	=	A ' B ' C '	

Example:

Find the complement of the functions.

F1 = x' yz' + x'y'z and F2 = x (y'z' + yz)

F1' = (x' y z' + x' y' z)'
= (x' y z')' (x' y' z)'
= (x + y' + z) (x + y + z')

F2' = [x (y' z' + yz)]'
= x' + (y'z' + yz)'
= x' + (y'z')' . (yz)'
= x' + (y + z) (y' + z')

Example:

Find the complement of the functions F1 and F2 by taking their duals and complementing each literal.

F1 = x'yz' + x' y' z

The dual of F1 is : (x' + y + z') (x' + y' + z)
Complement each literal : F1' = (x + y' + z) (x + y + z')

F2 = x (y' z' + yz)

The dual of F2 is x + (y' + z') (y + z)
Complement each literal: F2' = x' + (y + z) (y' + z')

Boolean Expressions

- A Boolean expression always produces a Boolean value.
- A Boolean expression is composed of a combination of the Boolean constants (True or False), Boolean variables and logical connectives.
- Each Boolean expression represents a Boolean function.

Example − AB'C is a Boolean expression

CANONICAL AND STANDARD FORMS:

a) CANONICAL FORMS:

For a Boolean expression there are two kinds of canonical forms −

- The sum of minterms (SOM) form
- The product of maxterms (POM) form

The Sum of Minterms (SOM) or Sum of Products (SOP) form

- A minterm is a product of all variables taken either in their direct or complemented form.
- Any Boolean function can be expressed as a sum of its 1-minterms and the inverse of the function can be expressed as a sum of its 0-minterms. Hence,

F (list of variables) = ∑ (list of 1-minterm indices)and

F' (list of variables) = ∑ (list of 0-minterm indices)

A	B	C	Minterm
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

Example

Let, F(x, y, z) = x' y' z' + x y' z + x y z' + x y z

Or, F(x, y, z) = $m_0 + m_5 + m_6 + m_7$

Hence,

F(x, y, z) = $\sum (0, 5, 6, 7)$

Now we will find the complement of F(x, y, z)

F' (x, y, z) = x' y z + x' y' z + x' y z' + x y' z'

Or, F'(x, y, z) = $m_3 + m_1 + m_2 + m_4$

Hence,

F'(x, y, z) = ∑ (3, 1, 2, 4) = ∑ (1, 2, 3, 4)

Example:

Express F = A + B'C as a sum of minterms.

F = A + B'C
= A (B + B') + (A + A')B'C
= AB + AB' + AB'C + A'B'C

Note the terms AB and AB' are still lack the variable C, we will expand them by (C + C ')

= AB (C + C') + AB' (C + C ') + AB'C + A'B'C
= ABC + ABC' + AB'C + AB'C ' + AB'C + A'B'C
= ABC + ABC' + AB'C + AB'C ' + A'B'C
= $m_7 + m_6 + m_5 + m_4 + m_1$
= ∑ (1, 4, 5, 6, 7)

The Product of Maxterms (POM) or Product of Sums (POS) form

A maxterm is addition of all variables taken either in their direct or complemented form. Any Boolean function can be expressed as a product of its 0-maxterms and the inverse of the function can be expressed as a product of its 1-maxterms. Hence,

F (list of variables) = π (list of 0-maxterm indices) and
F'(list of variables) = π (list of 1-maxterm indices).

A	B	C	Maxterm
0	0	0	M_0
0	0	1	M_1
0	1	0	M_2
0	1	1	M_3
1	0	0	M_4
1	0	1	M_5
1	1	0	M_6
1	1	1	M_7

Example

Let, F(x, y, z) = (x+y+z) • (x+y+z') • (x+y'+z) • (x'+y+z)

Or, $F(x, y, z) = M_0 \cdot M_1 \cdot M_2 \cdot M_4$

Hence,

$F(x, y, z) = \pi(0, 1, 2, 4)$

$F'(x, y, z) = (x+y'+z') \cdot (x'+y+z') \cdot (x'+y'+z) \cdot (x'+y'+z')$

Or, $F(x, y, z) = M_3 \cdot M_5 \cdot M_6 \cdot M_7$

Hence, $F'(x, y, z) = \pi(3, 5, 6, 7)$

EXAMPLE:

Express F = xy + x'z in product of maxterm form.

F = (xy + x'z)
= (x' + xy) (z + xy)
= (x' + x) (x' + y) (z + x) (z + y)
= (1) (x' + y) (x + z) (y + z)
= (x' + y + zz') (x + z + yy') (y + z + xx')
= (x'+y+z) (x'+y+z') (x+z+y) (x+z+y') (y+z+x) (y+z+x')
= Reduce the terms which appear twice
= (x + y + z) (x + y' + z) (x' + y + z) (x' + y + z')
= 000 010 100 101

The convenient way to express this function is
$F(x,y,z) = \pi (0, 2, 4, 5)$
Symbol **π** denotes **ANDing of maxterms**; **numbers** are the indices of the **maxterm** of the function.

CONVERSION BETWEEN CANONICAL FORMS

Consider the function $F (A, B, C) = \sum (1, 4, 5, 6, 7)$

This function has a complement that can be expressed as

$$F'(A,B, C) = \sum (0, 2, 3) = m_0 + m_2 + m_3$$

If we take the complement of F ' by the De Morgan's theorem, we obtain F in a different form:

$F (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0.M_2.M_3 = \pi (0, 2, 3)$

$$m'_j = M_j$$

To convert from one canonical form to another interchange the symbols and list those numbers missing from the original form.

EXAMPLE:
F (x, y, z) = (0, 2, 4, 5) is expressed in the product of MAXTERM form.
It's conversion to sum of MINTERMS is:
F (x, y, z) =∑(1, 3, 6, 7)

b) STANDARD FORMS:

Another way to express Boolean function is in standard forms. There are two types of standard forms:

. The sum of products
. The product of sums

The SUM of PRODUCTs is a boolean expression containing AND terms, called PRODUCT terms, of one or more literals each.
The SUM denotes the OR ing of these terms.

Example:

A function expressed in sum of products is

$F_1 = y' + xy + x'yz'$

The expression has three product terms of one, two, and three literals each, respectively.
The PRODUCT of SUMs is a boolean expression containing OR terms, called SUM terms. Each term may have any number of literals. The PRODUCT denotes the AND ing of these terms.

Karnaugh Maps

- The Karnaugh map provides a simple and straight-forward method of minimizing Boolean expressions.
- With the Karnaugh map Boolean expressions having up to four and even six variables can be simplified.
  A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables.

- The map is a simple table containing 1s and 0s that can express a truth table or complex Boolean expression describing the operation of a digital circuit.
- K-Maps are a convenient way to simplify Boolean functions.
- They can be used for simplifying functions having up to 4 or
- 5 variables.They are a visual (graphical) representation of a truth table.
- They are used to simplify S.O.P and P.O.S forms of Boolean
- function.
- One ***map cell*** corresponds to a row in the truth table.
- Also, one map cell corresponds to a minterm or a maxterm in the Boolean expression.

Rules for K-Maps

1. Each cell with a 1 must be included in at least one group.
2. Try to form the largest possible groups.
3. Try to end up with as few groups as possible.
4. Groups may be in sizes that are powers of 2: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, ...
5. Groups may be square or rectangular only (including wraparound at the grid edges).
   No diagonals or zig-zags can be used to form a group.

6. The larger a group is, the more redundant inputs there are:
   i. A group of 1 has no redundant inputs.

   ii. A group of 2 has 1 redundant input.

   iii. A group of 4 has 2 redundant inputs.

   iv. A group of 8 has 3 redundant inputs.

   v. A group of 16 has 4 redundant inputs.

Grouping the 1s (rules)

1. A group must contain either 1,2,4,8,or 16 cells (depending on number of variables in the expression)

2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.

3. Always include the largest possible number of 1's in a group in accordance with rule 1.

4. Each 1 on the map must be included in at least one group.

5. The 1's already in a group can be included in another group as long as the overlapping groups include at least one noncommon 1's.

- The following rules are applied to find the minimum product terms and the minimum SOP expression:

    1. Group the cells that have 1's. Each group of cell containing 1's creates one product term composed of all variables that occur in only one form (either complemented or complemented) within the group.

        Variables that occur both complemented and uncomplemented within the group are eliminated → called *contradictory variables*.

Determining the Minimum SOP Expression from the Map

1. Determine the minimum product term for each group.

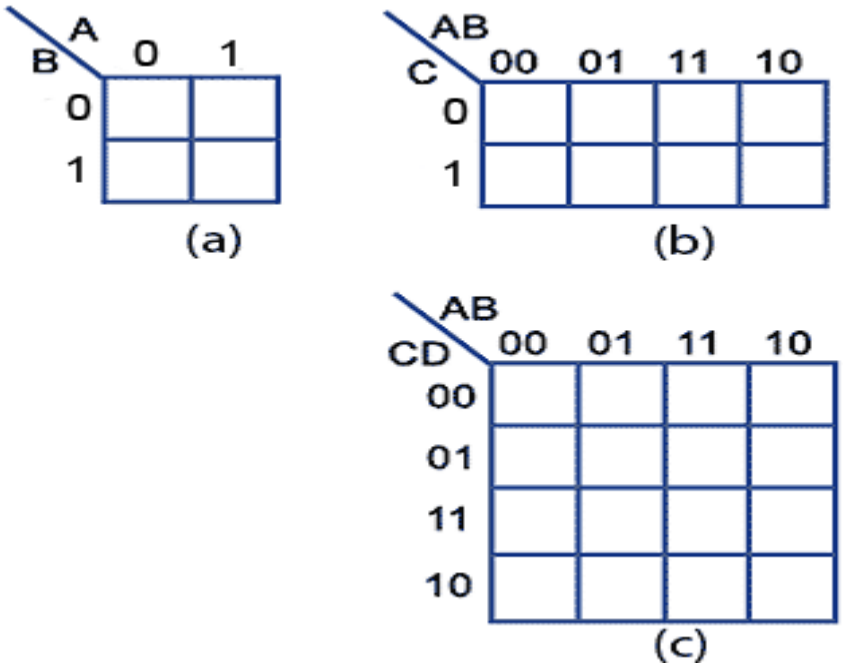
    - For a 3-variable map:

        1. A 1-cell group yields a 3-variable product term
        2. A 2-cell group (pair) yields a 2-variable product term
        3. A 4-cell group (quad) yields a 1-variable product term
        4. An 8-cell group (octet) yields a value of 1 for the expression.

    - For a 4-variable map:

        1. A 1-cell group yields a 4-variable product term
        2. A 2-cell group yields a 3-variable product term
        3. A 4-cell group yields a 2-variable product term
        4. An 8-cell group yields a 1-variable product term
        5. A 16-cell group yields a value of 1 for the expression.

Constructing Karnaugh Maps



The shape and size of the map is dependent on the number of binary inputs in the circuit to be analysed. The map needs one cell for each possible binary word applied to the inputs.

Therefore:

2 input circuits with inputs A and B require maps with $2^2 = 4$ cells

3 input circuits with inputs A B and C require maps with $2^3 = 8$ cells

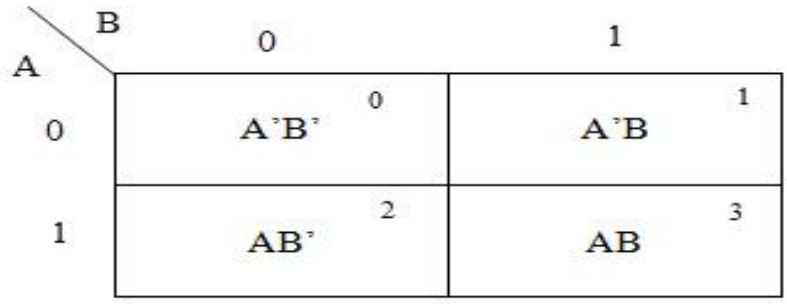
4 input circuits with inputs A B C and D require maps with $2^4 = 16$ cells.

Two Variable Map:

- The Possible Min Terms with 2 Variables (A And B) Are A.B, A.B', A'.B And A'.B'.
- The following table shows the positions of all the possible outputs of 2-variable Boolean function on a K-map.

A	B	Possible Outputs	Location on K-map
0	0	A'B'	0
0	1	A'B	1
1	0	AB'	2
1	1	AB	3

A general representation of a 2 variable K-map plot is shown below.



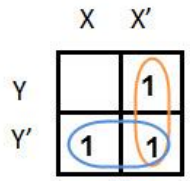
Example

Simplify the given 2-variable Boolean equation by using K-map.F = X Y' + X' Y + X'Y'

First, let's construct the truth table for the given equation,

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

We put 1 at the output terms given in equation.



- In this K-map, we can create 2 groups by following the rules for grouping,
- one is by combining (X', Y) and (X', Y') terms and the other is by combining (X, Y') and (X', Y') terms.
- Here the lower right cell is used in both groups. After grouping the variables, the next step is determining the minimized expression.

- By reducing each group, we obtain a conjunction of the minimized expression such as by taking out the common terms from two groups, i.e. X' and Y'.
- So the reduced equation will be X' +Y'.

3 variable K-maps

- For a 3-variable Boolean function, there is a possibility of 8 output min terms.
- The general representation of all the min terms using 3-variables is shown below.

A	B	C	Output Function	Location on K-map
0	0	0	A'B'C'	0
0	0	1	A'B'C	1
0	1	0	A'BC'	2
0	1	1	A'BC	3
1	0	0	AB'C'	4
1	0	1	AB'C	5
1	1	0	ABC'	6
1	1	1	ABC	7

A general representation of a 3 variable K-map plot is shown below.

A \ BC	00	01	11	10
0	A'B'C' 0	A'B'C 1	A'BC 3	A'BC' 2
1	AB'C' 4	AB'C 5	ABC 7	ABC' 6

Example

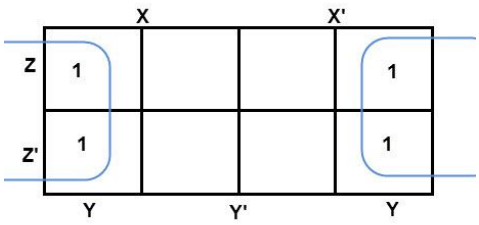
Simplify the given 3-variable Boolean equation by using k-map.

F = X' Y Z + X' Y' Z + X Y Z' + X' Y' Z' + X Y Z + X Y' Z'

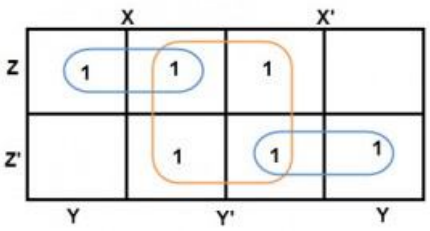
First, let's construct the truth table for the given equation,

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- We put 1 at the output terms given in equation. There are 8 cells (23) in the 3-variable k-map. It will look like (see below image).
- The largest group size will be 8 but we can also form the groups of size 4 and size 2, by possibility.
- In the 3 variable Karnaugh map, we consider the left most column of the k-map as the adjacent column of rightmost column. So the size 4 group is formed as shown below.



- And in both the terms, we have 'Y' in common.
- So the group of size 4 is reduced as the conjunction Y.
- To consume every cell which has 1 in it, we group the rest of cells to form size 2 group, as shown below.



- The 2 size group has no common variables, so they are written with their variables and its conjugates. So the reduced equation will be X Z' + Y' + X' Z. In this equation, no further minimization is possible.

4 variable K-maps

- There are 16 possible min terms in case of a 4-variable Boolean function. The general representation of minterms using 4 variables is shown below.

A	B	C	D	Output function	K-map location
0	0	0	0	A' B' C' D'	0
0	0	0	1	A' B' C' D	1
0	0	1	0	A' B' C D'	2
0	0	1	1	A' B' C D	3
0	1	0	0	A' B C' D'	4
0	1	0	1	A' B C' D	5
0	1	1	0	A' B C D'	6
0	1	1	1	A' B C D	7
1	0	0	0	A B' C' D'	8
1	0	0	1	A B' C' D	9
1	0	1	0	A B' C D'	10
1	0	1	1	A B' C D	11
1	1	0	0	A B C' D'	12
1	1	0	1	A B C' D	13
1	1	1	0	A B C D'	14
1	1	1	1	A B C D	15

- A typical 4-variable K-map plot is shown below. It can be observed that both the columns and rows of 10 and 11 are interchanged.

AB \ CD	00	01	11	10
00	0 A' B' C' D'	1 A' B' C' D	3 A' B' C D	2 A' B' C D'
01	4 A' B C' D'	5 A' B C' D	7 A' B C D	6 A' B C D'
11	12 A B C' D'	13 A B C' D	15 A B C D	14 A B C D'
10	8 A B' C' D'	9 A B' C' D	11 A B' C D	10 A B' C D'

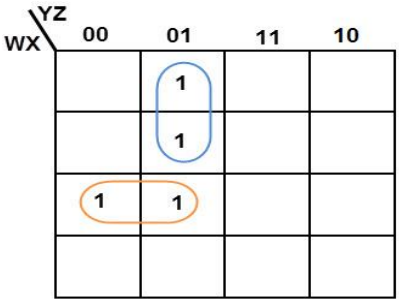
- The possible number of cells that can be grouped together are 1, 2, 4, 8 and 16.

Example

Simplify the given 4-variable Boolean equation by using k-map.

F (W, X, Y, Z) = (1, 5, 12, 13)

Sol: F (W, X, Y, Z) = (1, 5, 12, 13)

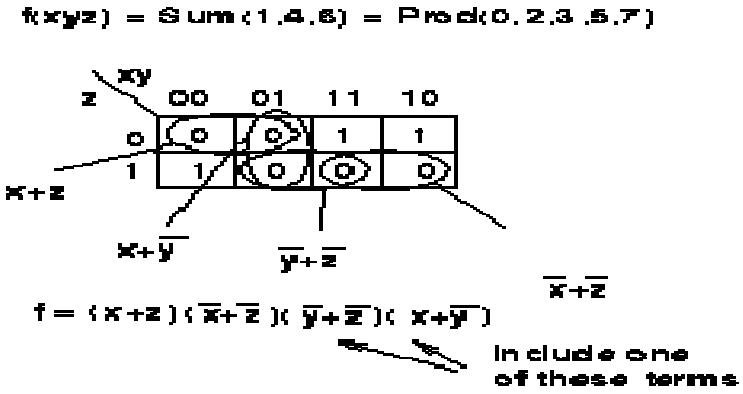


By preparing k-map, we can minimize the given Boolean equation as F = W Y' Z + W 'Y' Z

K-maps for Product-of-Sum Design

Product-of-sums design uses the same principles, but applied to the zeros of the function.

Example:



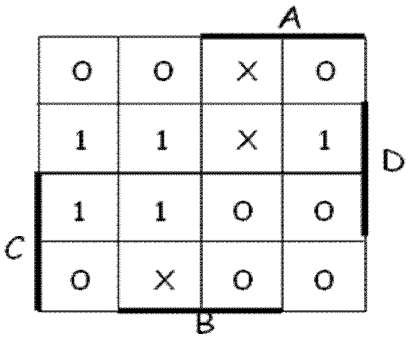
K-map with "Don't care" conditions

- The "Don't care" conditions are used to replace the empty cell to form a possible grouping of variables.
- They can be used as either 0 or 1, based on the adjacent variables in the group.
- The cells that contain "don't care" conditions are represented by 'X' symbol among the normal 0's and 1's.

EXAMPLE:

f(A,B,C,D) = S m(1,3,5,7,9) + d(6,12,13)

f = A'D + B'C'D without don't cares
f = A^1D+C^1D with don't cares


A
0 0 X 0
1 1 X 1
1 1 0 0
0 X 0 0
D
C
B

UNIT-II
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. The basic logic gate whose output is the complement of the input is the:

   A.OR gate [ ]

   B.AND gate

   C.INVERTER gate

   D. Comparator

2. Logically, the output of a NOR gate would have the same Boolean expression

   a. AND gate immediately followed by an INVERTER [ ]

   b. NAND gate immediately followed by an INVERTER

   c. OR gate immediately followed by an INVERTER

   d. NOR gate immediately followed by an INVERTER

3. The format used to present the logic output for the various combinations of logic inputs to a gate is called: [ ]

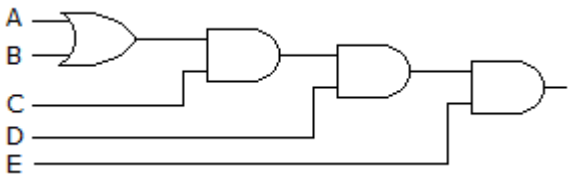
   a. truth table.

   b. input logic function.

   c. Boolean constant.

   d. Boolean variable.

4. Derive the Boolean expression for the logic circuit shown below: [ ]



A. $C(A + B)DE$

B. $\left[C(A + B)D + \overline{E}\right]$

C. $\left[\left[C(A + B)D\right]\overline{E}\right]$

D. $ABCDE$

5. Which Boolean law is described by the equation A•(B+C) = A•B+A•C ? [ ]

a) Commutative law**.** b) Associative law.

c) Distributive law**.** d) Complement law.

6. e*x=x*e=x is the [ ]

a. commutative property

b. inverse property

c. associative property

d. identity element

7. To perform product of max terms Boolean function must be brought into

a. AND terms [ ]

b. OR terms

c. NOT terms

d. NAND terms

8. Universal logic gates are: [ ]

a. NAND and NOR

b. OR and AND

c. NOT and OR

d. OR and XOR

9. NAND gate is a combination of: [ ]

a. AND and NOT gates

b. AND and OR gates

c. AND and XOR gates

d. OR and NOR gates

10. What does an EX-OR gate do? [ ]

a. Give a high output when odd number of inputs are high

b. Give a high output when even number of inputs are high

c. Give a low output when odd number of inputs are high

d. Give a low output when even number of inputs are high

11. Determine the values of A, B, C, and D that make the product term $\bar{A}B\bar{C}D$ equalto1. [ ]

A. A = 0, B = 1, C = 0, D = 1

B. A = 0, B = 0, C = 0, D = 1

C. A = 1, B = 1, C = 1, D = 1

D. A = 0, B = 0, C = 1, D = 0

12. The simplification of the Boolean expression (ABC) + $(ABC)^1$ is [ ]

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

13. How many gates would be required to implement the following Boolean expression after simplification? XY + X(X + Z) + Y(X + Z) [ ]

(A) 1 (B) 2 (C) 4 (D) 5

14. Applying DeMorgan's theorem to the expression $\overline{\overline{(X + Y)} + \bar{Z}}$, we get _____

A. $(X + Y)Z$ [ ]

B. $(\bar{X} + \bar{Y})Z$

C. $(X + Y)\bar{Z}$

D. $(\bar{X} + \bar{Y})\bar{Z}$

15. Use Boolean algebra to find the most simplified SOP expression for $F = ABD + CD + ACD + ABC + ABCD$. [ ]

A. $F = ABD + ABC + CD$

B. $F = CD + AD$

C. $F = BC + AB$

D. $F = AC + AD$

16. Convert the f(x,y,z)=∑(1,3,5) to the other canonical form [ ]

A) π(0,1,2,3,4,6,7) B) π(0,2,4,6,7) C) π(0,2,4,6,7) D) ∑(0,1,2,3,4,6,7)

17. Simplified expression of $Y=AB^1+AB+A^1B$ [ ]

A) A^1B B) A^1+B C) A+B D) AB

SECTION-B

SUBJECTIVE QUESTIONS

1. Simplify the Boolean expressions:

a. AB + A (B + C) + B(B + C).

b. [AB (C + BD) + A B]C

c. $A^1BC + ABC^1 + A\ B\ C + A^1BC^1$

2. Convert each of the following Boolean expressions to SOP and POS form:

a. $(u+xw)\ (x+u^1v)$

b. $x^1 +x(x+y^1)(y+z^1)$

3. Express the following functions as a sum of min terms and as a product of max terms:

$F(A,B,C)=B^1C+A^1C+BC$

4. Use Boolean algebra to simplify the following expression, then draw a logic circuit for the simplified expression: A (B + AB) + AC

5. Reduce the following Boolean expressions to the indicated number of literals

a. A'C'+ABC +AC' to THREE literals

b. $ABC^1D+A^1BD+ABCD$ to TWO literals

c. A'B(D'+CD)+B(A+A'CD) to ONE literals

6. Simplify the following Boolean function using K –map :

a. F(X,Y,Z)=∑(0,1,2,5,7)

b. F(A,B,C,D)= ∑(4,5,6,7,15)

7. Simplify the following using De Morgan's theorem [((AB)^1 C)^{1}D]$^{1.}$

8. Show that (X+Y'+XY) (X+Y') (X' Y)=0.

9. Realize the Boolean function using logic gates Y = CD+EF+G.

10. Implement the following POS function using NOR gates only

a. F = (X+Z) (Y'+Z) (X'+Y+Z)

11. Implement the following function

a. F=(XZ+Y^{1}Z+X^{1}YZ)^1 OR F^1=XZ+Y^{1}Z+X^{1}YZ using two level NAND circuit.

12. Simplify the following function and implement with two - level NOR gate F=WX'+Y'Z'+W'YZ'

13. Simplify the following Boolean function together with the don't care conditions and simplify in SOP form

a. F(A,B,C,D)= ∑ (0,6,8,13,14) d(A,B,C,D)= ∑ (2,4,10)

b. F(A,B,C,D)= ∑ (4,5,6,7,12,13,14) d(A,B,C,D)= ∑ (1,9,11,15)

14. Simplify the following Boolean function to product of sums form:

a. F(W,X,Y,Z)= ∑ (0,1,2,5,8,10,13)

b. F(W,X,Y,Z)= Π(1,3,5,7,12,13,14,15)

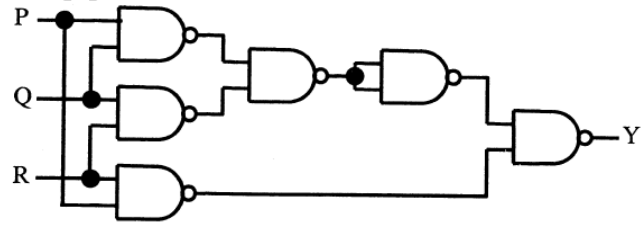
SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. Which one of the following expressions does NOT represent exclusive NOR of x and y? **[Gate 2013]**

a. xy+x′y′ b. x⊕y′ c. x′⊕y d. x′⊕y′

2. The output Y in the circuit below is always '1', when **[Gate 2011]**



a. Two or more of the inputs P,Q, R are '0'

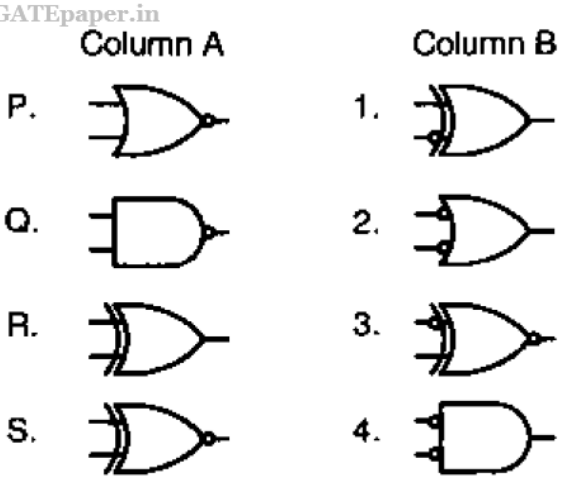
b. Two or more of the inputs P,Q, R are '1'

c. Any odd number of the inputs P,Q,R is '0'

d. Any odd number of the inputs P,Q,R is '1'

3. Match the logic gates in Column A with their equivalents in Column B.

[Gate 2010]



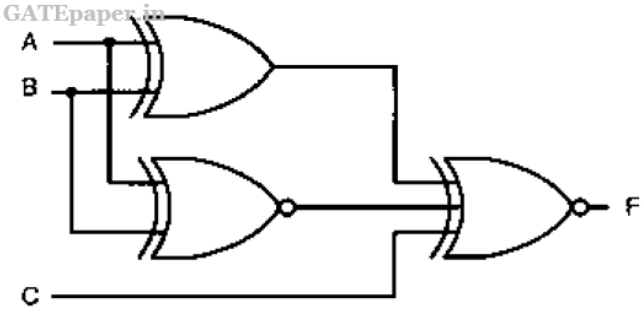
e. P – 2, Q – 4, R – 1, S – 3

f. P – 4, Q – 2, R – 1, S – 3

g. P – 2, Q – 4, R – 3, S – 1

h. P – 4, Q – 2, R – 3, S - 1

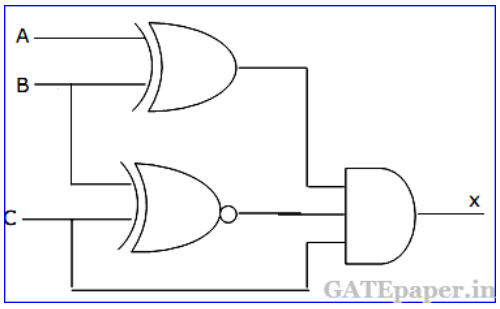
4. For the output F to be 1 in the logic circuit shown, the input combination should be **[Gate 2010]**



a. A = 1, B = 1, C = 0
b. A = 1, B = 0, C = 0
c. A = 0, B = 1, C = 0
d. A = 0, B = 0, C = 1

5. Evaluate the minimum number of gates required to implement the Boolean function (AB+C) if we have to use only 2-input NOR gates? **[Gate 2009]**

6. For the logic circuit shown in the figure, the required input combination (A,B,C) to make the output X = 1 is **[Gate 2000]**



a. 1, 0, 1
b. 0, 0, 1
c. 1, 1, 1
d. 0, 1, 1

7. Which function does NOT implement the Karnaugh map given below?

[Gate 2000]

wz → / xy↓	00	01	11	10
00	0	×	0	0
01	0	×	1	1
11	1	1	1	1
10	0	×	0	0

(a) $(w+x)y$ (b) $xy+yw$

(c) $(w+x)(\bar{w}+y)(\bar{x}+y)$ (d) None of the above

8. The minimum number of 2 input NAND gates required to implement the Boolean function Z = AB'C, assuming that A, B and C are available, is

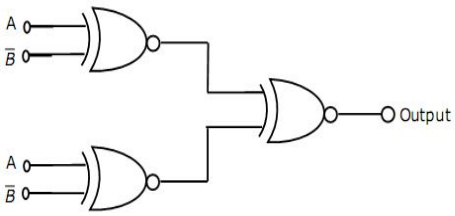
a. Two **[Gate 1998]**

b. Three

c. Five

d. Six

9. The output of the circuit shown in figure is equal to **[Gate 1995]**



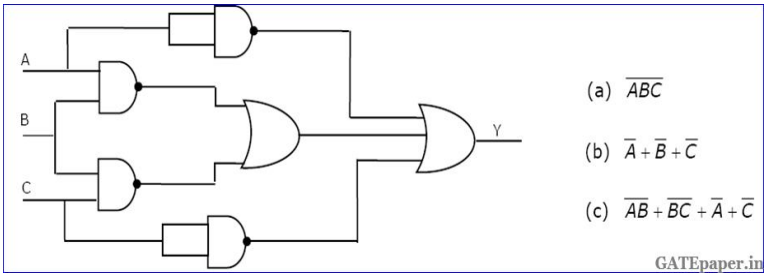
(a) 0

(b) 1

(c) $\bar{A}B + A\bar{B}$

(d) $\overline{(A*B)}*\overline{(A*B)}$

10. For the logic circuit shown in figure, the output Y is equal to **[Gate 1993]**

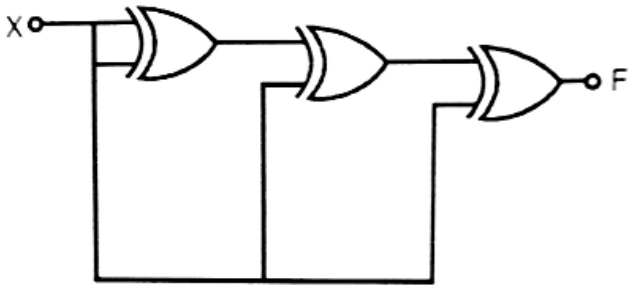


(a) $\overline{ABC}$

(b) $\bar{A}+\bar{B}+\bar{C}$

(c) $\overline{AB}+\overline{BC}+\bar{A}+\bar{C}$

11. For the circuit shown below, the output F is given by **[Gate 1988]**



(a) F = 1 (b) F = 0
(c) F = X (d) F = $\bar{X}$

12. Minimum number of 2 input NAND gates required to implement the function given below is **[Gate 1988]**

$$F = (\bar{X} + \bar{Y})(Z + W)$$

a. 3 b.4 c.5 d.6