[Unit-1] Digital Logical Circuit

Digital Computers

A Digital computer can be considered as a digital system that performs various computational tasks.

The first electronic digital computer was developed in the late 1940s and was used primarily for numerical computations.

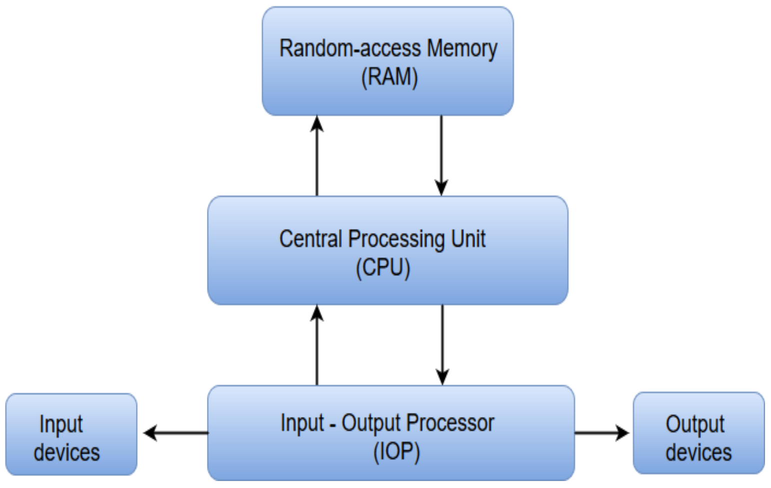
By convention, the digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit.

A computer system is subdivided into two functional entities: Hardware and Software.

The hardware consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.

The software of the computer consists of the instructions and data that the computer manipulates to perform various data-processing tasks.

Block diagram of a digital computer:

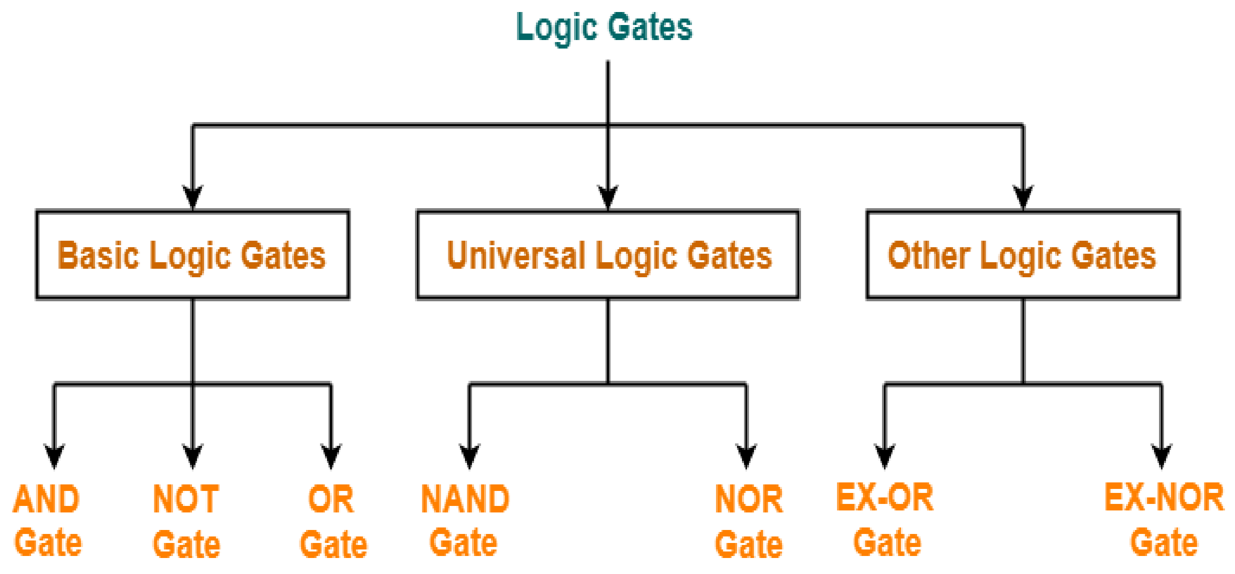

Random-access Memory (RAM)
Central Processing Unit (CPU)
Input devices
Input - Output Processor (IOP)
Output devices

- The Central Processing Unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and a control circuit for fetching and executing instructions.
- The memory unit of a digital computer contains storage for instructions and data.

- The Random Access Memory (RAM) for real-time processing of the data.
- The Input-Output devices for generating inputs from the user and displaying the final results to the user.
- The Input-Output devices connected to the computer include the keyboard, mouse, terminals, magnetic disk drives, and other communication devices.

Logic Gates

- The logic gates are the main structural part of a digital system.
- Logic Gates are a block of hardware that produces signals of binary 1 or 0 when input logic requirements are satisfied.
- Each gate has a distinct graphic symbol, and its operation can be described by means of algebraic expressions.
- The seven basic logic gates includes: AND, OR, XOR, NOT, NAND, NOR, and XNOR.
- The relationship between the input-output binary variables for each gate can be represented in tabular form by a truth table.
- Each gate has one or two binary input variables designated by A and B and one binary output variable designated by x.

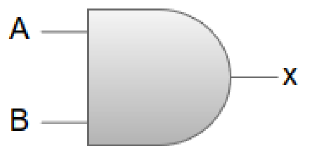


Types of Logic Gates

AND GATE:

The AND gate is an electronic circuit which gives a high output only if all its inputs are high. The AND operation is represented by a dot (.) sign.

AND Gate:



Algebraic Function: x = AB

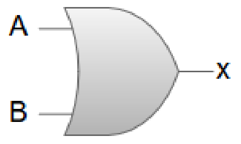
Truth Table:

A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:

The OR gate is an electronic circuit which gives a high output if one or more of its inputs are high. The operation performed by an OR gate is represented by a plus (+) sign.

OR Gate:



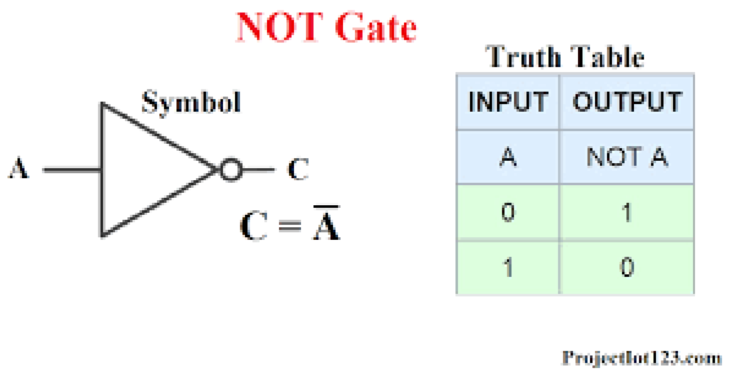
Algebraic Function: x = A + B

Truth Table:

A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:

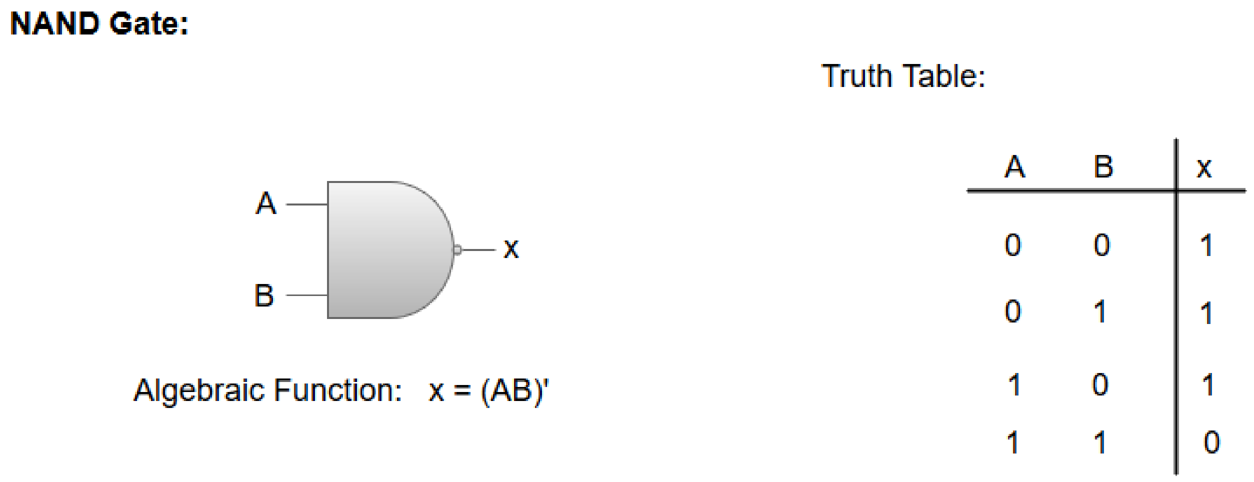
The NOT gate is an electronic circuit which produces an inverted version of the I nput at its output. It is also known as an **Inverter**.



INPUT	OUTPUT
A	NOT A
0	1
1	0

NAND GATE:

The NOT-AND (NAND) gate which is equal to an AND gate followed by a NOT gate. The NAND gate gives a high output if any of the inputs are low. The NAND gate is represented by a AND gate with a small circle on the output. The small circle represents inversion.



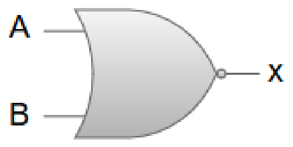
Algebraic Function: x = (AB)'

A	B	x
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE:

The NOT-OR (NOR) gate which is equal to an OR gate followed by a NOT gate. The NOR gate gives a low output if any of the inputs are high. The NOR gate is represented by an OR gate with a small circle on the output. The small circle represents inversion.

NOR Gate:



Algebraic Function: x = (A+B)'

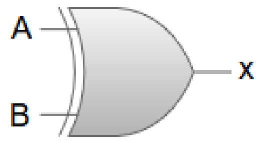
Truth Table:

A	B	x
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR/ XOR GATE:

The 'Exclusive-OR' gate is a circuit which will give a high output if one of its inputs is high but not both of them. The XOR operation is represented by an encircled plus sign.

XOR Gate:



Algebraic Function: $x = A \oplus B$

or

$x = A'B + AB'$

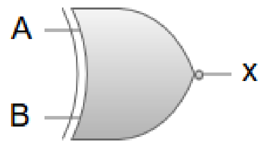
Truth Table:

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE-NOR/Equivalence GATE:

The 'Exclusive-NOR' gate is a circuit that does the inverse operation to the XOR gate. It will give a low output if one of its inputs is high but not both of them. The small circle represents inversion.

Exclusive-NOR Gate:



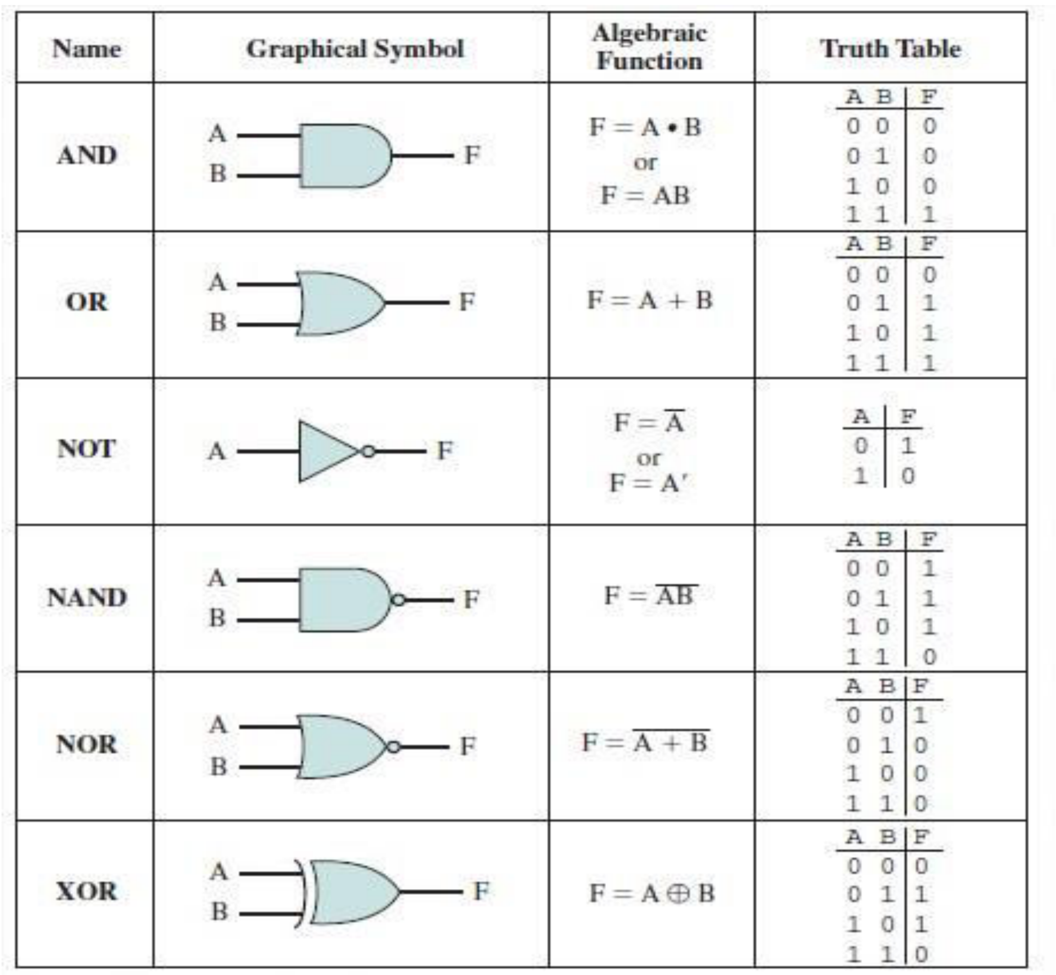
Algebraic Function: $x = (A \oplus B)'$

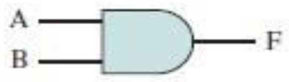
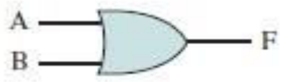
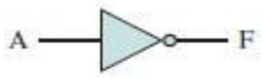
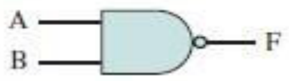
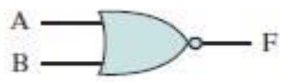
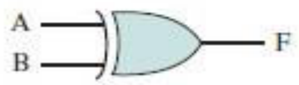
or

$x = A'B' + AB$

Truth Table:

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1



Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \bullet B$ or $F = AB$	A B | F; 0 0 | 0; 0 1 | 0; 1 0 | 0; 1 1 | 1
OR		$F = A + B$	A B | F; 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F; 0 | 1; 1 | 0
NAND		$F = \overline{AB}$	A B | F; 0 0 | 1; 0 1 | 1; 1 0 | 1; 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F; 0 0 | 1; 0 1 | 0; 1 0 | 0; 1 1 | 0
XOR		$F = A \oplus B$	A B | F; 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 0

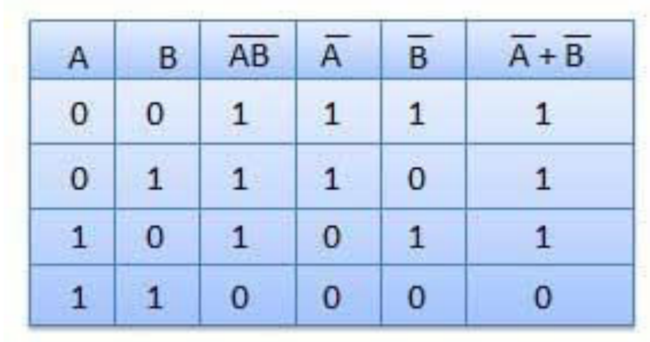
De Morgan's Theorems

Boolean e Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

Theorem 1

- The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.
- This OR gate is called as **Bubbled OR**.

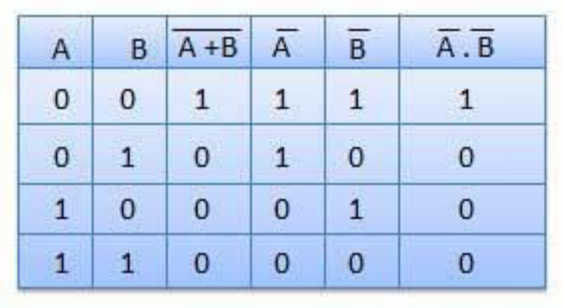
Table showing verification of the De Morgan's first theorem −



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

Theorem 2

- The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.
- This AND gate is called as **Bubbled AND**.



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

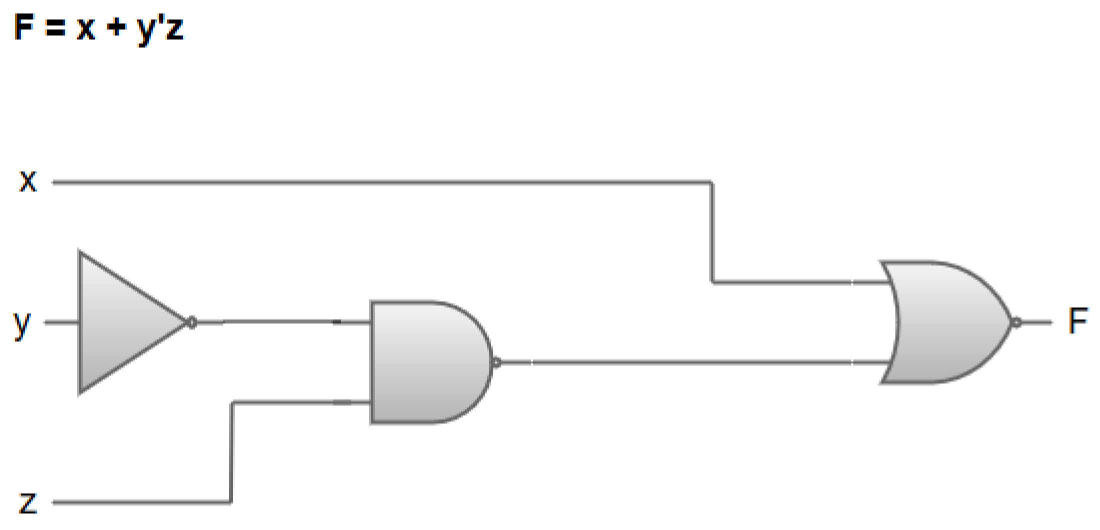
Boolean algebra

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:

$F = x + y'z$

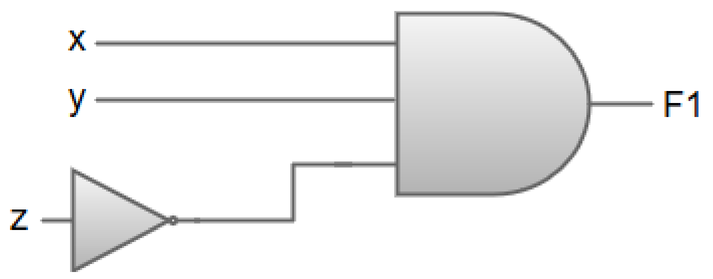
The logic diagram for the Boolean function $F = x + y'z$ can be represented as:



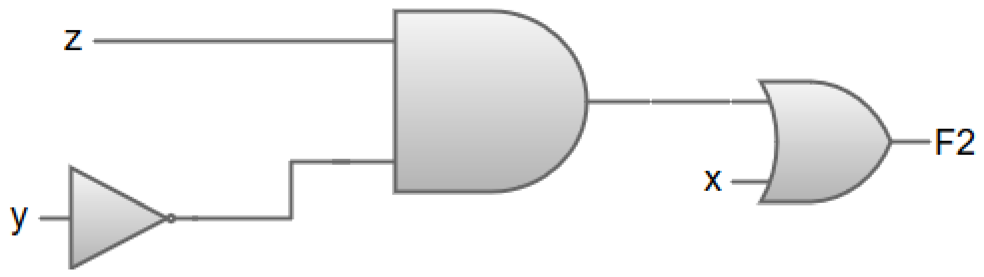
Examples of Boolean algebra simplifications using logic gates

In this section, we will look at some of the examples of Boolean algebra simplification using Logic gates.

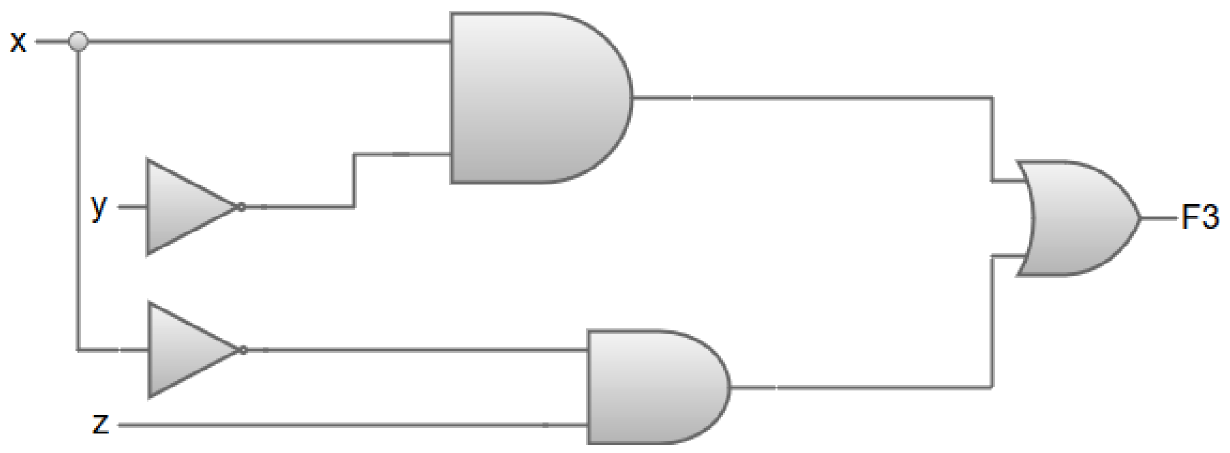
1. F1 = xyz'



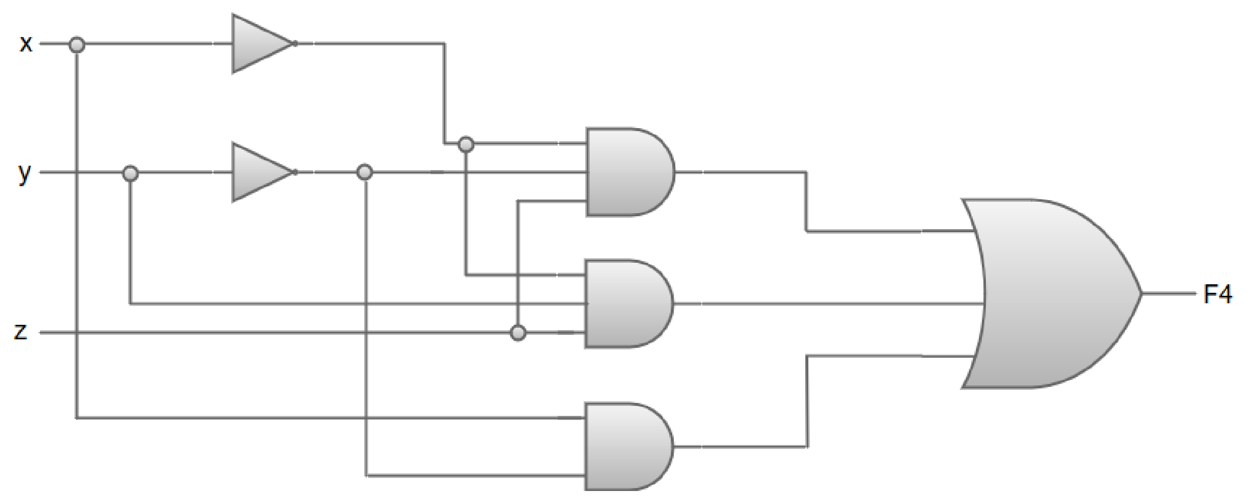
2. F2 = x + y'z



3. F3 = xy' + x'z



4. F4 = x'y'z + x'yz + xy'



→CANONICAL OR STANDARD FORMS

There are two ways to represent boolean function,one is standard (canonical) sum of products form (SOP), and another is the standard product of sums form (POS). These forms are important because functional comparison can be made using these forms and they are also useful to minize the boolean functions.

➢ 1. Sum of products form (SOP)

It is expression in which one or more product terms are logically added. In this kind of expression variables may or may not be complemented.

Ex.

Term X.Y.Z are the examples of product term,logical addition of these two product terms X.Y.Z. +X'Y'Z' are known as sum of product expression.

Ex1. F=A+BC

Ex2. F=ABC + A'BC + AB'C + ABC'

Ex3. F=XYZ + X'Y'Z' + XY'+ Z

Min terms

Sum of products function is also known as MIN terms. Each minterms obtained from an AND terms of the n variables, with each variable being primed if the corresponding bit of binary number is 0 and unprimed if a 1.

- 2. Products of Sum form (POS)

It is expression in which one or more product terms are logically multiplied. In this kind of expression variables may or may not be complemented.

Ex.

Term X+Y+Z and X'+Y'+Z' are sum terms, then logical multiplication of these two sum terms (X+Y+Z).(X'+Y'+Z') are known as product of sum expression.

Ex1. F=(A+B).C

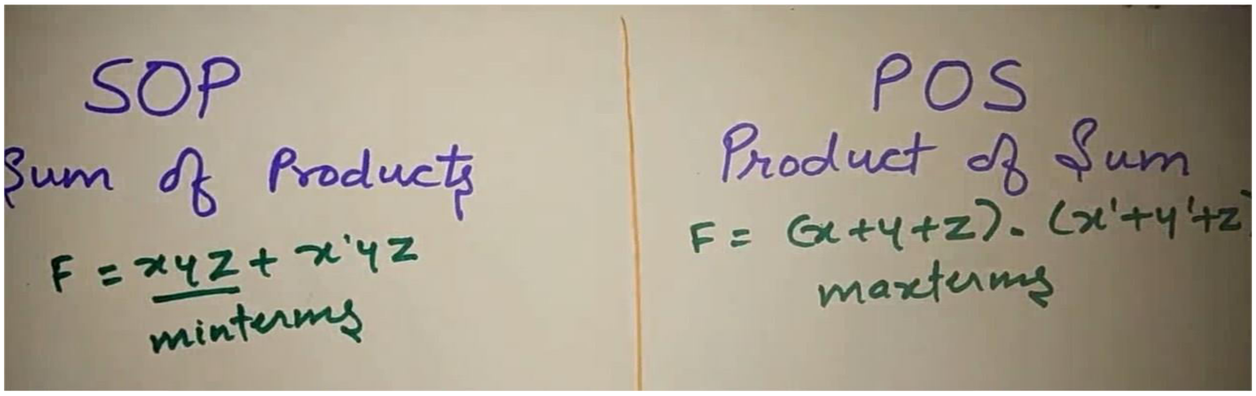
Ex2. F=(ABC).(A'BC) .(AB'C) . (ABC')

Ex3. F=(XYZ). (X'Y'Z') . (XY'). (Z)

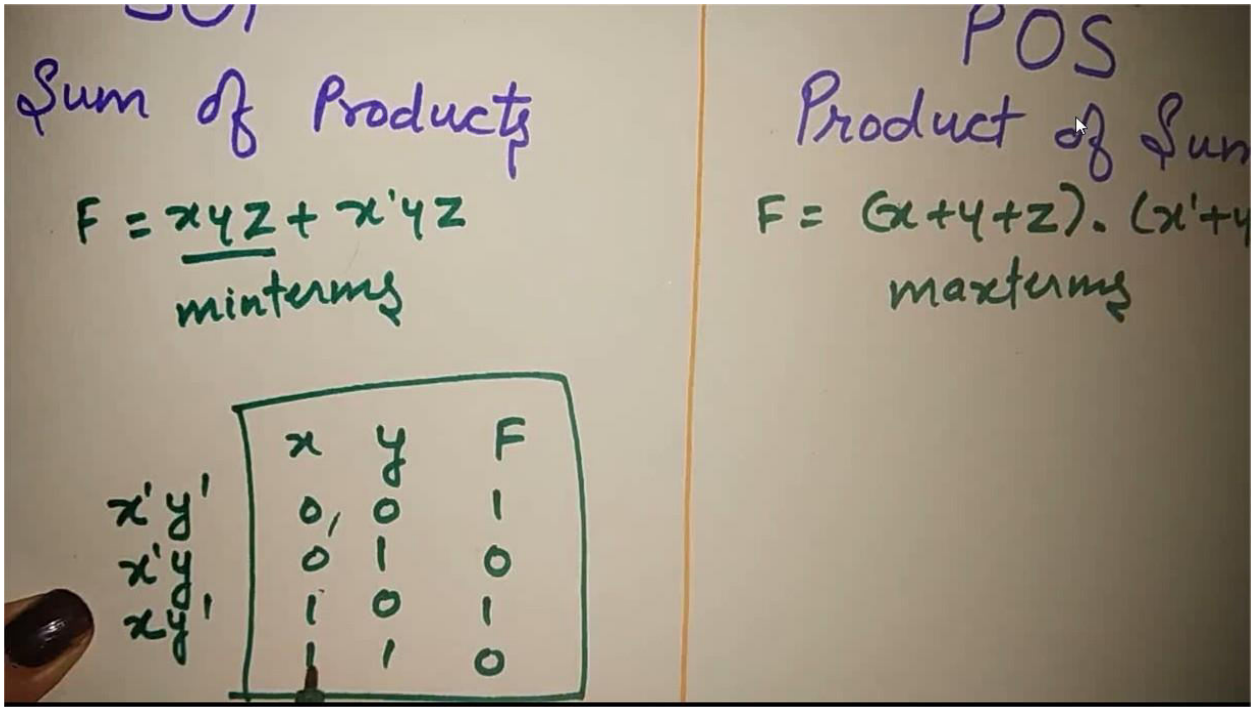
MAX terms

Products of Sum function is also known as MAX terms. Each maxterms obtained from an OR terms of the n variables, with each variable being primed if the corresponding bit of binary number is 1 and unprimed if a 0.

1.Min terms and Max Terms in SOP and POS.


SOP
Sum of Products
F = xyz + x'yz
minterms
POS
Product of Sum
F = (x+y+z). (x'+y'+z
maxterms

2. Truth table in SOP and POS. [Ex.(SOP) 0→X' and 1→X]



SOP

Sum of Products

$F = \underline{xyz} + x'yz$

minterms

POS

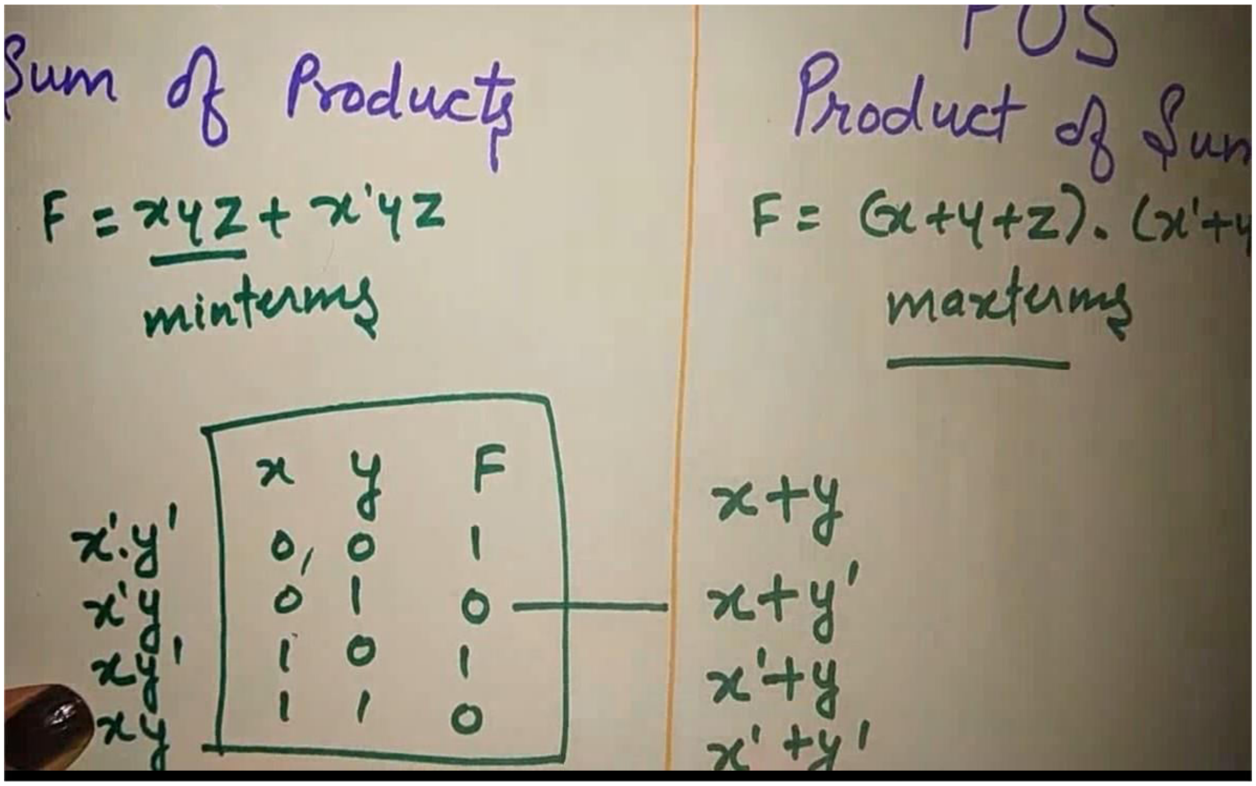
Product of Su

$F = (x+y+z).(x'+y$

maxterms

	x	y	F
x'y'	0	0	1
x'y	0	1	0
xy'	1	0	1
	1	1	0

3.SOP →0 is a compliment and 1 is don't. POS→1 is a compliment and 0 is don't.



Sum of Products

$F = \underline{xyz} + x'yz$

minterms

POS

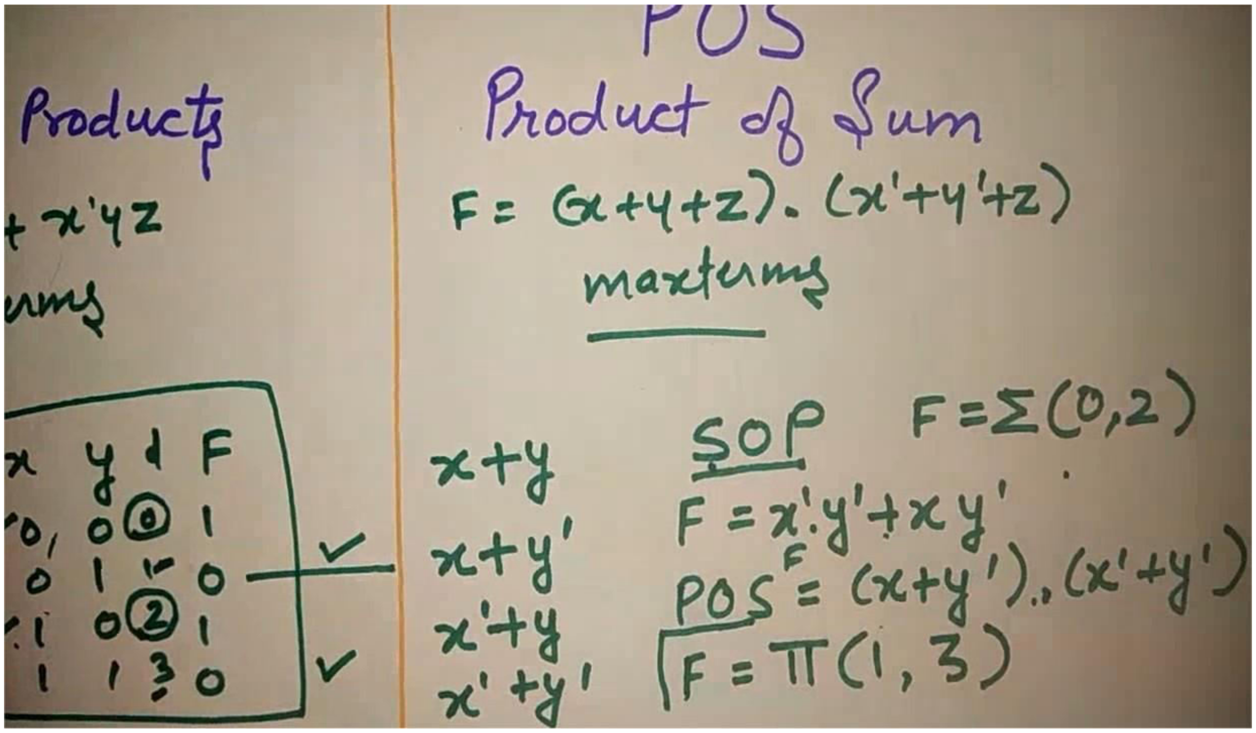
Product of Su

$F = (x+y+z).(x'+y$

maxterms

	x	y	F	
x.y'	0	0	1	x+y
x'y	0	1	0	x+y'
xy'	1	0	1	x'+y
xy	1	1	0	x'+y'

4.SOP is symbol Σ and POS are π symbol to complete statement.



5.Video of SOP and POS.(only show)

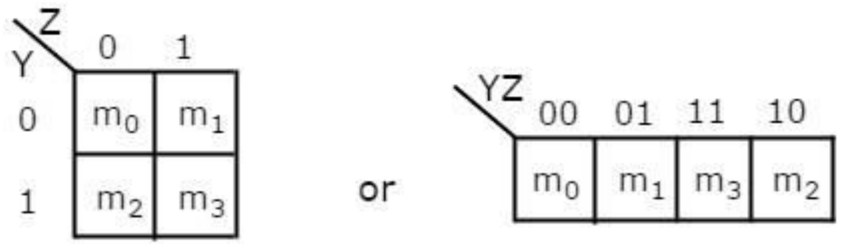
[https://www.youtube.com/watch?v=EYNyAy6-sq8]

→ KARNAUGH MAP (K-MAP)

- The booolean function may be simplified using algebra method, but this method is sometime difficult because it lacks specific rules for predicting each succeeding step.
- The map method provides a simple and straight forward procedure for minimizing boolean functions.
- This method is regarded either as a pictorial form of a truth table or as an extension of venn diagram.
- This method is known as a karnaugh map (K-MAP) or as Veitch Diagram.
- Two, three and four variable k-map are specification.

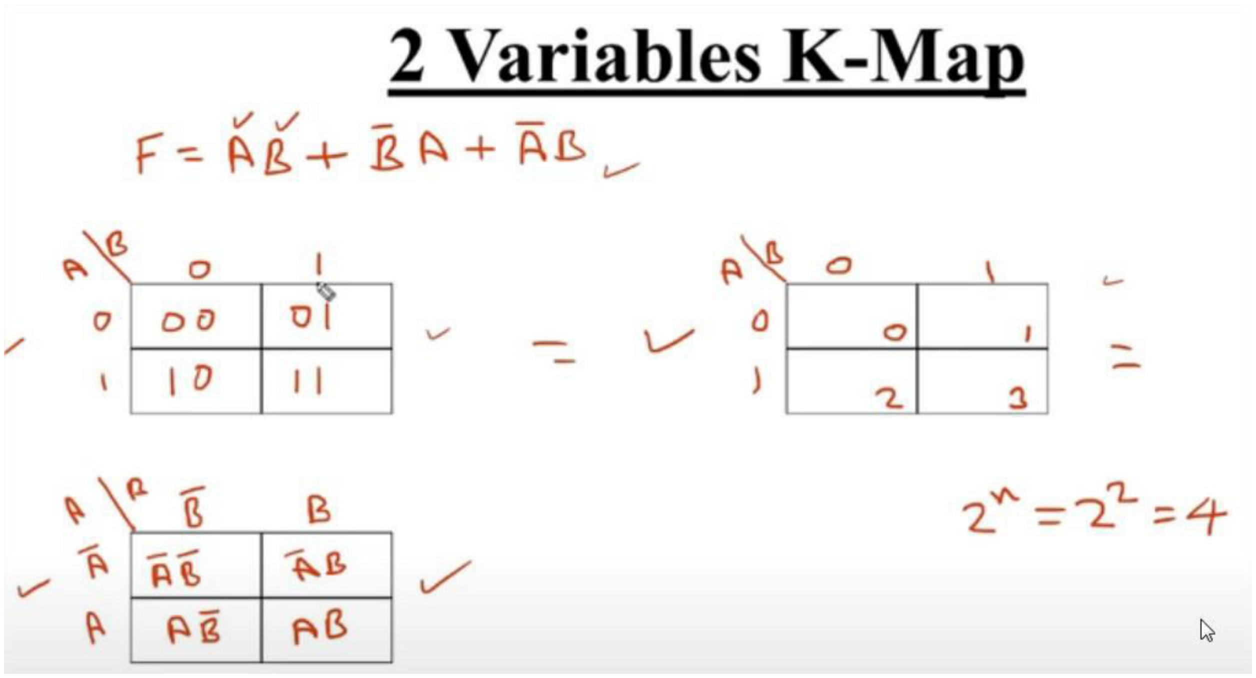
1.K-MAP for 2 variable

The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



As shown in the first row is for a A' and the second row for A. similarly ,the first column is for B' and the second column for B.

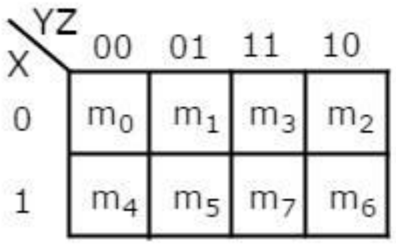
Example :→ F= AB + B'A + A'B



Show the video in 2 variable K-MAP.
(https://www.youtube.com/watch?v=_shQtM2MGD0)

2. K-MAP for 3 variable

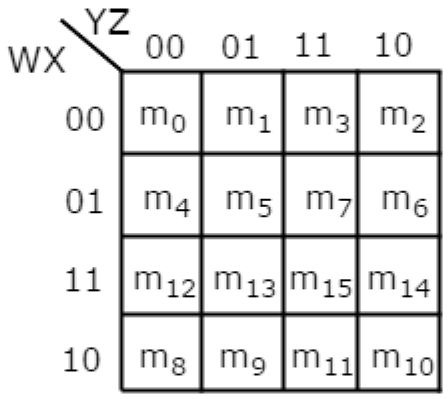
The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.



There is only one possibility of grouping 8 adjacent min terms.

3. K-MAP for 4 variable

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.



There is only one possibility of grouping 16 adjacent min terms.

Binary Table

Sr.no.	BINARY	OCTAL	DECIMAL	HEXA

				DECIMAL
0	0000	0	0	0
1	0001	1	1	1
2	0010	2	2	2
3	0011	3	3	3
4	0100	4	4	4
5	0101	5	5	5
6	0110	6	6	6
7	0111	7	7	7
8	1000		8	8
9	1001			9
10	1010			A
11	1011			B
12	1100			C
13	1101			D
14	1110			E
15	1111			F
16	0000			
17	0001			
18	0010			
19	0011			
20	0100			
21	0101			
22	0110			
23	0111			
24	1000			
25	1001			
26	1010			
27	1011			
28	1100			
29	1101			
30	1110			
31	1111			

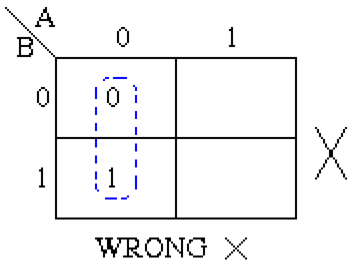
Rule for k-map

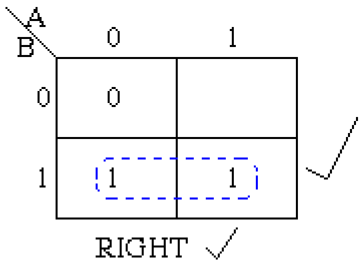
1) Groups may not include any cell containing a zero.(Only 1)
2) Groups may be horizontal or vertical, but not diagonal.

3) Groups must contain 1, 2, 4, 8, or in general 2^n cells. ...
4) Each group should be as large as possible.
5) Each cell containing a one must be in at least one group.
6) Groups may overlap.
7) Groups may wrap around the table.

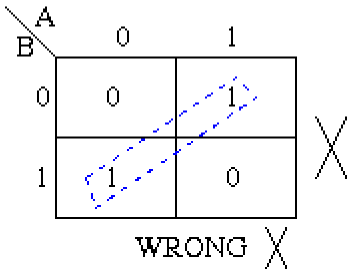
The Karnaugh map uses the following rules for the simplification of expressions by *grouping* together adjacent cells containing *ones*

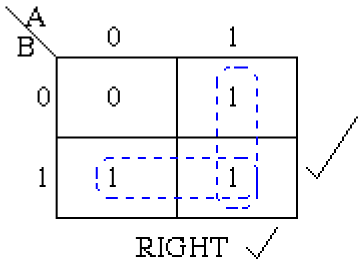
1. **Groups may not include any cell containing a zero**





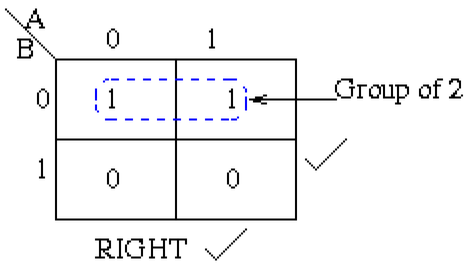
2. **Groups may be horizontal or vertical, but not diagonal.**

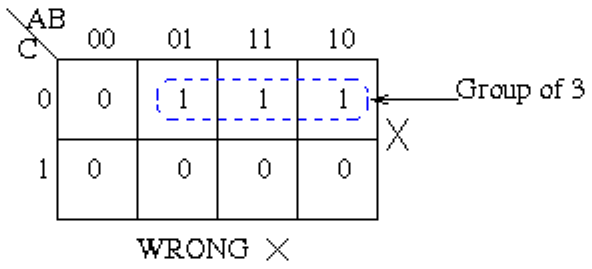


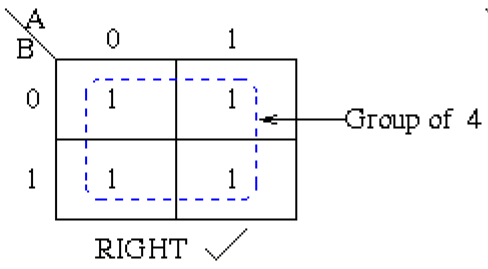


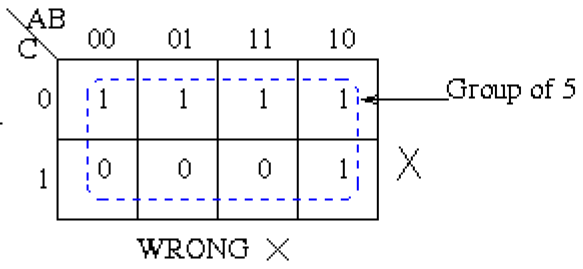
3. **Groups must contain 1, 2, 4, 8, or in general 2^n cells.**
   That is if n = 1, a group will contain two 1's since $2^1 = 2$.

If n = 2, a group will contain four 1's since $2^2 = 4$.

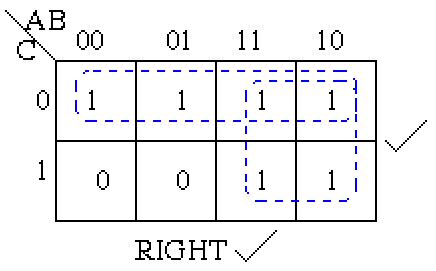


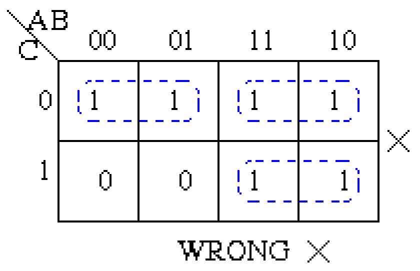






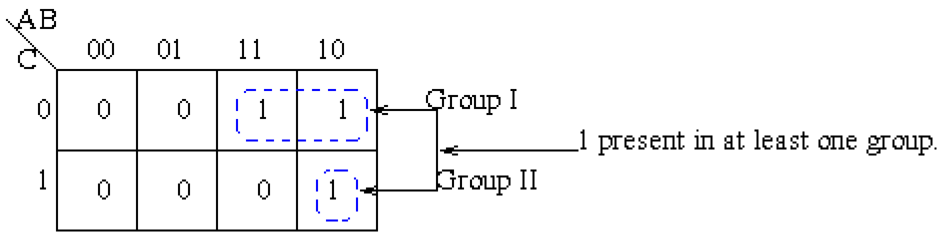
4. **Each group should be as large as possible.**



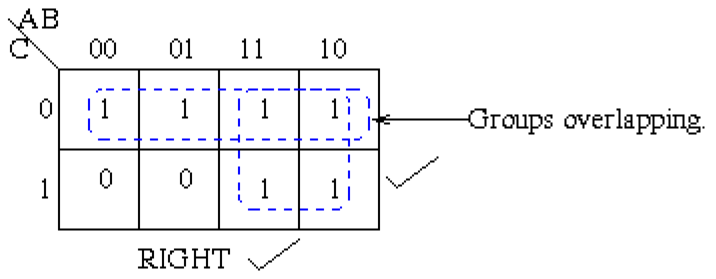


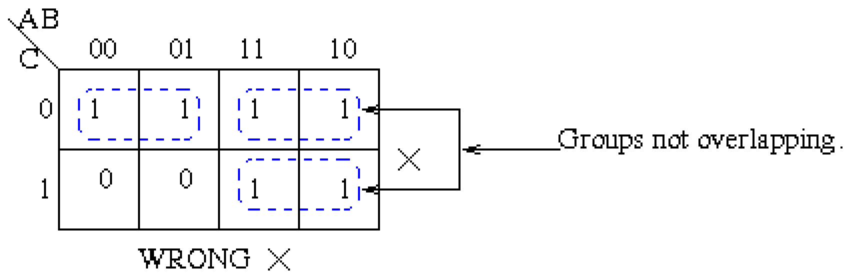
(Note that no Boolean laws broken, but not sufficiently minimal)

5. **Each cell containing a *one* must be in at least one group.**

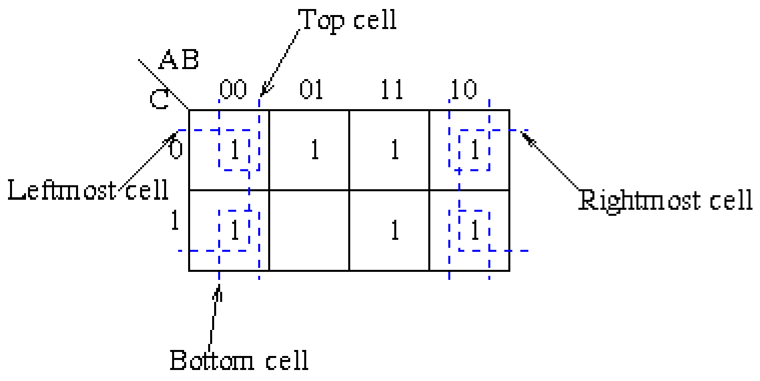


6. **Groups may overlap.**

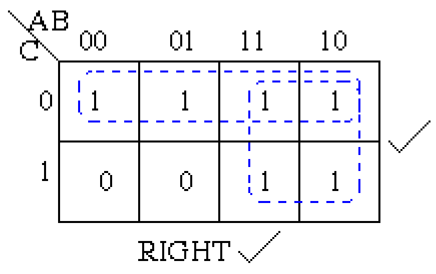


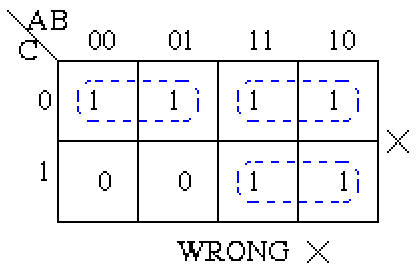


7. **Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.**



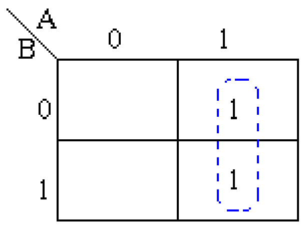
8. **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**





Example 1:

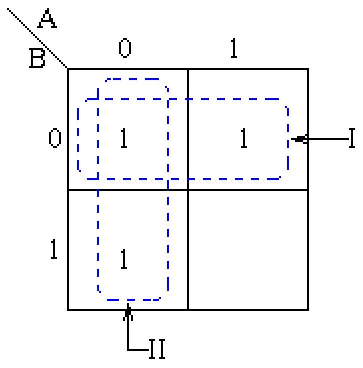
Consider the following map. The function plotted is: $Z = f(A,B) = A\bar{B} + AB$



Ans. Draw the circuit

Example 2:

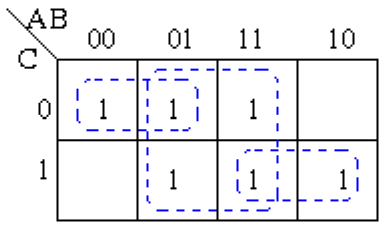
Consider the expression $Z = f(A,B) = \bar{A}\bar{B} + A\bar{B} + \bar{A}B$ plotted on the Karnaugh map:



Ans. Draw the circuit

Example 3:

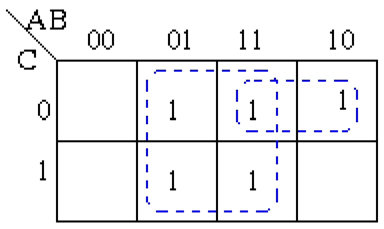
$Z = f(A,B,C) = \bar{A}\bar{B}\bar{C} + \bar{A}B + AB\bar{C} + AC$



Ans. Draw the circuit

Example 4:

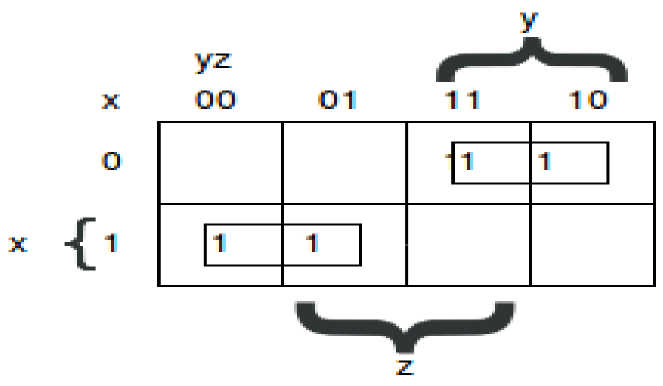
$Z = f(A,B,C) = \bar{A}B + B\bar{C} + BC + A\bar{B}\bar{C}$



WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

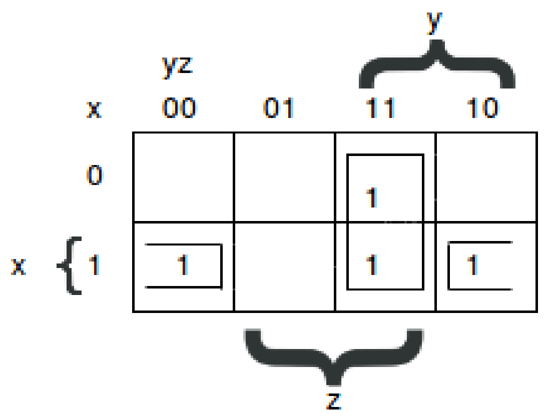
Ans. Draw the circuit

Example 5: F(x,y,z) = Σ (2,3,4,5)



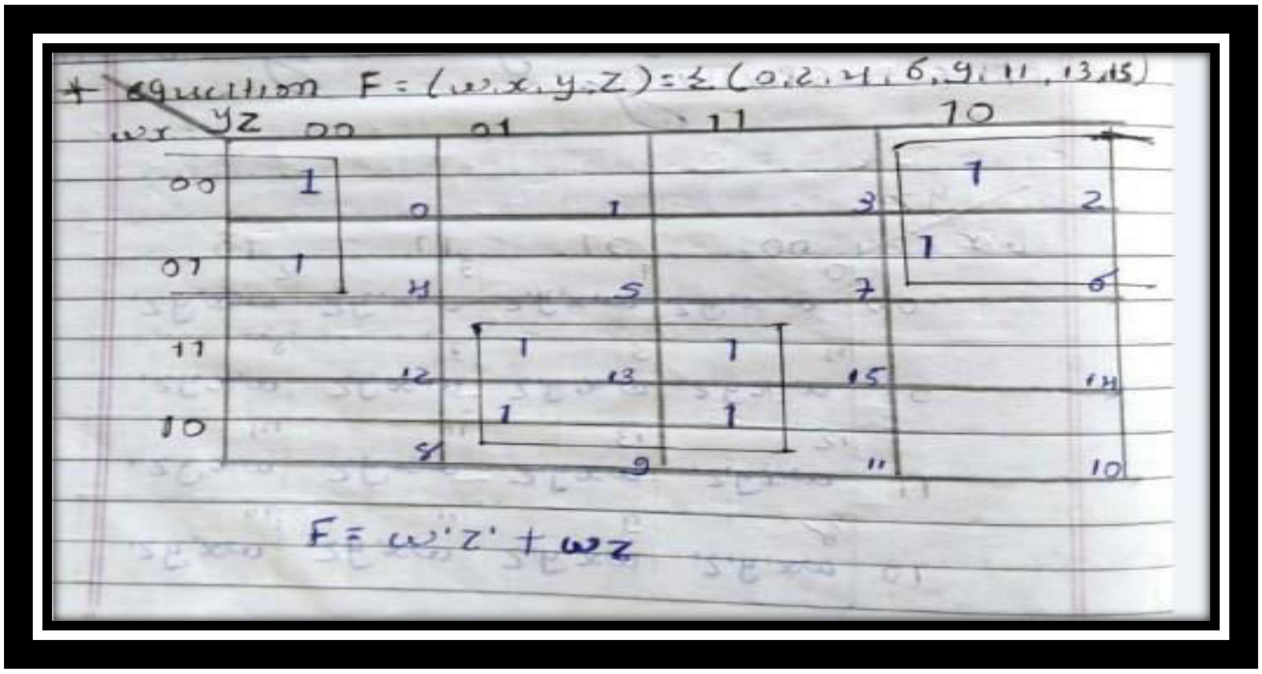
Σ(2,3,4,5)= x'y+xy'

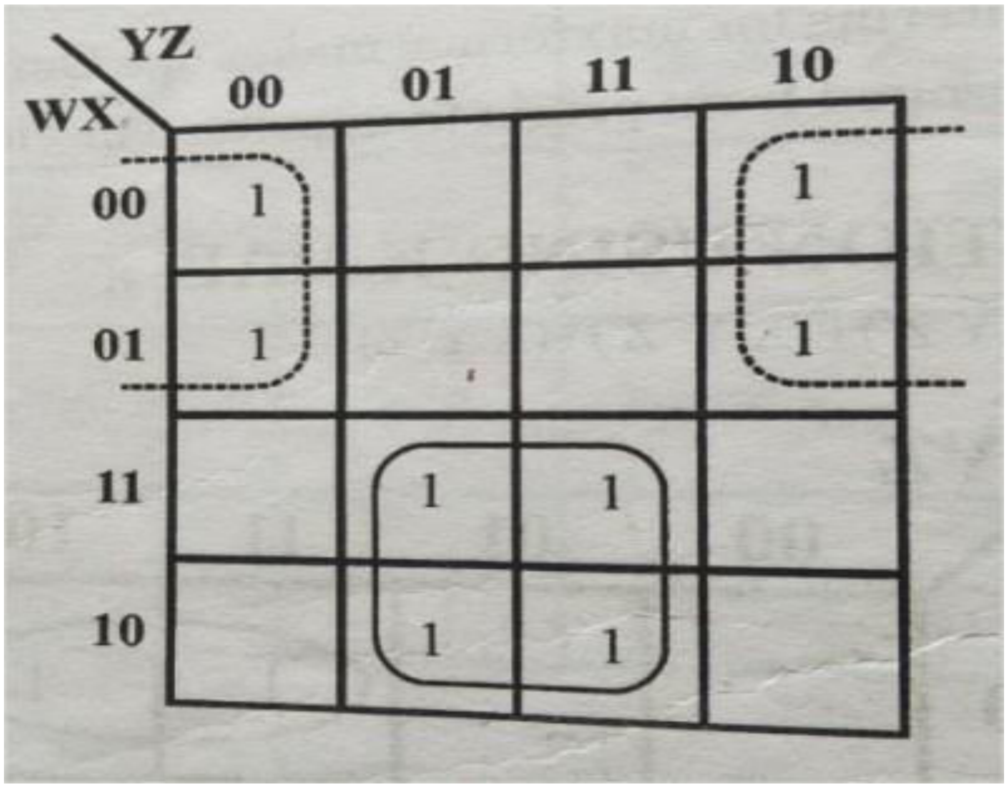
Example 6: F(x,y,z) = Σ(3,4,6,7)



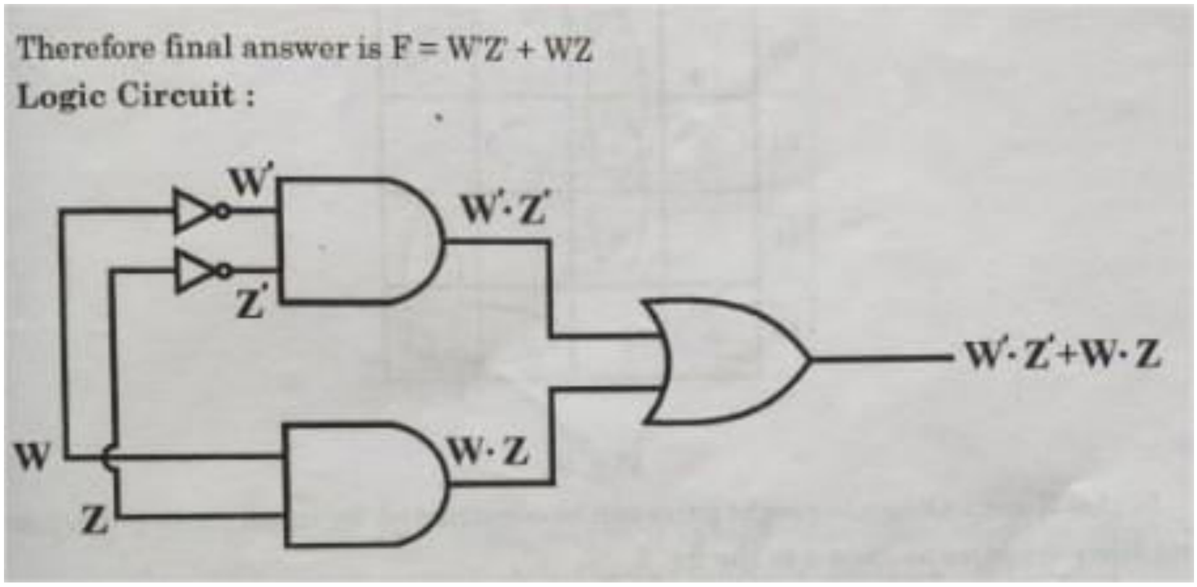
$\Sigma(3,4,6,7) = yz + xz'$

Example 7: Simplified the equation F(W,X,Y,Z)= Σ (0,2,4,6,9,11,13,15) using k-map and draw the logic circuit for the result that obtained from the k-map.





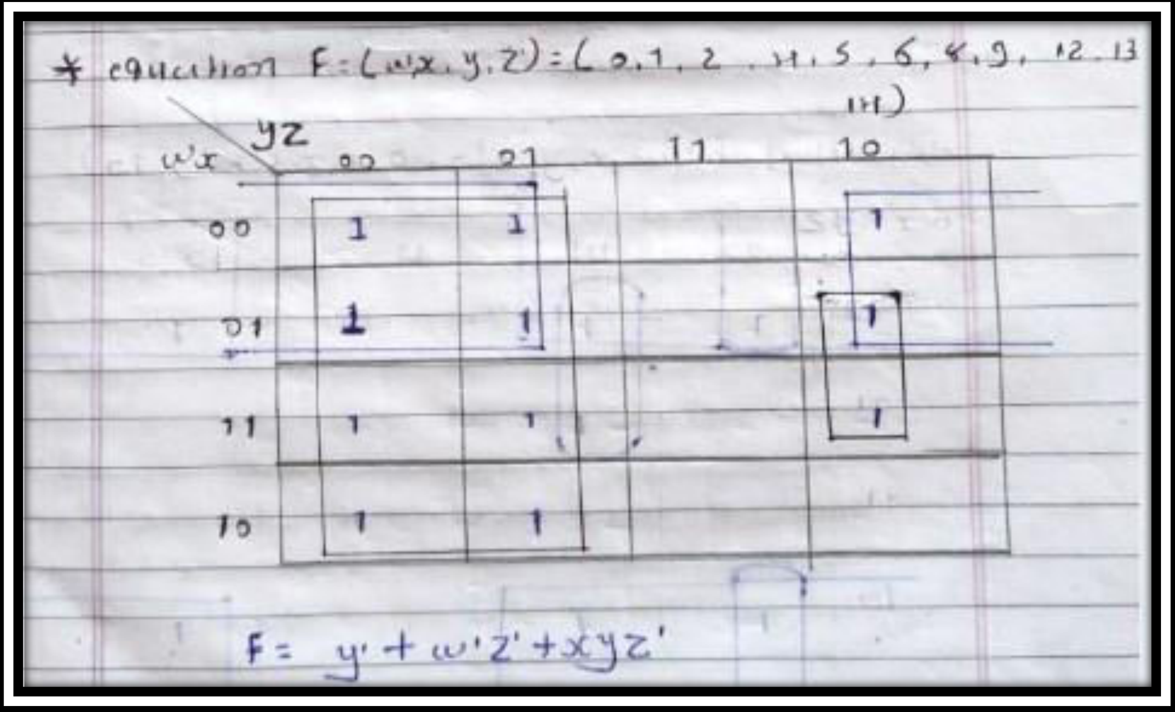
Circuit→ F=W'Z' + WZ



Therefore final answer is F = W'Z' + WZ

Logic Circuit :

Example 8: **Simplified the equation F(W,X,Y,Z)= Σ (0,1,2,4,5,6,8,9,12,13,14) using k-map and draw the logic circuit for the result that obtained from the k-map.**

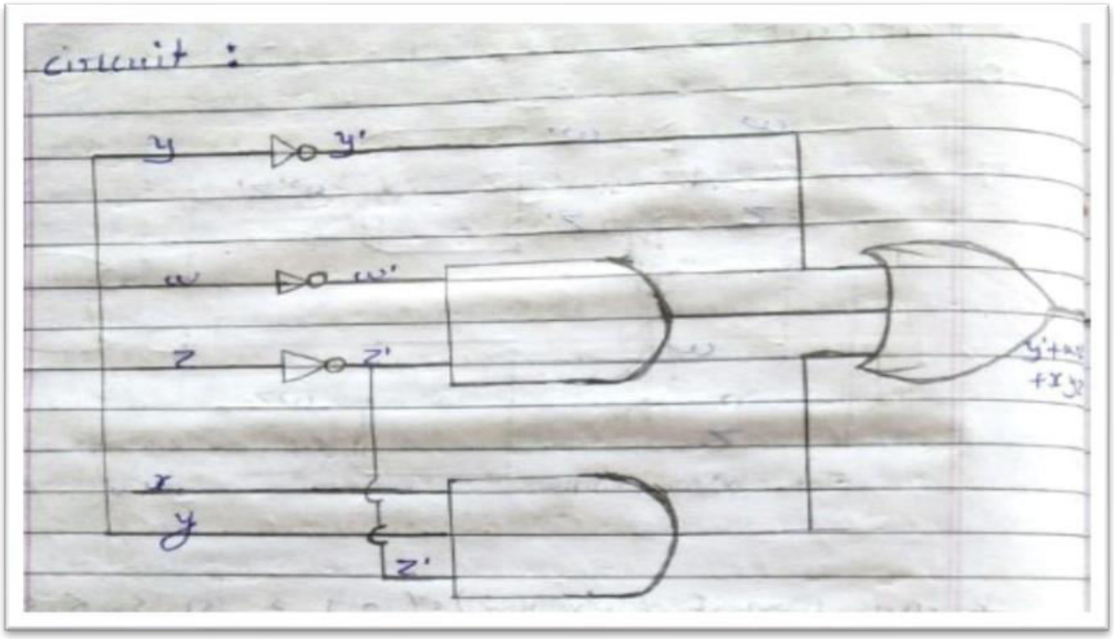


* equation F:(w,x,y,z)=(0,1,2,4,5,6,8,9,12,13,14)

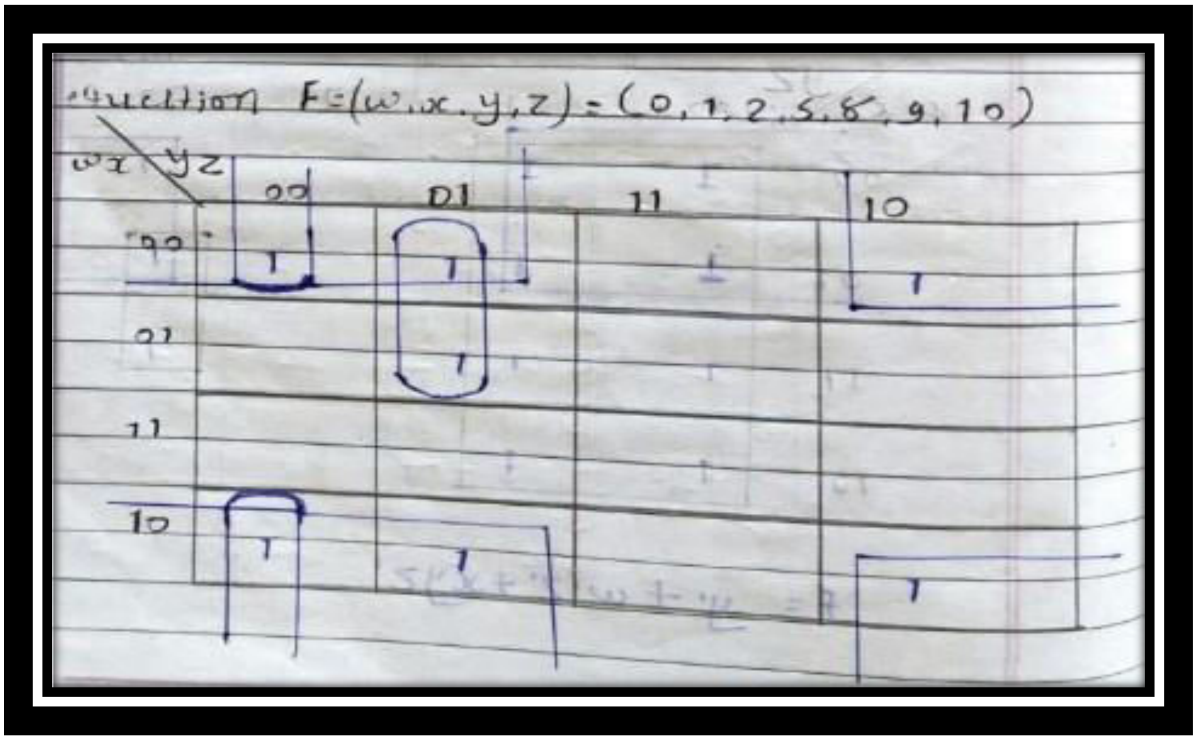
wx \ yz	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

F = y' + w'z' + xyz'

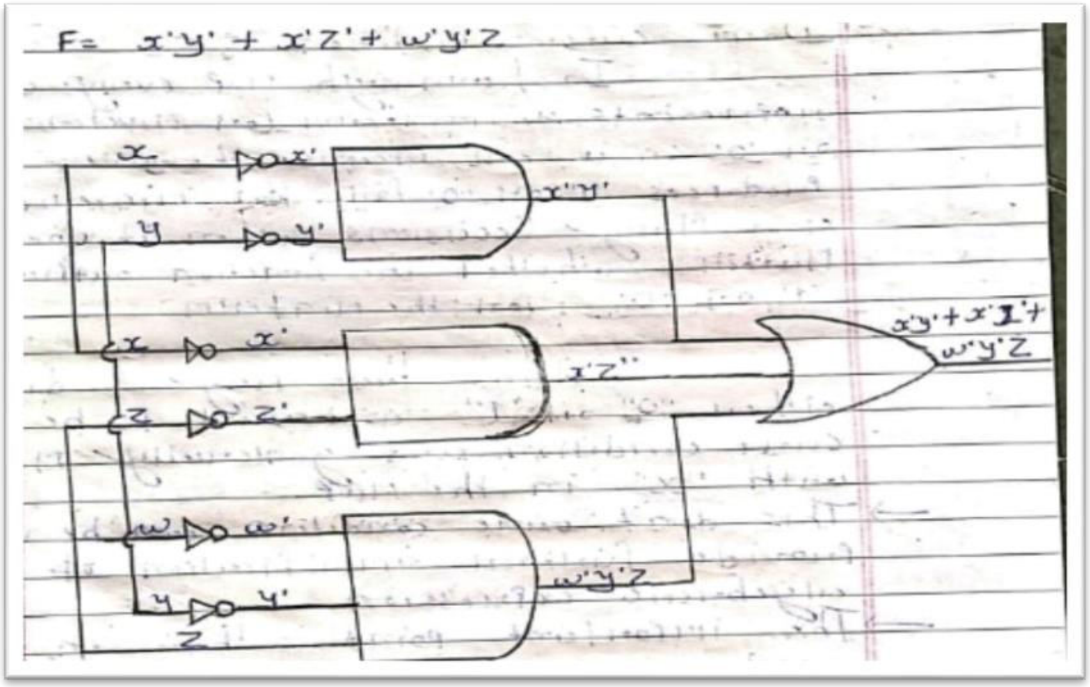
<u>Circuit</u>→ F=Y' + W'Z' + XYZ'



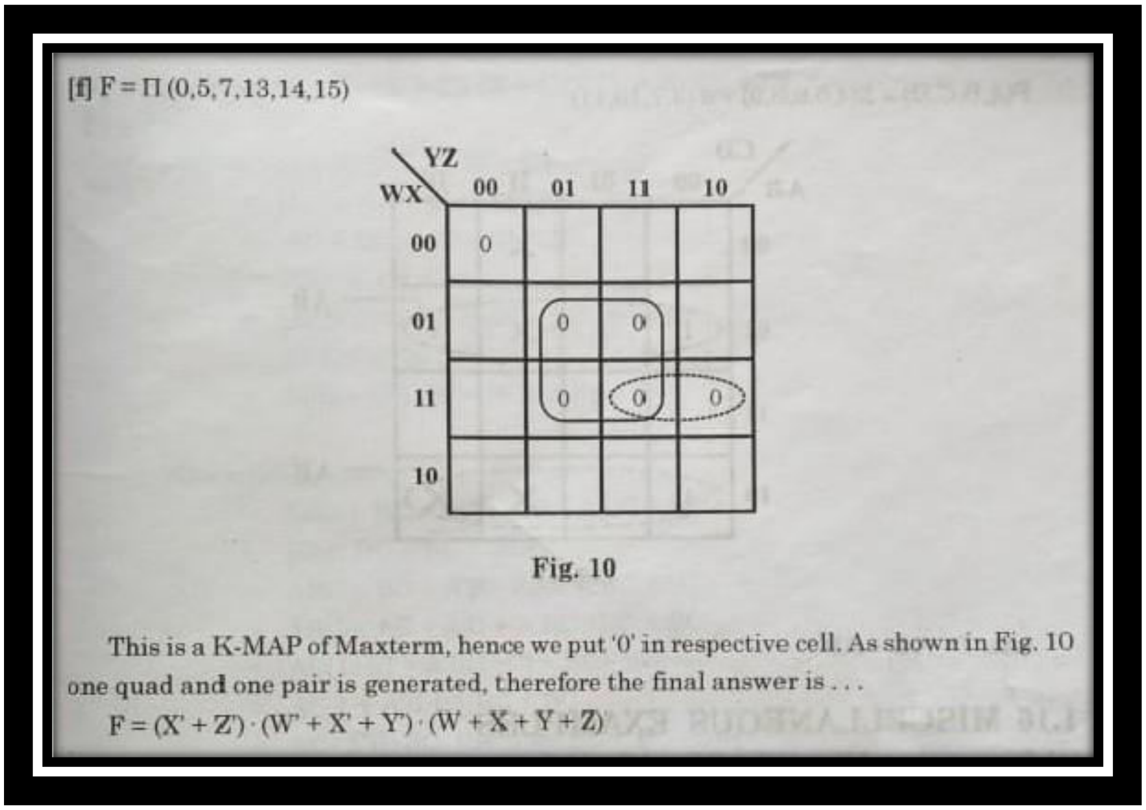
Example 9: Simplified the equation F(W,X,Y,Z)= Σ (0,1,2,4,5,8,9,10) using k-map and draw the logic circuit for the result that obtained from the k-map.



Circuit→ F=X'Y' + X'Z' + W'Y'Z



Example 10: **F=π (0,5,7,13,14,15)**



[f] F = Π (0,5,7,13,14,15)

WX \ YZ	00	01	11	10
00	0			
01		0	0	
11		0	0	0
10				

Fig. 10

This is a K-MAP of Maxterm, hence we put '0' in respective cell. As shown in Fig. 10 one quad and one pair is generated, therefore the final answer is . . .

$F = (X' + Z') \cdot (W' + X' + Y') \cdot (W + X + Y + Z)$

Don't care Condition:

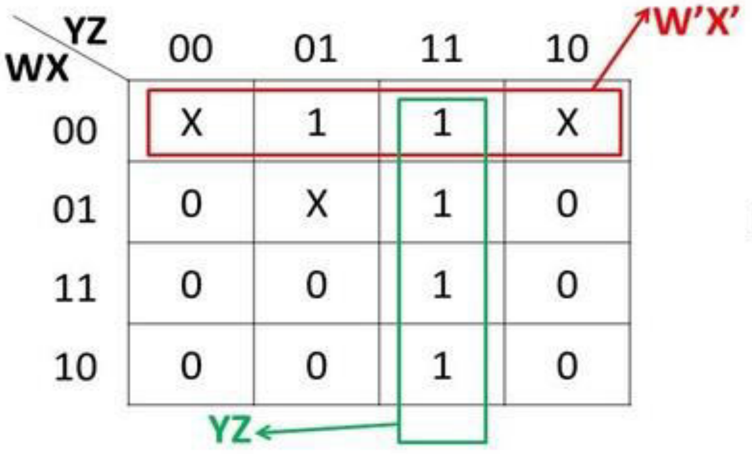
- In k-map every cell represents a minterm (or maxterm).
- The "Don't care" condition says that we can use the blank cells of a K-map to make a group of the variables.
- To make a group of cells, we can use the "don't care" cells as either 0 or 1, and if required, we can also ignore that cell.
- We mainly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

Show the video in don't care condition in K-MAP

[https://www.youtube.com/watch?v=OzBDlyW_B2o]

Don’t Care Conditions

- Simplify the Boolean function *F (w, x, y, z) = ∑(1, 3, 7, 11, 15)* which has the don’t-care conditions: *d (w, x, y, z) = (0, 2, 5)*

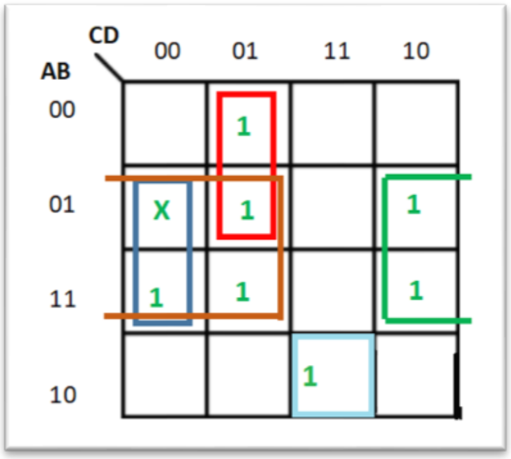


WX \ YZ	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

F= W’X’ + YZ

Example-1:
Minimise the following function in SOP minimal form using K-Maps:

f = m(1, 5, 6, 11, 12, 13, 14) + d(4)

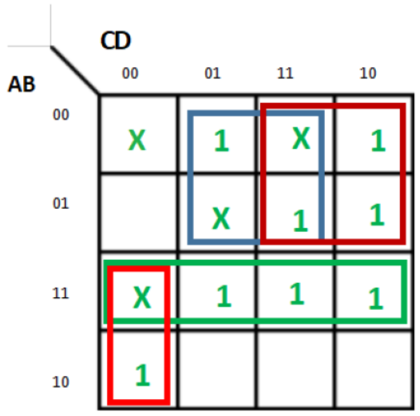


AB \ CD	00	01	11	10
00		1		
01	X	1		1
11	1	1		1
10			1	

Therefore, SOP minimal is,

f = BC' + BCD' + A'C'D + AB'CD

Example-2:
Minimise the following function in SOP minimal form using K-Maps:
F(A, B, C, D) = m(1, 2, 6, 7, 8, 13, 14, 15) + d(0, 3, 5, 12)



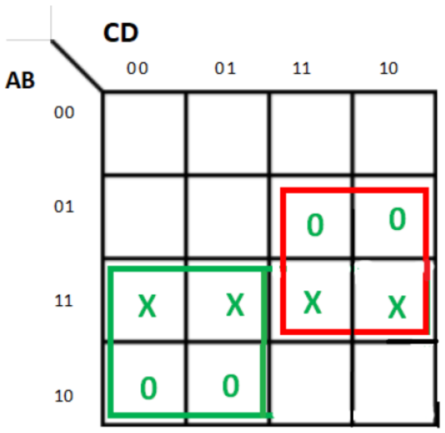
Therefore, SOP minimal is,
f = AC'D' + A'D + A'C + AB

Example-3:
Minimise the following function in POS minimal form using K-Maps:

F(A, B, C, D) = m(0, 1, 2, 3, 4, 5) + d(10, 11, 12, 13, 14, 15)

Explanation:
Writing the given expression in POS form:

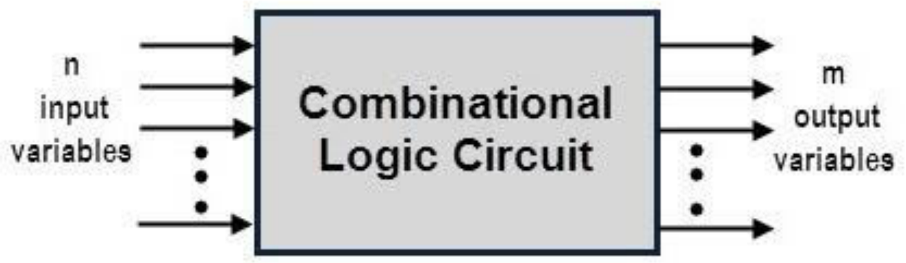
F(A, B, C, D) = M(6, 7, 8, 9) + d(12, 13, 14, 15)

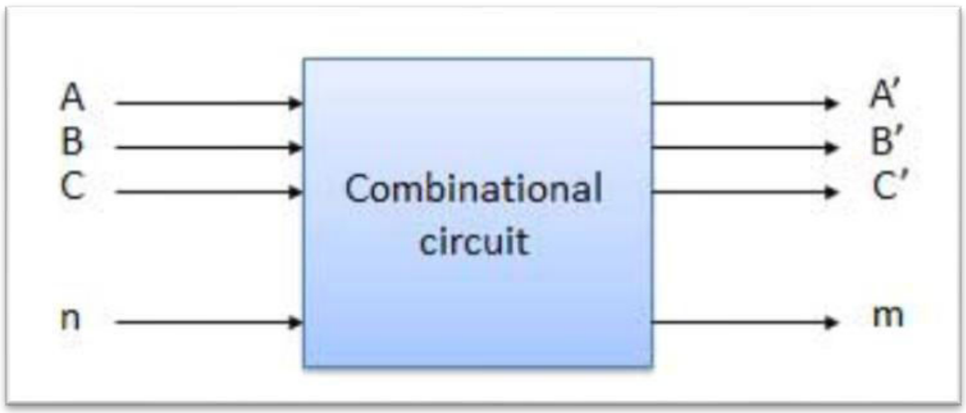


Therefore, POS minimal is, F = (A'+ C)(B' + C')

❖ Sequential and Combinational circuits

- A combinational circuit is a connected arrangement of logic gates with a set of input and outputs.
- The 'n' binary input variable come from an external source; the 'm' binary output variable go to an external destination, and in between there is an interconnection of logic gates.
- A combinational circuit transforms binary information from the given input data to the required output data.
- Combinational circuit are employed in digital computer for generating binary control decision and for providing digital components required for data processing.





Arithmetic Circuits

Digital computers perform a variety of arithmetic operations, like addition, subtraction etc. in this section we mainly discuss the Adder and Subtraction circuit.

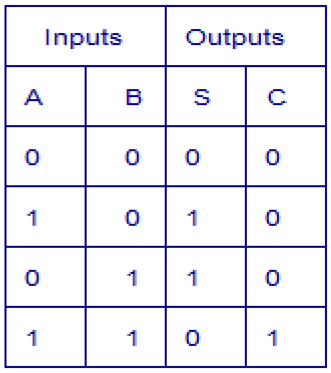
[A] Half-Adder

A combination circuit that performs the arithmetic addition of two bits is called a half-adder.

This circuit needs two inputs X and Y and two output variables S(sum) and C(carry).

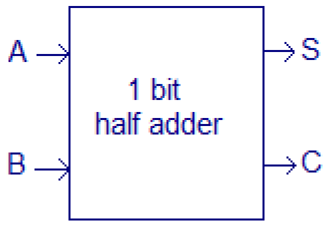
The truth table show that when both input A and B is high then output C produced 1-carry, while for other cases it will not generate any carry bit. The S output represents the least significant bit of the sum.

The logic diagram is show. It consists of an EX-OR gate and an AND gate.

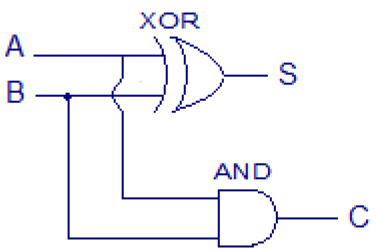


Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

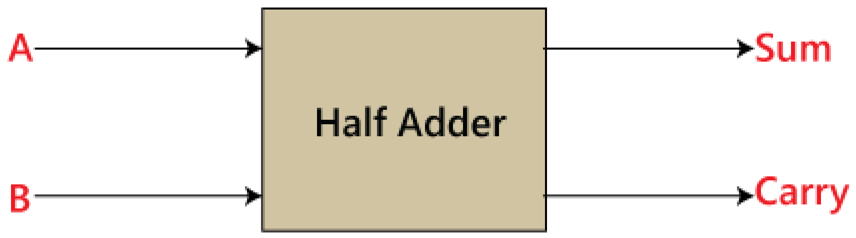
Truth table



Schematic



Realization

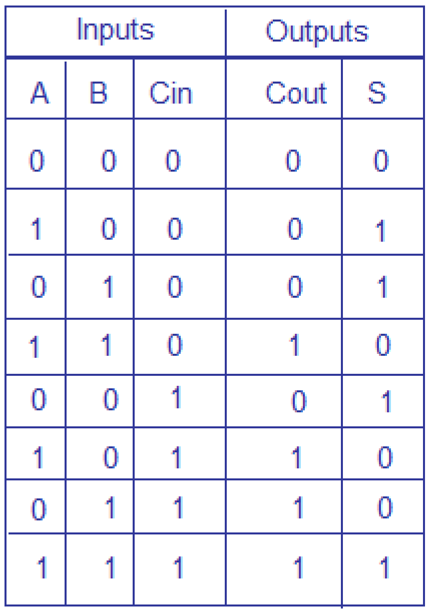


[B] Full-Adder

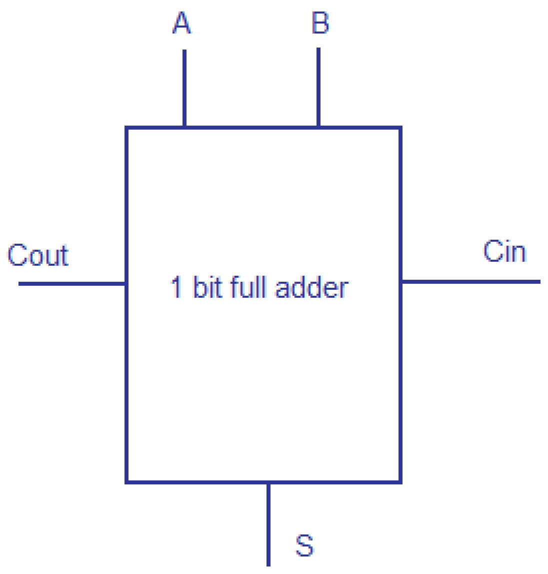
A full-adder is a combinational circuit that performs the arithmetic sum of three input bits.

It consists of three inputs and two output. When we want to add two binary number, each having two or more bits, the LSB (least significant bits) can be added by using a half-adder.

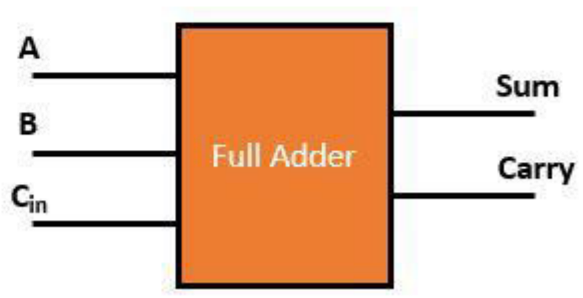
Two of the input variable, denoted by A and B, represent the two significant bits to be third input C, represent the carry from the previous lower significant position.



Inputs			Outputs	
A	B	Cin	Cout	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



1 bit full adder truth table & schematic


A
B
C_{in}
Full Adder
Sum
Carry