Digital Electronics
PCC11EC03 (Bridge Course)

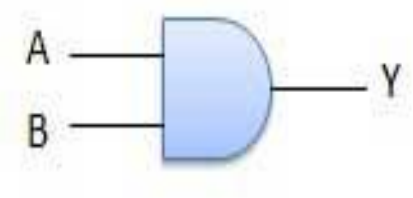
Module 1:Implementation of Logic functions

Logic Gates

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

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



Y=A.B

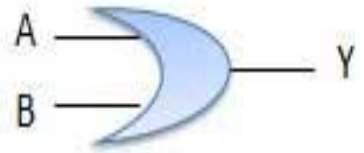
Truth Table

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

OR Gate

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

Y=A+B



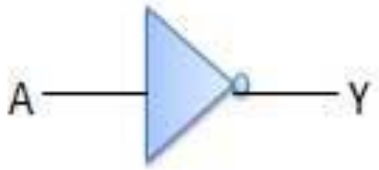
Truth Table

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

NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

$$Y=\overline{A}$$

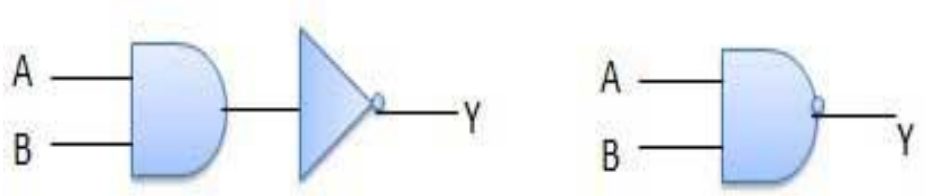


Inputs	Output
A	B
0	1
1	0

NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

$Y=\overline{A.B}$



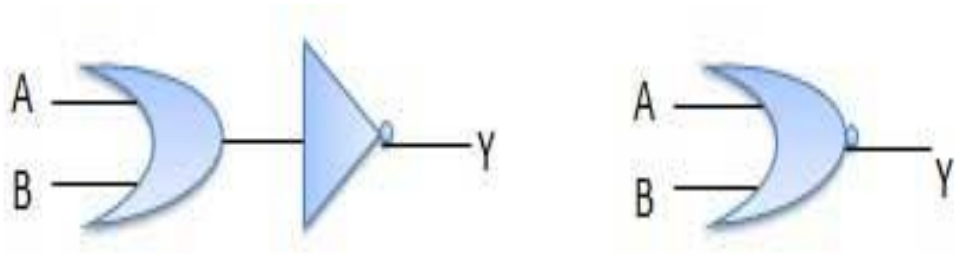
Truth Table

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 input (n >= 2) and one output.

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



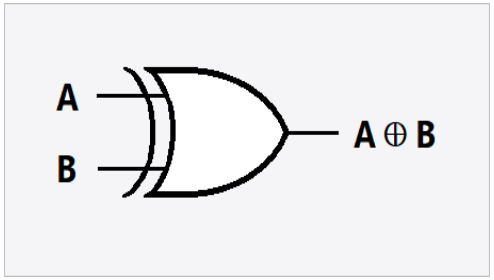
Truth Table

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

XOR Gate

XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

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



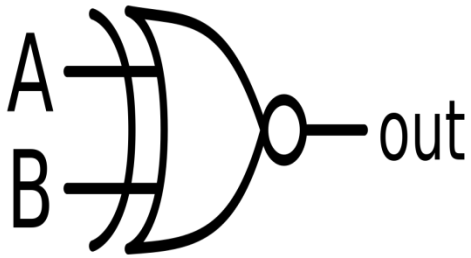
Truth Table

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

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 sometime as X-NOR gate. It has n input (n >= 2) and one output.

$$Y = A\,B + \overline{A}\;\;\overline{B}$$



Truth Table

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

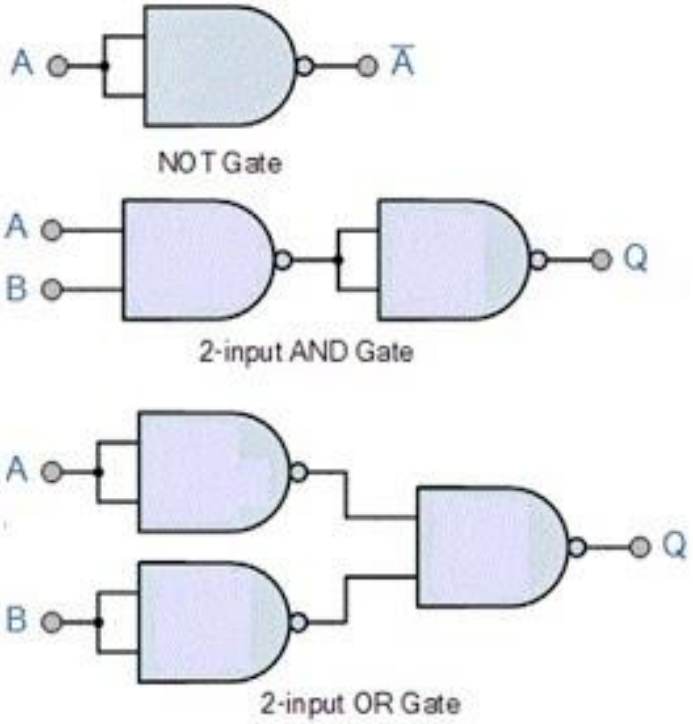
Home Assignment

Implement Basic logic functions AND,OR and NOT using NAND and NOR gates (Universal Gates)

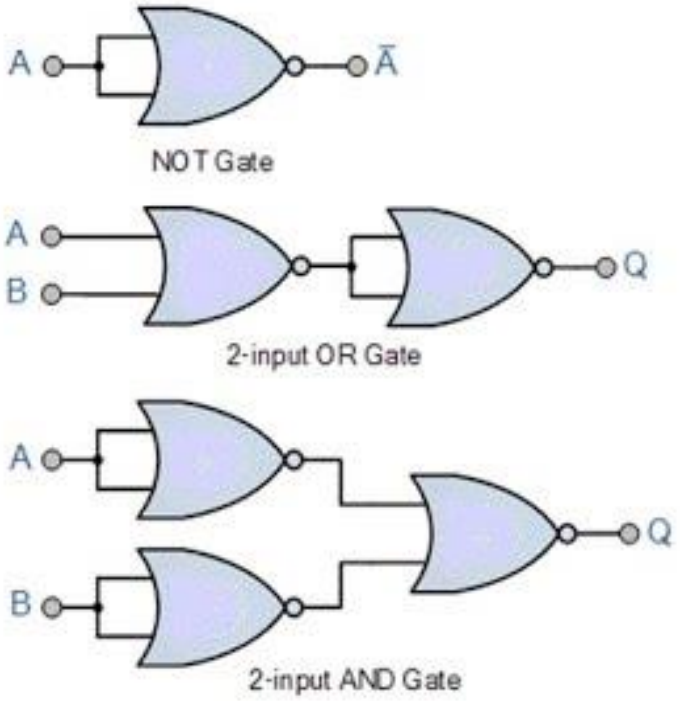
Note: A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

Realization of Basic Gates using Universal gates

Using NAND Gate



Using NOR Gate



Formulating a logic function

Boolean Logic in SOP (Sum of Products) and POS (Product of Sums) forms is foundational to digital circuit design, providing a systematic way of expressing and simplifying logical expressions.

SOP form combines multiple OR operations applied to ANDed terms, whereas POS form involves AND operations applied to ORed terms, each efficiently representing complex logic circuits.

These forms are pivotal in optimizing digital systems for both hardware efficiency and computational speed, serving as the cornerstone for understanding and implementing digital logic in various applications.

Boolean Function Representation

- Sum-of-Products (SOP) Form
- Product-of-sums (POS) form
- Canonical/Standard forms

Sum of Product (SOP) Form

A sum-of-products form can be formed by adding (or summing) two or more product terms using a Boolean addition operation. Here the product terms are defined by using the AND operation and the sum term is defined by using OR operation.

Examples

F(A,B,C)= AB + ABC + CDE

Product of Sums (POS) Form

Here the sum terms are defined by using the OR operation and the product term is defined by using AND operation. When two or more sum terms are multiplied by a Boolean OR operation, the resultant output expression will be in the form of product-of-sums form or POS form.

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

SOP example

F = A′BC + AB′C + ABC ‘ + ABC

Truth table:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

POS example

F = (A + B + C) (A + B + C ‘) (A + B’ + C) (A’ + B + C)

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Important Note:

1. SOP expressions can be implemented using AND-OR or NAND-NAND gates
2. POS expressions can be implemented using OR-AND or NOR-NOR gates

Show example

				Minterms		*Maxterms*
X	Y	Z		*Product Terms*		*Sum Terms*
0	0	0		$m_0 = \overline{X} \cdot \overline{Y} \cdot \overline{Z} = \min\left(\overline{X}, \overline{Y}, \overline{Z}\right)$		$M_0 = X + Y + Z = \max\left(X, Y, Z\right)$
0	0	1		$m_1 = \overline{X} \cdot \overline{Y} \cdot Z = \min\left(\overline{X}, \overline{Y}, Z\right)$		$M_1 = X + Y + \overline{Z} = \max\left(X, Y, \overline{Z}\right)$
0	1	0		$m_2 = \overline{X} \cdot Y \cdot \overline{Z} = \min\left(\overline{X}, Y, \overline{Z}\right)$		$M_2 = X + \overline{Y} + Z = \max\left(X, \overline{Y}, Z\right)$
0	1	1		$m_3 = \overline{X} \cdot Y \cdot Z = \min\left(\overline{X}, Y, Z\right)$		$M_3 = X + \overline{Y} + \overline{Z} = \max\left(X, \overline{Y}, \overline{Z}\right)$
1	0	0		$m_4 = X \cdot \overline{Y} \cdot \overline{Z} = \min\left(X, \overline{Y}, \overline{Z}\right)$		$M_4 = \overline{X} + Y + Z = \max\left(\overline{X}, Y, Z\right)$
1	0	1		$m_5 = X \cdot \overline{Y} \cdot Z = \min\left(X, \overline{Y}, Z\right)$		$M_5 = \overline{X} + Y + \overline{Z} = \max\left(\overline{X}, Y, \overline{Z}\right)$
1	1	0		$m_6 = X \cdot Y \cdot \overline{Z} = \min\left(X, Y, \overline{Z}\right)$		$M_6 = \overline{X} + \overline{Y} + Z = \max\left(\overline{X}, \overline{Y}, Z\right)$
1	1	1		$m_7 = X \cdot Y \cdot Z = \min\left(X, Y, Z\right)$		$M_7 = \overline{X} + \overline{Y} + \overline{Z} = \max\left(\overline{X}, \overline{Y}, \overline{Z}\right)$

Canonical Form (Standard SOP and POS Form)

Any Boolean function that is expressed as a sum of minterms or as a product of max terms is said to be in its “canonical form”.
It mainly involves in two Boolean terms**, “minterms”** and **“maxterms”.**

When the SOP form of a Boolean expression is in canonical form, then each of its product term is called ‘minterm’.

Similarly, when the POS form of a Boolean expression is in canonical form, then each of its sum term is called ‘maxterm’.

Conversion of SOP form to standard SOP form or Canonical SOP form

For getting the standard SOP form of the given non-standard SOP form, we will add all the variables in each product term which do not have all the variables. By using the Boolean algebraic law, (x + x' = 0) and by following the below steps we can easily convert the normal SOP function into standard SOP form.

E.g.

Convert the non-standard SOP function F = AB + A C + B C

F= AB+AC+BC
= A (C + C') + A (B + B') C + (A + A') B C
= A B C + A B C' + A B C + A B' C + A B C + A' B C
= **A B C + A B C' + A B' C + A' B C (standard/canonical form)**

Conversion of POS form to standard POS form or Canonical POS form

For getting the standard POS form of the given non-standard POS form, we will add all the variables in each product term that do not have all the variables.

Given the POS expression

F = (p' + q + r) * (q' + r + s') * (p + q' + r' + s)

Convert to standard/canonical POS

Hint: Add into each term the product of the missing variable and its complement

Answer:

F = (p' + q + r + s)* (p' + q + r + s')* (p + q' + r + s')* (p' + q' + r + s') * (p + q' + r' + s)

Conversion of SOP form to POS form

SOP function:

$$F(A,B,C) = \sum m\,(0, 2, 3, 5, 7) = A' B' C' + A B' C' + A B' C + ABC' + ABC$$

POS function:

$$F(A,B,C) = \Pi M\,(1, 4, 6) = (A + B + C') * (A' + B + C) * (A '+ B' + C)$$

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.

Rules in Boolean Algebra

Following are the important rules used in Boolean algebra.

•Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.

•Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as . Thus if B = 0 then = 1 and B = 1 then = 0.

•ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.

•Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

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.

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

Associative law

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

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

Distributive law

Distributive law states the following condition.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

AND law

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

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (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.

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

Boolean Theorems

Theorem No.	Theorem
1.1	$A + 0 = A$
1.2	$A \cdot 1 = A$
1.3	$A + 1 = 1$
1.4	$A \cdot 0 = 0$
1.5	$A + A = A$
1.6	$A \cdot A = A$
1.7	$A + \overline{A} = 1$
1.8	$A \cdot \overline{A} = 0$
1.9	$A \cdot (B + C) = AB + AC$
1.10	$A + BC = (A + B)(A + C)$
1.11	$A + AB = A$

Boolean Theorems revisited

Theorem No.	Theorem
1.12	$A\,(A + B) = A$
1.13	$A + \overline{A}B = (A + B)$
1.14	$A(\overline{A} + B) = AB$
1.15	$AB + A\overline{B} = A$
1.16	$(A + B) \cdot (A + \overline{B}) = A$
1.17	$AB + \overline{A}C = (A + C)\,(\overline{A} + B)$
1.18	$(A + B)\,(\overline{A} + C) = AC + \overline{A}B$
1.19	$AB + \overline{A}C + BC = AB + \overline{A}C$
1.20	$(A + B)\,(\overline{A} + C)\,(B + C) = (A + B)\,(\overline{A} + \mathrm{C})$
1.21	$\overline{A \cdot B \cdot C \cdot \ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots$
1.22	$\overline{A + B + C + \ldots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \ldots$

For example: To prove Theorem 1.10
A+BC=(A+B)(A+C)

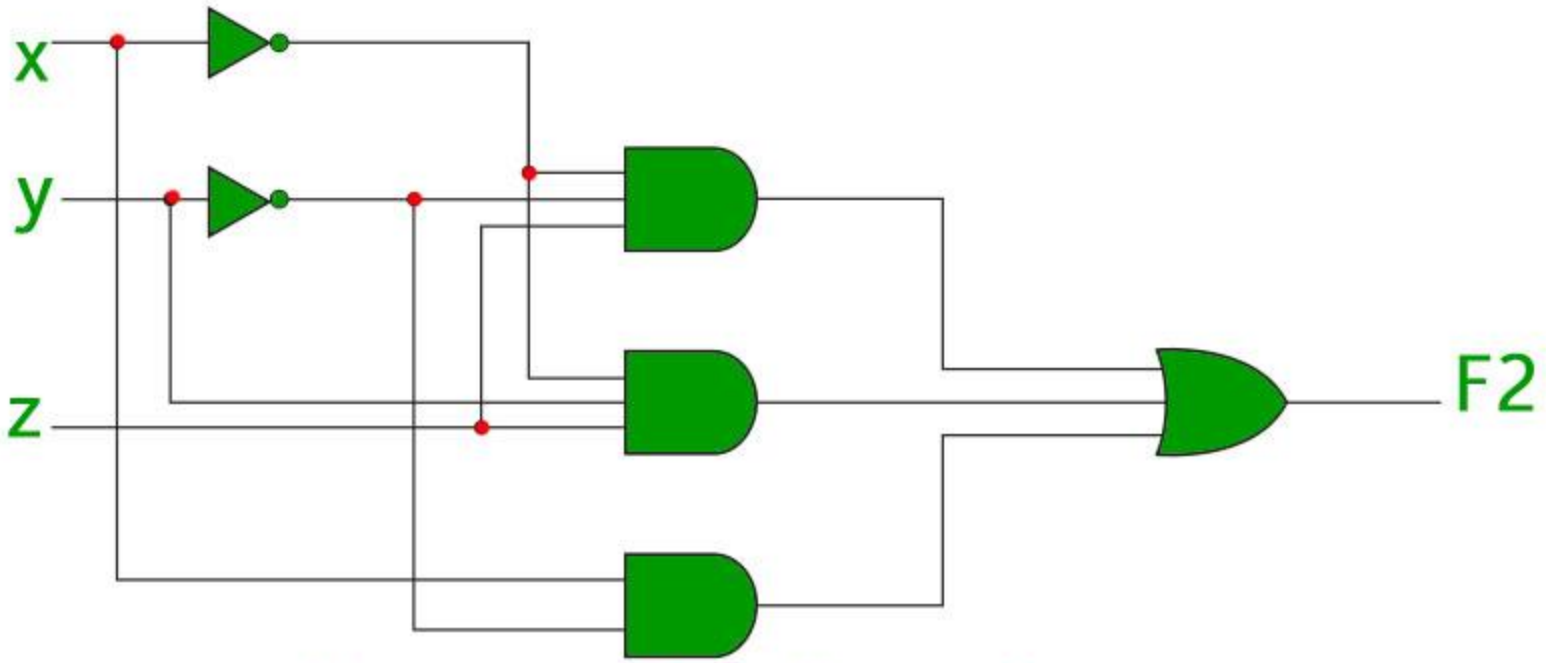
A	B	C	BC	$A + BC$	$A + B$	$A + C$	$(A + B) \cdot (A + C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Logic function minimization

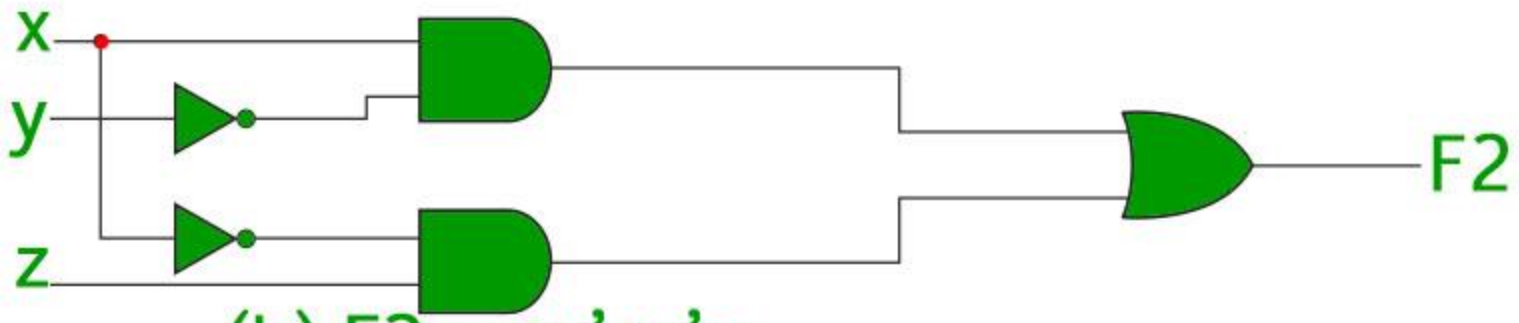
- Every boolean function can be expressed as a sum of minterms or a product of maxterms.

- Since the number of literals in such an expression is usually high, and the complexity of the digital logic gates that implement a Boolean function is directly related to the complexity of the algebraic expression from which the function is implemented, it is preferable to have the most simplified form of the algebraic expression. The process of simplifying the algebraic expression of a boolean function is called **minimization**.

- Minimization is important since it reduces the cost and complexity of the associated circuit.



(a) F2 = x'y'z+x'yz+xy'



(b) F2 = xy'+x'z

Example

Simplify the **Boolean function**

$F = AB + (AC)' + AB'C(AB + C)$

Solution.

$F = AB + (AC)' + AB'C(AB + C)$

$= AB + A' + C' + AB'C.AB + AB'C.C$

$= AB + A' + C' + 0 + AB'C$ $(B.B' = 0 \text{ and } C.C = C)$

$= ABC + ABC' + A' + C' + AB'C$ $(AB = AB(C + C') = ABC + ABC')$

$= AC(B + B') + C'(AB + 1) + A'$

$\mathbf{F = AC + C' + A'}$ $(B + B' = 1 \text{ and } AB + 1 = 1) = AC + (AC)' = 1$

Minimization using Karnaugh map(K map)

- The Karnaugh map provides a systematic method for simplification and manipulation of a Boolean expression.

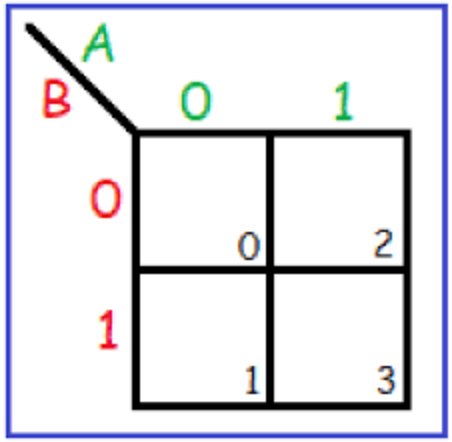
- The map is a diagram consisting of squares. For n variables on a Karnaugh map there are 2n numbers of squares. Each square or cell represents one of the minterms.

- Since any Boolean function can be expressed as a sum of minterms, it is possible to recognize a Boolean function graphically in the map from the area enclosed by those squares whose minterms appear in the function.

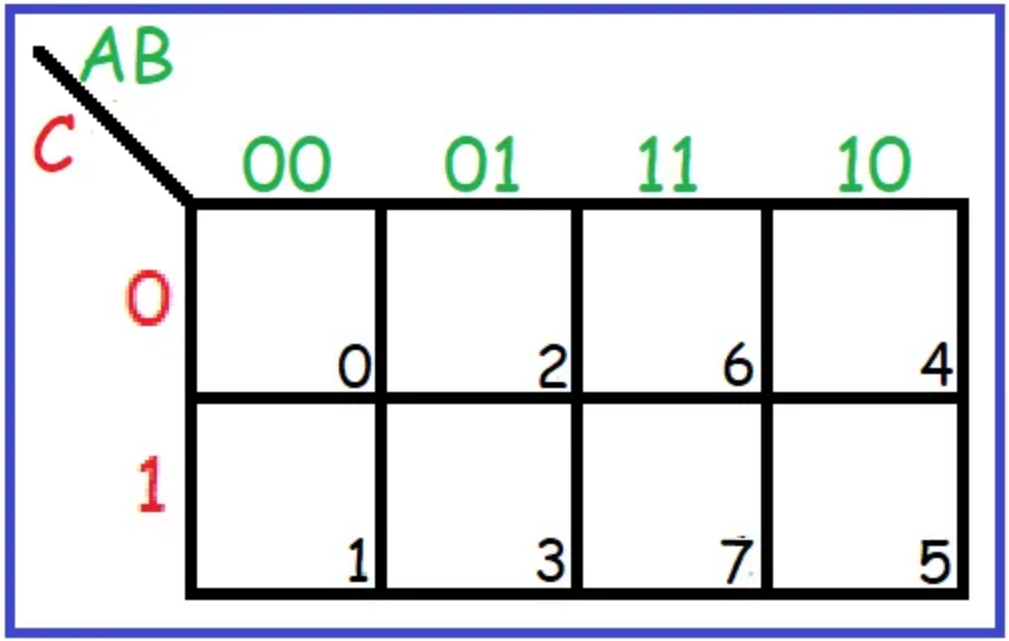
- It is also possible to derive alternative algebraic expressions or simplify the expression with a minimum number of variables or literals and sum of products or product of sums terms, by analyzing various patterns.

- In fact, the map represents a visual diagram of all possible ways a function can be expressed in a standard form and the simplest algebraic expression consisting of a sum of products or product of sums can be selected

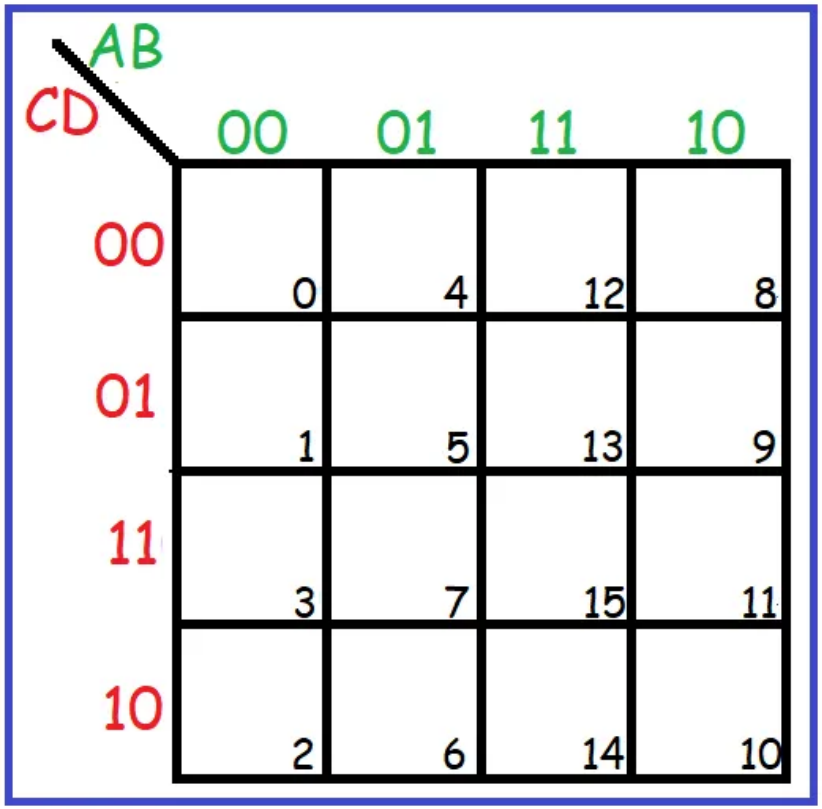
2 Variable K-map



3 variable K map

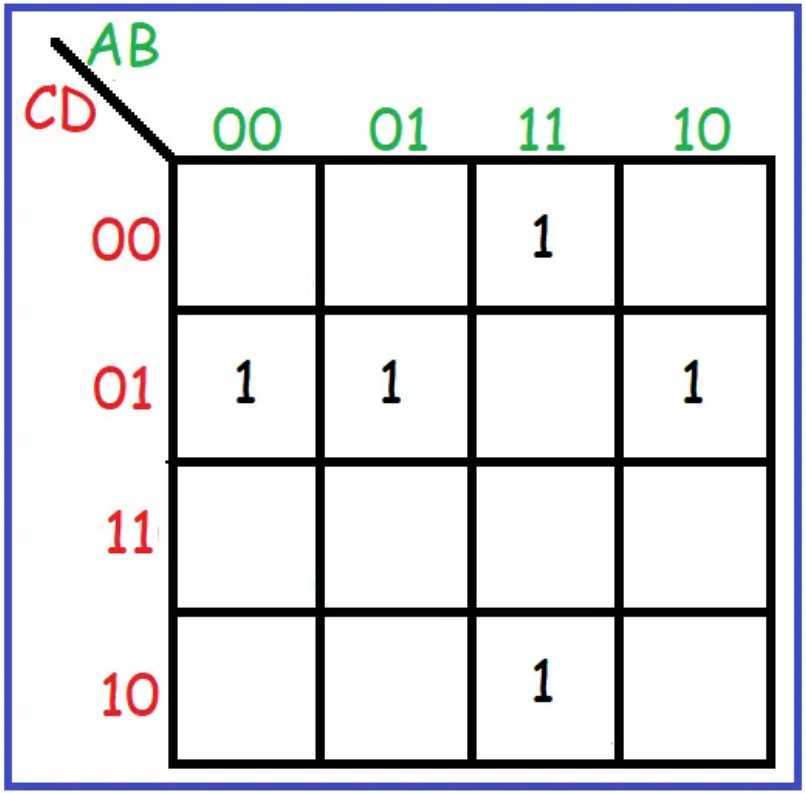


K map (continued)



Example

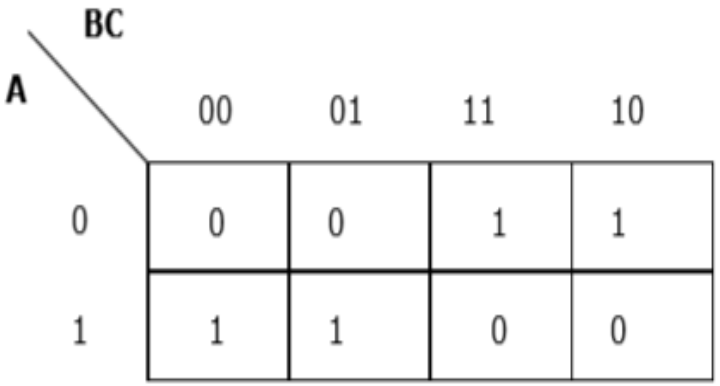
Y (A, B, C, D) = Σm(1, 5, 9, 12, 14)



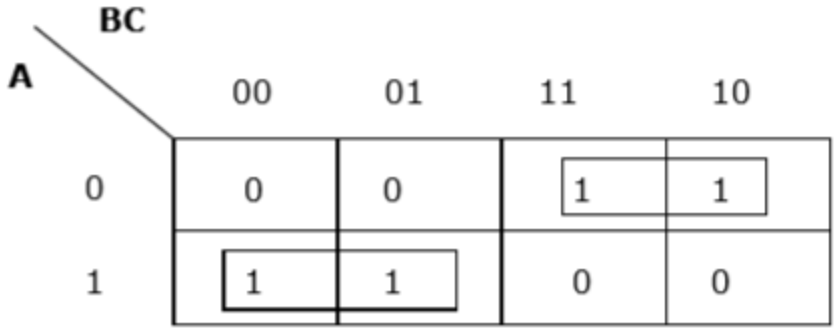
Minimize the following Boolean expression using K-map –

F(A,B,C)=A′BC+A′BC′+AB′C′+AB′C

Solution:



Now we will group the cells of 1 according to the rules



F(A,B,C)=A′B+AB′=A⊕B$F(A,B,C)=A'B+AB'=A \oplus B$.

Example:

Simplify the following Boolean expression using the 4-variable K-map.

$$f(A,B,C,D)=\sum m(2,3,6,7,8,10,13,15)$$

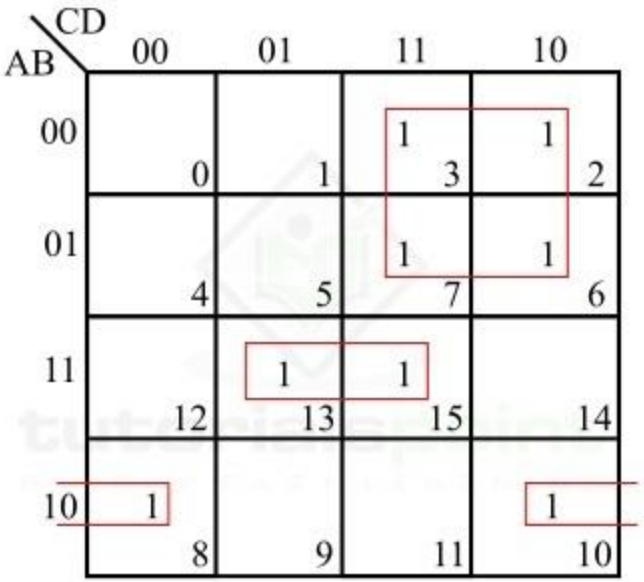


Figure 2 - SOP K-Map

So the simplified SOP expression is:

$$\mathbf{f(A,B,C,D)=\overline{A}C+A\overline{B}D+AB\overline{D}}$$