Fundamental Concepts of Boolean Algebra

When a variable is used in an algebraic formula, it is generally assumed that the variable may take any numerical value.

Ex:-

In the formula $2x - 5y = 2$ we assume that $x$, $y$, and $z$ may range through the entire field of real number.

• Boolean algebra is an algebra that deals with binary variables and logic operations. The variable are designated by letter such as $A, B, X$ and $Y$. The three basic logic operation **AND**, **OR** and **Complement**.

• A Boolean function can be expressed algebraic with binary variable, the logic operation **symbols**, **parentheses**, and **equal sign**.
Logic Multiplication

An important operation in Boolean algebra, we call logic Multiplication or the logic AND operation.

- The Rules for this operation are,
  
  \[ 0 \cdot 0 = 0 \]
  \[ 0 \cdot 1 = 0 \]
  \[ 1 \cdot 0 = 0 \]
  \[ 1 \cdot 1 = 1 \]
  
- Ex: If \( z = x \cdot y \) and \( x = 0, y = 1 \), then \( z = 0 \)

- Only when \( x \) and \( y \) are both 1 would \( z \) be a 1.

- Both + and \( \cdot \) obey a mathematical rule called the associative law.
  
  - This low for +, that \( (x + y) + z = x + (y + z) \).
  - And for \( \cdot \), that \( x \cdot (y \cdot z) = (x \cdot y) \cdot z \)

Note;-

While either +’s or \( \cdot \)’s can be used freely, the two cannot be mixed without ambiguity in the absence of further rule.

Ex:- \( A \cdot B + C \) means

\( (A \cdot B) + C \) or \( A \cdot (B + C) \)
AND Gates and OR Gates

What is Gate?

Binary logic deals with binary variables and with operations that assume a logic meaning. It is used to describe, in algebraic or tabular form. The manipulation of binary information is done by logic circuit called Gates.

- Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic of requirements are satisfied.
- A verity of logic gates are commonly used in digital computer system.
- Each gate has a distinct graphic symbol and it’s operation can be described by means of an algebraic expression.
- The Input - Output relationship of the binary variables for each gate can be represented in tabular form by truth table.
- The + and . Operation are physically realized by two types of electronic circuits, called OR gate and AND Gate.
AND Gates and OR Gates

OR Gates:-

The OR gate produces the inclusive OR functions that the output is 1, if the input A or B or both inputs are 1, otherwise the output is 0.

- The algebraic symbol of Or function is +, similar to arithmetic addition.
- Or gates may have more than two inputs and definition, the output is 1 if any input is 1.
AND Gates and OR Gates

AND Gates:

The AND gate produce the AND logic function i.e., the output is 1, if input A and input B both equal to 1, otherwise the output is 0.

- The algebraic operation symbol of the AND function is same as the multiplication symbol of ordinary arithmetic.
- AND gates may have more than two inputs, and definition, the output is 1 if and only if all inputs are 1.
Complementation and Inverters

Complementation:-

The symbol of complementation is ‘\( \overline{\cdot} \).

Ex:- \( \overline{x} \), meaning “take the complement of x”
or \( (\overline{x + y}) \), means take the complement of \( x + y \).

• The complement operation can be defined as
  \[
  \overline{0} = 1 \\
  \overline{1} = 0
  \]

Rules for Complement

\[
\overline{\overline{x}} = x
\]

Ex:-
\[
\overline{\overline{0}} = \overline{1} = 0 \\
\overline{1} = \overline{0} = 1
\]

This rule that double complement give the original value is an important characteristic of a Boolean algebra which does not generally hold for most unary operation.
Complementation and Inverters

Inverter:

The complementation operation is physically realized by a gate or circuit called inverter.

Block diagram of Inverter.

0 = 1
1 = 0

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Evaluation of Logical Expressions

| TABLE 3.1 |
|---|---|---|
| X | Y | Z |
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| TABLE 3.2 |
|---|---|---|---|
| X | Y | Z | $\overline{Z}$ |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |
### TABLE 3.3

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$\bar{Z}$</th>
<th>$Y \bar{Z}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### TABLE 3.4

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$\bar{Z}$</th>
<th>$Y \bar{Z}$</th>
<th>$X + Y \bar{Z}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### TABLE 3.6

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$\overline{X}$</th>
<th>$\overline{Y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### TABLE 3.7

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$\overline{X}$</th>
<th>$\overline{Y}$</th>
<th>$\overline{X} + \overline{Y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### TABLE 3.8

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$\overline{X}$</th>
<th>$\overline{Y}$</th>
<th>$\overline{X} + \overline{Y}$</th>
<th>$Y(\overline{X} + \overline{Y})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### TABLE 3.9

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$\overline{X}$</th>
<th>$\overline{Y}$</th>
<th>$\overline{X} + \overline{Y}$</th>
<th>$Y(\overline{X} + \overline{Y})$</th>
<th>$X + Y(\overline{X} + \overline{Y})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Basic Law of Boolean Algebra

1. \(0 + X = X\)
2. \(1 + X = 1\)
3. \(X + X = X\)
4. \(X + \overline{X} = 1\)
5. \(0 \cdot X = 0\)
6. \(1 \cdot X = X\)
7. \(X \cdot X = X\)
8. \(\overline{X} \cdot X = 0\)
9. \(\overline{X} = X\)

Rules 10 and 11 Which are know as Commutative law

10. \(X + Y = Y + X\)
11. \(X \cdot Y = Y \cdot X\)
12. \(X + (Y + Z) = (X + Y) + Z\)
13. \(X(YZ) = (XY)Z\)
14. \(X(Y + Z) = XY + XZ\)
15. \(X + XZ = X\)
16. \(X(X + Y) = X\)
17. \((X + Y)(X + Z) = X + YZ\)
18. \(X + \overline{X}Y = X + Y\)
19. \(XY + YZ + \overline{Y}Z = XY + Z\)

Rules 12 and 13 Which are know as Associative law

Rules 14 Which are know as Distributive law
Proof By Perfect Induction

\[(W+X)(Y+Z) = W(Y + Z) + X(Y+Z) = WY + WZ + XY + XZ\]
De Morgan’s Theorems

• Two rules of De Morgan’s Theorems.
  1. \( (x + y) = x \cdot \bar{y} \)
  2. \( (x \cdot y) = x + \bar{y} \)

• Two steps are used to form a complement.
  1. The + symbols are replaced with \( \cdot \) symbols and \( \cdot \) symbols with +.
  2. Each of the terms in the expression is complemented.

• The use of De Morgan’s theorems may be demonstrated by finding the complement of the expression.

  Ex. \( x + yz \)
  
  \[ = x + (y \cdot z) \]
  \[ = x \cdot (y \cdot z) \]
  \[ = \bar{x} \cdot (\bar{y} + \bar{z}) \]
De Morgan’s Theorems

• Sometime necessary to complement both sides of an equation.

Ex. $wx + yz = 0$

$wx + yz = 0$

$(\overline{w} + \overline{x}) (\overline{y} + \overline{z}) = 1$
Basic Duality of Boolean Algebra

De Morgan’s theorem expresses a basic duality which underlies all Boolean algebra.

• The theorems which have been presented can all be divided into pair.

Ex. \((x + y) + z = x + (y + z)\)

is dual of \((xy) z = x (yz)\)

and \(x + 0 = x\) in the dual of \(x \cdot 1 = x\)

• If \(x + xy = x\), immediately add the theorem

\(x (x + y) = x\)

to the list of theorems as the dual of the first expression.
Derivation of Boolean expression

When designing a logical circuit, the logical designer work from two sets of known values are

1. The various states which the inputs to the logical network can take.
2. The desired output for each input condition.

- Consider a specific problem A logical network has two input x and y and an output z.
- The relationship between inputs and output are
  1. When both x and y are 0, the output z is to be 1.
  2. When x is 0 and y is 1, the output z is to be 0.
  3. When x is 1 and y is 0, the output z is to be 1.
  4. When x is 1 and y is 1, the output z is to be 1.
Derivation of Boolean expression

In tabular form

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Product Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• The column will consist of a list of product term obtained from the values of the input variables.
• The new column will contain each of the input variables listed representing the respective input complement when the input value for this variable is 0 and not complemented when the input value is 1.
• The terms obtained in this manner are designated as product terms.
• With two input variable x and y, each row of the table will contain a product term consisting of x and y.
Derivation of Boolean expression

- Each row of the table will contain a product term consisting of $x$ and $y$ with $x$ or $y$ complemented or not, depending on the input values for the row.
- Whenever $z$ is equal to 1, the $x$ and $y$ product term from the same row is removed and formed into sum-of-product expression.
- Therefore, the product terms from the first, third and fourth rows are selected.
- The logical sum of these product is equal to the expression desired. This type of expression is often referred to as a canonical expansion for the function.

\[
\begin{align*}
\overline{x} \overline{y} + \overline{x} y + x y &= z \\
\overline{x} y + x (y + y) &= z \\
x y + x &= z
\end{align*}
\]
Interconnecting Gates

The combination of the OR, AND and Inverter.
Sum of Product and Product Of Sum

1. Product term :-

A product term is a single variable or the logical product of several variable. The variable may or may not be complemented.

2. Sum term :-

A sum term is a single variable or the sum of several variable. The variable may or may not be complemented.

Ex :- Product Term

   x.y.z

Ex :- Sum Term

   x + y + z
Sum of Product and Product Of Sum

Sum of Products Expression :-

A sum of product expression is a product term or several product terms logically add.

Ex :- Sum of Products Expression

1. \( \overline{x} y + x \overline{y} \)
2. \( x \)
3. \( x y + z \)
4. \( \overline{x} \overline{y} + \overline{x} \overline{y} \overline{z} \)
5. \( x + y \)
Product of sum expression :-

A product of sum expression is a sum term or several sum terms logically multiplied.

Ex :- Product of Sum expression

1. \((x + y)(\bar{x} + \bar{y})\)
2. \((x + y)(x + \bar{y})(\bar{x} + \bar{y})\)
3. \((x + y + z)(x + \bar{y})(\bar{x} + \bar{y})\)
4. \(x + z\)
5. \(\bar{x}\)
6. \((x + y)x\)

Note :-

When a variable \(x\) is available, and its complement \(\bar{x}\) is also available, that is no inverters are required to complement input. This is quite important and quite realistic, since most signals come from flip-flops.
Derivation of Product of Sum expression

The method for arriving at the desired expression is

1. Construct a table of the input and output values.

2. Construct an additional column of sum term containing complemented and un complemented variable (depending of the values in the input columns) for each row of the table. In each row of the table, a sum term is formed. However, in this case, if the input values for a given variable is 1, the variable will be complemented and if 0 not complemented.

3. The desired expression in the product of the sum terms from the rows in which the output is 0.
Derivation of Product of Sum expression

Based on Derivation of Boolean expression.

The relationship between input and output

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Sum Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- A column containing the input variable in sum-term from is now added in each row, A given variable is complemented if the input values for row and is not complemented of products Expressions.
- A product of sum expression is now formed by selecting those sum terms for which the output is 0 and multiplying them.
- when 0’s appear as in the above table the expression described.

\[( x + \overline{y} ) ( \overline{x} + y) \]
Derivation of Product of Sum expression

A sum of product expression found by multiplying the two terms.

\[(x + y)(\bar{x} + y)\]  
\[= (x + y)(x + y)\]  
\[= x x + x y + x y + y y\]  
\[= x y + \bar{x} \bar{y}\]
Sum of Product

Product of sum

FIGURE 3.7

(a) AND-to-OR gate networks. (b) OR-to-AND gate networks.
NAND gates and NOR gates

NAND :-

The NAND function is complement of the AND function, as indicated by the graphical symbol, which consists of an AND graphic symbol followed by a small circle. The designation NAND is derived from the abbreviation of NOT-AND.

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
NAND gates and NOR gates

NOR :-

The NOR gate is complement of the OR gate and uses an OR graphically Symbol followed by a small circle
Map Method for Simplification Expressions

K – Map :-

The map method provides a simple, straightforward procedure for simplifying Boolean Expression. This method may be regarded as a pictorial arrangement of the truth table which allows an easy interpretation for choosing the minimum number of terms needed to express the function algebraically. The map method is also known as Karnaugh map of K-map.

- Each combination of variable in the truth table is called minterm.
- The diagram in each case lists the $2^n$ different product terms which can be formed in exactly $n$ variables, each in a different square.
- For function $n$ variables, a product term in exactly these $n$ variable is called minterm.
- Three type of K-map
  1. Two Variable
  2. Three Variable
  3. Four Variable
K – Map

Venn diagram and equivalent K-map for two variables
K – Map

Venn diagram and equivalent K-map for four variables
In some logic circuits, the output responses for some input Conditions are don’t care whether they are 1 or 0.

In K-maps, don’t-care conditions are represented by d’s in the corresponding cells.

Don’t-care conditions are useful in minimizing the logic functions using K-map.
- Can be considered either 1 or 0
- Thus increases the chances of merging cells into the larger cells
  --> Reduce the number of variables in the product terms
Design using NAND gates

\[ A \cdot B \cdot C = \overline{A} + \overline{B} + \overline{C} \]
(a) \[ A \cdot B \] \[ C \cdot D \] \[ E \cdot F \]

\[ (A \cdot B) + (C \cdot D) + (E \cdot F) \]

\[ = A \cdot B + C \cdot D + E \cdot F \]

(b) \[ A \cdot B \] \[ C \cdot D \] \[ E \cdot F \]

\[ (A \cdot B) + (C \cdot D) + (E \cdot F) \]

\[ = A \cdot B + C \cdot D + E \cdot F \]

(c) \[ A \cdot B \] \[ C \cdot D \] \[ E \cdot F \]

\[ AB + CD + EF \]
\[(A \cdot B \cdot C) \cdot (C \cdot \overline{B}) = (A \cdot B \cdot C) + (C \cdot \overline{B}) = \overline{A}BC + \overline{CB} \]

\[\overline{A \cdot B \cdot C} + (\overline{C} \cdot \overline{B}) = \overline{A}BC + \overline{CB} \]

\[\overline{A}BC + \overline{CB} \]
Design using NOR gates

\[ \overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C} \]

NOR gate

Equivalent gate
(a) \[ (A + B) + (C + D) + (E + F) \]
\[ = (A + B)(C + D)(E + F) \]
\[ = (A + B)(C + D)(E + F) \]

(b) \[ (A + B)(C + D)(E + F) \]
\[ = (A + B)(C + D)(E + F) \]