ELEMENTARY PHYSICS AND DIGITAL ELECTRONICS

UNIT-3

Logic Gates:

Logic gates are an important concept if you are studying electronics. These are important digital devices that are mainly based on the Boolean function. Logic gates are used to carry out logical operations on single or multiple binary inputs and give one binary output. In simple terms, logic gates are the electronic circuits in a digital system.

Universal gates:

A universal gate is a logic gate that can implement any Boolean function without using another logic gate. The universal gates are the NOR and NAND gates.

Positive and Negative logic:

Positive Logic:

In positive logic representation Bit 1 represents Logic high and Bit 0 represent a Logic low as shown in fig 2 a and b. High is represented by +5 Volts and low is represented by -5 Volts or 0 Volts.

Negative Logic:

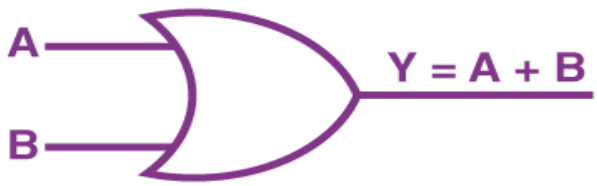
In Negative logic representation Bit 1 represents logic low and Bit 0 represents logic high as shown in Fig 3 a and b. In terms of voltage level, bit 1 can be represented as +5V and bit 0 can be represented as 0 V or -5 Volts.

Types of Logic Gates:

There are several basic logic gates used in performing operations in digital systems.

1. OR Gate

In an OR gate, the output of an OR gate attains state 1 if one or more inputs attain state 1.



The Boolean expression of the OR gate is Y = A + B, read as Y equals A 'OR' B.

The truth table of a two-input OR basic gate is given as;

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

2. AND Gate

In the AND gate, the output of an AND gate attains state 1 if and only if all the inputs are in state 1.

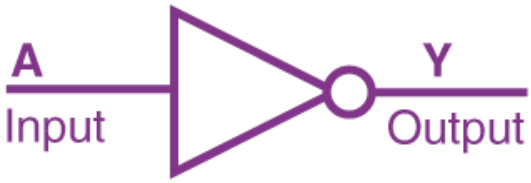
The Boolean expression of AND gate is Y = A.B

The truth table of a two-input AND basic gate is given as;

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

3. NOT Gate

In a NOT gate, the output of a NOT gate attains state 1 if and only if the input does not attain state 1.



The Boolean expression is:

$Y=A^-$

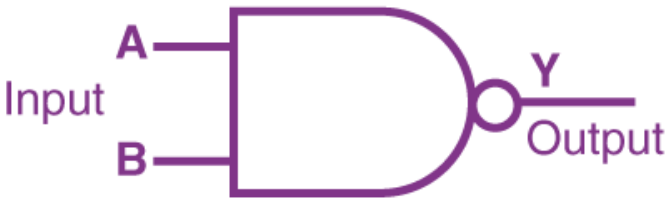
The truth table of NOT gate is as follows;

A	Y
0	1
1	0

When connected in various combinations, the three gates (OR, AND and NOT) give us basic logic gates such as NAND, and NOR gates, which are the universal building blocks of digital circuits.

4. NAND Gate

This basic logic gate is the combination of AND and NOT gates.



The Boolean expression of the NAND gate is:

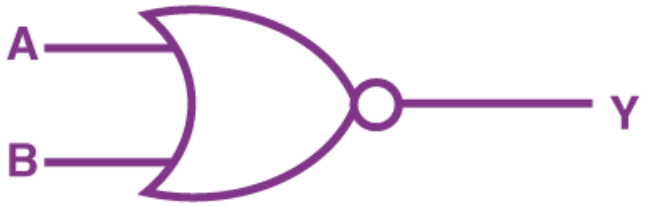
$Y=A.B—$

The truth table of a NAND gate is given as;

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

5. NOR Gate

This gate is the combination of OR and NOT gate.



The Boolean expression of NOR gate is:

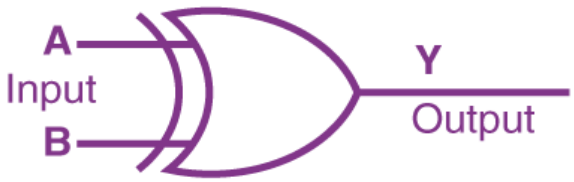
$Y=A+B—$

The truth table of a NOR gate is as follows;

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

6. Exclusive-OR gate (XOR Gate)

In an XOR gate, the output of a two-input XOR gate attains state 1 if one adds only input attains state 1.



The Boolean expression of the XOR gate is:

$A.B^{-}+A^{-}.B$

or

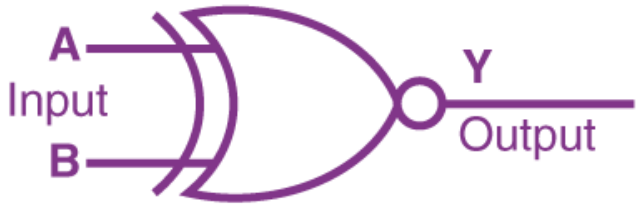
$Y = A \oplus B$

The truth table of an XOR gate is;

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

7. Exclusive-NOR Gate (XNOR Gate)

In the XNOR gate, the output is in state 1 when both inputs are the same, that is, both 0 or both 1.



The Boolean expression of the XNOR gate

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

The truth table of an XNOR gate is given below;

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

Application Of Logic Gates:

Logic gates have a lot of applications, but they are mainly based upon their mode of operations or their truth table. Basic logic gates are often found in circuits such as safety thermostats, push-button locks, automatic watering systems, light-activated burglar alarms and many other electronic devices.

One of the primary benefits is that basic logic gates can be used in various combinations if the operations are advanced. Besides, there is no limit to the number of gates that can be used in a single device. However, it can be restricted due to the given physical space in the device. In digital integrated circuits (ICs), we will find an array of the logic gate area unit.

De Morgan's Theorem:

First theorem – It states that the NAND gate is equivalent to a bubbled OR gate.

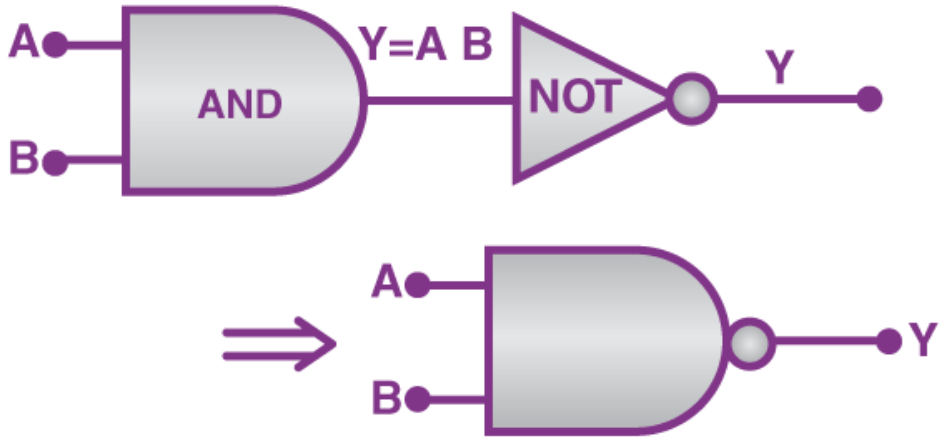
$A.B^{-}=A^{-}+B^{-}$

Second theorem – It states that the NOR gate is equivalent to a bubbled AND gate.

$A+B—=A^{-}.B^{-}$

Important Conversions:

1. **The 'NAND' gate:** From 'AND' and 'NOT' gate.



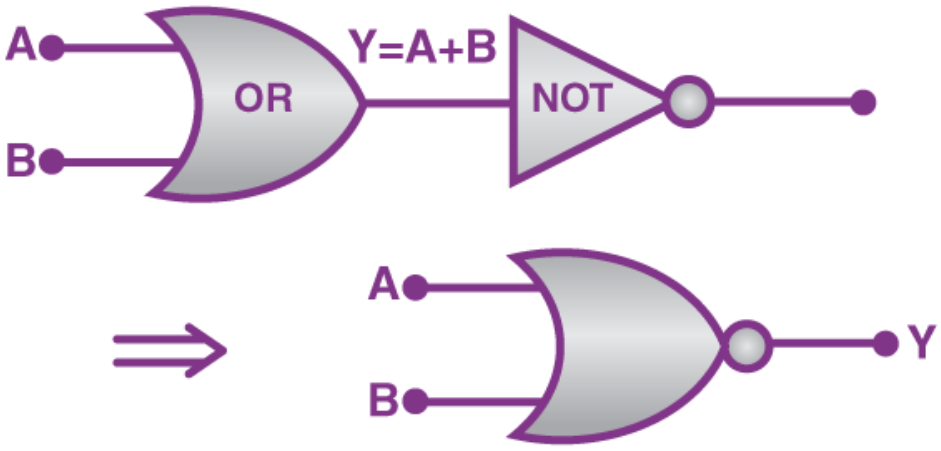
Boolean expression and truth table :

$Y=A.B—$

A	B	Y'=A· B	$Y=A.B—$
0	0	0	1
0	1	0	1
1	0	0	1

1	1	1	0

2. **The 'NOR' gate:** From 'OR' and 'NOT' gate



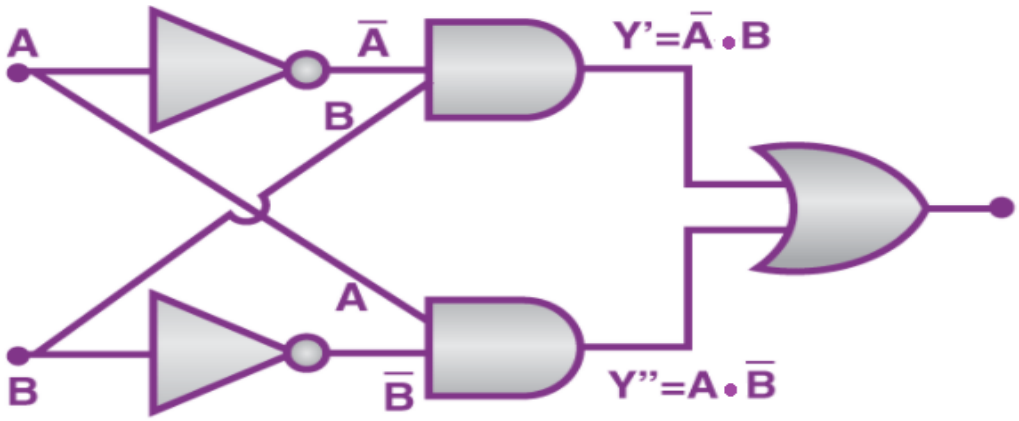
Boolean expression and truth table:

Y=A+B—

A	B	Y'=A+B	*Y=A+B—*
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

3. The 'XOR' gate: From 'NOT', 'AND' and 'OR' gate.

The logic gate, which gives a high output (i.e., 1) if either input A or input B but not both are high (i.e. 1), is called the exclusive OR gate or the XOR gate. It may be noted that if both the inputs of the XOR gate are high, then the output is low (i.e., 0).



Boolean expression and truth table:

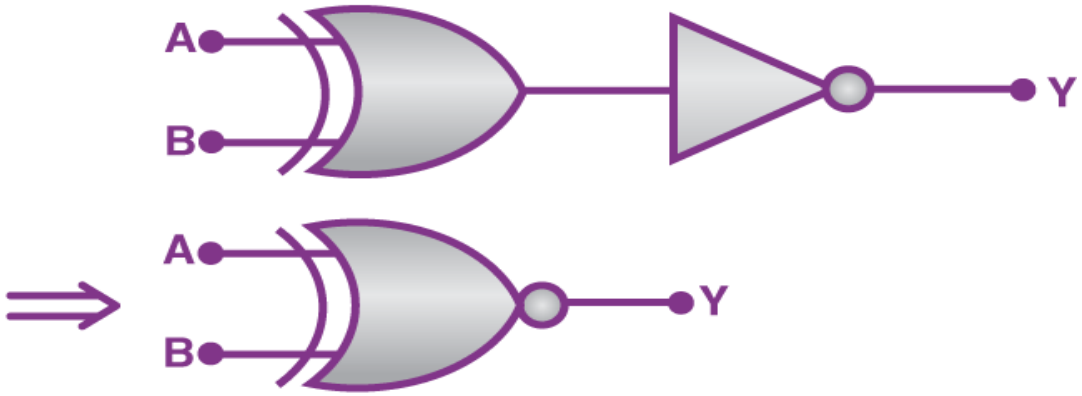
$A.B^{-}+A^{-}.B$

or

$Y=A \oplus B$

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

4. The Exclusive nor (XNOR) gate: XOR + NOT



Boolean expression:

$Y=(A \oplus B)^{-}$

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Boolean algebra:

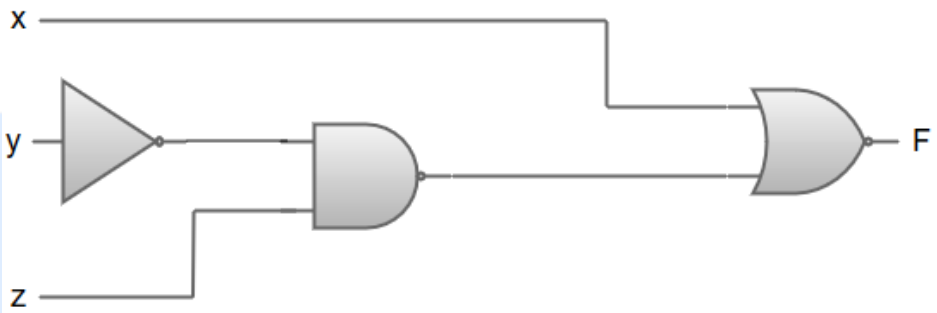
Boolean algebra can be considered as an algebra that deals with binary variables and logic operations. Boolean algebraic variables are designated by letters such as A, B, x, and y. The basic operations performed are AND, OR, and complement.

The Boolean algebraic functions are mostly expressed with binary variables, logic operation symbols, parentheses, and equal sign. For a given value of variables, the Boolean function can be either 1 or 0. For instance, consider the Boolean function:

F = x + y'z

The logic diagram for the Boolean function F = x + y'z can be represented as:

F = x + y'z



- The Boolean function F = x + y'z is transformed from an algebraic expression into a logic diagram composed of AND, OR, and inverter gates.
- Inverter at input 'y' generates its complement y'.
- There is an AND gate for the term y'z, and an OR gate is used to combine the two terms (x and y'z).
- The variables of the function are taken to be the inputs of the circuit, and the variable symbol of the function is taken as the output of the circuit.

Note: A truth table can represent the relationship between a function and its binary variables. To represent a function in a truth table, we need a list of the 2^n combinations of n binary variables.

The truth table for the Boolean function F = x + y'z can be represented as:

F = x + y'z

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

Standard Forms for Logical Expressions:

SOP (Sum of products):

SOP stands for Sum of Product. SOP form is a set of product(AND) terms that are summed(OR) together. When an expression or term is represented in a sum of binary terms known as minterms and sum of products.

POS (product of sum):

POS stands for product of sum. A technique of explaining a Boolean expression through a set of max terms or sum terms, is known as POS (product of sum).

Difference between SOP and POS in Digital Logic

S.No.	SOP	POS
1	SOP stands for Sum of Products.	POS stands for Product of Sums.
2	It is a technique of defining the Boolean terms as the sum of product terms.	It is a technique of defining Boolean terms as a product of sum terms.
3	It prefers minterms.	It prefers maxterms.
4	In the case of SOP, the minterms are defined as ‘m’.	In the case of POS, the Maxterms are defined as ‘M’
5	It gives HIGH(1) output.	It gives LOW(0) output.
6	In SOP, we can get the final term by adding the product terms.	In POS, we can get the final term by multiplying the sum terms.

Minterm and Maxterm:

There are two ways in which we can put the Boolean function. These ways are the minterm canonical form and maxterm canonical form.

Literal

A Literal signifies the Boolean variables including their complements. Such as B is a boolean variable and its complements are ~B or B', which are the literals.

Minterm

The product of all literals, either with complement or without complement, is known as **minterm**.

Example

The minterm for the Boolean variables A and B is:

1. A.B
2. A.~B
3. ~A.B

The complement variables ~A and ~B can also be written as A' and B' respectively. Thus, we can write the minterm as:

1. A.B'
2. A'.B

Minterm from values:

Using variable values, we can write the minterms as:

1. If the variable value is 1, we will take the variable without its complement.
2. If the variable value is 0, take its complement.

Maxterm:

The sum of all literals, either with complement or without complement, is known as **maxterm**.

Example:

The maxterm for the Boolean variables A and B will be:

1. A+B
2. A+~B
3. ~A+B

We know that the complement variables ~A and ~B can be written as A' and B' respectively. So, the above maxterm can be written as

1. A+B'
2. A'+B

Maxterm from values:

Using the given variable values, we can write the maxterm as:

1. If the variable value is 1, then we will take the variable without a complement.

2. If the variable value is 0, take the complement of the variable.

Karnaugh Map:

In realization of digital electronic systems, the simplification of Boolean expressions is one of the most crucial steps because it reduces the hardware complexity and cost of production. There are several tools and methods available for simplifying complex Boolean expression. K-Map or Karnaugh Map is one of such simplification methods. K-Map was developed by Maurice Karnaugh in the year of 1953. It is a visual or graphical method used to simplify the Boolean expressions.

K-Map is one of the most efficient simplification tools when the number of variables in the Boolean expression are less than or equal to four. However, for five, six, and more variables, the K-Map becomes quite difficult.

The K-Map or Karnaugh map makes the use of two-dimensional table for simplification of the Boolean functions. The size of this table increases considerably with the increase in the number of variables in the Boolean functions.

If n is the number of variables in the given Boolean function, then the corresponding Karnaugh map (K-Map) will have 2^n squares or cells. For examples, if the number of variables in the Boolean function is 3, then the corresponding K-Map will have 8 (= 2^3) cells.

Structure of Karnaugh Map:

A typical K-Map has a table of certain cells. On the top-left corner of this table, a set of variables are represented as A, B, C, D. These variables are basically the input variables involved in the logical expression that requires to be simplified.

The values of these inputs variables in binary form are represented along their respective sides, i.e., on the top and left of the table.

K-Map Simplification:

The procedure of K-Map or Karnaugh map simplification is started with the entering the values of the variables, either in their SOP (Sum of Products) form or in POS (Product of Sums) form, in the right K-map cells. After that we need to group the maximum number of 1s (in the case of SOP form) or the maximum number of 0s (in the case of POS form). Each of these groups must be in powers of 2 and must be carried on in decreasing order only.

Once the grouping is done, each group has to be expressed in terms of combinations of input variables which are corresponding to the common binary values along the associated rows and columns. At last, all the combinations express the output expression of the Boolean function.

Advantages of Karnaugh Map:

The following are the important advantages of the Karnaugh map −

- For simplifying Boolean expression, the K-map does not require the knowledge of theorems of Boolean algebra.

- Karnaugh map involves a smaller number of steps in simplification process of logical expressions as compared to other simplification techniques.

Limitations of Karnaugh Map:

The following are the major limitations of the Karnaugh map −

- The most significant limitation of the Karnaugh map is that it is only efficient when the Boolean expression has less number of variables. It becomes quite complicated with the higher number of variables in the logical expression.
- The simplification of a Boolean function having more than or equal to five variables using K-Map is quite complex.
- It is very difficult to get equations correct with more than 5 variables using the K-map.

Don't Care Condition:

The "Don't care" condition says that we can use the blank cells of a K-map to make a group of the variables. To make a group of cells, we can use the "don't care" cells as either 0 or 1, and if required, we can also ignore that cell. We mainly use the "don't care" cell to make a large group of cells.

The cross(×) symbol is used to represent the "don't care" cell in K-map. This cross symbol represents an invalid combination. The "don't care" in excess-3 code are 0000, 0001, 0010, 1101, 1110, and 1111 because they are invalid combinations. Apart from this, the 4-bit BCD to Excess-3 code, the "don't care" are 1010, 1011, 1100, 1101, 1110, and 1111.
We can change the standard SOP function into a POS expression by making the "don't care" terms the same as they are. The missing minterms of the POS form are written as maxterms of the POS form. In the same way, we can change the standard POS function into an SOP expression by making the "don't care" terms the same as they are. The missing maxterms of the SOP form are written as minterm of the SOP form.

Significance of "Don't Care" Conditions:

Don't Care conditions has the following significance with respect to the digital circuit design:

Simplification:

These conditions denote the set of inputs that never occurs for given digital circuits. Therefore, to simplify the boolean output expressions, the 'don't care' are used.

Reduced Power Consumption:

The switching of the state is reduced when we group the terms long with "don't care". This reduces the required memory space resulting in lower power consumption.

Lesser number of gates:

For reducing the number of gates that are used to implement the given expression, simplification places an important role. So, the 'don't care' makes the logic design more economical.

Prevention of Hazards:

In the digital system, the 'don't care' place an important role in hazards prevention.

States in Code Converters:

These also play an important role in code conversion. For example- In the design of a 4-bit BCD-to-XS-3 code converter, the input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are don't cares.