- **Boolean Algebra**
- **De Morgan’s Theorem**
- **Representation of Boolean Algebra**
- **De Morgan’s Theorem through Logic Circuits**
- **Karnaugh Map Mathod**
- **Binary/BCD Subtraction and Addition**

Boolean Algebra

Boolean Algebra is Algebra of logic. This is an algebra that deals with logical propositions, which are either true or false. This algebra is suitable for binary number system and is very useful in designing digital circuits, which operates under logic.

For example,

A+A =A, not 2A

Also 1+1 = 1 not 2 as it is logical expression.

One can visualize this as,

TRUE+TRUE=TRUE

FALSE+FALSE=FALSE

A Boolean algebraic expression is composed of variables, constants and operators. The **variables are** generally represented by the letters of the Alphabet (say A) which can have two possible values 1 or 0. The interpretation of 1 may be that the variable is presented input signal is ON, is TRUE, and is a positive voltage. If A is 0, then it mean that the variable is absent, input signal if OFF, is FALSE, and is a negative voltage. Similarly, the **Boolean Constant** can have any two values, either 1 or 0.

Boolean Operators are used in Boolean Algebra where a mathematical function called **Boolean Function** is constructed. These operators are the Symbols

PLUS (+) meaning an OR operation

DOT (.) meaning and AND operation

BAR ‘A read as COMPLEMENT meaning a NOT operation.

Postulates of Boolean Algebra

A set of Boolean postulates are the following

(a) $A = 0$ iff $A \neq 1$

$A = 1$ iff $A \neq 0$

(b) $0.0=0$

(c) $1+1=1$

(d) $0+0=0$

(e) $1.1=1$

(f) $1.0=0.1=0$

(g) $1+0=0+1=1$

The realization of any Boolean Expression can be obtained with the help of a table called **TRUTH TABLE**. To simplify a Boolean Expression, one requires certain laws of Boolean Algebra.

Laws of Boolean Algebra and their Truth Table

a) Commutative Law

i) $A+B=B+A$

ii) $A.B=B.A$

Truth Table

A	*B*	*A+B*	*A.B*	*B+A*	*B.A*
0	0	0	0	0	0
0	1	1	0	1	0
1	0	1	0	1	0
1	1	1	1	1	1

b) Associative Law

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

ii) A.(B.C)=(A.B).C

Truth Table

A	*B*	*C*	*B+C*	*A+B*	*A+(B+C)*	*(A+B)+C*	*B.C*	*A.B*	*A.(B.C)*	*(A.B).C*
0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	0	0	0	0
0	1	0	1	1	1	1	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0
1	0	0	0	1	1	1	0	0	0	0
1	0	1	1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1

C) Distributive Law

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

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

Truth Table

A	*B*	*C*	*B+C*	*A.B*	*A.C*	*A.(B+C)*	*(A.B+A.C)*	*B.C*	*A+B*	*A+C*	*A+(B.C)*	*(A+B).(A+C)*
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
1	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**
1	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

De Morgan's Theorem

This is a useful theorem in Boolean Algebra which states how to complement a Boolean expression. They allow us to convert back and forth from minterm to maxterm forms of Boolean expression. It helps to eliminate long over-bars that cover several variables.

a) First Theorem

$\overline{A+B} = \overline{A}\,.\,\overline{B}$ (For two variables)

In general $\overline{A+B+C}+\ldots\ldots = \overline{A}\,.\,\overline{B}\,.\,\overline{C}\,\ldots.$ (For many variables)

b) Second Theorem

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

In general $\overline{A}\,.\,\overline{B}\,.\,\overline{C} = \overline{A}+\overline{B}+\overline{C}+\ldots\ldots.$

Truth Table

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

Representation of Boolean Algebra

& De Morgan's Theorem through Logic Circuits

There are nine basic identities commonly used in converting complex Boolean expression to their simple forms. A Boolean identity usually consists of one variable and one constant equates two expressions, which are equal for all possible combinations of the variables. The equivalence of two expressions is presented through Truth Table and representative circuits symbol.

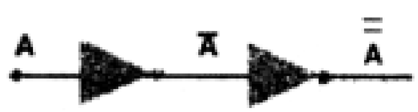
Double Complementation

A double complementation is the complementation of a single complement. The single complement is called **NOT** operation.

Expression : $\overline{A} = A$

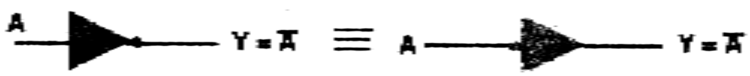
Truth Table

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



The triangle with a circle at the vertex is known as **inverter.** This circle is called '**Bubble'** and the expression for output A is read as 'A complement' or 'A NOT'.

In the language of logic gate, this is called NOT Gate. The equivalent inverter symbol looks like



AND Function Identities

The identity states that

A.1 = A

Corresponding **Truth Table** and circuit are given below

A	1	Y
0	**1**	**0**
1	**1**	**1**

The Boolean expression for the output is A.1, which is read as 'A and 1". In genral this is A.B.

Similarly, there are other identities using AND operator.

i) A.0=0

A	0	Y
0	0	0
1	0	0

i) A.A=A

A	A	Y
0	0	0
1	1	1

i) $A.\overline{A}=0$

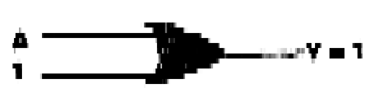
A	$\overline{A}$	Y
0	1	0
1	0	0

OR Function Identities

This identity can be realized with one input permanently tied to logic 1 and the other is varying

A+1=1

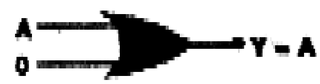
A	1	Y
0	1	1
1	1	1



Other three identities using OR operator are

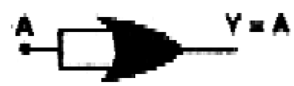
i) **A+0 = A**

A	0	Y
0	0	0
1	0	1



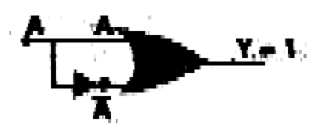
ii) **A+A = A**

A	0	Y
0	0	0
1	0	1



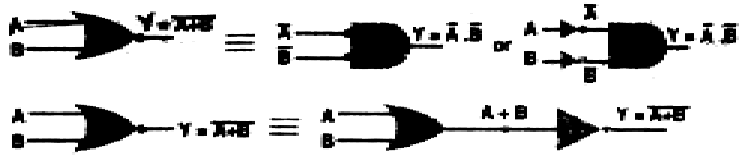
iii) **$A + \overline{A} = 1$**

A	0	Y
0	0	0
1	0	1



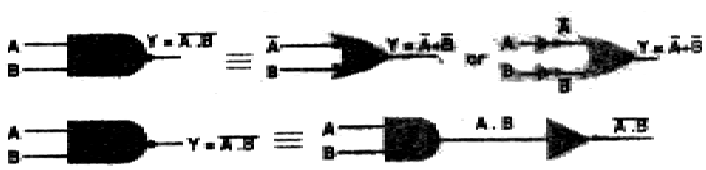
The Circuit Representation of De Morgan's Theorem

1. A+B = A.B known as NOR circuit, which is equivalent to Bubbled input AND circuit.



A	B	A+B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}\,.\,\overline{B}$
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**

2. = $\overline{A} + \overline{B}$ known as NAND circuit, which is equivalent to Bubbled input OR circuit.



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

$\overline{A.B}$

Karnaugh Map Method

There are two methods for developing the required logic diagram from a given Truth Table. The first method requires "**Boolean Algebra"** and "**De Morgan's Theorem**" to reduce the expressions produced to lowest term (Minimal expressions). The second method is a variation of the first and uses a tool called the "**Karnaugh Map (K-Map)**". The K-Map is the simple3st and most commonly used method. It is a graphical method (in the form of table) extensively used to simplify Boolean equation.

The K-Map method uses a table or map to reduce its expressions. Each position in the table is called a "CELL". CELLs are filled with ones and zeros according to the expressions to be reduced.

Adjacent ones are grouped together in clusters, called “subcobes”, following definite rules: a subcube must be of size 1,2,4,8,16, etc. All 1s must be included in a subcube of maximum size. These rules are explained through examples below

Assignment 1

Designed a circuit that will behave according to this Truth Table

Inputs			**Output**
C	**B**	**A**	**Y**
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**1**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**
1	**1**	**1**	$\overline{C}\,\overline{B}$ **1**

<u>Solution</u>

Step 1. Draw the table. Choose two of the variables to use as column headings across the top. We will chose C and B. From all combinations of C and ‘C with B and ‘B. Each column heading should differ from all adjacent column by one variable only.

Part 1

$\overline{C}\,\overline{B}$	$\overline{C}\,B$	$C\,B$	

Start with $\overline{C}\,\overline{B}$ and change $\overline{B}$ to B to from the heading for column 2, .

Then change $\overline{C}$ to C for the third column CB, and finally . The fourth column wraps around to the first column and should differ by one variable only, which it does.

Part 1

$\overline{C}\,\overline{B}$	$\overline{C}\,B$	$C\ B$	

Use the third variable, A for row headings $\overline{A}$ and A.

Step 2. Fill the table with ones and zeros from the Truth Table. The output Y is 1 in line 3 when we have and B and . Place a 1 in the table in cell B . The output Y is also 1 on line 4, which is represented by BA, on line 6, which is C A, on line 7, whichis CB , and on line 8, which is CBA. Fill those cells with ones and the remaining cells with zeros.

Part 1

	$\overline{C}\,B$	$C\ B$	
0	**1**	**1**	**0**
0	**1**	$\overline{C}\,\overline{B}$ **1**	**1**

Step 3. Combine adjacent cells that contain ones in sub cubes of maximum size. The four ones in the centre of the table compose a sub cube of size 4.

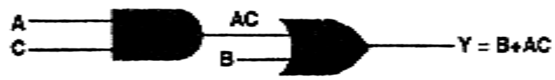
Part 1

$\overline{C}\,\overline{B}$	$\overline{C}\,B$	$C\ B$	
0	**1**	**1**	**0**
0	**1**	**1**	**1**

The 1 in cell $C\overline{B}A$ has not been included in a sub cube so it is used with its adjacent 1 in a sub cube of size 2.

Step 4. Write the expression that 3ach sub cube represents. In the sub cube of size 4, find the variable that occurs in all four cells. In these case B is the only variable that appears in all four cells. The sub cube of size 4 represents B. In the sub cube of size two, A and C appear in each cell, so the sub cube represents AC.

Steps 5. From the output expression. The output Y is the expression from each sub cube ORed together. In this case Y=B+AC.



The Truth Table can be implemented by the above logic diagram.

Verification

In the circuit, when A and C are both 1s, the output of the AND will also be 1. A 1 into an OR gives a 1 out. In the Truth Table, A and C are both 1s on line6 and 8, and the required output is 1. In the circuit, any time B is 1 the output is 1. In the Truth Table, B is 1 on lines 3, 4, 7 and 8, and the required output is 1. The rest of the time both inputs into the OR gate will be 0, and the result will be 0. This occur on lines of 1, 3 and 5 of the Truth Table where the output is 0. In all cases the circuit produces the results required by the Truth Table.

Assignment 2

Use a Karnaugh map to design a logic diagram to implement the following Truth Table.

Inputs Output

Inputs			Output
C	**B**	**A**	**Y**
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	**1**	**1**	**1**

<u>Solution</u>

Step 1. Draw the table.

$\overline{C}\,\overline{B}$	$\overline{C}\,B$	$C\ B$	
0	**1**	**1**	**0**
0	**1**	**1**	**1**

Step 2. Fill the table with 1s and 0s from the Truth Table.

Step 3. Combination adjacent cells that contain 1s into sub cubes (size 1, 2, 4, or 8).

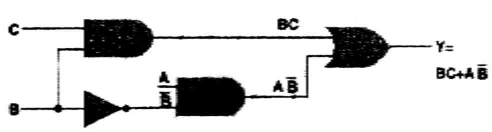
$\overline{C}\,\overline{B}$	$\overline{C}\,B$	$C\ B$	
0	**0**	**1**	**0**
1	**0**	**1**	**1**

The right side of the table **"wraps around"** to the other side so that the table is continuos. The 1s in $C\,\overline{B}$ the lower corners from a sub cube of size 2. The two sub cube **"cover"** the map in that all 1s are contained in sub cube. Any additional sub cube drawn would add un-needed terms to the final expression.

Step 4. Write the expression that each sub cube represents. In the vertical sub cube, C and B remain constant. In the horizontal sub cube, ‘B and A are constant.

Step 5. From the output expression.

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



Assignment 3

Use a Karnaugh map to design a logic diagram to implement the following Truth Table.

Inputs				Output
D	**C**	**B**	**A**	**Y**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Solution

Step 1. Draw the table. Since four variables are needed, use two across the top and down the side.

$\bar{D}\,\bar{C}$	$\bar{D}\,C$	$D\,C$	$D\,C$
1	0	1	1
0	1	1	0
0	0	1	0
1	0	1	1

Step 2. Fill the table with 1s and 0s from the truth Table.

Step 3. Combine adjacent cells that contain 1s into subcubes of size 1, 2, 4, 8 or 16.

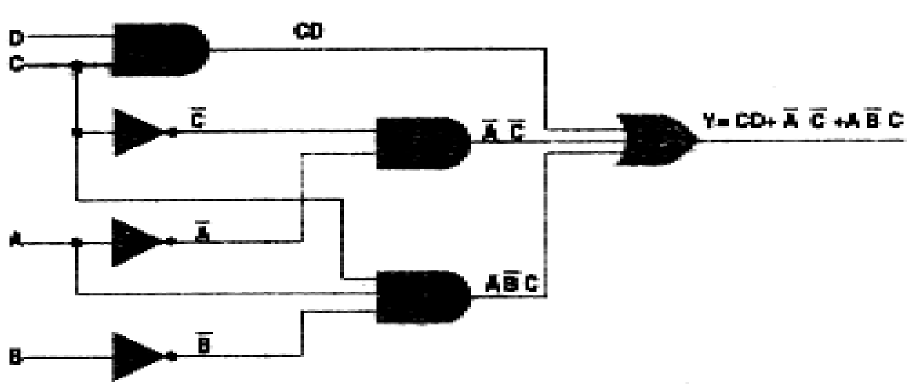
$\bar{D}\,\bar{C}$	$\bar{D}\,C$	$D\,C$	
1	**0**	**1**	**1**
0	**1**	**1**	**0**
0	**0**	**1**	**0**
1	**0**	**1**	**1**

Since the map is continuous top to bottom and side to side, the four corners are adjacent and form a sub cube of size 4. The DC column forms another sub cube of size 4. One cell remains uncovered. ‘DC‘BA forms a sub cube of size 2 with the cell on its right.

Step 4. Write the expression that each sub cube $D\bar{C}$ represents. The sub cube formed by the four corners represents the term ‘A‘C. The vertical sub cube represents the expression CD, and the sub cube of size 2 represents the expression A‘BC.

Step 5. From the output expression.

$$Y = CD + \bar{A}\,\bar{C} + A\bar{B}C$$



Binary/BCD Subtraction and Addition

Write working with digital equipment, one has to convert from the binary code to decimal numbers. If a binary number, say 110011 is given, what would be it equals in decimal? First write down the binary numbers as

Binary

1	**1**	**0**	**0**	**1**	**1**

Decimal

$1x2^5+1x2^4+0x2^3+0x2^2+1x2^1+1x2^0$

or $32+16+0+0+2+1$

or 51

Shortly, it is written as

$(110011)_2=(51)_{10}$

In another example, $(101010)_2=(?)_{10}$

Binary

1	**0**	**1**	**0**	**1**	**0**

Decimal

$1x2^5+0x2^4+1x2^3+0x2^2+1x2^1+0x2^0$

or $32+0+8+0+2+0$

or 42

Thus, $(101010)_2=(42)_{10}$

What about $(1101010.101)_2=(?)_{10}$

Binary

1	**1**	**0**	**1**	**0**	**1**	**0**	**.**	**1**	**0**	**1**

Position

6	**5**	**4**	**3**	**2**	**1**	**0**	**-1**	**-2**	**-3**

Decimal

$1x2^6+1x2^5+0x2^4+1x2^3+0x2^2+1x2^1+0x2^0.1x2^{-1}+0x2^{-2}+1x2^{-3}$

or $64+32+0+8+0+2+0.1/_2+0/_4+1/_8$

or $106(5/_8)$or 106.625

Thus $(1101010.101)_2=(106.625)_{10}$

Now, let us look at the method of converting decimal number to binary number.

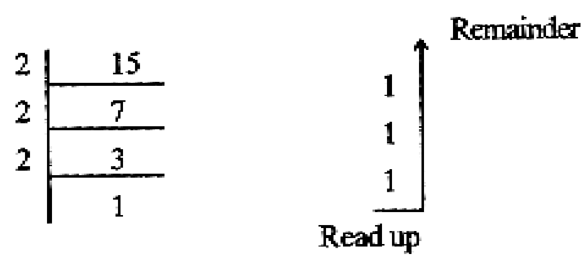
The most popular method to convert decimal number to binary number is the "DOUBLE-DABBLE" method. In this method, one progressively divides the given decimal number by 2 and writes down the remainder after each division. The remainder is read in reverse order.

However, to convert a fraction number into binary number, multiply the decimal number by 2 and record the carry in the integer position and down ward read these carries. Let us take few examples to understand the method of conversion.

Assignment 4

$(15)_{10}=(?)_2$

Solution



$(15)_{10} = (1111)_2$

Assignment 5

$(.35)_{10}=(?)_2$

Solution

0.35x2=0.70 with a carry 0

0.70x2=1.40 with a carry 1

0.40x2=0.80 with a carry 0

0.80x2=1.60 with a carry 1

0.60x2=1.20 with a carry 1

0.20x2=0.40 with a carry 0 $\rightarrow$ **Stop when the number started repeating**

$(.35)_{10}=(.010110....)_2$

Binary Addition

Binary addition is performed in the same manner as decimal Addition. However, since Binary system has only two digits, the addition table for Binary Arithmetic is very simple consisting of only four entries.

The complete table for Binary addition is as follows;

0+0=0

0+1=1

1+0=1

1+1=0 plus a carry of 1 to next higher column

Alternately, 1+1=10 (sum 0 with carry 1)

Carry-overs are performed in the same way as in decimal arithmetic.

Since 1 is the largest digit in Binary system, any sum greater than 1 requires that a digit be carried over. For instance, 10 plus 10 binary requires the addition of two 1's in the second position. Since 1+1=0 plus a carry over1, the sum of 10+10 is 100 in binary.

Assignment 6

$(1010)_2+(101)_2=(1111)_2$

Solution

Binary		Decimal
$(1010)_2$		$(10)_{10}$
$\underline{+(101)}_2$		$\underline{+(5)}_{10}$
$(1111)_2$	=	$(15)_{10}$

Assignment 7

$(1011)_2+(111)_2=(10010)_2$

Solution

Binary		Decimal
$(1011)_2$	Carry bits(1+1+1=11)	$(11)_{10}$
$+(111)_2$		$+(7)_{10}$
$(10010)_2$	=	$(18)_{10}$

Assignment 8

$(100011)_2+(11011)_2=(1000010)_2$

Solution

Binary		Decimal
1111 Carry bits		$(39)_{10}$
$(100011)_2$		$(27)_{10}$
$+(11011)_2$		
$(1000010)_2$	=	$(66)_{10}$

Binary Subtraction

The principle of binary subtractions consists of two steps. The first step is to determine if it is necessary to borrow. If the subtrahend (the lower digit) is larger than the minuend (the upper digit), it is necessary to borrow from the column to the left. It is important to note here that the value borrowed depends on the base of the number and is always the decimal equivalent of the base. Thus, in decimal, 10 is borrowed; in binary, 2 is borrowed. The second step is simply to subtract the lower value from the upper value.

The complete table for binary subtraction is as follows

0-0=0

1-0=1

1-1=0

0-1=1 with a borrow from the next column.

Alternately, 10-1=1

Note that the only case in which it is necessary to borrow is when 1 is subtracted from 0. Let us take few more examples to make the operation more clear

Assignment 9

$(10101)_2 - (01110)_2 = (0011)_2$

Solution

Binary		Decimal
12		
0202 borrow		
$(10101)_2$		$(21)_{10}$
$\underline{-(01110)}_2$		$\underline{(14)}_{10}$
$\underline{(00111)}_2$	=	$\underline{(7)}_{10}$

Assignment 10

$(10100)_2 - (1111)_2 = (00101)_2$

Solution

Binary		Decimal
01212 borrow		
$(10100)_2$		$(20)_{10}$
$\underline{-(1111)}_2$		$\underline{(15)}_{10}$
$\underline{(00101)}_2$	=	$\underline{(5)}_{10}$

Assignment 11

$(101.01)_2 - (010.11)_2 = 010.10$

Solution

Binary		Decimal
0202 borrow		
$(101.01)_2$		$(5.25)_{10}$
$-(010.11)_2$		$(2.75)_{10}$
$(010.10)_2$	=	$(2.50)_{10}$

Exercise

Truth Table

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

1. Use the Boolean algebra method to develop a circuit to implement the truth table above.
2. Use the Karnaugh map method to develop a circuit to implement the truth table above.