<u>**Digital Electronics**</u>

Boolean Algebra & Logic Gates

Logic gates are electronic circuits which can be used to implement the most elementary logical expressions, which are also known as Boolean expressions. The logic gate is the most basic building block of combinational logic.

There are three basic logic gates, namely the OR gate, the AND gate and the NOT gate. Other logic gates that are derived from these basic gates are the NAND gate, the NOR gate, the EXCLUSIVE-OR gate and the EXCLUSIVE-NOR gate.

This chapter deals with logic gates and implementations using NAND and NOR gates followed by simplification of Boolean functions using Boolean Laws and theorems and using K-maps.

Positive and Negative Logic

The binary variables can have either of the two states, i.e., the logic '0' state or the logic '1' state. These logic states in digital systems such as computers, for instance, are represented by two different voltage levels or two different current levels. If the more positive of the two voltage or current levels represents a logic '1' and the less positive of the two levels represents a logic '0', then the logic system is referred to as a *positive logic system*. If the more positive of the two voltage or current levels represents a logic '0' and the less positive of the two levels represents a logic '1', then the logic system is referred to as a *negative logic system*.

If the two voltage levels are 0V and +5V, then in the positive logic system the 0V represents logic '0' and the +5V represents logic '1'. In the negative logic system, 0V represents logic '1' and 5V represents logic '0'. If the two voltage levels are 0V and −5V, then in the positive logic system the 0V represents a logic '1' and the −5V represents a logic '0'. In the negative logic system, 0V represents logic '0' and −5V represents logic '1'.

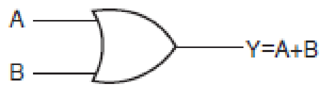
Logic Gates

The Logic Gate is the most basic building block of any digital system, including computers. Each one of the basic logic gates is a piece of hardware or an electronic circuit that can be used to implement some basic logic expression. While laws of Boolean algebra could be used to do manipulation with binary variables and simplify logic expressions, these are actually implemented in a digital system with the help of electronic circuits called logic gates. The three basic logic gates are the OR gate, the AND gate and the NOT gate.

OR Gate

A logic gate used to perform the operation of ***logical addition*** is called an OR gate. An OR gate performs an *OR* operation on two or more than two logic variables. The OR operation on two independent logic variables A and B is written as Y = A+B and reads as Y equals A OR B. An OR gate is a logic circuit with two or more inputs and one output. The output of an OR gate is LOW only when all of its inputs are LOW. For all other possible input combinations, the output is HIGH. A truth table lists all possible combinations of input binary variables and the corresponding outputs of a logic system. Figure shows the circuit symbol and the truth table of a two-input OR gate. The operation of a two-input OR gate is explained by the logic expression

$$Y = A+B$$



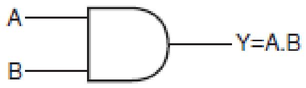
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Two input OR Gate

AND Gate

A logic gate used to perform ***logical multiplication*** is known as AND gate. An AND gate is a logic circuit having two or more inputs and one output. The output of an AND gate is HIGH only when all of its inputs are in the HIGH state. In all other cases, the output is LOW. The logic symbol and truth table of a two-input AND gate is shown in figure. The AND operation on two independent logic variables A and B is written as Y = A.B and reads as Y equals A AND B. The operation of a two-input AND gate is explained by the logic expression

$$Y = A.B$$



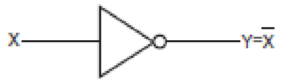
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Two input AND Gate

NOT Gate

A logic gate used to perform ***logical inversion*** is known as a NOT gate. A NOT gate is a one-input, one-output logic circuit whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa. If X is the input to a NOT circuit, then its output Y is given by Y = $\overline{X}$ or X' and reads as Y equals NOT X. The logic symbol and truth table of a NOT gate is shown in figure. The operation of a NOT gate is explained by the logic expression

$$Y = \overline{X}$$



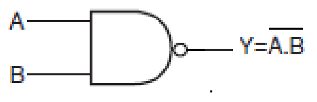
X	Y
0	1
1	0

NOT Gate

NAND Gate

NAND stands for NOT AND. An AND gate followed by a NOT circuit makes it a NAND gate. The output of a NAND gate is logic ‘0’ when all its inputs are logic ‘1’. For all other input combinations, the output is logic ‘1’. The symbol and truth table of a NAND gate is as shown. NAND gate operation is logically expressed as

$$Y = \overline{AB}$$



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

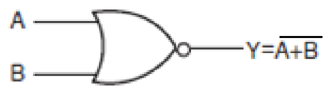
Two input NAND Gate

NAND Gate is known as Universal gate as it can be used alone to implement any gate operation. Hence it is said to be functionally complete.

NOR Gate

NOR stands for NOT OR. An OR gate followed by a NOT circuit makes it a NOR gate. The output of a NOR gate is logic ‘1’ when all its inputs are logic ‘0’. For all other input combinations, the output is logic ‘0’. The symbol and truth table of a NOR gate is as shown. The output of a two-input NOR gate is logically expressed as

$$Y = (\overline{A + B})$$



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

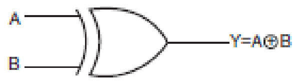
Two input NOR Gate

NOR gate is also known as Universal gate as it is used alone to implement any gate operation and hence it is also functionally complete.

EXCLUSIVE-OR Gate

The EXCLUSIVE-OR gate, commonly written as EX-OR gate, is a two-input, one-output gate. The output of an EX-OR gate is logic ‘1’ when the inputs are unlike and logic ‘0’ when the inputs are like. Although EX-OR gates are available in integrated circuit form only as two-input gates, unlike other gates which are available in multiple inputs also, multiple-input EX-OR logic functions can be implemented using more than one two-input gates. The output of a multiple-input EX-OR logic function is logic ‘1’ when the number of 1s in the input sequence is odd and logic ‘0’ when the number of 1s in the input sequence is even, including zero. The symbol and truth table of an EX-OR gate is shown in figure. The output of a two-input EX-OR gate is logically expressed as

$$Y = A \oplus B = A'B + AB'$$



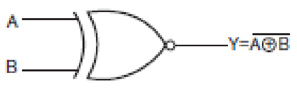
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Two input EX-OR Gate

EXCLUSIVE-NOR Gate

EXCLUSIVE-NOR, commonly written as EX-NOR, means NOT of EX-OR, i.e., the logic gate that we get by complementing the output of an EX-OR gate. The truth table of an EX-NOR gate is obtained from the truth table of an EX-OR gate by complementing the output entries as shown in figure. Logically,

$$Y = \overline{A \oplus B} = A'B' + AB$$



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Two input EX-NOR Gate

The output of a two-input EX-NOR gate is logic '1' when the inputs are like and logic '0' when they are unlike. In general, the output of a multiple-input EX-NOR logic function is logic '0' when the number of 1s in the input sequence is odd and a logic '1' when the number of 1s in the input sequence is even including zero.

Boolean algebra:

Boolean algebra is an algebraic structure defined on a set of elements B together with two binary operators + and · provided the following postulates are satisfied:

1. a) Closure with respect to the operator +.

   b) Closure with respect to the operator ·.

2. a) An identity element with respect to +, designated by 0: $x + 0 = 0 + x = x$.

   b) An identity element with respect to ·, designated by 1:$x \cdot 1 = 1 \cdot x = x$.

3. a) Commutative with respect to +: $x + y = y + x$.

   b) Commutative with respect to ·: $x \cdot y = y \cdot x$.

4. a) · is distributive over +: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

   b) + is distributive over ·: $x + (y \cdot z) = (x + y) \cdot (x + z)$.

5. For every element $x \in B$, there exists an element $x' \in B$ (complement of x) such that

$$x + x' = 1 \text{ and } x \cdot x' = 0.$$

Boolean Laws & Theorems:

Duality Principle:

It states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. If the dual of an algebraic expression is desired, OR and AND operators are interchanged and 1's are replaced by 0's and 0's by 1's.

1. $a)\ x + 0 = x \qquad b)\ x \cdot 1 = x$

2. $a)\ x + x' = 1 \qquad b)\ x \cdot x' = 0$

3. $a)\ x + x = x \qquad b)\ x \cdot x = x$

4. $a)\ x + 1 = 1 \qquad b)\ x \cdot 0 = 0$

5. DeMorgan's Theorem: $a)\ (x + y)' = x'y' \qquad b)\ (xy)' = x' + y'$

6. Absorption Theorem: $a)\ x + xy = x \qquad b)\ x(x + y) = x$

Simplification using Boolean Laws & Theorems:

The Boolean functions can be simplified by using appropriate Boolean laws and theorems.

Examples:

Simplify the following functions using Boolean laws and theorems:

1. $F = ABCD + ABC'D' + ABCD' + ABC'D + ABCDE + ABC'D'E' + ABC'DE$

Sol: $F = ABCD + ABC'D' + ABCD' + ABC'D + ABCDE + ABC'D'E' + ABC'DE$

$$= ABC(D + D') + ABC'(D + D') + ABDE(C + C') + ABC'D'E'$$

$$= AB(C + C') + ABDE + ABC'D'E' = AB(1 + DE + C'D'E') = AB$$

2. $F = xy + x'z + yz$

Sol: $F = xy + x'z + yz$

$$= xy + x'z + yz(x + x') = xy(1 + z) + x'z(1 + y)$$

$$= xy + x'z$$

3. $F = ABC + A'B'C + A'BC + ABC' + A'B'C'$

Sol: $F = ABC + A'B'C + A'BC + ABC' + A'B'C'$

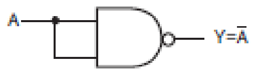
$$= BC + A'B' + ABC' = B(C + AC') + A'B' = AB + BC + A'B'$$

Universal Gates

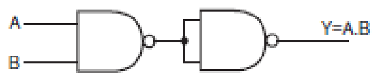
OR, AND and NOT gates are the three basic logic gates as they together can be used to construct the logic circuit for any given Boolean expression. NOR and NAND gates have the property that they individually can be used to hardware-implement a logic circuit corresponding to any given Boolean expression. That is, it is possible to use either only NAND gates or only NOR gates to implement any Boolean expression. This is so because a combination of NAND gates or a combination of NOR gates can be used to perform functions of any of the basic logic gates. It is for this reason that NAND and NOR gates are universal gates.

Implementation of gates using NAND gates

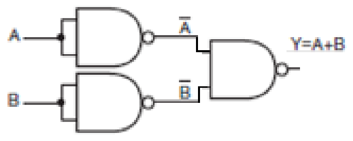
a) **NOT gate:**



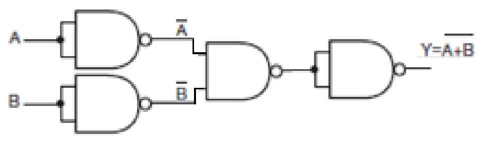
b) **AND gate:**



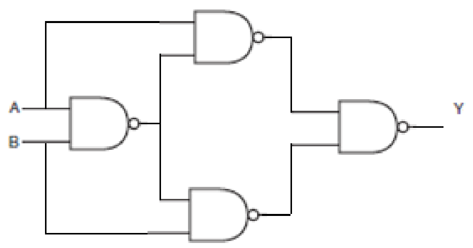
c) **OR gate:**



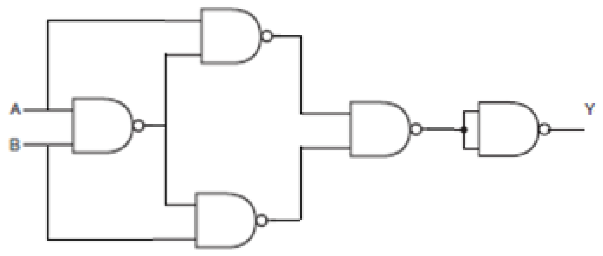
d) **NOR gate:**



e) **Ex-OR gate:**

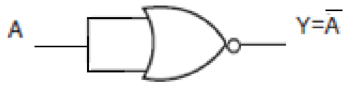


f) **Ex-NOR gate:**

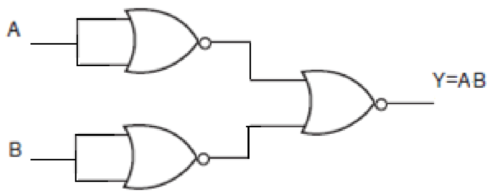


Implementation of gates using NOR gates

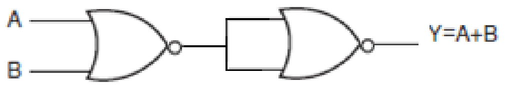
a) **NOT gate:**



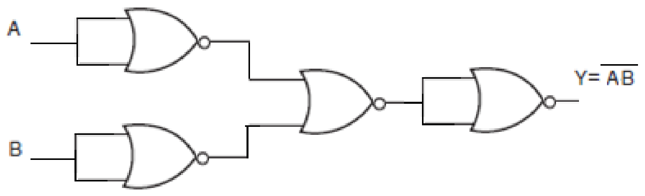
b) **AND gate:**



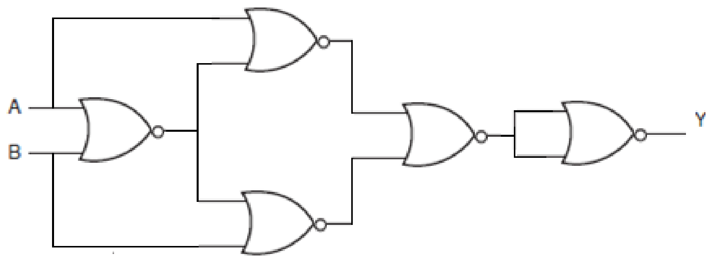
c) **OR gate:**



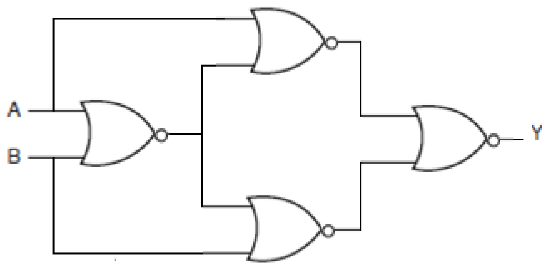
d) **NAND gate:**



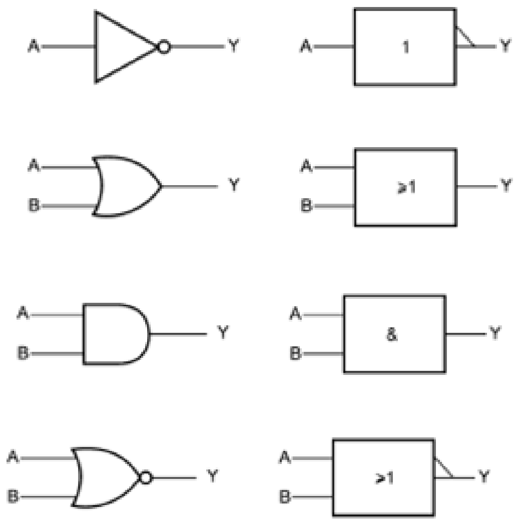
e) **Ex-OR gate:**

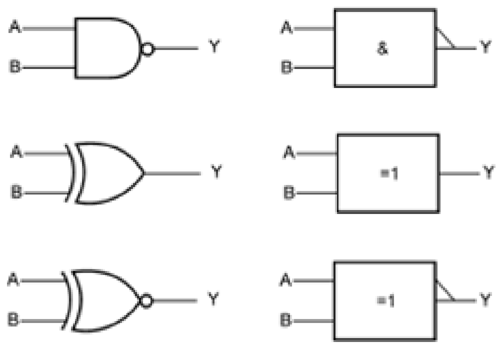


f) **Ex-NOR gate:**



IEEE/ANSI Symbols:





Boolean Expressions:

A Boolean expression or a function is an expression which consists of binary variables joined by the Boolean connectives AND and OR along with NOT operation.

Any Boolean expression can be expressed in two forms:

a) Canonical form

b) Standard form

Canonical Form:

An expanded form of Boolean expression, where each term contains all Boolean variables in their true or complemented form, is known as the canonical form of the expression.

a) **Sum of minterms:** Any Boolean function can be expressed as a sum of minterms expression. A minterm is a standard product which consists of all variables in either complemented or un-complemented form for which the output is 1. For example,

$$Y = A'BC + AB'C' + ABC$$
$$= \sum m(3,4,7)$$

is a sum of minterms expression with three variables.

b) **Product of maxterms:** Any Boolean function can be expressed as a product of maxterms expression. A maxterm is a standard sum which consists of all variables in either complemented or un-complemented form for which the output is 0. For example,

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

$$= \prod M\ (0,1,6)$$

is a product of maxterms expression with three variables.

Standard Form:

A simplified form of a Boolean expression which may consist of one or more number of variables in each term in either complemented or un-complemented form is known as Standard form of the expression.

a) **Sum of Products (SOP):** The sum of products is a Boolean expression containing AND terms, called Product terms, of one or more literals each; the sum denotes the ORing of these terms. For example,

$$Y = A'B + BC' + AC$$

is a SOP expression with three variables.

b) **Product of Sums (POS):** It is a Boolean expression containing OR terms called Sum terms and the product denotes the ANDing of these terms.

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

is a POS expression with three variables.

**** Canonical form is obtained when a function is taken from a truth table. When implementing a Boolean function with gates, standard form is preferred.**

<u>**Simplification of Boolean expressions:**</u>

The primary objective of all simplification procedures is to obtain an expression that has the minimum number of terms. Obtaining an expression with the minimum number of literals is usually the secondary objective. The Boolean functions can be simplified by using

a) Boolean Laws and theorems

b) K maps

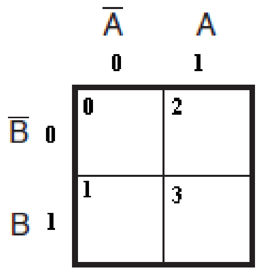
c) Quine Mc-Cluskey or Tabulation Method

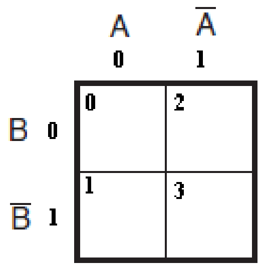
Simplification using K-maps:

A Karnaugh map is a graphical representation of the logic system. It can be drawn directly from either minterm (sum-of-products) or maxterm (product-of-sums) Boolean expressions. Drawing a Karnaugh map from the truth table involves an additional step of writing the minterm or maxterm expression depending upon whether it is desired to have a minimized sum-of-products or a minimized product of sums expression.

An n-variable Karnaugh map has 2^n squares, and each possible input is allotted a square. In the case of a minterm Karnaugh map, '1' is placed in all those squares for which the output is '1', and '0' is placed in all those squares for which the output is '0'. 0s are omitted for simplicity. An 'X' is placed in squares corresponding to 'don't care' conditions.

a) Two Variable K-map:





i) Sum of minterms representation

ii) Product of maxterms representation

b) Three Variable K-map:

	$\overline{A}\overline{B}$ 00	$\overline{A}B$ 01	AB 11	$A\overline{B}$ 10
$\overline{C}$ 0	0	2	6	4
C 1	1	3	7	5

	A+B 00	A+B' 01	A'+B' 11	A'+B 10
C 0	0	2	6	4
C' 1	1	3	7	5

i) Sum of minterms representation ii) Product of

maxterms representation

c) Four Variable K-map:

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

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

i) Sum of minterms representation ii) Product of maxterms

representation

d) Five Variable K-map:

BC / DE	A				A'			
	00	01	11	10	00	01	11	10
00	0	4	12	8	16	20	28	24
01	1	5	13	9	17	21	29	25
11	3	7	15	11	19	23	31	27
10	2	6	14	10	18	22	30	26

Simplification Algorithm:

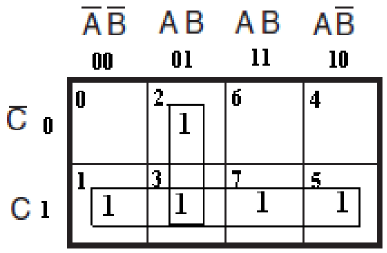
Simplification of logical functions using K-maps is based on the principle of combining terms in adjacent cells. Two cells are said to be adjacent if they differ in only one variable.

1. Identify the ones which cannot be combined with any other ones and encircle them. These are called essential prime implicants.
2. Identify the ones that can be combined in groups of two in only one way. Encircle them.
3. Identify the ones that can be combined with three other ones, to make a group of four adjacent ones, in only one way. Encircle such group of ones.
4. Identify the ones that can be combined with seven other ones, to make a group of eight adjacent ones, in only one way. Encircle them.
5. After identifying the essential groups of 2, 4, and 8 ones, if there still remain some ones which have not been encircled, then these are to be combined with each other or with other already encircled ones.

Examples:

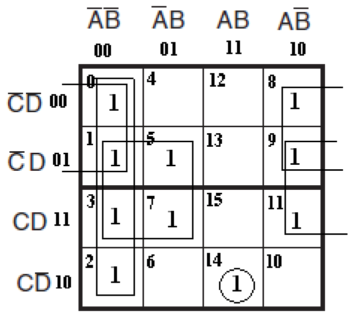
1. Simplify the Boolean function $F = A^{'}C + A^{'}B + AB^{'}C + BC$

Sol: $F = A^{'}C + A^{'}B + AB^{'}C + BC = A^{'}B^{'}C + A^{'}BC^{'} + A^{'}BC + AB^{'}C + AB'C$



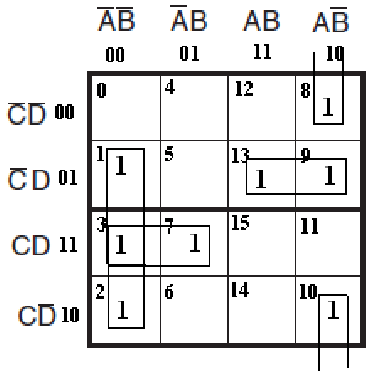
$$F = C + A'B$$

2. $F(A,B,C,D) = \sum m\,(0,1,2,3,5,7,8,9,11,14)$



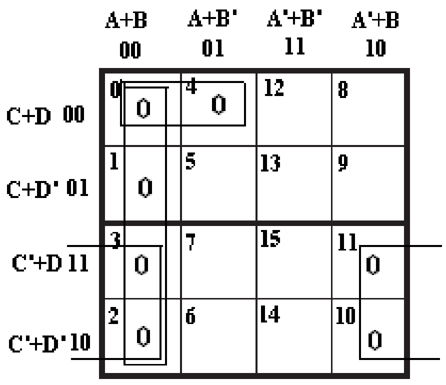
$$F = A'B' + A'D + ABCD' + B'C' + B'D$$

3. $F(A,B,C,D) = \sum m\,(1,2,3,5,7,8,9,10,13)$



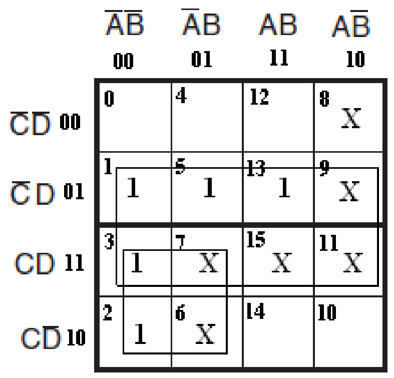
$$F = A'CD + AC'D + AB'D' + A'B'D + A'B'C$$

4. $F(A,B,C,D) = \prod M\,(0,1,2,3,4,10,11)$



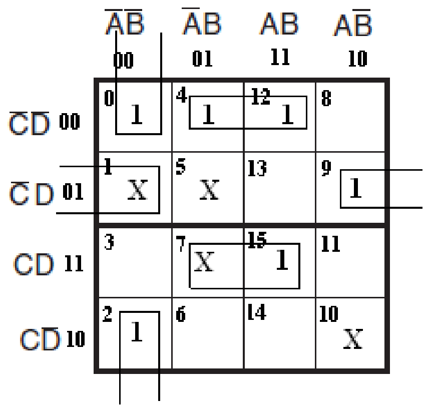
$$F = (A+B)(A+C+D)(B+C')$$

5. $F = \sum m\,(1,2,3,5,13) + \sum d\,(6,7,8,9,11,15)$



$$F = D + A'C$$

6. $F = \sum m(0,2,4,9,12,15) + \sum \emptyset(1,5,7,10)$



$$F = A'B'D' + BCD + BC'D' + B'C'D$$

Two level Implementation:

Any logic circuit can be implemented in two levels by representing the Boolean function either in SOP or POS form. Two level NAND and NOR circuits can be obtained by representing the expression in SOP and POS form respectively. Minimum propagation delay will be obtained by using two level implementation. But as the number of terms increases, the number of inputs increases for the second level gate.

NAND Implementation:

By expressing the given function in SOP form, the logic circuit can be implemented using two level NAND gates.

Example:

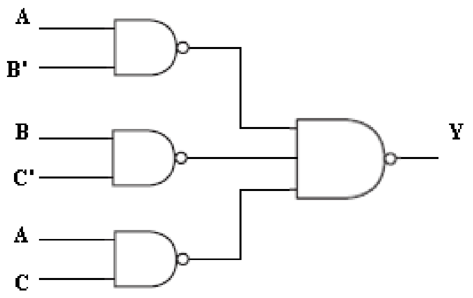
Implement $Y = A(A' + B') + BC' + AC$ using NAND gates.

Sol: $Y = A(A' + B') + BC' + AC$

$= AB' + BC' + AC$ ----- SOP form

By using double complements we get,

$$Y = (Y')' = \{(AB' + BC' + AC)'\}' = (AB')'(BC')'(AC)'$$



NOR Implementation:

By expressing the given function in POS form, the logic circuit can be implemented using two level NOR gates.

Example:

Implement $Y = A'B + BC'$ using NOR gates.

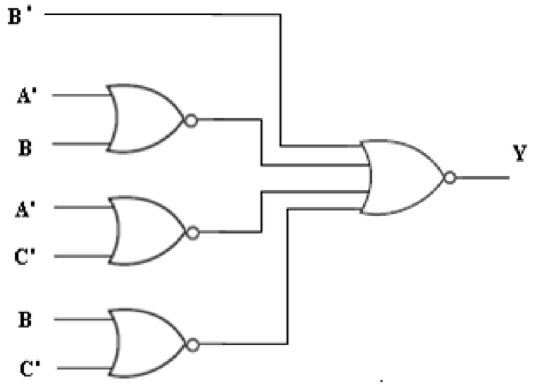
Sol: $Y = A'B + BC'$

$= (A'B + B)(A'B + C')$ ……. Distributive law

$Y = B(A' + B)(A' + C')(B + C')$ ……. Distributive law

and the expression is in POS form.

$$Y = (Y')' = [\{B(A' + B)(A' + C')(B + C')\}']'$$
$$= [(B)' + (A' + B)' + (A' + C')' + (B + C')']'$$



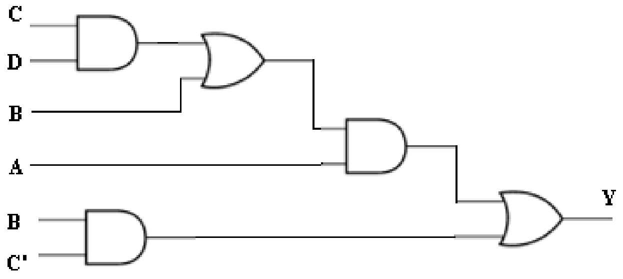
Multilevel Implementation:

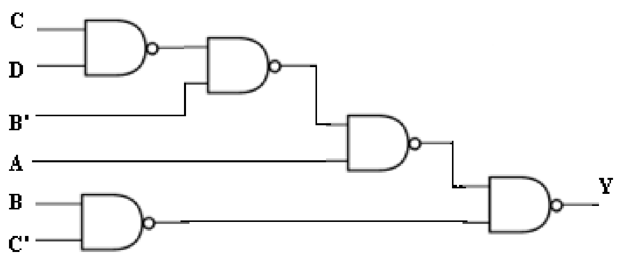
Multi level NAND or NOR implementation of a Boolean circuit can be achieved by replacing each gate in the circuit with the NAND or NOR equivalent circuits respectively. With this type of implementation, very large propagation delay is achieved and only two input gates are required.

NAND Implementation:

By replacing each gate in the circuit with the NAND equivalent, multi level NAND circuit is achieved. When two single input NAND gates (inverters) are in series, they can be removed.

Example: $Y = A(B + CD) + BC'$





NOR Implementation:

By replacing each gate in the circuit with the NOR equivalent, multi level NOR circuit is achieved. When two single input NOR gates (inverters) are in series, they can be removed.

Example: $Y = A(B + CD) + BC'$